

GSR-010***rev 1.00.00******OpenGL ES 1.1 Common-Lite
(Addition Only)***

변경 내역

버전	배포 날짜	변경사항
1.00.00	2005.10.12	초안작성

지적 재산권에 대한 공지

본 문서 및 문서와 함께 제공되는 일체의 정보는 SK 텔레콤의 WIPI 추가 규격을 명시함으로써, SK 텔레콤에 납품하는 단말기 제조사들 및 콘텐츠 개발업체가 WIPI 추가 규격을 구현하도록 하기 위함이다. 본 문서 및 문서와 함께 제공되는 일체의 정보는 SK 텔레콤의 소유물이다. 이의 일부 또는 전부는 SKT WIPI 를 위해서만 SKT 의 승인 아래 복사되거나 배포될 수 있다.

에스케이텔레콤 주식회사

목 차

1 일반사항	6
1.1 개요	6
1.2 목적	6
1.3 규격의 범위.....	6
1.4 용어 정의	6
2 의존성	7
2.1 WIPI 의존성.....	7
2.2 GIGA 의존성	7
2.3 GSR간 의존성	7
2.4 OAT 의존성.....	7
3 제조사 포팅 가이드	8
4 개발자 가이드	9
4.1 Fundamentals	9
4.2 Type 정의.....	9
4.3 Basic 3D API Overview (OpenGL/ES 1.0 Common addition only)	9
4.4 Basic 3D API (OpenGL/ES 1.0 Common addition only) Description	13

세부목차

1	일반사항	6
1.1	개요	6
1.2	목적	6
1.3	규격의 범위.....	6
1.4	용어 정의	6
2	의존성	7
2.1	WIPI 의존성.....	7
2.2	GIGA 의존성	7
2.3	GSR간 의존성	7
2.4	OAT 의존성.....	7
3	제조사 포팅 가이드	8
4	개발자 가이드	9
4.1	Fundamentals	9
4.1.1	Matrix Convention	9
4.1.2	Coordinate system	9
4.2	Type 정의.....	9
4.3	Basic 3D API Overview (OpenGL/ES 1.0 Common addition only)	9
4.4	Basic 3D API (OpenGL/ES 1.0 Common addition only) Description	13
4.4.1	OEMC_glBindBuffer.....	13
4.4.2	OEMC_glBufferData	15
4.4.3	OEMC_glBufferSubData.....	17
4.4.4	OEMC_glClipPlanex	19
4.4.5	OEMC_glColor4ub.....	21
4.4.6	OEMC_glDeleteBuffers.....	22
4.4.7	OEMC_glGetBooleanv	23
4.4.8	OEMC_glGenBuffers	25
4.4.9	OEMC_glGetBufferParameteriv	26
4.4.10	OEMC_glGetClipPlanex	28
4.4.11	OEMC_glGetFixedv	29
4.4.12	OEMC_glGetLightxv	32
4.4.13	OEMC_glGetMaterialxv	35
4.4.14	OEMC_glGetPointerv	37
4.4.15	OEMC_glGetTexEnviv	38

4.4.16 OEMC_glGetTexEnvxv	40
4.4.17 OEMC_glGetTexParameteriv	42
4.4.18 OEMC_glGetTexParameterxv	44
4.4.19 OEMC_gllsBuffer	46
4.4.20 OEMC_gllsEnabled	47
4.4.21 OEMC_gllsTexture.....	49
4.4.22 OEMC_glLightModelxv	50
4.4.23 OEMC_glPointParameterx	52
4.4.24 OEMC_glPointParameterxv.....	54
4.4.25 OEMC_glTexEnvi.....	56
4.4.26 OEMC_glTexEnviv	59
4.4.27 OEMC_glTexParameteri	62
4.4.28 OEMC_glTexParameteriv	66
4.4.29 OEMC_glTexParameterxv	70
4.4.30 OEMC_glCurrentPaletteMatrixOES.....	74
4.4.31 OEMC_glLoadPaletteFromModelViewMatrixOES	75
4.4.32 OEMC_glMatrixIndexPointerOES.....	76
4.4.33 OEMC_glWeightPointerOES	78
4.4.34 OEMC_glPointSizePointerOES	80
4.4.35 OEMC_glDrawTexsOES	82
4.4.36 OEMC_glDrawTexiOES	84
4.4.37 OEMC_glDrawTexxOES	86
4.4.38 OEMC_glDrawTexsvOES	88
4.4.39 OEMC_glDrawTexivOES	90
4.4.40 OEMC_glDrawTexxvOES	92

1 일반사항

1.1 개요

OpenGL/ES 1.1 Common-Lite (Addition Only)에 대한 요구사항 및 규격에 대해서 설명하기로 한다.

1.2 목적

본 규격은 SK Telecom(주)(이하 “SK Telecom” 이라 한다.)의 GSR-010 (OpenGL/ES 1.1 Common-Lite Addition Only)에 대한 요구사항 및 규격을 명시하는데 목적이 있다

1.3 규격의 범위

본 규격은 SK Telecom에서 서비스 예정인 GSR-010을 위한 규격에 대하여 기술한다.

1.4 용어 정의

본 문서에서 언급되는 용어들을 정의한다.

2 의존성

본 규격 구현을 위하여, 필요로 하는 WIPI 버전과, 타 WSR 및 본 규격의 검증을 위해 필요한 OAT 버전을 명시한다.

2.1 WIPI 의존성

본 WSR 규격은 아래 WIPI Core 버전 이상이 적용된 단말에 포팅 되어야 한다.

Base WIPI Core 버전
WIPI 1.08.08

2.2 GIGA 의존성

본 GSR 규격은 아래 GIGA Core 버전 이상이 적용된 단말에 포팅 되어야 한다.

Base GIGA Core 버전
GIGA 2.00.00

2.3 GSR간 의존성

의존성을 갖는 GSR	설명
GSR-003	EGL 1.0
GSR-001	OpenGL / ES 1.0 Common-Lite
GSR-016	EGL 1.1

2.4 OAT 의존성

2.4.1.1 본 규격의 테스트를 위해서는 다음 버전의 OAT가 필요하다.

OAT 버전

3 제조사 포팅 가이드

추후 배포 예정임.

4 개발자 가이드

4.1 Fundamentals

4.1.1 Matrix Convention

Prefix convention is used. That is $R = M2 * M1 * V$, where R is the result matrix, M1 and M2 are arbitrary matrices, and V is a vertex vector.

4.1.2 Coordinate system

Right-handed coordinate system is used.

4.2 Type 정의

Type	Description
M_GLboolean	unsigned char
M_GLubyte	unsigned char
M_GLshort	short
M_GLenum	unsigned int
M_GLint	int
M_GLuint	unsigned int
M_GLsizei	int
M_GLfixed	int
M_GLintptr	int
M_GLsizeiptr	int
M_GLvoid	void

■ API Prefix

Type	Define	Adapt
M_GL	OpenGL ES 1.1 common profile	Mandatory

4.3 Basic 3D API Overview (OpenGL/ES 1.0 Common addition only)

Function	Description
void OEMC_glBindBuffer(M_GLenum target, M_GLuint buffer)	bind a named buffer to a target

void OEMC_glBufferData(M_GLenum target, M_GLsizei size, const M_GLvoid *data, M_GLenum usage)	creates and initializes the data store of a buffer object.
void OEMC_glBufferSubData(M_GLenum target, M_GLintptr offset, M_GLsizei size, const M_GLvoid *data)	modifies some or all of the data contained in a buffer object's data store.
void OEMC_glClipPlanex(M_GLenum plane, const M_GLfixed *equation)	specify a plane against which all geometry is clipped
void OEMC_glColor4ub(M_GLubyte red, M_GLubyte green, M_GLubyte blue, M_GLubyte alpha)	set the current color
void OEMC_glDeleteBuffers(M_GLsizei n, const M_GLuint *buffers)	delete named buffer objects
void OEMC_glGetBooleanv(M_GLenum pname, M_GLboolean *params)	Return the boolean value or boolean values of a selected parameter
void OEMC_glGetBufferParameteriv(M_GLenum target, M_GLenum pname, M_GLint *params)	return texture parameter values
void OEMC_glGetClipPlanex(M_GLenum pname, M_GLfixed eqn[4])	return the coefficients of the specified clipping plane
void OEMC_glGenBuffers(M_GLsizei n, M_GLuint *buffers)	generate buffer object names
void OEMC_glGetFixedv(M_GLenum pname, M_GLfixed *params)	Return the fixed-point value or fixed-point values of a selected parameter
void OEMC_glGetLightxv(M_GLenum light, M_GLenum pname, M_GLfixed *params)	return light source parameter values
void OEMC_glGetMaterialxv(M_GLenum face, M_GLenum pname, M_GLfixed *params)	return material parameters values
void OEMC_glGetPointerv(M_GLenum pname, void **params)	return the address of the specified pointer
void OEMC_glGetTexEnviv(M_GLenum env, M_GLenum pname, M_GLint *params)	return texture environment parameters
void OEMC_glGetTexEnvxv(M_GLenum env, M_GLenum pname, M_GLfixed *params)	return texture environment parameters
void OEMC_glGetTexParameteriv(M_GLenum target, M_GLenum pname, M_GLint *params)	return texture parameter values
void OEMC_glGetTexParameterxv(M_GLenum target, M_GLenum pname, M_GLfixed *params)	return texture parameter values

M_GLenum pname, M_GLfixed *params)	
M_GLboolean OEMC_gIsBuffer(M_GLuint buffer)	determine if a name corresponds to a buffer object
M_GLboolean OEMC_gIsEnabled(M_GLenum cap)	test whether a capability is enabled
M_GLboolean OEMC_gIsTexture(M_GLuint texture)	determine if a name corresponds to a texture
void OEMC_gLightModelxv(M_GLenum pname, const M_GLfixed *params)	set the lighting model parameters
void OEMC_gPointSizeParameterx(M_GLenum pname, M_GLfixed param)	specify parameters for point rasterization
void OEMC_gPointSizeParameterxv(M_GLenum pname, const M_GLfixed *params)	specify parameters for point rasterization
void OEMC_gTexEnv(M_GLenum target, M_GLenum pname, M_GLint param)	set texture environment parameters
void OEMC_gTexEnviv(M_GLenum target, M_GLenum pname, const M_GLint *params)	set texture environment parameters
void OEMC_gTexParameter(M_GLenum target, M_GLenum pname, M_GLint param)	set texture parameters
void OEMC_gTexParameteriv(M_GLenum target, M_GLenum pname, const M_GLint *params)	set texture parameters
void OEMC_gTexParameterxv(M_GLenum target, M_GLenum pname, const M_GLfixed *params)	set texture parameters
void OEMC_gCurrentPaletteMatrixOES(M_GLuint matrixpaletteindex)	defines which of the palette's matrices is affected by subsequent matrix operations
void OEMC_gLoadPaletteFromModelViewMatrixOES(void)	copies the current model view matrix to a matrix in the current matrix palette
void OEMC_gMatrixIndexPointerOES(M_GLint size, M_GLenum type, M_GLsizei stride, const M_GLvoid *pointer)	define an array of matrix indices
void OEMC_gWeightPointerOES(M_GLint size, M_GLenum type, M_GLsizei stride, const M_GLvoid *pointer)	define an array of weights
void OEMC_gPointSizePointerOES(M_GLenum type, M_GLsizei stride, const M_GLvoid *pointer)	define an array of point sizes
void OEMC_gDrawTexsOES(M_GLshort x, M_GLshort y, M_GLshort z, M_GLshort width, M_GLshort height)	draws a texture rectangle to the screen

void OEMC_glDrawTexiOES(M_GLint x, M_GLint y, M_GLint z, M_GLint width, M_GLint height)	draws a texture rectangle to the screen
void OEMC_glDrawTexxOES(M_GLfixed x, M_GLfixed y, M_GLfixed z, M_GLfixed width, M_GLfixed height)	draws a texture rectangle to the screen
void OEMC_glDrawTexsvOES(const M_GLshort *coords)	draws a texture rectangle to the screen
void OEMC_glDrawTexivOES(const M_GLint *coords)	draws a texture rectangle to the screen
void OEMC_glDrawTexxvOES(const M_GLfixed *coords)	draws a texture rectangle to the screen

4.4 Basic 3D API (OpenGL/ES 1.0 Common addition only) Description

4.4.1 OEMC_glBindBuffer

Description

bind a named buffer to a target

Prototypes

```
void OEMC_glBindBuffer( M_GGLenum target, M_GLuint buffer )
```

Parameters

target

Specifies the target to which the buffer is bound. Which must be GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER.

buffer

Specifies the name of a buffer.

