The OpenGL® Graphics System: A Specification (Version 2.0 - October 22, 2004)

Mark Segal
Kurt Akeley

Editor (version 1.1): Chris Frazier
Editor (versions 1.2-1.5): Jon Leech
Editors (version 2.0): Jon Leech and Pat Brown

Contents

List of Tables

Chapter 1

Introduction

or texturing is enabled) relies on the existence of a framebuffer. Further, some of OpenGL is specifically concerned with framebuffer manipulation.

1.3 Programmer's View of OpenGL

To the programmer, OpenGL is a set of commands that allow the specification of

Chapter 2

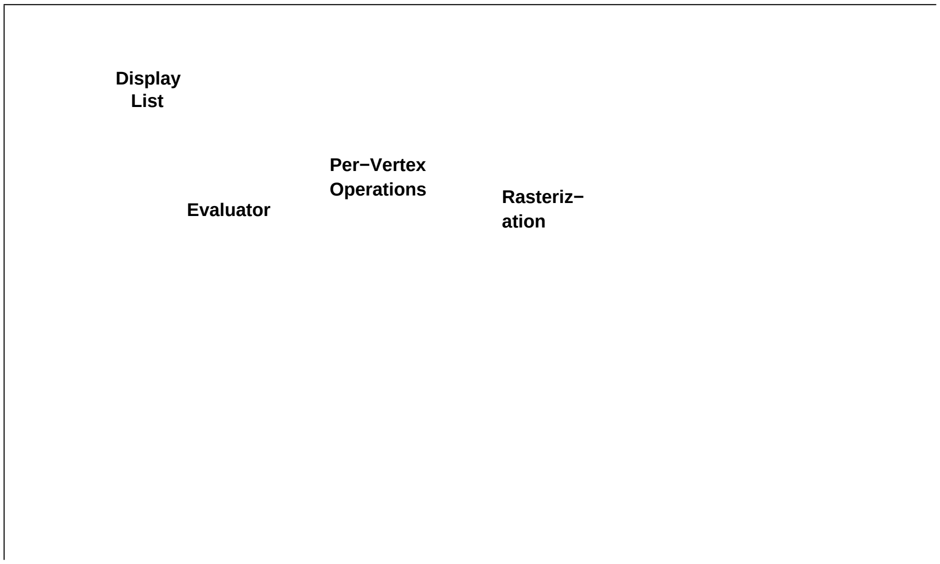
OpenGL Operation

2.1 OpenGL Fundamentals

OpenGL (henceforth, the “GL”) is concerned only with rendering into a frame-

Finally, command names, constants, and types are prefixed in the GL (by **gl**,

2.3. GL GL GL4sCOMMAND0(GL4sSYNT93)(AX)TJ/F34.798687.12353.2098-GL


Display
List
Per−Vertex
Operations
Evaluator
Rasteriz−
ation

Current

Figure 2.5. (a) A quad strip. (b) Independent quads. The numbers give the sequencing of the vertices between **Begin** and **End**.

2.6.2 Polygon Edges

Each edge of each primitive generated from a polygon, triangle strip, triangle fan, separate triangle set, quadrilateral strip, or separate quadrilateral set, is flagged as either *boundary* or *non-boundary*. These classifications are used during polygon rasterization; some modes affect the interpretation of polygon boundary edges (see section 3.5.4). By default, all edges are boundary edges, but the flagging of poly-

MAX_VERTEX_ATTRIBS available slots. Matrices are loaded into these slots in

should be normalized when converted to floating-point. If *normalized* is TRUE, fixed-point data are converted as specified in table 2.9; otherwise, the fixed-point values are converted directly.

The one, two, three, or four values in an array that correspond to a single vertex

Specifying an invalid *texture*

with one exception: the current normal coordinates, color, secondary color, color index, edge flag, fog coordinate, texture coordinates, and generic attributes are each indeterminate after the execution of **DrawElements**, if the corresponding array is enabled. Current values corresponding to disabled arrays are not modified by the execution of **DrawElements**.

The command

`void` **MultiDrawElements**(`enum` *mode*, `sizei` **count*,
`enum` *type*, `void` ***indices*, `sizei` *primcount*)

format	e_t	e_c	e_n	s_t	s_c	s_v	t

format

```
    DisableClientState(NORMAL_ARRAY);
  EnableClientState(VERTEX_ARRAY);
  VertexPointer(s_v, FLOAT, str, pointer + p_v);
}
```

If the number of supported texture units (the value of MAX_TEXTURE_COORDS) is m and the number of supported generic vertex attributes (the value of

2.9. Bi1123cUER OBJECTS

2.9.2 Array Indices in Buffer Objects

is exactly the same as the following sequence of commands:

Begin(POLYGON);
  Vertex2(x_1, y_1);
  Vertex2(x_2, y_1);
  Vertex2(x_2, y_2);
  Vertex2(x_1, y_2);
End();

The appropriate **Vertex2** command would be invoked depending on which of the **Rect** commands is issued.

