Specifying an invalid *texture* generates the error INVALID_ENUM. Valid values of *texture* are the same as for the **MultiTexCoord** commands described in section 2.7.

There is a stack of matrices for each of matrix modes MODELVIEW and PROJECTION, and for each texture unit. For MODELVIEW mode, the stack depth is at least 16 (that is, there is a stack of at least 16 model-view matrices). For the other modes, the depth is at least 2. Texture matrix stacks for all texture units have the same depth. The current matrix in any mode is the matrix on the top of the stack for that mode.

`void` **PushMatrix**(`void`);

pushes the stack down by one, duplicating the current matrix in both the top of the stack and the entry below it.

`void` **PopMatrix**(`void`);

pops the top entry off of the stack, replacing the current matrix with the matrix that was the second entry in the stack. The pushing or popping takes place on the stack corresponding to the current matrix mode. Popping a matrix off a stack with only one entry generates the error STACK_UNDERFLOW; pushing a matrix onto a full stack generates STACK_OVERFLOW.

When the current matrix mode is TEXTURE, the texture matrix stack of the active texture unit is pushed or popped.

The state required to implement transformations consists of an integer for the active texture unit selector, a four-valued integer indicating the current matrix mode, one stack of at least two 4×4 matrices for each of PROJECTION and each texture unit, TEXTURE; and a stack of at least 16 4×4 matrices for MODELVIEW. Each matrix stack has an associated stack pointer. Initially, there is only one matrix on each stack, and all matrices are set to the identity. The initial active texture unit selector is TEXTURE0, and the initial matrix mode is MODELVIEW.

2.10.3 Normal Transformation

Finally, we consider how the model-view matrix and transformation state affect normals. Before use in lighting, normals are transformed to eye coordinates by a matrix derived from the model-view matrix. Rescaling and normalization operations are performed on the transformed normals to make them unit length prior to use in lighting. Rescaling and normalization are controlled by

`void` **Enable**(`enum` *target*);

relationships generates the error INVALID_VALUE:

$$x < 0$$
$$x + w > w_s$$
$$y < 0$$
$$y + h > h_s$$

Counting from zero, the nth pixel group is assigned to the texel with internal integer coordinates $[i, j]$, where

$$i = x + (n \bmod w)$$
$$j = y + (\lfloor \frac{n}{w} \rfloor \bmod h)$$

3.7.3 Compressed Texture Images

Texture images may also be specified or modified using image data already stored in a known compressed image format. The GL defines some specific compressed formats, and others may be defined by GL extensions. There is a mechanism to obtain token values for compressed formats; the number of specific compressed internal formats supported can be obtained by querying the value of NUM_COMPRESSED_TEXTURE_FORMATS. The set of specific compressed internal formats supported by the renderer can be obtained by querying the value of COMPRESSED_TEXTURE_FORMATS. The only values returned by this query are those corresponding to *internalformat* parameters accepted by **CompressedTexImage2D** and suitable for general-purpose usage. The renderer will not enumerate formats with restrictions that need to be specifically understood prior to use.

The command

```
void CompressedTexImage2D( enum target, int level,
    enum internalformat, sizei width, sizei height,
    int border, sizei imageSize, void *data );
```

defines a two-dimensional texture image, with incoming data stored in a specific compressed image format. The *target*, *level*, *internalformat*, *width*, *height*, and *border* parameters have the same meaning as in **TexImage2D**. *data* points to compressed image data stored in the compressed image format corresponding to *internalformat*.

For all compressed internal formats, the compressed image will be decoded according to the definition of *internalformat*. Compressed texture images are treated

as an array of *imageSize* `ubyte`s beginning at address *data*. All pixel storage and pixel transfer modes are ignored when decoding a compressed texture image. If the *imageSize* parameter is not consistent with the format, dimensions, and contents of the compressed image, an `INVALID_VALUE` error results. If the compressed image is not encoded according to the defined image format, the results of the call are undefined.

Specific compressed internal formats may impose format-specific restrictions on the use of the compressed image specification calls or parameters. For example, the compressed image format might not allow *width* or *height* values that are not a multiple of 4. Any such restrictions will be documented in the extension specification defining the compressed internal format; violating these restrictions will result in an `INVALID_OPERATION` error.

Any restrictions imposed by specific compressed internal formats will be invariant with respect to image contents, meaning that if the GL accepts and stores a texture image in compressed form, **CompressedTexImage2D** will accept any properly encoded compressed texture image of the same width, height, compressed image size, and compressed internal format for storage at the same texture level.

The specific compressed texture formats supported by **CompressedTexImage2D**, and the corresponding base internal format for each specific format, are defined in table 3.10.

Compressed Texture Format	Base Internal Format
`PALETTE4_RGB8_OES`	`RGB`
`PALETTE4_RGBA8_OES`	`RGBA`
`PALETTE4_R5_G6_B5_OES`	`RGB`
`PALETTE4_RGBA4_OES`	`RGBA`
`PALETTE4_RGB5_A1_OES`	`RGBA`
`PALETTE8_RGB8_OES`	`RGB`
`PALETTE8_RGBA8_OES`	`RGBA`
`PALETTE8_R5_G6_B5_OES`	`RGB`
`PALETTE8_RGBA4_OES`	`RGBA`
`PALETTE8_RGB5_A1_OES`	`RGBA`

Table 3.10: Specific compressed texture formats.