Remarks

glBindBuffer lets you create or use a named buffer. Calling glBindBuffer with target set to GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER and buffer set to the buffer name, binds the buffer name to the target.

Buffer names are unsigned integers. The value 0 is reserved for GL.

When a buffer is bound to a target, any previous binding for that target is automatically broken.

If an unused buffer name is specified, a buffer object is created. The resulting buffer object is a new state vector, initialized with a zero-sized memory buffer, and with the following state values.

GL_BUFFER_SIZE initialized to 0.

GL_BUFFER_USAGE initialized to GL_STATIC_DRAW.

GL_BUFFER_ACCESS initialized to GL_WRITE_ONLY.

glBindBuffer may also be used to bind an existing buffer object. If the bind is successful no change is made to the state of the newly bound buffer object.

While a buffer is bound, GL operations on the target to which it is bound affect the bound buffer object, and queries of the target to which a buffer object is bound return state from the bound object.

Notes

In the initial state the reserved name zero is bound to GL_ARRAY_BUFFER and GL_ELEMENT_ARRAY_BUFFER. There is no buffer object corresponding to the name zero, so client attempts to modify or query buffer object state for the target GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER while zero is bound will generate GL errors.

While a buffer object is bound, any GL operations on that object affect any other bindings of that object.

Errors

GL_INVALID_ENUM is generated if target is not one of the allowable values.

4.4.2 OEMC_glBufferData

Description

creates and initializes the data store of a buffer object.

Prototypes

```
void OEMC_glBufferData( M_GGLenum target, M_GLsizeiptr size, const M_GLvoid *data,  
M_GGLenum usage );
```

Parameters

target

Specifies the buffer object target. Which must be GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER.

size

Specifies the size of the data store in basic machine units.

data

Specifies the source data in client memory.

usage

Specifies the expected application usage pattern of the data store. Accepted values are GL_STATIC_DRAW and GL_DYNAMIC_DRAW.

Remarks

glBufferData lets you create and initialize the data store of a buffer object.

If data is non-null, then the source data is copied to the buffer object's data store. If data is null, then the contents of the buffer object's data store are undefined.

The options for usage are:

GL_STATIC_DRAW

Where the data store contents will be specified once by the application, and used many times as the source for GL drawing commands.

GL_DYNAMIC_DRAW

Where the data store contents will be respecified repeatedly by the application, and used many times as the source for GL drawing commands.

glBufferData deletes any existing data store, and sets the values of the buffer object's state variables thus:

GL_BUFFER_SIZE initialized to size.

GL_BUFFER_USAGE initialized to usage.

GL_BUFFER_ACCESS initialized to GL_WRITE_ONLY.

Clients must align data elements consistent with the requirements of the client platform, with an additional base-level requirement that an offset within a buffer to a datum comprising N basic machine units be a multiple of N.

Notes

usage is provided as a performance hint only. The specified usage value does not constrain the actual usage pattern of the data store.

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

Errors

GL_INVALID_ENUM is generated if target is not one of the allowable values.

GL_INVALID_ENUM is generated if usage is not one of the allowable values.

GL_OUT_OF_MEMORY is generated if the GL is unable to create a data store of the requested size.

4.4.3 OEMC_glBufferSubData

Description

modifies some or all of the data contained in a buffer object's data store.

Prototypes

```
void OEMC_glBufferSubData( M_GLenum target, M_GLintptr offset, M_GLsizeiptr size,
const M_GLvoid *data )
```

Parameters

target

Specifies the buffer object target. Must be GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER.

offset

Specifies the starting offset of the data to be replaced in basic machine units.

size

Specifies the size of the data to be replaced in basic machine units.

data

Specifies a region of client memory size basic machine units in length, containing the data that replace the specified buffer range.

Remarks

glBufferSubData lets you modify some or all of the data contained in a buffer object's data store.

Notes

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

Errors

GL_INVALID_ENUM is generated if target is not one of the allowable values.

GL_INVALID_VALUE is generated if offset or size is less than zero, or if offset + size is greater than the value of BUFFER_SIZE.

4.4.4 OEMC_glClipPlanex

Description

specify a plane against which all geometry is clipped

Prototypes

```
void OEMC_glClipPlanex( M_GGLenum plane, const M_GLfixed *equation )
```

Parameters

plane

Specifies which clipping plane is being positioned. Symbolic names of the form GL_CLIP_PLANE i where $0 < i < GL_MAX_CLIP_PLANES$ are accepted.

equation

Specifies the address of an array of four fixed-point or fixed-point values. These values are interpreted as a plane equation.

Remarks

Geometry is always clipped against the boundaries of a six-plane frustum in x, y, and z. glClipPlane allows the specification of additional planes, not necessarily perpendicular to the x, y, or z axis, against which all geometry is clipped. To determine the maximum number of additional clipping planes, call glGet with argument GL_MAX_CLIP_PLANES. All implementations support at least one such clipping planes. Because the resulting clipping region is the intersection of the defined half-spaces, it is always convex.

glClipPlane specifies a half-space using a four-component plane equation. When glClipPlane is called, equation is transformed by the inverse of the modelview matrix and stored in the resulting eye coordinates. Subsequent changes to the modelview matrix have no effect on the stored plane-equation components. If the dot product of the eye coordinates of a vertex with the stored plane equation components is positive or zero, the vertex is in with respect to that clipping plane. Otherwise, it is out.

To enable and disable clipping planes, call glEnable and glDisable with the argument GL_CLIP_PLANE i , where i is the plane number.

All clipping planes are initially defined as (0, 0, 0, 0) in eye coordinates and are disabled.

Notes

It is always the case that GL_CLIP_PLANE i = GL_CLIP_PLANE0 + i .

Errors

GL_INVALID_ENUM is generated if plane is not an accepted value.

4.4.5 OEMC_glColor4ub

Description

set the current color

Prototypes

```
void OEMC_glColor4ub( M_GLubyte red, M_GLubyte green, M_GLubyte blue,  
M_GLubyte alpha )
```

Parameters

red, green, blue, alpha

Specify new red, green, blue, and alpha values for the current color. The initial value is (1, 1, 1, 1).

Remarks

The GL stores a current four-valued RGBA color. glColor sets a new four-valued RGBA color.

Current color values are stored in fixed-point or fixed-point. In case the values are stored in fixed-point, the mantissa and exponent sizes are unspecified.

fixed-point values are not clamped to the range [0, 1] before the current color is updated. However, color components are clamped to this range before they are interpolated or written into the color buffer.

4.4.6 OEMC_glDeleteBuffers

Description

delete named buffer objects

Prototypes

```
void OEMC_glDeleteBuffers( M_GLsizei n, const M_GLuint *buffers )
```

Parameters

n

Specifies the number of buffer objects to be deleted.

buffers

Specifies an array of buffer object names to be deleted.

Remarks

glDeleteBuffers deletes n buffer objects named by the elements of the array buffers. After a buffer object is deleted, it has no contents, and its name is free for reuse.

glDeleteBuffers silently ignores zero and names that do not correspond to existing buffer objects.

Notes

If a buffer object is deleted while it is bound, all bindings to that object in the current context (i.e. in the thread that called glDeleteBuffers) are reset to zero. Bindings to that buffer in other contexts and other threads are not affected, but attempting to use a deleted buffer in another thread produces undefined results, including but not limited to possible GL errors and rendering corruption. Using a deleted buffer in another context or thread may not, however, result in program termination.

Errors

GL_INVALID_VALUE is generated if n is negative.

4.4.7 OEMC_glGetBooleanv

Description

These functions return the value or values of a selected parameter.

Prototypes

```
void OEMC_glGetBooleanv( M_GGLenum pname, M_GLboolean *params )
```

Parameters

pname

The parameter value to be returned. The following symbolic constants are accepted:

GL_LIGHT_MODEL_TWO_SIDE

The pname parameter returns a single Boolean value indicating whether separate materials are used to compute lighting for front- and back-facing polygons.

GL_COLOR_WRITEMASK

The pname parameter returns four Boolean values: the red, green, blue, and alpha write enables for the color buffers. See glColorMask.

GL_DEPTH_WRITEMASK

The pname parameter returns a single Boolean value indicating if the depth buffer is enabled for writing.

GL_SAMPLE_COVERAGE_INVERT

Remarks

These four functions return values for simple state variables in OpenGL. The pname parameter is a symbolic constant indicating the state variable to be returned, and params is a pointer to an array of the indicated type in which to place the returned data.

Type conversion is performed if params has a different type from the state variable value being requested. If you call glGetBooleanv, a fixed-point or integer value is converted to GL_FALSE if and only if it is zero. Otherwise, it is converted to GL_TRUE.

If you call glGetIntegerv, Boolean values are returned as GL_TRUE or GL_FALSE, and most fixed-point values are rounded to the nearest integer value. Fixed-point colors and normals, however, are returned with a linear mapping that maps 1.0 to the most positive representable integer value and -1.0 to the most negative representable integer value.

If you call `glGetFloatv` or `glGetDoublev`, Boolean values are returned as `GL_TRUE` or `GL_FALSE`, and integer values are converted to fixed-point values.

You can query many of the Boolean parameters more easily with `glIsEnabled`.

Errors

The following are the error codes generated and their conditions.

Error Code	Condition
<code>GL_INVALID_ENUM</code>	<code>pname</code> was not an accepted value.
<code>GL_INVALID_OPERATION</code>	<code>glGet</code> was called between a call to <code>glBegin</code> and the corresponding call to <code>glEnd</code> .