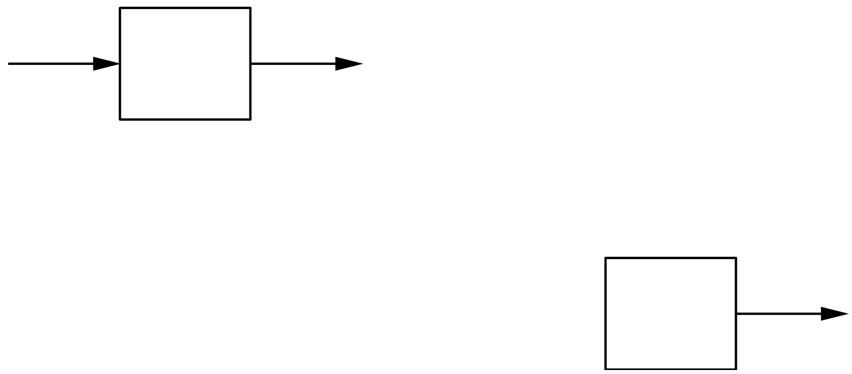


Figure 2.6. Vertex transformation sequence.

```
void
```

MultTransposeMatrix[fd](m);

is the same as the effect of

MultMatrix[fd](m^T);

The command

The active texture unit selector also selects the texture image unit accessed mode, one stack of at least two 4×4 matrices for each of COLOR, PROJECTION, and each texture coordinate set, TEXTURE; and a stack of at least 32 4×4 matrices for MODELVIEW. Each matrix stack has an associated stack pointer. Initially, there is only one matrix on each stack, and all matrices are set to the identity. The initial active texture unit selector is TEXTURE0, and the initial matrix mode is MODELVIEW.

2.11.3 Normal Transformation

where M_u is the upper leftmost 3x3 matrix taken from M.

Rescale multiplies the transformed normals by a scale factor

$$(n_x \quad n_y \quad n_z) = f \, (n_x \quad n_y \quad n_z)$$

If rescaling is disabled, then f

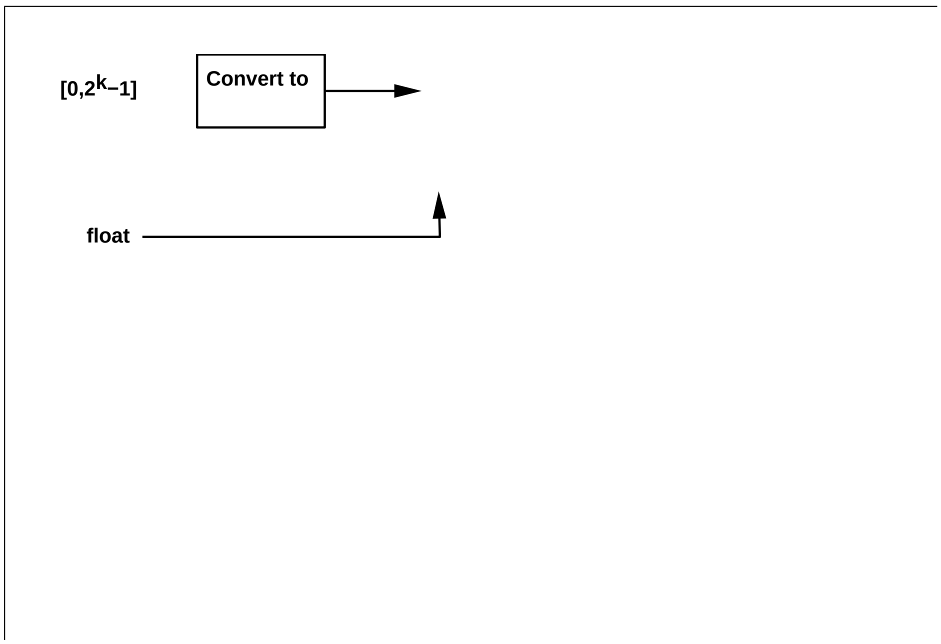
and let $m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$. Then the value assigned to an s coordinate (the first **TexGen** argument value is S) is $s = r_x/m + \frac{1}{2}$; the value assigned to a t coordinate is $t = r_{z\psi f e \psi i s}$

undefined. The user must ensure that the clip vertex and client-defined clip planes are defined in the same coordinate space.

Client-defined clip planes are enabled with the generic **Enable** command and disabled with the **Disable** command. The value of the argument to either command is CLIP_PLANEi where i is an integer between 0 and n; specifying a value of i enables or disables the plane equation with index i. The constants obey CLIP_PLANEi = CLIP_PLANE0 + i.

