Name

glTexImage2D — specify a two-dimensional texture image

C Specification

void **glTexImage2D**(GLenum *target*,
GLint *level*,
GLint *internalformat*,
GLsizei *width*,
GLsizei *height*,
GLint *border*,
GLenum *format*,
GLenum *type*,
const void * *data*);

Parameters

target

Specifies the target texture. Must be `GL_TEXTURE_2D`, `GL_PROXY_TEXTURE_2D`, `GL_TEXTURE_1D_ARRAY`, `GL_PROXY_TEXTURE_1D_ARRAY`, `GL_TEXTURE_RECTANGLE`, `GL_PROXY_TEXTURE_RECTANGLE`, `GL_TEXTURE_CUBE_MAP_POSITIVE_X`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_X`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Y`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Y`, `GL_TEXTURE_CUBE_MAP_POSITIVE_Z`, `GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`, or `GL_PROXY_TEXTURE_CUBE_MAP`.

level

Specifies the level-of-detail number. Level 0 is the base image level. Level *n* is the *n*th mipmap reduction image. If *target* is `GL_TEXTURE_RECTANGLE` or `GL_PROXY_TEXTURE_RECTANGLE`, *level* must be 0.

internalformat

Specifies the number of color components in the texture. Must be one of base internal formats given in Table 1, one of the sized internal formats given in Table 2, or one of the compressed internal formats given in Table 3, below.

width

Specifies the width of the texture image. All implementations support texture images that are at least 1024 texels wide.

height

Specifies the height of the texture image, or the number of layers in a texture array, in the case of the `GL_TEXTURE_1D_ARRAY` and

GL_PROXY_TEXTURE_1D_ARRAY targets. All implementations support 2D texture images that are at least 1024 texels high, and texture arrays that are at least 256 layers deep.

border

This value must be 0.

format

Specifies the format of the pixel data. The following symbolic values are accepted: GL_RED, GL_RG, GL_RGB, GL_BGR, GL_RGBA, GL_BGRA, GL_RED_INTEGER, GL_RG_INTEGER, GL_RGB_INTEGER, GL_BGR_INTEGER, GL_RGBA_INTEGER, GL_BGRA_INTEGER, GL_STENCIL_INDEX, GL_DEPTH_COMPONENT, GL_DEPTH_STENCIL.

type

Specifies the data type of the pixel data. The following symbolic values are accepted: GL_UNSIGNED_BYTE, GL_BYTE, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_HALF_FLOAT, GL_FLOAT, GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_8_REV, GL_UNSIGNED_INT_10_10_10_2, and GL_UNSIGNED_INT_2_10_10_10_REV.

data

Specifies a pointer to the image data in memory.

Description

Texturing allows elements of an image array to be read by shaders.

To define texture images, call **glTexImage2D**. The arguments describe the parameters of the texture image, such as height, width, width of the border, level-of-detail number (see glTexParameter), and number of color components provided. The last three arguments describe how the image is represented in memory.

If *target* is GL_PROXY_TEXTURE_2D, GL_PROXY_TEXTURE_1D_ARRAY, GL_PROXY_TEXTURE_CUBE_MAP, or GL_PROXY_TEXTURE_RECTANGLE, no data is read from *data*, but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see glGetError). To query for an entire mipmap array, use an image array level greater than or equal to 1.

If *target* is GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE or one of the GL_TEXTURE_CUBE_MAP targets, data is read from *data* as a sequence of signed or unsigned bytes, shorts, or longs, or single-precision floating-point values, depending on *type*. These values are grouped into sets of one, two, three, or four values, depending on *format*, to form elements. Each data byte is treated

as eight 1-bit elements, with bit ordering determined by GL_UNPACK_LSB_FIRST (see glPixelStore).

If *target* is GL_TEXTURE_1D_ARRAY, data is interpreted as an array of one-dimensional images.

If a non-zero named buffer object is bound to the GL_PIXEL_UNPACK_BUFFER target (see glBindBuffer) while a texture image is specified, *data* is treated as a byte offset into the buffer object's data store.