4.4.8 OEMC_glGenBuffers

Description

generate buffer object names

Prototypes

```
void OEMC_glGenBuffers( M_GLsizei n, M_GLuint *buffers )
```

Parameters

n

Specifies the number of buffer object names to be generated.

buffers

Specifies an array in which the generated buffer object names are stored.

Remarks

glGenBuffers returns n buffer object names in buffers.

These names are marked as used, for the purposes of glGenBuffers only, but they acquire buffer state only when they are first bound, just as if they were unused.

Errors

GL_INVALID_VALUE is generated if n is negative.

4.4.9 OEMC_glGetBufferParameteriv

Description

return texture parameter values

Prototypes

```
void OEMC_glGetBufferParameteriv( M_GGLenum target, M_GGLenum pname, M_GLint  
*params )
```

Parameters

target

Specifies the buffer object target. Which must be GL_ARRAY_BUFFER.

pname

Specifies the symbolic name of a buffer object parameter. Which can be either GL_BUFFER_SIZE, GL_BUFFER_USAGE, or GL_BUFFER_ACCESS.

params

Returns the buffer object parameters.

Remarks

glGetBufferParameter returns in params the value or values of the buffer object parameter specified as pname. target defines the target buffer object, which must be GL_ARRAY_BUFFER.

GL_BUFFER_SIZE

Returns the size of the data store in basic machine units.

GL_BUFFER_USAGE

Returns the expected application usage pattern of the data store. Possible values are:

GL_STATIC_DRAW

Where the data store contents will be specified once by the application, and used many times as the source for GL drawing commands.

GL_DYNAMIC_DRAW

Where the data store contents will be respecified repeatedly by the application, and used many times as the source for GL drawing commands.

GL_BUFFER_ACCESS

Returns the access capability for the data store. Will always be GL_WRITE_ONLY.

Errors

GL_INVALID_ENUM is generated if target or pname is not one of the accepted defined values.

4.4.10 OEMC_glGetClipPlanex

Description

return the coefficients of the specified clipping plane

Prototypes

```
void OEMC_glGetClipPlanex( M_GGLenum pname, M_GLfixed equation[4] )
```

Parameters

plane

Specifies a clipping plane. The number of clipping planes depends on the implementation, but at least six clipping planes are supported. They are identified by symbolic names of the form `GL_CLIP_PLANEi` where $0 < i < GL_MAX_CLIP_PLANES$

equation

Returns four fixed-point or fixed-point values that are the coefficients of the plane equation of plane in eye coordinates. The initial value is (0, 0, 0, 0).

Remarks

`glGetClipPlane` returns in equation the four coefficients of the plane equation for plane.

Notes

It is always the case that `GL_CLIP_PLANEi = GL_CLIP_PLANE0 + i`.

If an error is generated, no change is made to the contents of equation.

Errors

`GL_INVALID_ENUM` is generated if plane is not an accepted value.

4.4.11 OEMC_glGetFixedv

Description

These functions return the value or values of a selected parameter.

Prototypes

```
void OEMC_glGetFixedv( M_GLEnum pname, M_GLfixed *params )
```

Parameters

pname

The parameter value to be returned. The following symbolic constants are accepted:

GL_CURRENT_TEXTURE_COORDS

The params parameter returns four values: the s, t, r, and q current texture coordinates.

See glTexCoord

GL_CURRENT_NORMAL

The params parameter returns three values: the x, y, and z values of the current normal. Integer values, if requested, are linearly mapped from the internal fixed-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. See glNormal.

GL_MODELVIEW_MATRIX

The params parameter returns 16 values: the modelview matrix on the top of the modelview matrix stack. See glPushMatrix

GL_PROJECTION_MATRIX

The params parameter returns 16 values: the projection matrix on the top of the projection matrix stack. See glPushMatrix.

GL_TEXTURE_MATRIX

The params parameter returns 16 values: the texture matrix on the top of the texture matrix stack. See glPushMatrix.

GL_DEPTH_RANGE

The params parameter returns two values: the near and far mapping limits for the depth buffer. Integer values, if requested, are linearly mapped from the internal fixed-point

representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. See `glDepthRange`.

GL_FOG_COLOR

The `params` parameter returns four values: the red, green, blue, and alpha components of the fog color. Integer values, if requested, are linearly mapped from the internal fixed-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. See `glFog`.

GL_FOG_DENSITY

The `params` parameter returns one value: the fog density parameter. See `glFog`.

GL_FOG_START

The `params` parameter returns one value: the start factor for the linear fog equation. See `glFog`.

GL_FOG_END

The `params` parameter returns one value: the end factor for the linear fog equation. See `glFog`.

GL_LIGHT_MODEL_AMBIENT

The `params` parameter returns four values: the red, green, blue, and alpha components of the ambient intensity of the entire scene. Integer values, if requested, are linearly mapped from the internal fixed-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. See `glLightModel`.

GL_POINT_SIZE

The `params` parameter returns one value: the point size as specified by `glPointSize`.

GL_POINT_SIZE_MIN

The `params` parameter returns two values: the smallest supported sizes for antialiased points. See `glPointSize`.

GL_POINT_SIZE_MAX

The `params` parameter returns two values: the largest supported sizes for antialiased

points. See glPointSize.

GL_POINT_FADE_THRESHOLD_SIZE

GL_POINT_DISTANCE_ATTENUATION

GL_LINE_WIDTH

The params parameter returns one value: the line width as specified with glLineWidth.

GL_POLYGON_OFFSET_FACTOR

GL_POLYGON_OFFSET_UNITS

GL_SAMPLE_COVERAGE_VALUE

GL_COLOR_CLEAR_VALUE

The params parameter returns four values: the red, green, blue, and alpha values used to clear the color buffers. Integer values, if requested, are linearly mapped from the internal fixed-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. See glClearColor.

GL_ALIASED_POINT_SIZE_RANGE

GL_SMOOTH_POINT_SIZE_RANGE

GL_ALIASED_LINE_WIDTH_RANGE

GL_SMOOTH LINE_WIDTH_RANGE

Remarks

This function return values for simple state variables in OpenGL/ES. The pname parameter is a symbolic constant indicating the state variable to be returned, and params is a pointer to an array of the indicated type in which to place the returned data.

Type conversion is performed if params has a different type from the state variable value being requested.

You can query many of the Boolean parameters more easily with glIsEnabled.

4.4.12 OEMC_glGetLightxv

Description

return light source parameter values

Prototypes

```
void OEMC_glGetLightxv( M_GGLenum light, M_GGLenum pname, M_GLfixed *params )
```

Parameters

light

Specifies a light source. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form `GL_LIGHTi` where $0 < i < \text{GL_MAX_LIGHTS}$

pname

Specifies a light source parameter for light. Accepted symbolic names are `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_EMISSION`, `GL_SPOT_DIRECTION`, `GL_SPOT_EXPONENT`, `GL_SPOT_CUTOFF`, `GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, and `GL_QUADRATIC_ATTENUATION`.

params

Returns the requested data.

Remarks

`glGetLight` returns in `params` the value or values of a light source parameter. `light` names the light and is a symbolic name of the form `GL_LIGHTi` for $0 < i < \text{GL_MAX_LIGHTS}$ where `GL_MAX_LIGHTS` is an implementation dependent constant that is greater than or equal to eight. `pname` specifies one of ten light source parameters, again by symbolic name.

The ten light parameters are defined:

`GL_AMBIENT`

`params` returns four fixed-point values that specify the ambient RGBA intensity of the light. Both fixed-point and fixed-point values are mapped directly. Neither fixed-point nvalues are clamped. The initial ambient light intensity is (0, 0, 0, 1).

`GL_DIFFUSE`

`params` returns four fixed-point values that specify the diffuse RGBA intensity of the light. Both fixed-point and fixed-point values are mapped directly. Neither fixed-point nor fixed-

point values are clamped. The initial value for `GL_LIGHT0` is (1, 1, 1, 1). For other lights, the initial value is (0, 0, 0, 0).

`GL_SPECULAR`

`params` returns four fixed-point values that specify the specular RGBA intensity of the light. Both fixed-point and fixed-point values are mapped directly. Neither fixed-point nor fixed-point values are clamped. The initial value for `GL_LIGHT0` is (1, 1, 1, 1). For other lights, the initial value is (0, 0, 0, 0).

`GL_EMISSION`

`params` returns four fixed-point values representing the emitted light intensity of the material. Both fixed-point and fixed-point values are mapped directly. The initial value is (0, 0, 0, 1).

`GL_SPOT_DIRECTION`

`params` returns three fixed-point values that specify the direction of the light in homogeneous object coordinates. Both fixed-point and fixed-point values are mapped directly. fixed-point values is not clamped. The initial direction is (0, 0, -1).

`GL_SPOT_EXPONENT`

`params` returns a single fixed-point value that specifies the intensity distribution of the light. Fixed-point and fixed-point values are mapped directly. Only values in the range [0, 128] are accepted. The initial spot exponent is 0.

`GL_SPOT_CUTOFF`

`params` returns a single fixed-point value that specifies the maximum spread angle of a light source. Fixed-point and fixed-point values are mapped directly. Only values in the range [0, 90] and the special value 180 are accepted. If the angle between the direction of the light and the direction from the light to the vertex being lighted is greater than the spot cutoff angle, the light is completely masked. Otherwise, its intensity is controlled by the spot exponent and the attenuation factors. The initial spot cutoff is 180.

`GL_CONSTANT_ATTENUATION`, `GL_LINEAR_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`

`params` returns a single fixed-point value that specifies one of the three light attenuation factors. Fixed-point and fixed-point values are mapped directly. Only nonnegative values are accepted. If the light is positional, rather than directional, its intensity is attenuated by the reciprocal of the sum of the constant factor, the linear factor times the distance between the light and the vertex being lighted, and the quadratic factor times the square of the same distance. The initial attenuation factors are (1, 0, 0).

Notes

It is always the case that $GL_LIGHTi = GL_LIGHT0 + i$.

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if light or pname is not an accepted value.

4.4.13 OEMC_glGetMaterialxv

Description

return material parameters values

Prototypes

```
void OEMC_glGetMaterialxv( M_GGLenum face, M_GGLenum pname, M_GLfixed
*params )
```

Parameters

face

Specifies which of the two materials is being queried. GL_FRONT or GL_BACK are accepted, representing the front and back materials, respectively.

pname

Specifies the material parameter to return. Accepted symbolic names are GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, and GL_SHININESS.

params

Returns the requested data.

Remarks

glGetMaterial returns in params the value or values of parameter pname of material face.

Five parameters are defined:

GL_AMBIENT

params returns four fixed-point values that specify the ambient RGBA reflectance of the material. The values are not clamped. The initial ambient reflectance is (0.2, 0.2, 0.2, 1.0).

GL_DIFFUSE

params returns four fixed-point values that specify the diffuse RGBA reflectance of the material. The values are not clamped. The initial diffuse reflectance is (0.8, 0.8, 0.8, 1.0).

GL_SPECULAR

params returns four fixed-point values that specify the specular RGBA reflectance of the material. The values are not clamped. The initial specular reflectance is (0, 0, 0, 1).