If the primitive under consideration is a point, then clipping passes it unchanged if it lies within the clip volume; otherwise, it is discarded. If the primitive isdii0199 mine27o0(ds15(gernt)-,27o06hen)-43o0(dipping)-43199 doe-27o0(dnoin)--43199 d isisdes istdt

Primitives rendered with clip planes must satisfy a complementarity crite-


[0,2^k–1]
Convert to
float

The value of A produced by lighting is the alpha value associated with $\mathbf{d}_{cm}$. A is always associated with the primary color $\mathbf{c}_{pri}$

Parameter	Name	Number of values
Material Parameters (**Material**)		
a_{cm}	`AMBIENT`	4
d_{cm}	`DIFFUSE`	4
		4d

shader is active such property changes are not guaranteed to update material parameters, defined in table 2.11, until the following **End** command.

2.14.4 Lighting State

The state required for lighting consists of all of the lighting parameters (front and back material parameters, lighting model parameters, and at least 8 sets of light parameters), a bit indicating whether a back color distinct from the front ahether Sighting Senble Sistble In-3751ithe-435(in)ital state sighting paramet

The final color index is

$$c = \min\{c, s_m\}.$$

2.14.9 Final Color Processing

For an RGBA color, each color component (which lies in $[0, 1]$) is converted (by rounding to nearest) to a fixed-point value with m bits. We assume that the fixed-point representation used represents each value during rasterization, and are described in section 3.11. A single program object can contain both vertex and fragment shaders.

When the program object currently in use includes a vertex shader, its vertex shader is considered *active* and is used to process vertices. If the program object

Vertex Attributes

Vertex shaders can access built-in vertex attribute variables corresponding to the per-vertex state set by commands such as **Vertex**, **Normal**, **Color**. Vertex shaders can also define named attribute variables, which are bound to the generic vertex attributes that are 06(s(v))-33952 0 Td[(V)10i8rui*0.909 Tf 5.67 0 Td[64.941

have been linked successfully. The link could have failed because the number of active attributes exceeded the limit.

The name of the selected attribute is returned as a null-terminated string in *name*. The actual number of characters written into *name*, excluding the null terminator, is returned in *length*. If *length* was bound previouslybound2evitoundviound

Uniform Variables

Shaders can declare named *uniform variables*, as described in the OpenGL Shading Language Specification. Values for these uniforms are constant over a primitive, and typically they are constant across many primitives. Uniforms are program object-specific state. They retain their values once loaded, and their values are restored whenever a program object is used, as long as the program object has not been re-linked. A uniform is considered *active* if it is determined by the compiler and linker that the uniform will actually be accessed when the executable code is executed. In cases where the compiler and linker cannot make a conclusive determination, the uniform will be considered active.

uniform array, then the location of the first element of that array can be retrieved by either using the name of the uniform array, or the name of the uniform array appended with `"[0]"`.

To determine the set of active uniform attributes used by a program, and to determine their sizes and types, use the command:

void **GetActiveUniform**(uint *program*, uint *index*,
  sizei *bufSize*, sizei **length*, int **size*, enum **type*,
  char **name*);

This command provides information about the uniform selected by *index*. An *index* of 0 selects the first active uniform, and an *index* of ACTIVE_

image unit number i. The values of i range from zero to the implementation-dependent maximum supported number of texture image units.

The type of the sampler identifies the target on the texture image unit. The texture object bound to that texture image unit's target is then used for the texture

- Clipping, including client-defined clip planes (section 2.12).
-

Texture lookups involving textures with depth component data can either reit cannot be executed then no fragments will be rendered, and **Begin**, **RasterPos**, or any command that performs an implicit **Begin** will generate the error INVALID_OPERATION.

This error is generated by **Begin**, **RasterPos**, or any command that performs an implicit **Begin** if:

- any two active samplers in the current program object are of different types, but refer to the same texture image unit,

- any active sampler in the current program object refers to a texture image unit where fixed-function fragment processing accesses a texture target that

Chapter 3

Rasterization

Rasterization is the process by which a primitive is converted to a two-dimensional image. Each point of this image contains such information as color and depth. Thus, rasterizing a primitive consists of two parts. The first is to determine which squares of an integer grid in window coordinates are occupied by the primitive. The second is assigning a depth value and one or more color values to each such square. The results of this process are passed on to the next stage of the GL (per-fragment operations), which uses the information to update the appropriate locations in the framebuffer. Figure 3.1 diagrams the rasterization process. The color values

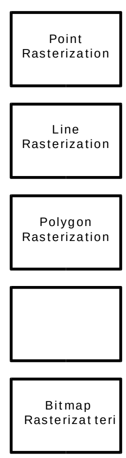


Figure 3.1. Rasterization.

Several factors affect rasterization. Lines and polygons may be stippled. Points may be given differing diameters and line segments differing widths. A point, line segment, or polygon may be antialiased.

uniform intensity. The square is called a *fragment square* and has lower left corner (x, y) and upper right corner $(x+1, y+1)$. We recognize that this simple box filter may not produce the most favorable antialiasing results, but it provides a simple, well-defined model.