The first element corresponds to the lower left corner of the texture image. Subsequent elements progress left-to-right through the remaining texels in the lowest row of the texture image, and then in successively higher rows of the texture image. The final element corresponds to the upper right corner of the texture image.

format determines the composition of each element in *data*. It can assume one of these symbolic values:

GL_RED

Each element is a single red component. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for green and blue, and 1 for alpha. Each component is clamped to the range [0,1].

GL_RG

Each element is a red/green double. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for blue, and 1 for alpha. Each component is clamped to the range [0,1].

GL_RGB, GL_BGR

Each element is an RGB triple. The GL converts it to floating point and assembles it into an RGBA element by attaching 1 for alpha. Each component is clamped to the range [0,1].

GL_RGBA, GL_BGRA

Each element contains all four components. Each component is clamped to the range [0,1].

GL_DEPTH_COMPONENT

Each element is a single depth value. The GL converts it to floating point and clamps to the range [0,1].

GL_DEPTH_STENCIL

Each element is a pair of depth and stencil values. The depth component of the pair is interpreted as in GL_DEPTH_COMPONENT. The stencil component is interpreted based on specified the depth + stencil internal format.

If an application wants to store the texture at a certain resolution or in a certain format, it can request the resolution and format with *internalformat*. The GL will choose an internal representation that closely approximates that requested by *internalformat*, but it may not match exactly. (The representations specified by GL_RED, GL_RG, GL_RGB, and GL_RGBA must match exactly.)

internalformat may be one of the base internal formats shown in Table 1, below

Table 1. Base Internal Formats

Base Internal Format	RGBA, Depth and Stencil Values	Internal Components
GL_DEPTH_COMPONENT	Depth	D
GL_DEPTH_STENCIL	Depth, Stencil	D, S
GL_RED	Red	R
GL_RG	Red, Green	R, G
GL_RGB	Red, Green, Blue	R, G, B
GL_RGBA	Red, Green, Blue, Alpha	R, G, B, A

internalformat may also be one of the sized internal formats shown in Table 2, below

Table 2. Sized Internal Formats

Sized Internal Format	Base Internal Format	Red Bits	Green Bits	Blue Bits	Alpha Bits	Shared Bits
GL_R8	GL_RED	8				
GL_R8_SNORM	GL_RED	s8				
GL_R16	GL_RED	16				
GL_R16_SNORM	GL_RED	s16				
GL_RG8	GL_RG	8	8			
GL_RG8_SNORM	GL_RG	s8	s8			
GL_RG16	GL_RG	16	16			
GL_RG16_SNORM	GL_RG	s16	s16			
GL_R3_G3_B2	GL_RGB	3	3	2		
GL_RGB4	GL_RGB	4	4	4		
GL_RGB5	GL_RGB	5	5	5		
GL_RGB8	GL_RGB	8	8	8		
GL_RGB8_SNORM	GL_RGB	s8	s8	s8		
GL_RGB10	GL_RGB	10	10	10		
GL_RGB12	GL_RGB	12	12	12		
GL_RGB16_SNORM	GL_RGB	16	16	16		

Sized Internal Format	Base Internal Format	Red Bits	Green Bits	Blue Bits	Alpha Bits	Shared Bits
GL_RGBA2	GL_RGB	2	2	2	2	
GL_RGBA4	GL_RGB	4	4	4	4	
GL_RGB5_A1	GL_RGBA	5	5	5	1	
GL_RGBA8	GL_RGBA	8	8	8	8	
GL_RGBA8_SNORM	GL_RGBA	s8	s8	s8	s8	
GL_RGB10_A2	GL_RGBA	10	10	10	2	
GL_RGB10_A2UI	GL_RGBA	ui10	ui10	ui10	ui2	
GL_RGBA12	GL_RGBA	12	12	12	12	
GL_RGBA16	GL_RGBA	16	16	16	16	
GL_SRGB8	GL_RGB	8	8	8		
GL_SRGB8_ALPHA8	GL_RGBA	8	8	8	8	
GL_R16F	GL_RED	f16				
GL_RG16F	GL_RG	f16	f16			
GL_RGB16F	GL_RGB	f16	f16	f16		
GL_RGBA16F	GL_RGBA	f16	f16	f16	f16	
GL_R32F	GL_RED	f32				
GL_RG32F	GL_RG	f32	f32			
GL_RGB32F	GL_RGB	f32	f32	f32		
GL_RGBA32F	GL_RGBA	f32	f32	f32	f32	
GL_R11F_G11F_B10F	GL_RGB	f11	f11	f10		
GL_RGB9_E5	GL_RGB	9	9	9		5
GL_R8I	GL_RED	i8				
GL_R8UI	GL_RED	ui8				
GL_R16I	GL_RED	i16				
GL_R16UI	GL_RED	ui16				
GL_R32I	GL_RED	i32				
GL_R32UI	GL_RED	ui32				
GL_RG8I	GL_RG	i8	i8			
GL_RG8UI	GL_RG	ui8	ui8			
GL_RG16I	GL_RG	i16	i16			