GL_EMISSION

params returns four fixed-point values that specify the RGBA emitted light intensity of the material. The values are not clamped. The initial emission intensity is (0, 0, 0, 1).

GL_SHININESS

params returns a single fixed-point value that specifies the RGBA specular exponent of the material. The initial specular exponent is 0.

Notes

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if face or pname is not an accepted value.

4.4.14 OEMC_glGetPointerv

Description

return the address of the specified pointer

Prototypes

```
void OEMC_glGetPointerv( M_GGLenum pname, void **params )
```

Parameters

pname

Specifies the array or buffer pointer to be returned. Accepted symbolic names are
GL_COLOR_ARRAY_POINTER, GL_NORMAL_ARRAY_POINTER,
GL_TEXTURE_COORD_ARRAY_POINTER, GL_VERTEX_ARRAY_POINTER,
GL_MATRIX_INDEX_ARRAY_POINTER_OES,
GL_POINT_SIZE_ARRAY_POINTER_OES, and GL_WEIGHT_ARRAY_POINTER_OES.

params

Returns the pointer value specified by pname.

Remarks

glGetPointer returns pointer information. pname is a symbolic constant indicating the pointer to be returned, and params is a pointer to a location in which to place the returned data.

Notes

The pointers are all client-side state. The initial value for each pointer is 0.

Errors

GL_INVALID_ENUM is generated if pname is not an accepted value.

4.4.15 OEMC_glGetTexEnviv

Description

return texture environment parameters

Prototypes

```
void OEMC_glGetTexEnviv( M_GGLenum env, M_GGLenum pname, M_GLint *params )
```

Parameters

target

Specifies a texture environment, which must be GL_TEXTURE_ENV.

pname

Specifies the symbolic name of a texture environment parameter. Which can be either GL_TEXTURE_ENV_MODE, GL_TEXTURE_ENV_COLOR, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

params

Returns the requested data.

Remarks

glGetTexParameter returns in params selected values of a texture environment that was specified with glTexEnv. target specifies a texture environment. Currently, only one texture environment is defined and supported: GL_TEXTURE_ENV.

pname names a specific texture environment parameter, as follows:

GL_TEXTURE_ENV_MODE

params returns the single-valued texture environment mode, a symbolic constant. The initial value is GL_MODULATE.

GL_TEXTURE_ENV_COLOR

params returns four fixed values that are the texture environment color. The initial value is (0, 0, 0, 0).

Notes

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if target or pname is not one of the accepted defined

values.

4.4.16 OEMC_glGetTexEnvxv

Description

return texture environment parameters

Prototypes

```
void OEMC_glGetTexEnvxv( M_GGLenum env, M_GGLenum pname, M_GLfixed *params )
```

Parameters

target

Specifies a texture environment, which must be GL_TEXTURE_ENV.

pname

Specifies the symbolic name of a texture environment parameter. Which can be either GL_TEXTURE_ENV_MODE, GL_TEXTURE_ENV_COLOR, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

params

Returns the requested data.

Remarks

glGetTexParameter returns in params selected values of a texture environment that was specified with glTexEnv. target specifies a texture environment. Currently, only one texture environment is defined and supported: GL_TEXTURE_ENV.

pname names a specific texture environment parameter, as follows:

GL_TEXTURE_ENV_MODE

params returns the single-valued texture environment mode, a symbolic constant. The initial value is GL_MODULATE.

GL_TEXTURE_ENV_COLOR

params returns four fixed values that are the texture environment color. The initial value is (0, 0, 0, 0).

Notes

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if target or pname is not one of the accepted defined

values.

4.4.17 OEMC_glGetTexParameteriv

Description

return texture parameter values

Prototypes

```
void OEMC_glGetTexParameteriv( M_GGLenum target, M_GGLenum pname, M_GLint  
*params )
```

Parameters

target

Specifies the target texture, which must be GL_TEXTURE_2D.

pname

Specifies the symbolic name of a texture parameter. Which can be one of the following: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

params

Returns the texture parameters.

Remarks

glGetTexParameter returns in params the value or values of the texture parameter specified as pname. target defines the target texture, which must be GL_TEXTURE_2D which specifies two-dimensional texturing. pname accepts the same symbols as glGetTexParameter, with the same interpretations:

GL_TEXTURE_MIN_FILTER

Returns the texture minifying function. Which can one of the following: GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, or GL_LINEAR_MIPMAP_LINEAR. The initial value is GL_NEAREST_MIPMAP_LINEAR.

GL_TEXTURE_MAG_FILTER

Returns the texture magnification function. Which can be either GL_NEAREST or GL_LINEAR. The initial value is GL_LINEAR.

GL_TEXTURE_WRAP_S

Returns the wrap parameter for texture coordinate s. Which can be either: GL_CLAMP, GL_CLAMP_TO_EDGE, or GL_REPEAT. The initial value is GL_REPEAT.

GL_TEXTURE_WRAP_T

Returns the wrap parameter for texture coordinate t. Which can be either: GL_CLAMP, GL_CLAMP_TO_EDGE, or GL_REPEAT. The initial value is GL_REPEAT.

GL_GENERATE_MIPMAP

Returns the automatic mipmap generation parameter. The initial value is GL_FALSE.

Notes

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if target or pname is not one of the accepted defined values.

4.4.18 OEMC_glGetTexParameterv

Description

return texture parameter values

Prototypes

```
void OEMC_glGetTexParameterv( M_GGLenum target, M_GGLenum pname, M_GLfixed  
*params )
```

Parameters

target

Specifies the target texture, which must be GL_TEXTURE_2D.

pname

Specifies the symbolic name of a texture parameter. Which can be one of the following: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

params

Returns the texture parameters.

Remarks

glGetTexParameter returns in params the value or values of the texture parameter specified as pname. target defines the target texture, which must be GL_TEXTURE_2D which specifies two-dimensional texturing. pname accepts the same symbols as glGetTexParameter, with the same interpretations:

GL_TEXTURE_MIN_FILTER

Returns the texture minifying function. Which can one of the following: GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, or GL_LINEAR_MIPMAP_LINEAR. The initial value is GL_NEAREST_MIPMAP_LINEAR.

GL_TEXTURE_MAG_FILTER

Returns the texture magnification function. Which can be either GL_NEAREST or GL_LINEAR. The initial value is GL_LINEAR.

GL_TEXTURE_WRAP_S

Returns the wrap parameter for texture coordinate s. Which can be either: GL_CLAMP, GL_CLAMP_TO_EDGE, or GL_REPEAT. The initial value is GL_REPEAT.

GL_TEXTURE_WRAP_T

Returns the wrap parameter for texture coordinate t. Which can be either: GL_CLAMP, GL_CLAMP_TO_EDGE, or GL_REPEAT. The initial value is GL_REPEAT.

GL_GENERATE_MIPMAP

Returns the automatic mipmap generation parameter. The initial value is GL_FALSE.

Notes

If an error is generated, no change is made to the contents of params.

Errors

GL_INVALID_ENUM is generated if target or pname is not one of the accepted defined values.

4.4.19 OEMC_gllsBuffer

Description

determine if a name corresponds to a buffer object

Prototypes

M_GLboolean OEMC_gllsBuffer(M_GLuint buffer)

Parameters

buffer

Specifies a value that may be the name of a buffer object.

Remarks

gllsBuffer returns GL_TRUE if buffer is currently the name of a buffer object. If buffer is zero, or is a non-zero value that is not currently the name of a buffer object, gllsBuffer returns GL_FALSE.

4.4.20 OEMC_gllsEnabled

Description

test whether a capability is enabled

Prototypes

M_GLboolean OEMC_gllsEnabled(M_GLenum cap)

Parameters

cap

Specifies a symbolic constant indicating a GL capability.

Remarks

gllsEnabled returns GL_TRUE if cap is an enabled capability and returns GL_FALSE otherwise.

The following capabilities are accepted for cap:

Constant	See function:
GL_ALPHA_TEST	glAlphaFunc
GL_ARRAY_BUFFER_BINDING	glBindBuffer
GL_BLEND	glBlendFunc, glLogicOp
GL_CLIP_PLANE i	glClipPlane
GL_COLOR_ARRAY	glColorPointer
GL_COLOR_LOGIC_OP	glLogicOp
GL_COLOR_MATERIAL	glColorMaterial
GL_CULL_FACE	glCullFace
GL_DEPTH_TEST	glDepthFunc, glDepthRange
GL_DITHER	glEnable
GL_FOG	glFog
GL_LIGHT i	glLight, glLightModel
GL_LIGHTING	glLight, glLightModel, glMaterial
GL_LINE_SMOOTH	glLineWidth
GL_MATRIX_PALETTE_OES	glMatrixMode
GL_MATRIX_INDEX_ARRAY_OES	glEnableClientState
GL_MULTISAMPLE	glEnable
GL_NORMAL_ARRAY	glNormalPointer

GL_NORMALIZE	glNormal
GL_POINT_SIZE_ARRAY_OES	glEnableClientState
GL_POINT_SMOOTH	glPointSize
GL_POINT_SPRITE_OES	glEnable, glTexEnv
GL_POLYGON_OFFSET_FILL	glPolygonOffset
GL_RESCALE_NORMAL	glEnable
GL_SAMPLE_ALPHA_TO_COVERAGE	glEnable
GL_SAMPLE_ALPHA_TO_ONE	glEnable
GL_SAMPLE_COVERAGE	glEnable
GL_SCISSOR_TEST	glScissor
GL_STENCIL_TEST	glStencilFunc, glStencilOp
GL_TEXTURE_2D	glTexImage2D
GL_TEXTURE_COORD_ARRAY	glTexCoordPointer
GL_WEIGHT_ARRAY_OES	glEnableClientState
GL_VERTEX_ARRAY	glVertexPointer

Notes

If an error is generated, glIsEnabled returns 0.

Errors

GL_INVALID_ENUM is generated if cap is not an accepted value.

4.4.21 OEMC_gIsTexture

Description

determine if a name corresponds to a texture

Prototypes

M_GLboolean OEMC_gIsTexture(M_GLuint texture)

Parameters

texture

Specifies a value that may be the name of a texture.

Remarks

gIsTexture returns GL_TRUE if texture is currently the name of a texture. If texture is zero, or is a non-zero value that is not currently the name of a texture, gIsTexture returns GL_FALSE.

4.4.22 OEMC_gLightModelxv

Description

set the lighting model parameters

Prototypes

```
void OEMC_gLightModelxv( M_GGLenum pname, const M_GLfixed *params )
```

Parameters

`pname`

Specifies a single-valued lighting model parameter. Must be `GL_LIGHT_MODEL_TWO_SIDE`.

`param`

Specifies the value that `param` will be set to.

Remarks

`gLightModel` sets the lighting model parameter. `pname` names a parameter and `params` gives the new value. There are two lighting model parameters:

`GL_LIGHT_MODEL_AMBIENT`

`params` contains four fixed-point values that specify the ambient intensity of the entire scene. The values are not clamped. The initial value is (0.2, 0.2, 0.2, 1.0).