A GL implementation may use other methods to perform antialiasing, subject to the following conditions:

1. If f_1 and f_2 are two fragments, and the portion of f_1 covered by some primitive is a subset of the corresponding portion of f_2 covered by the primitive, then the coverage computed for f_1 must be less than or equal to that computed for f_2.

2. The coverage computation for a fragment ff

exact positions, rather than regions or areas, and each is referred to as a sample point. The sample points associated with a pixel may be located inside or outside of the unit square that is considered to bound the pixel. Furthermore, the relative locations of sample points may be identical for each pixel in the framebuffer, or they may differ.

All fragments produced in rasterizing a non-antialiased point are assigned the

the fragment, and x_w and y_w are the exact, unrounded window coordinates of the vertex for the point.

The widths supported for point sprites must be a superset of those supported

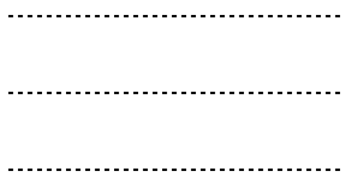


Figure 3.4. Visualization of Bresenham's algorithm. A portion of a line segment is shown. A diamond shaped region of height 1 is placed around each fragment center; those regions that the line segment exits cause rasterization to produce corresponding fragments.

1.

$$t = (\mathbf{p}_r - \mathbf{p}_a$$

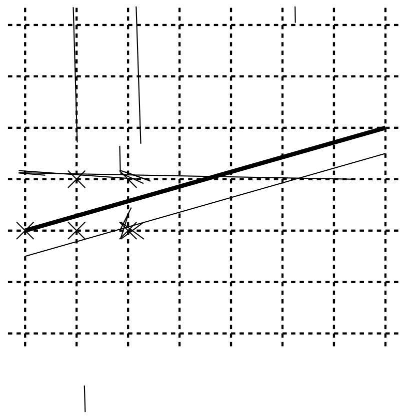


Figure 3.5. Rasterization of non-antialiased wide lines. x-major line segments are

this may yield acceptable results for color values (it *must* be used for depth values), but will normally lead to unacceptable distortion effects if used for texture coordinates or clip w coordinates.

For a polygon with more than three edges, we require only that a convex combination of the values of the datum at the polygon's vertices can be used to obtain the value assigned to each fragment produced by the rasterization algorithm. That is, it must be the case that at every fragment

$$f =$$

Polygon stippling may be enabled or disabled with

Boolean state values POLYGON_OFFSET_POINT,

3.5.7 Polygon Rasterization State

The state required for polygon rasterization consists of a polygon stipple pattern, whether stippling is enabled or disabled, the current state of polygon antialiasing (enabled or disabled), the current values of the **PolygonMode** setting for each of front and back facing polygons, whether point, line, and fill mode polygon offsets are enabled or disabled, and the factor and bias values of the polygon offset equa-

Commands, **Color Table State and Proxy State**, **Color Table Lookup**, **Post Convolution Color Table Lookup**, and **Post Color Matrix Color Table Lookup**, as well as the query commands described in section 6.1.7.

2. Convolution, including all commands and enumerants described in subsections **Convolution Filter Specification**, **Alternate Convolution Filter Specification Commands**, and **Convolution**, as well as the query commands described in section 6.1.8.

3. Color matrix, including all commands and enumerants described in subsections **Color Matrix Specification** and

Parameter Name

Map Name	

RGBA, with zero-sized components). The initial value of the scale parameters is (1,1,1,1) and the initial value of the bias parameters is (0,0,0,0).

In addition to the color lookup tables, partially instantiated proxy color lookup tables are maintained. Each proxy table includes width and internal format state be one of the formats in table 3.15 or table 3.16, other than the `DEPTH` formats in

Special facilities are provided for the definition of two-dimensional *separable* filters – filters whose image can be represented as the product of two one-dimensional images, rather than as full two-dimensional images. A two-dimensional separable convolution filter is specified with meanings, as the equivalent arguments of **ConvolutionFilter2D**. *format* is taken to be `RGBA`.

The command

`void` **CopyConvolutionFilter1D**(`enum` *target*, `enum` *internalformat*, `int` *x*, `int` *y*, `sizei` *width*);

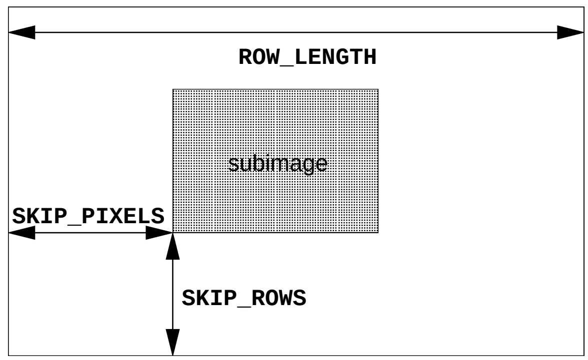
defines a one-dimensional filter in exactly the manner of **ConvolutionFilter1D**, except that image data are taken from the framebuffer, rather than from client memory. *target* must be `CONVOLUTION_1D`. *x*, *y*, and *width* correspond precisely to the corresponding arguments of **CopyPixels** (refer to section 4.3.3); they specify the image's *width* and the lower left (x, y) coordinates of the framebuffer region to be copied. The image is taken from the framebuffer exactly as if these arguments were passed to **CopyPixels** with argument *type* set to `COLOR` and *height* set toseLOR

set to zero. If the histogram table would be accomodated by **Histogram** called with *target* set to HISTOGRAM, the proxy state values are set exactly as though the actual histogram table were being specified. Calling **Histogram** with *target* PROXY_HISTOGRAM has no effect on the actual histogram table.