Respecifying Subimages of Compressed Textures

The commands

```
void CompressedTexSubImage2D( enum target, int level,
    int xoffset, int yoffset, sizei width, sizei height,
    enum format, sizei imageSize, void *data );
```

respecify only a rectangular region of an existing texture array, with incoming data stored in a known compressed image format. The *target*, *level*, *xoffset*, *yoffset*, *width*, *height*, and *format* parameters have the same meaning as in **TexSubImage2D**. *data* points to compressed image data stored in the compressed image format corresponding to *format*.

The image pointed to by *data* and the *imageSize* parameter is interpreted as though it was provided to **CompressedTexImage2D**. This command does not provide for image format conversion, so an INVALID_OPERATION error results if *format* does not match the internal format of the texture image being modified. If the *imageSize* parameter is not consistent with the format, dimensions, and contents of the compressed image (too little or too much data), an INVALID_VALUE error results.

As with **CompressedTexImage** calls, compressed internal formats may have additional restrictions on the use of the compressed image specification calls or parameters. Any such restrictions will be documented in the specification defining the compressed internal format; violating these restrictions will result in an INVALID_OPERATION error.

Any restrictions imposed by specific compressed internal formats will be invariant with respect to image contents, meaning that if the GL accepts and stores a texture image in compressed form, **CompressedTexSubImage2D** will accept any properly encoded compressed texture image of the same width, height, compressed image size, and compressed internal format for storage at the same texture level.

Calling **CompressedTexSubImage2D** will result in an INVALID_OPERATION error if *xoffset* or *yoffset* is not equal to zero (border width), or if *width* and *height* do not match the values of TEXTURE_WIDTH and TEXTURE_HEIGHT respectively. The contents of any texel outside the region modified by the call are undefined. These restrictions may be relaxed for specific compressed internal formats whose images are easily modified.

3.7.4 Compressed Paletted Textures

If *internalformat* is PALETTE4_RGB8, PALETTE4_RGBA8, PALETTE4_R5_G6_B5, PALETTE4_RGBA4, PALETTE4_RGB5_A1, PALETTE8_RGB8, PALETTE8_RGBA8, PALETTE8_R5_G6_B5, PALETTE8_RGBA4, or PALETTE8_RGB5_A1, the compressed texture is a compressed paletted texture. *data* contains the palette data following by the mipmap levels, where the number of mipmap levels stored is

given by $|level| + 1$. The number of bits that represent a texel is 4 bits if *interalformat* is PALETTE4_* and is 8 bits if *internalformat* is PALETTE8_*.

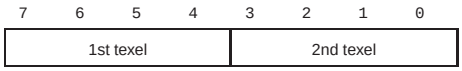
The palette data is formatted as an image containing 16 (for PALETTE4_*) or 256 (for PALETTE8_*) palette entries (pixels). The equivalent *format* and *type* of each palette entry is shown in table 3.11.

Compressed Texture Format	Palette entry *format*	Palette entry *type*
PALETTE4_RGB8_OES	RGB	UNSIGNED_BYTE
PALETTE4_RGBA8_OES	RGBA	UNSIGNED_BYTE
PALETTE4_R5_G6_B5_OES	RGB	UNSIGNED_SHORT_5_6_5
PALETTE4_RGBA4_OES	RGBA	UNSIGNED_SHORT_4_4_4_4
PALETTE4_RGB5_A1_OES	RGBA	UNSIGNED_SHORT_5_5_5_1
PALETTE8_RGB8_OES	RGB	UNSIGNED_BYTE
PALETTE8_RGBA8_OES	RGBA	UNSIGNED_BYTE
PALETTE8_R5_G6_B5_OES	RGB	UNSIGNED_SHORT_5_6_5
PALETTE8_RGBA4_OES	RGBA	UNSIGNED_SHORT_4_4_4_4
PALETTE8_RGB5_A1_OES	RGBA	UNSIGNED_SHORT_5_5_5_1

Table 3.11: Palette entry pixel formats.

Image data immediately follows the palette image. Each mipmap level image present in the image data immediately follows the previous level, starting with mipmap level zero and proceeding through the number of levels defined by $|level| + 1$. Texels within each mipmap level image are formatted as shown in table 3.12 and are packed contiguously starting at the lower left.

PALETTE4_*:



PALETTE8_*:

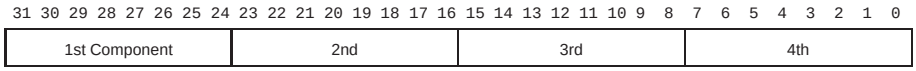


Table 3.12: Texel data formats for compressed paletted textures.