`GL_LIGHT_MODEL_TWO_SIDE`

`params` is a single fixed-point value that specifies whether one- or two-sided lighting calculations are done for polygons. It has no effect on the lighting calculations for points, lines, or bitmaps. If `params` is 0, one-sided lighting is specified, and only the front material parameters are used in the lighting equation. Otherwise, two-sided lighting is specified. In this case, vertices of back-facing polygons are lighted using the back material parameters, and have their normals reversed before the lighting equation is evaluated. Vertices of front-facing polygons are always lighted using the front material parameters, with no change to their normals. The initial value is 0.

The lighted color of a vertex is the sum of the material emission intensity, the product of the material ambient reflectance and the lighting model full-scene ambient intensity, and the contribution of each enabled light source. Each light source contributes the sum of three terms: ambient, diffuse, and specular. The ambient light source contribution is the product of the material ambient reflectance and the light's ambient intensity. The diffuse light source contribution is the product of the material diffuse reflectance, the light's

diffuse intensity, and the dot product of the vertex's normal with the normalized vector from the vertex to the light source. The specular light source contribution is the product of the material specular reflectance, the light's specular intensity, and the dot product of the normalized vertex-to-eye and vertex-to-light vectors, raised to the power of the shininess of the material. All three light source contributions are attenuated equally based on the distance from the vertex to the light source and on light source direction, spread exponent, and spread cutoff angle. All dot products are replaced with 0 if they evaluate to a negative value.

The alpha component of the resulting lighted color is set to the alpha value of the material diffuse reflectance.\

Errors

GL_INVALID_ENUM is generated if pname is not an accepted value.

4.4.23 OEMC_glPointParameterx

Description

specify parameters for point rasterization

Prototypes

void OEMC_glPointParameterx(M_GGLenum pname, M_GLfixed param)

Parameters

pname

Specifies the single-valued parameter to be updated. Can be either GL_POINT_SIZE_MIN, GL_POINT_SIZE_MAX, or GL_POINT_FADE_THRESHOLD_SIZE.

param

Specifies the value that the parameter will be set to.

Remarks

glPointParameter assigns values to point parameters.

glPointParameter takes two arguments. pname, specifies which of several parameters will be modified. param, specifies what value or values will be assigned to the specified parameter.

The parameters that can be specified using glPointParameter, and their interpretations are as follows:

GL_POINT_SIZE_MIN

param specifies, or param points to the lower bound to which the derived point size is clamped.

GL_POINT_SIZE_MAX

param specifies, or param points to the upper bound to which the derived point size is clamped.

GL_POINT_FADE_THRESHOLD_SIZE

param specifies, or param points to the point fade threshold.

GL_POINT_DISTANCE_ATTENUATION

param points to the distance attenuation function coefficients a, b, and c.

Notes

If the point size lower bound is greater than the upper bound, then the point size after

clamping is undefined.

Errors

GL_INVALID_ENUM is generated if pname is not an accepted value.

GL_INVALID_VALUE is generated if assigned values for GL_POINT_SIZE_MIN, GL_POINT_SIZE_MAX, or GL_POINT_FADE_THRESHOLD_SIZE are less than zero.

4.4.24 OEMC_glPointParameterxv

Description

specify parameters for point rasterization

Prototypes

```
void OEMC_glPointParameterxv( M_GGLenum pname, const M_GLfixed *params )
```

Parameters

`pname`

Specifies the single-valued parameter to be updated. Can be either `GL_POINT_SIZE_MIN`, `GL_POINT_SIZE_MAX`, or `GL_POINT_FADE_THRESHOLD_SIZE`.

`param`

Specifies the value that the parameter will be set to.

Remarks

`glPointParameter` assigns values to point parameters.

`glPointParameter` takes two arguments. `pname`, specifies which of several parameters will be modified. `params`, specifies what value or values will be assigned to the specified parameter.

The parameters that can be specified using `glPointParameter`, and their interpretations are as follows:

`GL_POINT_SIZE_MIN`

`param` specifies, or `params` points to the lower bound to which the derived point size is clamped.

`GL_POINT_SIZE_MAX`

`param` specifies, or `params` points to the upper bound to which the derived point size is clamped.

`GL_POINT_FADE_THRESHOLD_SIZE`

`param` specifies, or `params` points to the point fade threshold.

`GL_POINT_DISTANCE_ATTENUATION`

`params` points to the distance attenuation function coefficients `a`, `b`, and `c`.

Notes

If the point size lower bound is greater than the upper bound, then the point size after

clamping is undefined.

Errors

GL_INVALID_ENUM is generated if pname is not an accepted value.

GL_INVALID_VALUE is generated if assigned values for GL_POINT_SIZE_MIN, GL_POINT_SIZE_MAX, or GL_POINT_FADE_THRESHOLD_SIZE are less than zero.

4.4.25 OEMC_glTexEnvi

Description

set texture environment parameters

Prototypes

void OEMC_glTexEnvi(M_GGLenum target, M_GGLenum pname, M_GLint param)

Parameters

target

Specifies a texture environment. Can be either GL_TEXTURE_ENV or GL_POINT_SPRITE_OES.

pname

Specifies the symbolic name of a single-valued texture environment parameter. Must be one of GL_TEXTURE_ENV_MODE, GL_TEXTURE_ENV_COLOR, GL_COMBINE_RGB, GL_COMBINE_ALPHA, or GL_COORD_REPLACE_OES.

param

Specifies a single symbolic constant, one of GL_REPLACE, GL_MODULATE, GL_DECAL, GL_BLEND, or GL_ADD.

Remarks

If target is GL_TEXTURE_ENV, then the following applies:

A texture environment specifies how texture values are interpreted when a fragment is textured.

If pname is GL_TEXTURE_ENV_MODE, then param is (or points to) the symbolic name of a texture function:

A texture function acts on the fragment to be textured using the texture image value that applies to the fragment (see glTexParameter) and produces an RGBA color for that fragment. The following table shows how the RGBA color is produced for each of the three texture functions that can be chosen. C is a triple of color values (RGB) and A is the associated alpha value. RGBA values extracted from a texture image are in the range [0, 1]. The subscript f refers to the incoming fragment, the subscript t to the texture image, the subscript c to the texture environment color, and subscript v indicates a value produced by the texture function.

A texture image can have up to four components per texture element (see glTexImage2D,

and glCopyTexImage2D). In a one-component image, Lt indicates that single component. A two-component image uses Lt and At. A three-component image has only a color value Ct. A four-component image has both a color value Ct and an alpha value At.

For texture functions: GL_REPLACE, GL_MODULATE, GL_DECAL, GL_BLEND, or GL_ADD:

Base internal format	Texture functions				
	GL_REPLACE	GL_MODULATE	GL_DECAL	GL_BLEND	GL_ADD
GL_ALPHA	Cv = Cf Av = At	Cv = Cf Av = Af At	undefined	Cv = Cf Av = Af At	Cv = Cf Av = Af At
GL_LUMINANCE	Cv = Lt Av = Af	Cv = Cf Lt Av = Af	undefined	Cv = Cf (1 - Lt) + Cc Lt Av = Af	Cv = Cf + Lt Av = Af
GL_LUMINANCE_ALPHA	Cv = Lt Av = At	Cv = Cf Lt Av = Af At	undefined	Cv = Cf (1 - Lt) + Cc Lt Av = Af At	Cv = Cf + Lt Av = Af At
GL_RGB	Cv = Ct Av = Af	Cv = Cf Ct Av = Af	Cv = Ct Av = Af	Cv = Cf (1 - Ct) + Cc Ct Av = Af	Cv = Cf + Ct Av = Af
GL_RGBA	Cv = Ct Av = At	Cv = Cf Ct Av = Af At	Cv = Cf (1 - At) + Ct At Av = Af	Cv = Cf (1 - Ct) + Cc Ct Av = Af At	Cv = Cf + Ct Av = Af At

If the value of GL_TEXTURE_ENV_MODE is GL_COMBINE, then the form of the texture function depends on the values of GL_COMBINE_RGB and GL_COMBINE_ALPHA, The RGB and ALPHA results of the texture function are then multiplied by the values of GL_RGB_SCALE and GL_ALPHA_SCALE, respectively.

The results are clamped to [0, 1].

The arguments Arg0, Arg1, Arg2 are determined by the values of GL_SRCn_RGB, GL_SRCn_ALPHA, GL_OPERANDn_RGB, GL_OPERANDn_ALPHA, where n = 0,1, or 2, Cs and As denote the texture source color and alpha from the texture image bound to texture unit n

The state required for the current texture environment, for each texture unit, consists of a

six-valued integer indicating the texture function, an eight-valued integer indicating the RGB combiner function and a six-valued integer indicating the ALPHA combiner function, six four-valued integers indicating the combiner RGB and ALPHA source arguments, three four-valued integers indicating the combiner RGB operands, three two-valued integers indicating the combiner ALPHA operands, and four fixed-point environment color values. In the initial state, the texture and combiner functions are each `GL_MODULATE`, the combiner RGB and ALPHA sources are each `GL_TEXTURE`, `GL_PREVIOUS`, and `GL_CONSTANT` for sources 0, 1, and 2 respectively, the combiner RGB operands for sources 0 and 1 are each `SRC_COLOR`, the combiner RGB operand for source 2, as well as for the combiner ALPHA operands, are each `GL_SRC_ALPHA`, and the environment color is (0, 0, 0, 0).

The state required for the texture filtering parameters, for each texture unit, consists of a single fixed-point level of detail bias. The initial value of the bias is 0.0.

If pname is `GL_TEXTURE_ENV_COLOR`, then params is a pointer to an array that holds an RGBA color consisting of four values. The values are clamped to the range [0, 1] when they are specified. Cc takes these four values.

The initial value of `GL_TEXTURE_ENV_MODE` is `GL_MODULATE`. The initial value of `GL_TEXTURE_ENV_COLOR` is (0, 0, 0, 0).

If target is `GL_POINT_SPRITE_OES` then the following applies:

If pname is `GL_COORD_REPLACE_OES`, then the point sprite texture coordinate replacement mode is set from the value given by param, which may either be `GL_FALSE` or `GL_TRUE`. The default value for each texture unit is for point sprite texture coordinate replacement to be disabled.

Errors

`GL_INVALID_ENUM` is generated when target or pname is not one of the accepted values, or when params should have a defined constant value (based on the value of pname) and does not.

4.4.26 OEMC_glTexEnviv

Description

set texture environment parameters

Prototypes

```
void OEMC_glTexEnviv( M_GGLenum target, M_GGLenum pname, const M_GLint  
*params )
```

Parameters

target

Specifies a texture environment. Can be either GL_TEXTURE_ENV or GL_POINT_SPRITE_OES.

pname

Specifies the symbolic name of a single-valued texture environment parameter. Must be one of GL_TEXTURE_ENV_MODE, GL_TEXTURE_ENV_COLOR, GL_COMBINE_RGB, GL_COMBINE_ALPHA, or GL_COORD_REPLACE_OES.

params

Specifies a pointer to a parameter array that contains either a single symbolic constant or an RGBA color.

Remarks

If target is GL_TEXTURE_ENV, then the following applies:

A texture environment specifies how texture values are interpreted when a fragment is textured.