There is no image associated with PROXY_HISTOGRAM. It cannot be used as a histogram, and its image must never queried using **GetHistogram**. The error INVALID_ENUM results if this is attempted.

3Minmax T5F3474 -d(speci cati0(oM)]TJ/F34 10.909 T067 101249 Td[(T3474 -m(Minmax)-25t2(5F3474 -

Format Name	Element Meaning and Order	Target Buffer
COLOR_INDEX	Color Index	Color
STENCIL_INDEX	Stencil Index	Stencil
DEPTH_COMPONENT		


ROW_LENGTH
subimage
SKIP_PIXELS
SKIP_ROWS

UNSIGNED_SHORT_5_6_5:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st Component					2nd						3rd				

UNSIGNED_SHORT_5_6_5_REV:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
3rd					2nd						1st Component				

UNSIGNED_SHORT_4_4_4_4:

propriate formula in table 2.9 (section 2.14). For packed pixel types, each element

Stencil indices are masked by $2^n - 1$, where n is the number of bits in the stencil buffer.

Conversion to Fragments

The conversion of a group to fragments is controlled with

void **PixelZoom**(float z_x, float z_y);

Base Filter Format	R	G	B	A
ALPHA	R_s	G_s	B_s	$A_s \quad A_f$
LUMINANCE	$R_s \quad L_f$	$G_s \quad L_f$	$B_s \quad L_f$	

the RGBA color to be used as the image border. Integer color components are interpreted linearly such that the most positive integer maps to 1.0, and the most

group component values that are outside the representable range.

If the **Minmax** *sink* parameter is FALSE, minmax operation has no effect on the stream of pixel groups being processed. Otherwise, all RGBA pixel groups are discarded immediately after the minmax operation is completed. No pixel fragments are generated, no change is made to texture memory contents, and no pixel values are returned. However, texture object state is modified whether or not pixel groups are discarded.

3.6.6 Pixel Rectangle Multisample Rasterization

If MULTISAMPLE is enabled, and the value of SAMPLE_BUFFERS is one, then pixel rectangles are rasterized using the following algorithm. Let (X_{rp}, Y_{rp}) be the current raster position. (If the current raster position is invalid, then **DrawPixels** is

The GL provides two ways to specify the details of how texturing of a primitive is effected. The first is referred to as fixed-functionality, and is described in

Sized Internal Format	Base Internal Format	R bits	G bits	B	A	L	I	D

`INVALID`

two-dimensional image tokens such as TEXTURE_CUBE_MAP_POSITIVE_X are used when specifying, updating, or querying one of a cube map's six two-dimensional images, but when enabling cube map texturing or binding to a cube map texture object (that is when the cube ng(us59)-2a0(is)-2a(is)-239oles59isus59

ture array that is modified. If *level* is less than zero or greater than the base 2 logarithm of the maximum texture width, height, or depth, the error INVALID_VALUE is generated.

TexSubImage3D

$$z + d > d$$

Counting from zero, the n

the GL provides no specific image formats, using any of the six generic compressed internal formats as *internalformat*

TEXTURE

Major Axis Direction	Target	s_c	t_c	m_a

and

k

Mipmapping

TEXTURE_MIN_FILTER values NEAREST_MIPMAP_NEAREST, NEAREST_MIPMAP_LINEAR, LINEAR_MIPMAP_NEAREST, and LINEAR

For mipmap filters NEAREST_MIPMAP_LINEAR

level of detail, two integers describing the base and maximum mipmap array, a boolean flag indicating whether the texture is resident, a boolean indicating whether automatic mipmap generation should be performed, three integers describing the depth texture mode, compare mode, and compare function, and the priority associated with each set of properties. The value of the resident flag is

SRCn_RGB

DEPTH_TEXTURE_

If the value of TEXTURE_MAG_FILTER is not NEAREST, or the value of TEXTURE_

results of texture blending are undefined.

the eye, $(0, 0, 0, 1)$ in eye coordinates, to the fragment center. The equation and the a color in the color index buffer (buffers are discussed in chapter 4). The value of values that can be held in uniform variable storage for a fragment shader. A link error will be generated if an attempt is made to utilize more than the space available for fragment shader uniform variables.