Sized Internal Format	Base Internal Format	Red Bits	Green Bits	Blue Bits	Alpha Bits	Shared Bits
GL_RG16UI	GL_RG	ui16	ui16			
GL_RG32I	GL_RG	i32	i32			
GL_RG32UI	GL_RG	ui32	ui32			
GL_RGB8I	GL_RGB	i8	i8	i8		
GL_RGB8UI	GL_RGB	ui8	ui8	ui8		
GL_RGB16I	GL_RGB	i16	i16	i16		
GL_RGB16UI	GL_RGB	ui16	ui16	ui16		
GL_RGB32I	GL_RGB	i32	i32	i32		
GL_RGB32UI	GL_RGB	ui32	ui32	ui32		
GL_RGBA8I	GL_RGBA	i8	i8	i8	i8	
GL_RGBA8UI	GL_RGBA	ui8	ui8	ui8	ui8	
GL_RGBA16I	GL_RGBA	i16	i16	i16	i16	
GL_RGBA16UI	GL_RGBA	ui16	ui16	ui16	ui16	
GL_RGBA32I	GL_RGBA	i32	i32	i32	i32	
GL_RGBA32UI	GL_RGBA	ui32	ui32	ui32	ui32	

Finally, *internalformat* may also be one of the generic or compressed texture formats shown in Table 3 below

Table 3. Compressed Internal Formats

Compressed Internal Format	Base Internal Format	Type
GL_COMPRESSED_RED	GL_RED	Generic
GL_COMPRESSED_RG	GL_RG	Generic
GL_COMPRESSED_RGB	GL_RGB	Generic
GL_COMPRESSED_RGBA	GL_RGBA	Generic
GL_COMPRESSED_SRGB	GL_RGB	Generic
GL_COMPRESSED_SRGB_ALPHA	GL_RGBA	Generic
GL_COMPRESSED_RED_RGTC1	GL_RED	Specific
GL_COMPRESSED_SIGNED_RED_RGTC1	GL_RED	Specific
GL_COMPRESSED_RG_RGTC2	GL_RG	Specific
GL_COMPRESSED_SIGNED_RG_RGTC2	GL_RG	Specific

Compressed Internal Format	Base Internal Format	Type
GL_COMPRESSED_RGBA_BPTC_UNORM	GL_RGBA	Specific
GL_COMPRESSED_SRGB_ALPHA_BPTC_UNORM	GL_RGBA	Specific
GL_COMPRESSED_RGB_BPTC_SIGNED_FLOAT	GL_RGB	Specific
GL_COMPRESSED_RGB_BPTC_UNSIGNED_FLOAT	GL_RGB	Specific

If the *internalformat* parameter is one of the generic compressed formats, GL_COMPRESSED_RED, GL_COMPRESSED_RG, GL_COMPRESSED_RGB, or GL_COMPRESSED_RGBA, the GL will replace the internal format with the symbolic constant for a specific internal format and compress the texture before storage. If no corresponding internal format is available, or the GL can not compress that image for any reason, the internal format is instead replaced with a corresponding base internal format.

If the *internalformat* parameter is GL_SRGB, GL_SRGB8, GL_SRGB_ALPHA, or GL_SRGB8_ALPHA8, the texture is treated as if the red, green, or blue components are encoded in the sRGB color space. Any alpha component is left unchanged. The conversion from the sRGB encoded component c_s to a linear component c_l is:

$$c_l = \begin{cases} \frac{c_s}{12.92} & \text{if } \ c_s \leq 0.04045 \\ \left(\frac{c_s + 0.055}{1.055} \right)^{2.4} & \text{if } \ c_s > 0.04045 \end{cases}$$

Assume c_s is the sRGB component in the range [0,1].