If pname is GL_TEXTURE_ENV_MODE, then params is (or points to) the symbolic name of a texture function:

A texture function acts on the fragment to be textured using the texture image value that applies to the fragment (see glTexParameter) and produces an RGBA color for that fragment. The following table shows how the RGBA color is produced for each of the three texture functions that can be chosen. C is a triple of color values (RGB) and A is the associated alpha value. RGBA values extracted from a texture image are in the range [0, 1]. The subscript f refers to the incoming fragment, the subscript t to the texture image, the subscript c to the texture environment color, and subscript v indicates a value produced by the texture function.

A texture image can have up to four components per texture element (see `glTexImage2D`, and `glCopyTexImage2D`). In a one-component image, `Lt` indicates that single component. A two-component image uses `Lt` and `At`. A three-component image has only a color value `Ct`. A four-component image has both a color value `Ct` and an alpha value `At`.
For texture functions: `GL_REPLACE`, `GL_MODULATE`, `GL_DECAL`, `GL_BLEND`, or `GL_ADD`:

Base internal format	Texture functions				
	<code>GL_REPLACE</code>	<code>GL_MODULATE</code>	<code>GL_DECAL</code>	<code>GL_BLEND</code>	<code>GL_ADD</code>
<code>GL_ALPHA</code>	$Cv = Cf$ $Av = At$	$Cv = Cf$ $Av = Af At$	undefined	$Cv = Cf$ $Av = Af At$	$Cv = Cf$ $Av = Af At$
<code>GL_LUMINANCE</code>	$Cv = Lt$ $Av = Af$	$Cv = Cf Lt$ $Av = Af$	undefined	$Cv = Cf (1 - Lt)$ $+ Cc Lt$ $Av = Af$	$Cv = Cf + Lt$ $Av = Af$
<code>GL_LUMINANCE_ALPHA</code>	$Cv = Lt$ $Av = At$	$Cv = Cf Lt$ $Av = Af At$	undefined	$Cv = Cf (1 - Lt)$ $+ Cc Lt$ $Av = Af At$	$Cv = Cf + Lt$ $Av = Af At$
<code>GL_RGB</code>	$Cv = Ct$ $Av = Af$	$Cv = Cf Ct$ $Av = Af$	$Cv = Ct$ $Av = Af$	$Cv = Cf (1 - Ct)$ $+ Cc Ct$ $Av = Af$	$Cv = Cf + Ct$ $Av = Af$
<code>GL_RGBA</code>	$Cv = Ct$ $Av = At$	$Cv = Cf Ct$ $Av = Af At$	$Cv = Cf (1 - At)$ $+ Ct At$ $Av = Af$	$Cv = Cf (1 - Ct)$ $+ Cc Ct$ $Av = Af At$	$Cv = Cf + Ct$ $Av = Af At$

If the value of `GL_TEXTURE_ENV_MODE` is `GL_COMBINE`, then the form of the texture function depends on the values of `GL_COMBINE_RGB` and `GL_COMBINE_ALPHA`, The RGB and ALPHA results of the texture function are then multiplied by the values of `GL_RGB_SCALE` and `GL_ALPHA_SCALE`, respectively.
The results are clamped to `[0, 1]`.
The arguments `Arg0`, `Arg1`, `Arg2` are determined by the values of `GL_SRCn_RGB`, `GL_SRCn_ALPHA`, `GL_OPERANDn_RGB`, `GL_OPERANDn_ALPHA`, where `n = 0,1, or 2`, `Cs` and `As` denote the texture source color and alpha from the texture image bound to texture unit `n`

The state required for the current texture environment, for each texture unit, consists of a six-valued integer indicating the texture function, an eight-valued integer indicating the RGB combiner function and a six-valued integer indicating the ALPHA combiner function, six four-valued integers indicating the combiner RGB and ALPHA source arguments, three four-valued integers indicating the combiner RGB operands, three two-valued integers indicating the combiner ALPHA operands, and four fixed-point environment color values. In the initial state, the texture and combiner functions are each `GL_MODULATE`, the combiner RGB and ALPHA sources are each `GL_TEXTURE`, `GL_PREVIOUS`, and `GL_CONSTANT` for sources 0, 1, and 2 respectively, the combiner RGB operands for sources 0 and 1 are each `SRC_COLOR`, the combiner RGB operand for source 2, as well as for the combiner ALPHA operands, are each `GL_SRC_ALPHA`, and the environment color is (0, 0, 0, 0).

The state required for the texture filtering parameters, for each texture unit, consists of a single fixed-point level of detail bias. The initial value of the bias is 0.0.

If pname is `GL_TEXTURE_ENV_COLOR`, then params is a pointer to an array that holds an RGBA color consisting of four values. The values are clamped to the range [0, 1] when they are specified. Cc takes these four values.

The initial value of `GL_TEXTURE_ENV_MODE` is `GL_MODULATE`. The initial value of `GL_TEXTURE_ENV_COLOR` is (0, 0, 0, 0).

If target is `GL_POINT_SPRITE_OES` then the following applies:

If pname is `GL_COORD_REPLACE_OES`, then the point sprite texture coordinate replacement mode is set from the value given by param, which may either be `GL_FALSE` or `GL_TRUE`. The default value for each texture unit is for point sprite texture coordinate replacement to be disabled.

Errors

`GL_INVALID_ENUM` is generated when target or pname is not one of the accepted values, or when params should have a defined constant value (based on the value of pname) and does not.

4.4.27 OEMC_glTexParameterI

Description

set texture parameters

Prototypes

```
void OEMC_glTexParameterI( M_GGLenum target, M_GGLenum pname, M_GLint param )
```

Parameters

target

Specifies the target texture, which must be GL_TEXTURE_2D.

pname

Specifies the symbolic name of a single-valued texture parameter. Which can be one of the following: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

param

Specifies the value of pname.

Remarks

Texture mapping is a technique that applies an image onto an object's surface as if the image were a decal or cellophane shrink-wrap. The image is created in texture space, with an (s, t) coordinate system. A texture is a one- or two-dimensional image and a set of parameters that determine how samples are derived from the image.

glTexParameterI assigns the value or values in param to the texture parameter specified as pname. target defines the target texture, which must be GL_TEXTURE_2D.

The following symbols are accepted in pname:

GL_TEXTURE_MIN_FILTER

The texture minifying function is used whenever the pixel being textured maps to an area greater than one texture element. There are six defined minifying functions. Two of them use the nearest one or nearest four texture elements to compute the texture value. The other four use mipmaps.

A mipmap is an ordered set of arrays representing the same image at progressively lower resolutions. If the texture has dimensions $2n \times 2m$, there are $\max(n, m) + 1$ mipmaps. The first mipmap is the original texture, with dimensions $2n \times 2m$. Each subsequent mipmap has dimensions $2^{k-1} \times 2^{l-1}$, where $2^k \times 2^l$ are the dimensions of the previous

mipmap, until either $k = 0$ or $l = 0$. At that point, subsequent mipmaps have dimension $1 \times 2l - 1$ or $2k - 1 \times 1$ until the final mipmap, which has dimension 1×1 . To define the mipmaps, call `glTexImage2D` or `glCopyTexImage2D` with the level argument indicating the order of the mipmaps. Level 0 is the original texture. Level $\max(n, m)$ is the final 1×1 mipmap.

`param` supplies a function for minifying the texture as one of the following:

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

`GL_NEAREST_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value.

`GL_LINEAR_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.

`GL_NEAREST_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

`GL_LINEAR_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

As more texture elements are sampled in the minification process, fewer aliasing artifacts will be apparent. While the `GL_NEAREST` and `GL_LINEAR` minification functions can be faster than the other four, they sample only one or four texture elements to determine the texture value of the pixel being rendered and can produce moire patterns or ragged

transitions.

The initial value of `GL_TEXTURE_MIN_FILTER` is `GL_NEAREST_MIPMAP_LINEAR`.

`GL_TEXTURE_MAG_FILTER`

The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texture element. It sets the texture magnification function to either `GL_NEAREST` or `GL_LINEAR` (see below). `GL_NEAREST` is generally faster than `GL_LINEAR`, but it can produce textured images with sharper edges because the transition between texture elements is not as smooth.

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

The initial value of `GL_TEXTURE_MAG_FILTER` is `GL_LINEAR`.

`GL_TEXTURE_WRAP_S`

Sets the wrap parameter for texture coordinate `s` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`.

`GL_CLAMP`

causes `s` coordinates to be clamped to the range `[0, 1]` and is useful for preventing wrapping artifacts when mapping a single image onto an object.

`GL_CLAMP_TO_EDGE`

causes `s` coordinates to be clamped to the range `[1/2N , 1 - 1/2N]`, where `N` is the size of the texture in the direction of clamping.

`GL_REPEAT`

causes the integer part of the `s` coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern. Border texture elements are accessed only if wrapping is set to `GL_CLAMP`.

The initial value of `GL_TEXTURE_WRAP_S` is `GL_REPEAT`.

`GL_TEXTURE_WRAP_T`

Sets the wrap parameter for texture coordinate `t` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`. See the discussion under `GL_TEXTURE_WRAP_S`.

The initial value of `GL_TEXTURE_WRAP_T` is `GL_REPEAT`.

GL_GENERATE_MIPMAP

Sets the automatic mipmap generation parameter. If set to `GL_TRUE`, making any change to the interior or border texels of the levelbase array of a mipmap will also compute a complete set of mipmap arrays derived from the modified levelbase array. Array levels `levelbase + 1` through `p` are replaced with the derived arrays, regardless of their previous contents. All other mipmap arrays, including the levelbase array, are left unchanged by this computation.

The initial value of `GL_GENERATE_MIPMAP` is `GL_FALSE`.

Notes

Suppose that a program has enabled texturing (by calling `glEnable` with argument `GL_TEXTURE_2D` and has set `GL_TEXTURE_MIN_FILTER` to one of the functions that requires a mipmap. If either the dimensions of the texture images currently defined (with previous calls to `glTexImage2D`, or `glCopyTexImage2D`) do not follow the proper sequence for mipmaps (described above), or there are fewer texture images defined than are needed, or the set of texture images have differing numbers of texture components, then it is as if texture mapping were disabled.

Linear filtering accesses the four nearest texture elements.

`glTexParameter` specifies the texture parameters for the active texture unit, specified by calling `glActiveTexture`.

Errors

`GL_INVALID_ENUM` is generated if `target` or `pname` is not one of the accepted defined values.

`GL_INVALID_ENUM` is generated if `param` should have a defined constant value (based on the value of `pname`) and does not.

4.4.28 OEMC_glTexParameteriv

Description

set texture parameters

Prototypes

```
void OEMC_glTexParameteriv( M_GLenum target, M_GLenum pname, const M_GLint
*params )
```

Parameters

target

Specifies the target texture, which must be GL_TEXTURE_2D.

pname

Specifies the symbolic name of a single-valued texture parameter. Which can be one of the following: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

param

Specifies the value of pname.

Remarks

Texture mapping is a technique that applies an image onto an object's surface as if the image were a decal or cellophane shrink-wrap. The image is created in texture space, with an (s, t) coordinate system. A texture is a one- or two-dimensional image and a set of parameters that determine how samples are derived from the image.

glTexParameteriv assigns the value or values in param to the texture parameter specified as pname. target defines the target texture, which must be GL_TEXTURE_2D.