Fragment shaders can read varying variables that correspond to the attributes of the fragments produced by rasterization. The OpenGL Shading Language Spec-

The GL returns a four-component vector (R

section 2.11

`gl_FragDepth`

does differ, it should be defined relative to window, not screen, coordinates, so that rendering results are invariant with respect to window position.

Next, if SAMPLE_ALPHA_TO_ONE is enabled, each alpha value is replaced by the maximum representable alpha value. Otherwise, the alpha values are not changed.

Finally, if SAMPLE_COVERAGE

If **BeginQuery** is called with an *id* of zero, while another query is already in progress with the same *target*, or where *id*

Dithering is enabled with **Enable** and disabled with **Disable** using the symbolic constant `DITHER`. The state required is thus a single bit. Initially, dithering is enabled.

4.1.10 Logical Operation

Finally, a logical operation is applied between the incoming fragment's color or index values and the color or index values stored at the corresponding location in

Argument value	Operation
CLEAR	0
AND	$s \quad d$

symbolic	front	

The depth buffer can be enabled or disabled for writing void **Clear**(bitfield *buf*);

is the bitwise OR of a number of values indicating which buffers are to be cleared. The values are COLOR_BUFFER_BIT, DEPTH_BUFFER_BIT, STENCIL_BUFFER_BIT, and

location in the framebuffer, subject to the current front stencil mask (set with **StencilMask** or **StencilMaskSeparate**). If a depth component is present, and the setting of **DepthMask** is not `FALSE`, is also written to the framebuffer; the setting of **DepthTest** is ignored.

The error `INVALID_OPERATION` results if there is no stencil buffer.

4.3.2 Reading Pixels

The method for reading pixels from the framebuffer and placing them in client memory is diagrammed in figure 4.2

Conversion of RGBA values

type Parameter	GL Data Type	Component Conversion Formula
UNSIGNED_BYTE	ubyte	$c = (2$

target

for $i = q_1$ to $q_2 - 1$ step 1.

5.2. SELECTION

LoadName replaces the value on the top of the stack with *name*. Loading a name onto an empty stack generates the error written. The minimum and maximum (each of which lies in the range [0 *buffer* is a pointer to an array of floating-point values into which feedback information will be placed, and *n* is a number indicating the maximum number of values that can be written to that array. *type* is a symbolic constant describing the information to be fed back for each vertex (see figure 5.2). The error `INVALID_OPERATION` results if the GL is placed in feedback mode before a call to **FeedbackBuffer**

Type

feedback-list:
    feedback-item feedback-list
    feedback-item

feedback-item:
    point
    line-segment
    polygon
    bitmap
    pixel-rectangle
    passthrough

point:
    `POINT_TOKEN` vertex
line-segment:
    `LINE_TOKEN` vertex vertex
    `LINE_RESET_`

`void` **CallLists**(`sizei` *n*, `enum` *type*, `void` **lists* the effect of creating an empty display list for each of the indices $n, \ldots, n+s-1$, so that these indices all become used.

TexImage3D, **TexImage2D**, **TexImage1D**, **Histogram**, and

Chapter 6

Queries of *value* TEXTURE_WIDTH

6.1.9 Histogram Query

target must be `MINMAX`. *type* and *format* accept the same values as do the corresponding parameters of **GetTexImage**. A one-dimensional image of width 2 is returned to *values*

of bits allowed is a function of the implementation's maximum viewport dimensions (

6.1.75.7910(UER)65(YING)-250(GL)-250(ST)93(A)111(TE)]TJ/F34909Tf125.342.292-16.1.256

shader

returns properties of the shader object named *shader* in *params*. The parameter value to return is specified by *pname*.

If *pname* is SHADER_TYPE, VERTEX_SHADER

The command

void **GetAttachedShaders**(

table

Stack	Attribute	Constant

Type code	Explanation
B	Boolean

6.2 State Tables

The tables on the following pages indicate which state variables are obtained with what commands. State variables that can be obtained using any of **GetBooleanv**, **GetIntegerv**, **GetFloatv**, or **GetDoublev** are listed with just one of these commands – the one that is most appropriate given the type of the data to be returned. 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**, **GetFloatv**, and **GetDoublev**. State variables

6.2. STATE TABLES

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
CLIENT_ACTIVE_TEXTURE	$Z_{2\square}$	**GetIntegerv**	`TEXTURE0`	Client active texture unit selector	2.7	vertex-array
VERTEX_ARRAY	B	**IsEnabled**				

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute

TEXTURE_

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
LIGHTING	B	**IsEnabled**	*False*	True if lighting is enabled	2.14.1	

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
AMBIENT	$8 \times C$	**GetLightfv**	(0.0,0.0,0.0,1.0)	Ambient intensity of light *i*	2.14.1	lighting
DIFFUSE	$8 \times C$	**GetLightfv**	see table 2.10	Diffuse intensity of light *i*	2.14.1	lighting
SPECULAR	$8 \times C$	**GetLightfv**				