If a compressed paletted texture is specified with a positive *level* argument to

Name	Type	Legal Values
TEXTURE_WRAP_S	integer	CLAMP_TO_EDGE, REPEAT
TEXTURE_WRAP_T	integer	CLAMP_TO_EDGE, REPEAT
TEXTURE_MIN_FILTER	integer	NEAREST, LINEAR, NEAREST_MIPMAP_NEAREST, NEAREST_MIPMAP_LINEAR, LINEAR_MIPMAP_NEAREST, LINEAR_MIPMAP_LINEAR,
TEXTURE_MAG_FILTER	integer	NEAREST, LINEAR
GENERATE_MIPMAP	boolean	TRUE or FALSE

Table 3.13: Texture parameters and their values.

TexImage2D, an INVALID_VALUE error is generated.

Subimages may not be specified for compressed paletted textures. Calling **CompressedTexSubImage2D** with any of the PALETTE* arguments in table 3.11 will generate an INVALID_OPERATION error.

3.7.5 Texture Parameters

Various parameters control how the texture array is treated when specified or changed, and when applied to a fragment. Each parameter is set by calling

```
void TexParameter{ixf}( enum target, enum pname,
    T param );
void TexParameter{ixf}v( enum target, enum pname,
    T params );
```

target is the target, which must be TEXTURE_2D. *pname* is a symbolic constant indicating the parameter to be set; the possible constants and corresponding parameters are summarized in table 3.13. In the first form of the command, *param* is a value to which to set a single-valued parameter; in the second form of the command, *params* is an array of parameters whose type depends on the parameter being set.

If the value of texture parameter GENERATE_MIPMAP is TRUE, specifying or changing texture arrays may have side effects, which are discussed in the **Automatic Mipmap Generation** discussion of section 3.7.7.

6.1.3 Enumerated Queries

Other commands exist to obtain state variables that are identified by a category (clip plane, light, material, etc.) as well as a symbolic constant. These are

```
void GetClipPlane{xf}( enum plane, T eqn[4] );
void GetLight{xf}v( enum light, enum value, T data );
void GetMaterial{xf}v( enum face, enum value, T data );
void GetTexEnv{ixf}v( enum env, enum value, T data );
void GetTexParameter{ixf}v( enum target, enum value,
   T data );
void GetBufferParameteriv( enum target, enum value,
   T data );
```

GetClipPlane always returns four values in *eqn*; these are the coefficients of the plane equation of *plane* in eye coordinates (these coordinates are those that were computed when the plane was specified).

GetLight places information about *value* (a symbolic constant) for *light* (also a symbolic constant) in *data*. POSITION or SPOT_DIRECTION returns values in eye coordinates (again, these are the coordinates that were computed when the position or direction was specified).

GetMaterial, **GetTexEnv**, **GetTexParameter**, and **GetBufferParameter** are similar to **GetLight**, placing information about *value* for the target indicated by their first argument into *data*. The *face* argument to **GetMaterial** must be either FRONT or BACK, indicating the front or back material, respectively. The *env* argument to **GetTexEnv** must be TEXTURE_ENV.

GetTexParameter parameter *target* must be TEXTURE_2D, indicating the currently bound texture object. *value* is a symbolic value indicating which texture parameter is to be obtained. For **GetTexParameter**, *value* must be one of the symbolic values in table 3.13.

■

6.1.4 Texture Queries

The command

```
boolean IsTexture( uint texture );
```

returns TRUE if *texture* is the name of a texture object. If *texture* is zero, or is a nonzero value that is not the name of a texture object, or if an error condition occurs, **IsTexture** returns FALSE. A name returned by **GenTextures**, but not yet bound, is not the name of a texture object.

Remi Arnaud, Sony Computer Entertainment

Robert Simpson, Bitboys

Tero Sarkinnen, Futuremark

Timo Suoranta, Futuremark

Thomas Tannert, Silicon Graphics

Tomi Aarnio, Nokia

Tom McReynolds, Nvidia

Tom Olson, Texas Instruments

Ville Miettinen, Hybrid Graphics

C.4.4 Document History

version 1.1.10, draft of 2007/01/05 Initial revision of the full specification, based on the 1.1.09 diff specification.

version 1.1.10, draft of 2007/01/09 Add Khronos copyright page. Remove COLOR matrix from section 2.10.2. Reorganized compressed texture language (section 3.7.3) and moved language specific to compressed paletted textures into a new section 3.7.4; added more detail of the format of compressed paletted textures in memory and specified that **CompressedTexSubImage2D** may not be called for them. Removed state not present or not exposed in OpenGL ES , including all texture level-specific parameters from section 6.1.3, table 6.15 (state per texture image), and the state table entries for `COLOR_MATERIAL_PARAMETER`, `COLOR_MATERIAL_FACE`, `TEXTURE_INTENSITY_SIZE`, `TEXTURE_DEPTH_SIZE`, `DRAW_BUFFER`, `READ_BUFFER`, `AUX_BUFFERS`, `DOUBLEBUFFER`, `STEREO`, `SMOOTH_POINT_SIZE_GRANULARITY`, and `SMOOTH_LINE_WIDTH_GRANULARITY`.