The following symbols are accepted in pname:

GL_TEXTURE_MIN_FILTER

The texture minifying function is used whenever the pixel being textured maps to an area greater than one texture element. There are six defined minifying functions. Two of them use the nearest one or nearest four texture elements to compute the texture value. The other four use mipmaps.

A mipmap is an ordered set of arrays representing the same image at progressively lower resolutions. If the texture has dimensions $2n \times 2m$, there are $\max(n, m) + 1$ mipmaps. The first mipmap is the original texture, with dimensions $2n \times 2m$. Each subsequent

mipmap has dimensions $2^k - 1 \times 2^l - 1$, where $2^k \times 2^l$ are the dimensions of the previous mipmap, until either $k = 0$ or $l = 0$. At that point, subsequent mipmaps have dimension $1 \times 2^l - 1$ or $2^k - 1 \times 1$ until the final mipmap, which has dimension 1×1 . To define the mipmaps, call `glTexImage2D` or `glCopyTexImage2D` with the level argument indicating the order of the mipmaps. Level 0 is the original texture. Level max (n, m) is the final 1×1 mipmap.

param supplies a function for minifying the texture as one of the following:

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

`GL_NEAREST_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value.

`GL_LINEAR_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.

`GL_NEAREST_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

`GL_LINEAR_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

As more texture elements are sampled in the minification process, fewer aliasing artifacts will be apparent. While the `GL_NEAREST` and `GL_LINEAR` minification functions can be faster than the other four, they sample only one or four texture elements to determine the

texture value of the pixel being rendered and can produce moire patterns or ragged transitions.

The initial value of `GL_TEXTURE_MIN_FILTER` is `GL_NEAREST_MIPMAP_LINEAR`.

`GL_TEXTURE_MAG_FILTER`

The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texture element. It sets the texture magnification function to either `GL_NEAREST` or `GL_LINEAR` (see below). `GL_NEAREST` is generally faster than `GL_LINEAR`, but it can produce textured images with sharper edges because the transition between texture elements is not as smooth.

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

The initial value of `GL_TEXTURE_MAG_FILTER` is `GL_LINEAR`.

`GL_TEXTURE_WRAP_S`

Sets the wrap parameter for texture coordinate `s` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`.

`GL_CLAMP`

causes `s` coordinates to be clamped to the range `[0, 1]` and is useful for preventing wrapping artifacts when mapping a single image onto an object.

`GL_CLAMP_TO_EDGE`

causes `s` coordinates to be clamped to the range `[1/2N , 1 - 1/2N]`, where `N` is the size of the texture in the direction of clamping.

`GL_REPEAT`

causes the integer part of the `s` coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern. Border texture elements are accessed only if wrapping is set to `GL_CLAMP`.

The initial value of `GL_TEXTURE_WRAP_S` is `GL_REPEAT`.

`GL_TEXTURE_WRAP_T`

Sets the wrap parameter for texture coordinate `t` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`. See the discussion under `GL_TEXTURE_WRAP_S`.

The initial value of `GL_TEXTURE_WRAP_T` is `GL_REPEAT`.

`GL_GENERATE_MIPMAP`

Sets the automatic mipmap generation parameter. If set to `GL_TRUE`, making any change to the interior or border texels of the levelbase array of a mipmap will also compute a complete set of mipmap arrays derived from the modified levelbase array. Array levels `levelbase + 1` through `p` are replaced with the derived arrays, regardless of their previous contents. All other mipmap arrays, including the levelbase array, are left unchanged by this computation.

The initial value of `GL_GENERATE_MIPMAP` is `GL_FALSE`.

Notes

Suppose that a program has enabled texturing (by calling `glEnable` with argument `GL_TEXTURE_2D` and has set `GL_TEXTURE_MIN_FILTER` to one of the functions that requires a mipmap. If either the dimensions of the texture images currently defined (with previous calls to `glTexImage2D`, or `glCopyTexImage2D`) do not follow the proper sequence for mipmaps (described above), or there are fewer texture images defined than are needed, or the set of texture images have differing numbers of texture components, then it is as if texture mapping were disabled.

Linear filtering accesses the four nearest texture elements.

`glTexParameter` specifies the texture parameters for the active texture unit, specified by calling `glActiveTexture`.

Errors

`GL_INVALID_ENUM` is generated if `target` or `pname` is not one of the accepted defined values.

`GL_INVALID_ENUM` is generated if `param` should have a defined constant value (based on the value of `pname`) and does not.

4.4.29 OEMC_glTexParameterxv

Description

set texture parameters

Prototypes

```
void OEMC_glTexParameterxv( M_GGLenum target, M_GGLenum pname, const M_GLfixed
*params )
```

Parameters

target

Specifies the target texture, which must be GL_TEXTURE_2D.

pname

Specifies the symbolic name of a single-valued texture parameter. Which can be one of the following: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, or GL_GENERATE_MIPMAP.

param

Specifies the value of pname.

Remarks

Texture mapping is a technique that applies an image onto an object's surface as if the image were a decal or cellophane shrink-wrap. The image is created in texture space, with an (s, t) coordinate system. A texture is a one- or two-dimensional image and a set of parameters that determine how samples are derived from the image.

glTexParameter assigns the value or values in param to the texture parameter specified as pname. target defines the target texture, which must be GL_TEXTURE_2D.

The following symbols are accepted in pname:

GL_TEXTURE_MIN_FILTER

The texture minifying function is used whenever the pixel being textured maps to an area greater than one texture element. There are six defined minifying functions. Two of them use the nearest one or nearest four texture elements to compute the texture value. The other four use mipmaps.

A mipmap is an ordered set of arrays representing the same image at progressively lower resolutions. If the texture has dimensions $2n \times 2m$, there are $\max(n, m) + 1$ mipmaps. The first mipmap is the original texture, with dimensions $2n \times 2m$. Each subsequent

mipmap has dimensions $2^k - 1 \times 2^l - 1$, where $2^k \times 2^l$ are the dimensions of the previous mipmap, until either $k = 0$ or $l = 0$. At that point, subsequent mipmaps have dimension $1 \times 2^l - 1$ or $2^k - 1 \times 1$ until the final mipmap, which has dimension 1×1 . To define the mipmaps, call `glTexImage2D` or `glCopyTexImage2D` with the level argument indicating the order of the mipmaps. Level 0 is the original texture. Level max (n, m) is the final 1×1 mipmap.

param supplies a function for minifying the texture as one of the following:

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

`GL_NEAREST_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value.

`GL_LINEAR_MIPMAP_NEAREST`

Chooses the mipmap that most closely matches the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.

`GL_NEAREST_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_NEAREST` criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

`GL_LINEAR_MIPMAP_LINEAR`

Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the `GL_LINEAR` criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

As more texture elements are sampled in the minification process, fewer aliasing artifacts will be apparent. While the `GL_NEAREST` and `GL_LINEAR` minification functions can be faster than the other four, they sample only one or four texture elements to determine the

texture value of the pixel being rendered and can produce moire patterns or ragged transitions.

The initial value of `GL_TEXTURE_MIN_FILTER` is `GL_NEAREST_MIPMAP_LINEAR`.

`GL_TEXTURE_MAG_FILTER`

The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texture element. It sets the texture magnification function to either `GL_NEAREST` or `GL_LINEAR` (see below). `GL_NEAREST` is generally faster than `GL_LINEAR`, but it can produce textured images with sharper edges because the transition between texture elements is not as smooth.

`GL_NEAREST`

Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

`GL_LINEAR`

Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T`, and on the exact mapping.

The initial value of `GL_TEXTURE_MAG_FILTER` is `GL_LINEAR`.

`GL_TEXTURE_WRAP_S`

Sets the wrap parameter for texture coordinate `s` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`.

`GL_CLAMP`

causes `s` coordinates to be clamped to the range `[0, 1]` and is useful for preventing wrapping artifacts when mapping a single image onto an object.

`GL_CLAMP_TO_EDGE`

causes `s` coordinates to be clamped to the range `[1/2N , 1 - 1/2N]`, where `N` is the size of the texture in the direction of clamping.

`GL_REPEAT`

causes the integer part of the `s` coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern. Border texture elements are accessed only if wrapping is set to `GL_CLAMP`.

The initial value of `GL_TEXTURE_WRAP_S` is `GL_REPEAT`.

`GL_TEXTURE_WRAP_T`

Sets the wrap parameter for texture coordinate `t` to either `GL_CLAMP`, `GL_CLAMP_TO_EDGE`, or `GL_REPEAT`. See the discussion under `GL_TEXTURE_WRAP_S`.

The initial value of `GL_TEXTURE_WRAP_T` is `GL_REPEAT`.

`GL_GENERATE_MIPMAP`

Sets the automatic mipmap generation parameter. If set to `GL_TRUE`, making any change to the interior or border texels of the levelbase array of a mipmap will also compute a complete set of mipmap arrays derived from the modified levelbase array. Array levels `levelbase + 1` through `p` are replaced with the derived arrays, regardless of their previous contents. All other mipmap arrays, including the levelbase array, are left unchanged by this computation.

The initial value of `GL_GENERATE_MIPMAP` is `GL_FALSE`.

Notes

Suppose that a program has enabled texturing (by calling `glEnable` with argument `GL_TEXTURE_2D` and has set `GL_TEXTURE_MIN_FILTER` to one of the functions that requires a mipmap. If either the dimensions of the texture images currently defined (with previous calls to `glTexImage2D`, or `glCopyTexImage2D`) do not follow the proper sequence for mipmaps (described above), or there are fewer texture images defined than are needed, or the set of texture images have differing numbers of texture components, then it is as if texture mapping were disabled.

Linear filtering accesses the four nearest texture elements.

`glTexParameter` specifies the texture parameters for the active texture unit, specified by calling `glActiveTexture`.

Errors

`GL_INVALID_ENUM` is generated if `target` or `pname` is not one of the accepted defined values.

`GL_INVALID_ENUM` is generated if `param` should have a defined constant value (based on the value of `pname`) and does not.

4.4.30 OEMC_gICurrentPaletteMatrixOES

Description

defines which of the palette's matrices is affected by subsequent matrix operations

Prototypes

```
void OEMC_gICurrentPaletteMatrixOES( M_GLuint matrixpaletteindex )
```

Parameters

index

specifies the index into the palette's matrices.

Remarks

gICurrentPaletteMatrixOES defines which of the palette's matrices is affected by subsequent matrix operations when the current matrix mode is GL_MATRIX_PALETTE_OES.

Errors

GL_INVALID_VALUE is generated if index is not between 0 and GL_MAX_PALETTE_MATRICES_OES - 1.

4.4.31 OEMC_gILoadPaletteFromModelViewMatrixOES**Description**

copies the current model view matrix to a matrix in the current matrix palette

Prototypes

```
void OEMC_gILoadPaletteFromModelViewMatrixOES( void )
```

Remarks

gILoadPaletteFromModelViewMatrixOES copies the current model view matrix to a matrix in the current matrix palette, as specified by glCurrentPaletteMatrixOES.