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
POINT_SIZE	R^+	**GetFloatv**	1.0	Point size	3.3	point
POINT_SMOOTH	B	**IsEnabled**	*False*	Point antialiasing on	3.3	point/enable
POINT_SPRITE	B	**IsEnabled**	*False*	Point sprite enable		

Initial

Get Cmnd

Type

Get value

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
TEXTURE_BORDER_						

6.2. STATE TABLES

6.2. STATE TABLES

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
UNPACK_SWAP_						

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute

Get value	Type	Get Cmnd

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute
VERTEX						

Get value	Type	Get Cmnd	Minimum Value	Description	Sec.	Attribute

Get value	Type	Get Cmnd	Minimum Value	Description	Sec.	Attribute
EXTENSIONS						

Get value	Type	Get Cmnd	Initial Value	Description	Sec.	Attribute

Corollary 3 *Images rendered into different color buffers sharing the same framebuffer, either simultaneously or separately using the same command sequence, are pixel identical.*

Rule 4 *The same vertex or fragment shader will produce the same result when run multiple times with the same input. The wording 'the same shader' means a program object that is populated with the same source strings, which are compiled*
685 6521hlink10(esd)-241(prossiby)-2411(ultiple)-321h
psing the tame tG Tstted-250(Tvct)o,

17.

Appendix C

Version 1.1

OpenGL version 1.1 is the first revision since the original version 1.0 was released on 1 July 1992. Version 1.1 is upward compatible with version 1.0, meaning that any program that runs with a 1.0 GL implementation will also run unchanged with a 1.1 GL implementation. Several additions were made to the GL, especially to the texture mapping capabilities, but also to the geometry and fragment operations. Following are brief descriptions of each addition.

C.1 Vertex Array

Arrays of vertex data may be transferred to the GL with many fewer commands than were previously necessary. Six arrays are defined, one each storing vertex positions, normal coordinates, colors, color indices, texture coordinates, and edge flags. The arrays may be specified and enabled independently, or one of the pre-defined configurations may be selected with a single command.

The primary goal was to decrease the number of subroutine calls required to transfer non-display listed geometry data to the GL. A secondary goal was to improve the efficiency of the transfer; especially to allow direct memory access (DMA) hardware to be used to effect the transfer. 050Dd33.aat7(eaco)-32oserays to

becauseray5 enc4 lica(liste5)-330(th4)-interfardw7 ah4 were0-no(thnh4)-i00.2aag92(er0.2gd3.l3se)-307(of)-30

C.2 Polygon Offset

Depth values of fragments generated by the rasterization of a polygon may be shifted toward or away from the origin, as an affine function of the window coordinate depth slope of the polygon. Shifted depth values allow coplanar geometry, especially facet outlines, to be rendered without depth buffer artifacts. They may also be used by future shadow generation algorithms.

The additions match those of the `EXT_polygon_offset` extension, with two exceptions. First, the offset is enabled separately for `POINT`, `LINE`, and `FILL` rasterization modes, all sharing a single affine function definition. (Shifting the depth values of the outline fragments, instead of the fill fragments, allows the contents of the depth buffer to be maintained correctly.) Second, the offset bias is specified in units of depth buffer resolution, rather than in the [0,1] depth range.

C.3 Logical Operation

3.

Jeremy Morris, 3Dlabs
Israel Pinkas, Intel
Bimal Poddar, IBM
Lyle Ramshaw, Digital Equipment
Randi Rost, Hewlett Packard
John Schimpf, Silicon I8paphic

D.3.4. PACKED PIXEL FORMATS

Three independent lookups may be performed: prior to convolution; after convolution and prior to color matrix transformation; after color matrix transformation and prior to gathering pipeline statistics.

Methods to initialize the color lookup tables from the framebuffer, in addition to the standard memory source mechanisms, are provided.

Portions of a color lookup table may be redefined without reinitializing the

D.9.4 Pixel

David Blythe, Silicon Graphics
Jon Brewster, Hewlett Packard
Dan Brokenshire, IBM

Phil Lacroute, Silicon Graphics
Prakash Ladia, S3
Jon Leech, Silicon Graphics
Kevin Lefebvre, Hewlett Packard
David Ligon, Raycer Graphics
Kent Lin, S3
Dan McCabe, S3
Jack Middleton, Sun
Tim Misner, Intel
Bill Mitchell, National Institute of Standards
Jeremy Morris, 3Dlabs
Gene Munce, Intel
William Newhall, Real3D
Matthew Papakipos, Nvidia / Raycer
Garry Paxinos, Metro Link
Hanspeter Pfister, Mitsubishi Electric
Richard Pimentel, Parametric Technology
Bimal Poddar, IBM / Intel
Rob Putney, IBM
Mike Quinlan, Real3D
Nate Robins, University of Utah
Detlef Roettger, Elsa
Randi Rost, Hewlett Packard
Kevin Rushforth, Sun
Richard S. Wright, Real3D
Hock San Lee, Microsoft
John Schimpf, Silicon Graphics
Stefan Seeboth, ELSA
Mark Segal, Silicon Graphics
Bob Seitsinger, S3

kiti2i2i2iColshforth,Sharpl3D
WilliaMitchell,, IBM
Richard NewhalSwee(,)-250(IBM)]TJ 0 -13.549 Td[(Richard)-2Robheboth,uck,er
Garryw0(Ne-250(S3)]TJ 0 -113.549 Td[(Richard)-2Schimpf,)-Tyne,

