

GSR-011***rev 1.00.00******OpenGL ES 1.1 Common
(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	5
1.1 개요	5
1.2 목적	5
1.3 규격의 범위.....	5
1.4 용어 정의	5
2 의존성	6
2.1 WIPI 의존성.....	6
2.2 GIGA 의존성	6
2.3 GSR간 의존성	6
2.4 OAT 의존성.....	6
3 제조사 포팅 가이드	7
4 개발자 가이드	8
4.1 Fundamentals	8
4.2 Type 정의.....	8
4.3 Basic 3D API Overview (OpenGL/ES 1.0 Common addition only)	8
4.4 Basic 3D API (OpenGL/ES 1.0 Common addition only) Description	10

세부목차

1 일반사항	5
1.1 개요	5
1.2 목적	5
1.3 규격의 범위.....	5
1.4 용어 정의	5
2 의존성	6
2.1 WIPI 의존성.....	6
2.2 GIGA 의존성	6
2.3 GSR간 의존성	6
2.4 OAT 의존성.....	6
3 제조사 포팅 가이드	7
4 개발자 가이드	8
4.1 Fundamentals	8
4.1.1 Matrix Convention	8
4.1.2 Coordinate system	8
4.2 Type 정의.....	8
4.3 Basic 3D API Overview (OpenGL/ES 1.0 Common addition only)	8
4.4 Basic 3D API (OpenGL/ES 1.0 Common addition only) Description	10
4.4.1 OEMC_glClipPlanef	10
4.4.2 OEMC_glGetClipPlanef	12
4.4.3 OEMC_glGetFloatv	13
4.4.4 OEMC_glGetLightfv	16
4.4.5 OEMC_glGetMaterialfv	19
4.4.6 OEMC_glGetTexEnvfv	21
4.4.7 OEMC_glGetTexParameterfv	23
4.4.8 OEMC_glPointParameterf	25
4.4.9 OEMC_glPointParameterfv	27
4.4.10 OEMC_glTexParameterfv	29
4.4.11 OEMC_glDrawTexfOES	33
4.4.12 OEMC_glDrawTexfvOES	35

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_GLenum	unsigned int
M_GLfloat	float

■ 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_glClipPlanef(M_GLenum plane, const M_GLfloat *equation)	specify a plane against which all geometry is clipped
void OEMC_glGetClipPlanef(M_GLenum pname, M_GLfloat eqn[4])	return the coefficients of the specified clipping plane
void OEMC_glGetFloatv(M_GLenum pname, M_GLfloat *params)	Return the float-point value or float-point values of a selected parameter
void OEMC_glGetLightfv(M_GLenum light, M_GLenum pname, M_GLfloat *params)	return light source parameter values
void OEMC_glGetMaterialfv(M_GLenum face, M_GLenum pname, M_GLfloat *params)	return material parameters values
void OEMC_glGetTexEnvfv(M_GLenum env, M_GLenum	return texture environment parameters

pname, M_GLfloat *params)	
void OEMC_glGetTexParameterfv(M_GLenum target, M_GLenum pname, M_GLfloat *params)	return texture parameter values
void OEMC_glPointParameterf(M_GLenum pname, M_GLfloat param)	specify parameters for point rasterization
void OEMC_glPointParameterfv(M_GLenum pname, const M_GLfloat *params)	specify parameters for point rasterization
void OEMC_glTexParameterfv(M_GLenum target, M_GLenum pname, const M_GLfloat *params)	set texture parameters
void OEMC_glDrawTexfOES(M_GLfloat x, M_GLfloat y, M_GLfloat z, M_GLfloat width, M_GLfloat height)	draws a texture rectangle to the screen
void OEMC_glDrawTexfvOES(const M_GLfloat *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_glClipPlanef

Description

specify a plane against which all geometry is clipped

Prototypes

```
void OEMC_glClipPlanef( M_GGLenum plane, const M_GLfloat *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 float-point or float-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 $GL_CLIP_PLANEi = GL_CLIP_PLANE0 + i$.

Errors

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

4.4.2 OEMC_glGetClipPlanef

Description

return the coefficients of the specified clipping plane

Prototypes

```
void OEMC_glGetClipPlanef( M_GGLenum pname, M_GLfloat 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 float-point or float-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.

4.4.3 OEMC_glGetFloatv

Description

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

Prototypes

```
void OEMC_glGetFloatv( M_GLEnum pname, M GLfloat *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 float-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 float-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 float-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 float-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_LINE_WIDTH

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

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 float-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`.

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`.

4.4.4 OEMC_glGetLightfv

Description

return light source parameter values

Prototypes

```
void OEMC_glGetLightfv( M_GGLenum light, M_GGLenum pname, M_GLfloat *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 $0 < i < GL_MAX_LIGHTS$

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 $0 < i < GL_MAX_LIGHTS$ 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 float-point values that specify the ambient RGBA intensity of the light. Both float-point and float-point values are mapped directly. Neither float-point nvalues are clamped. The initial ambient light intensity is (0, 0, 0, 1).

`GL_DIFFUSE`

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

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 float-point values that specify the specular RGBA intensity of the light. Both float-point and float-point values are mapped directly. Neither float-point nor float-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 float-point values representing the emitted light intensity of the material. Both float-point and float-point values are mapped directly. The initial value is (0, 0, 0, 1).

`GL_SPOT_DIRECTION`

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

`GL_SPOT_EXPONENT`

`params` returns a single float-point value that specifies the intensity distribution of the light. Float-point and float-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 float-point value that specifies the maximum spread angle of a light source. Float-point and float-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 float-point value that specifies one of the three light attenuation factors. Float-point and float-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.

4.4.5 OEMC_glGetMaterialfv

Description

return material parameters values

Prototypes

```
void OEMC_glGetMaterialfv( M_GGLenum face, M_GGLenum pname, M_GLfloat *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 float-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 float-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 float-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 float-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 float-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.

4.4.6 OEMC_glGetTexEnvfv

Description

return texture environment parameters

Prototypes

```
void OEMC_glGetTexEnvfv( M_GGLenum env, M_GGLenum pname, M_GLfloat *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 float 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.

4.4.7 OEMC_glGetTexParameterfv

Description

return texture parameter values

Prototypes

```
void OEMC_glGetTexParameterfv( M_GGLenum target, M_GGLenum pname, M_GLfloat  
*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.

4.4.8 OEMC_glPointParameterf

Description

specify parameters for point rasterization

Prototypes

```
void OEMC_glPointParameterf( M_GLenum pname, M_GLfloat 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.

4.4.9 OEMC_glPointSizeParameterfv

Description

specify parameters for point rasterization

Prototypes

```
void OEMC_glPointSizeParameterfv( M_GLEnum pname, const M GLfloat *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

`glPointSizeParameter` assigns values to point parameters.

`glPointSizeParameter` 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 `glPointSizeParameter`, 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.

4.4.10 OEMC_glTexParameterfv

Description

set texture parameters

Prototypes

```
void OEMC_glTexParameterfv( M_GGLenum target, M_GGLenum pname, const M_GLfloat  
*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.

4.4.11 OEMC_gIDrawTexfOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_gIDrawTexfOES( M_GLfloat x, M_GLfloat y, M_GLfloat z, M_GLfloat width,  
M_GLfloat 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

gIDrawTexOES 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:

$$s = (U_{cr} + (X - x) * (W_{cr} / width)) / W_t$$

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

$$r = 0$$

$$q = 1$$

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.

4.4.12 OEMC_glDrawTexfvOES

Description

draws a texture rectangle to the screen

Prototypes

```
void OEMC_glDrawTexfvOES( const M_GLfloat *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.