4.4.32 OEMC_glMatrixIndexPointerOES

Description

define an array of matrix indices

Prototypes

```
void OEMC_glMatrixIndexPointerOES( M_GLint size, M_GLenum type, M_GLsizei stride,  
const M_GLvoid *pointer )
```

Parameters

size

Specifies the number of matrix indices per vertex. Must be is less than or equal to GL_MAX_VERTEX_UNITS_OES. The initial value is 0.

type

Specifies the data type of each matrix index in the array. Symbolic constant GL_UNSIGNED_BYTE is accepted. The initial value is GL_UNSIGNED_BYTE.

stride

Specifies the byte offset between consecutive matrix indices. If stride is 0, the matrix indices are understood to be tightly packed in the array. The initial value is 0.

pointer

Specifies a pointer to the first matrix index of the first vertex in the array. The initial value is 0.

Remarks

glMatrixIndexPointer specifies the location and data of an array of matrix indices to use when rendering. size specifies the number of matrix indices per vertex and type the data type of the coordinates. stride specifies the byte stride from one matrix index to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations.)

These matrices indices are used to blend corresponding matrices for a given vertex.

When a matrix index array is specified, size, type, stride, and pointer are saved as client-side state.

If the matrix index array is enabled, it is used when glDrawArrays, or glDrawElements is called. To enable and disable the vertex array, call glEnableClientState and

glDisableClientState with the argument GL_MATRIX_INDEX_ARRAY_OES. The matrix index array is initially disabled and isn't accessed when glDrawArrays or glDrawElements is called.

Use glDrawArrays to construct a sequence of primitives (all of the same type) from prespecified vertex and vertex attribute arrays. Use glDrawElements to construct a sequence of primitives by indexing vertices and vertex attributes.

Notes

glMatrixIndexPointer is typically implemented on the client side.

Errors

GL_INVALID_VALUE is generated if size is greater than GL_MAX_VERTEX_UNITS_OES.

GL_INVALID_ENUM is generated if type is not an accepted value.

GL_INVALID_VALUE is generated if stride is negative.

4.4.33 OEMC_gIWeightPointerOES

Description

define an array of weights

Prototypes

```
void OEMC_gIWeightPointerOES( M_GLint size, M_GLenum type, M_GLsizei stride,  
const M_GLvoid *pointer )
```

Parameters

size

Specifies the number of weights per vertex. Must be is less than or equal to GL_MAX_VERTEX_UNITS_OES. The initial value is 0.

type

Specifies the data type of each weight in the array. Symbolic constant GL_FIXED is accepted. However, the common profile also accepts the symbolic constant GL_FLOAT as well. The initial value is GL_FIXED for the common lite profile, or GL_FLOAT for the common profile.

stride

Specifies the byte offset between consecutive weights. If stride is 0, the weights are understood to be tightly packed in the array. The initial value is 0.

pointer

Specifies a pointer to the first weight of the first vertex in the array. The initial value is 0.

Remarks

glIWeightPointer specifies the location and data of an array of weights to use when rendering. size specifies the number of weights per vertex and type the data type of the coordinates. stride specifies the byte stride from one weight to the next allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations.)

These weights are used to blend corresponding matrices for a given vertex.

When a weight array is specified, size, type, stride, and pointer are saved as client-side state.

If the weight array is enabled, it is used when glDrawArrays, or glDrawElements is called.

To enable and disable the vertex array, call `glEnableClientState` and `glDisableClientState` with the argument `GL_WEIGHT_ARRAY_OES`. The weight array is initially disabled and isn't accessed when `glDrawArrays` or `glDrawElements` is called.

Use `glDrawArrays` to construct a sequence of primitives (all of the same type) from prespecified vertex and vertex attribute arrays. Use `glDrawElements` to construct a sequence of primitives by indexing vertices and vertex attributes.

Notes

`glWeightPointer` is typically implemented on the client side.

Errors

`GL_INVALID_VALUE` is generated if `size` is greater than `GL_MAX_VERTEX_UNITS_OES`.

`GL_INVALID_ENUM` is generated if `type` is not an accepted value.

`GL_INVALID_VALUE` is generated if `stride` is negative.

4.4.34 OEMC_glPointSizePointerOES

Description

define an array of point sizes

Prototypes

```
void OEMC_glPointSizePointerOES( M_GGLenum type, M_GLsizei stride, const  
M_GLvoid *pointer )
```

Parameters

type

Specifies the data type of each point size in the array. Symbolic constant GL_FIXED is accepted. However, the common profile also accepts the symbolic constant GL_FLOAT as well. The initial value is GL_FIXED for the common lite profile, or GL_FLOAT for the common profile.

stride

Specifies the byte offset between consecutive point sizes. If stride is 0, the point sizes are understood to be tightly packed in the array. The initial value is 0.

pointer

Specifies a pointer to the point size of the first vertex in the array. The initial value is 0.

Remarks

glPointSizePointer specifies the location and data of an array of point sizes to use when rendering points. type the data type of the coordinates. stride specifies the byte stride from one point size to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations.)

The point sizes supplied in the point size arrays will be the sizes used to render both points and point sprites.

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

When a point size array is specified, type, stride, and pointer are saved as client-side state.

If the point size array is enabled, it is used to control the sizes used to render points and

point sprites. To enable and disable the point size array, call `glEnableClientState` and `glDisableClientState` with the argument `GL_POINT_SIZE_ARRAY_OES`. The point size array is initially disabled.

Notes

If point size array is enabled, the point size defined by `glPointSize` is ignored.
`glPointSizePointer` is typically implemented on the client side.

Errors

`GL_INVALID_ENUM` is generated if `type` is not an accepted value.
`GL_INVALID_VALUE` is generated if `stride` is negative.

4.4.35 OEMC_gIDrawTexsOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_gIDrawTexsOES( M_GLshort x, M_GLshort y, M_GLshort z, M_GLshort width, M_GLshort height )
```

Parameters

x, y, z

Specify the position of the affected screen rectangle.

width, height

Specifies the width and height of the affected screen rectangle in pixels.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Z_w as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of `GL_DEPTH_RANGE` respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

Calling one of the DrawTex functions generates a fragment for each pixel that overlaps the screen rectangle bounded by (x, y) and (x + width), (y + height). For each generated fragment, the depth is given by Z_w as defined above, and the color by the current color.

Texture coordinates for each texture unit are computed as follows:

Let X and Y be the screen x and y coordinates of each sample point associated with the fragment. Let W_t and H_t be the width and height in texels of the texture currently bound to the texture unit. (If the texture is a mipmap, let W_t and H_t be the dimensions of the level specified by `GL_TEXTURE_BASE_LEVEL`). Let U_{cr} , V_{cr} , W_{cr} and H_{cr} be (respectively) the four integers that make up the texture crop rectangle parameter for the currently bound texture. The fragment texture coordinates (s, t, r, q) are given by:

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$

$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$

$$r = 0$$

$$q = 1$$

Notes

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

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

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

glDrawTexOES is available only if the GL_OES_draw_texture extension is supported by your implementation.

4.4.36 OEMC_gIDrawTexiOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_gIDrawTexiOES( M_GLint x, M_GLint y, M_GLint z, M_GLint width, M_GLint height )
```

Parameters

x, y, z

Specify the position of the affected screen rectangle.

width, height

Specifies the width and height of the affected screen rectangle in pixels.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Zw as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of GL_DEPTH_RANGE respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

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

Texture coordinates for each texture unit are computed as follows:

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

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$

$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$

$$r = 0$$

$$q = 1$$

Notes

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

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

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

glDrawTexOES is available only if the GL_OES_draw_texture extension is supported by your implementation.

4.4.37 OEMC_gIDrawTexxOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_gIDrawTexxOES( M_GLfixed x, M_GLfixed y, M_GLfixed z, M_GLfixed width,  
M_GLfixed height )
```

Parameters

x, y, z

Specify the position of the affected screen rectangle.

width, height

Specifies the width and height of the affected screen rectangle in pixels.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Z_w as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of GL_DEPTH_RANGE respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

Calling one of the DrawTex functions generates a fragment for each pixel that overlaps the screen rectangle bounded by (x, y) and (x + width), (y + height). For each generated fragment, the depth is given by Z_w as defined above, and the color by the current color.

Texture coordinates for each texture unit are computed as follows:

Let X and Y be the screen x and y coordinates of each sample point associated with the fragment. Let W_t and H_t be the width and height in texels of the texture currently bound to the texture unit. (If the texture is a mipmap, let W_t and H_t be the dimensions of the level specified by GL_TEXTURE_BASE_LEVEL). Let U_{cr} , V_{cr} , W_{cr} and H_{cr} be (respectively) the four integers that make up the texture crop rectangle parameter for the currently bound texture. The fragment texture coordinates (s, t, r, q) are given by:

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$

$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$

$$r = 0$$

$$q = 1$$

Notes

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

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

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

glDrawTexOES is available only if the GL_OES_draw_texture extension is supported by your implementation.

4.4.38 OEMC_glDrawTexsvOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_glDrawTexsvOES( const M_GLshort *coords )
```

Parameters

coords

Specifies a pointer to the coords for the affected screen rectangle.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Zw as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of GL_DEPTH_RANGE respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

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

Texture coordinates for each texture unit are computed as follows:

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

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$
$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$
$$r = 0$$
$$q = 1$$

Notes

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

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

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

`glDrawTexOES` is available only if the `GL_OES_draw_texture` extension is supported by your implementation.

4.4.39 OEMC_glDrawTexivOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_glDrawTexivOES( const M_GLint *coords )
```

Parameters

coords

Specifies a pointer to the coords for the affected screen rectangle.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Zw as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of GL_DEPTH_RANGE respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

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

Texture coordinates for each texture unit are computed as follows:

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

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$

$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$

$$r = 0$$

$$q = 1$$

Notes

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

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

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

`glDrawTexOES` is available only if the `GL_OES_draw_texture` extension is supported by your implementation.

4.4.40 OEMC_glDrawTexvOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_glDrawTexvOES( const M_GLfixed *coords )
```

Parameters

coords

Specifies a pointer to the coords for the affected screen rectangle.

Remarks

glDrawTexOES draws a texture rectangle to the screen.

x and y are given directly in window (viewport) coordinates.

z is mapped to window depth Zw as follows:

If $z \leq 0$ then $Z_w = n$

If $z \geq 1$ then $Z_w = f$

Otherwise $Z_w = n + z * (f - n)$

where n and f are the near and far values of GL_DEPTH_RANGE respectively.

width and height specify the width and height of the affected screen rectangle in pixels.

These values may be positive or negative; however if either of these are negative, nothing is drawn.

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

Texture coordinates for each texture unit are computed as follows:

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

$$s = (Ucr + (X - x) * (Wcr / width)) / Wt$$

$$t = (Vcr + (Y - y) * (Hcr / height)) / Ht$$

$$r = 0$$

$$q = 1$$

Notes

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

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

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

`glDrawTexOES` is available only if the `GL_OES_draw_texture` extension is supported by your implementation.