Appendix E

Appendix F

Version 1.3

one cube face two-dimensional image based on the largest magnitude coordinate (the major axis). A new (st) is calculated by dividing the two other coordinates (the minor axes values) by the major axis value, and the new (st) is used to lookup into the selected two-dimensional texture image face of the cube map.

Two new texture coordinate generation modes are provided for use in conjunction with cube map texturing. The `REFLECTION_`

image, the color returned is derived only from border texels. This behavior mirrors the behavior of the texture edge clamp mode introduced by OpenGL 1.2.

Bill Clifford, Intel
Bill Mannel, SGI
Bimal Poddar, Intel
Bob Beretta, Apple
Brent Insko, NVIDIA
Brian Goldiez, UCF
Brian Greenstone, Apple
Brian Paul, VA Linux
Brian Sharp, GLSetup
Bruce D'Amora, IBM
Bruce Stockwell, Compaq
Chris Brady, Alt.software
Chris Frazier, Raycer
Chris Hall, 3dlabs
Chris Hecker, GLSetup
Chris Lane, Intel
Chris Thornborrow, PixelFusion
Christopher Fraser, IMG
Chuck Smith, Intelligraphics
Craig Dunwoody, SGI
Dairsie Latimer, PixelFusion
Dale Kirkland, 3Dlabs / Intergraph
Dan Brokenshire, IBM
Dan Ginsburg, ATI
Dan McCabe, S3
Dave Aronson, Microsoft
Dave Gosselin, ATI
Dave Shreiner, SGI
Dave Zenz, Dell
David Aronson, Microsoft
David Blythe, SGI
David Kirk, NVIDIA
David Story, SGI
David Yu, SGI
Deanna Hohn, 3dfx
Dick Coulter, Silicon Magic
Don Mullis, 3dfx
Eamon O Dea, PixelFusion
Edward (Chip) Hill, Pixelfusion
Eiji Obata, NEC

Martin Amon, 3dfx
Martina Sourada, ATI
Matt Lavoie, Pixelfusion
Matt Russo, Matrox
Matthew Papakipos, NVIDIA
Michael Gold, NVIDIA
Miriam Geller, SGI
Morgan Von Essen, Metro Link
Naruki Aruga, PFU
Nathan Tuck, Raycer Graphics
Neil Trevett, 3Dlabs
Newton Cheung, S3
Nick Triantos, NVIDIA
Patrick Brown, Intel
Paul Jensen, 3dfx
Paul Keller, NVIDIA
Paul Martz, HP
Paula Womack, 3dfx
Peter Doenges, Evans & Sutherland
Peter Graffagnino, Apple
Phil Huxley, 3Dlabs
Ralf Biermann, Elsa AG
Randi Rost, 3Dlabs
Renee Rashid, Micron
Rich Johnson, HP
Richard Pimentel, PTC
Richard Schlein, Apple
Rick Hammerstone, ATI
Rik Faith, VA Linux

Tim Kelley, Real 3D
Tom Frisinger, ATI
Victor Vedovato, Micron
Vikram Simha, MERL
Yanjun Zhang, Sun
Zahid Hussain, TI

Appendix G

Version 1.4

OpenGL version 1.4, released on July 24, 2002, is the fourth revision since the original version 1.0. Version 1.4 is upward compatible with earlier versions, mean-

Texture environment crossbar was promoted from the ARB_texture_env_crossbar extension.

Luc Leblanc, Discreet
Jon Leech, SGI
Kevin Lefebvre, HP
Bill Licea-Kane, ATI
Barthold Lichtenbelt, 3Dlabs
Kent Lin, Intelc Leblanc, D2at0Licea-Kane,8 Td[(Lrld)-250(Lichrak0 -13.549 Td[(Barthold)-250-250(LRo

Neil Trevett, 3Dlabs
Nick Triantos, NVIDIA
Douglas Twilleager, Sun
Shawn Underwood, SGI
Steve Urquhart, Intelligraphics
Victor Vedovato, ATI
Daniel Vogel, Epic Games
Mik Wells, Softimage

I.1.3 OpenGL Shading Language

After the initial version of the OpenGL 2.0 was released, several more minor corrections were made in the specification revision approved on October 22, 2004:

- Corrected name of the fog source from FOG

J.7 Cube Map Textures

J.26 High-Level Vertex Programming

The name string for high-level vertex programming is ARB_vertex_shader. It

J.32 Multiple Render Targets

The name string for multiple render targets is _ _

Index