Use the GL_PROXY_TEXTURE_2D, GL_PROXY_TEXTURE_1D_ARRAY, GL_PROXY_TEXTURE_RECTANGLE, or GL_PROXY_TEXTURE_CUBE_MAP target to try out a resolution and format. The implementation will update and recompute its best match for the requested storage resolution and format. To then query this state, call glGetTexLevelParameter. If the texture cannot be accommodated, texture state is set to 0.

A one-component texture image uses only the red component of the RGBA color extracted from *data*. A two-component image uses the R and G values. A three-component image uses the R, G, and B values. A four-component image uses all of the RGBA components.

Image-based shadowing can be enabled by comparing texture r coordinates to depth texture values to generate a boolean result. See glTexParameter for details on texture comparison.

Notes

The glPixelStore mode affects texture images.

data may be a null pointer. In this case, texture memory is allocated to accommodate a texture of width *width* and height *height*. You can then download subtextures to initialize this texture memory. The image is undefined if the user tries to apply an uninitialized portion of the texture image to a primitive.

glTexImage2D specifies the two-dimensional texture for the current texture unit, specified with glActiveTexture.

GL_STENCIL_INDEX may be used for *format* only if the GL version is 4.4 or higher.

Errors

GL_INVALID_ENUM is generated if *target* is not GL_TEXTURE_2D, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_RECTANGLE, GL_PROXY_TEXTURE_2D, GL_PROXY_TEXTURE_1D_ARRAY, GL_PROXY_TEXTURE_RECTANGLE, GL_PROXY_TEXTURE_CUBE_MAP, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, or GL_TEXTURE_CUBE_MAP_NEGATIVE_Z.

GL_INVALID_ENUM is generated if *target* is one of the six cube map 2D image targets and the width and height parameters are not equal.

GL_INVALID_ENUM is generated if *type* is not a type constant.

GL_INVALID_VALUE is generated if *width* is less than 0 or greater than GL_MAX_TEXTURE_SIZE.

GL_INVALID_VALUE is generated if *target* is not GL_TEXTURE_1D_ARRAY or GL_PROXY_TEXTURE_1D_ARRAY and *height* is less than 0 or greater than GL_MAX_TEXTURE_SIZE.

GL_INVALID_VALUE is generated if *target* is GL_TEXTURE_1D_ARRAY or GL_PROXY_TEXTURE_1D_ARRAY and *height* is less than 0 or greater than GL_MAX_ARRAY_TEXTURE_LAYERS.

GL_INVALID_VALUE is generated if *level* is less than 0.

GL_INVALID_VALUE may be generated if *level* is greater than $log_2\left(max\right)$, where *max* is the returned value of GL_MAX_TEXTURE_SIZE.

GL_INVALID_VALUE is generated if *internalformat* is not one of the accepted resolution and format symbolic constants.

GL_INVALID_VALUE is generated if *width* or *height* is less than 0 or greater than GL_MAX_TEXTURE_SIZE.

GL_INVALID_VALUE is generated if *border* is not 0.

GL_INVALID_OPERATION is generated if *type* is one of GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, or GL_UNSIGNED_INT_10F_11F_11F_REV, and *format* is not GL_RGB.

GL_INVALID_OPERATION is generated if *type* is one of GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_8_REV, GL_UNSIGNED_INT_10_10_10_2, GL_UNSIGNED_INT_2_10_10_10_REV, or GL_UNSIGNED_INT_5_9_9_9_REV, and *format* is neither GL_RGBA nor GL_BGRA.

GL_INVALID_OPERATION is generated if *target* is not GL_TEXTURE_2D,

GL_PROXY_TEXTURE_2D, GL_TEXTURE_RECTANGLE, or GL_PROXY_TEXTURE_RECTANGLE, and *internalformat* is GL_DEPTH_COMPONENT, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24, or GL_DEPTH_COMPONENT32F.

GL_INVALID_OPERATION is generated if *format* is GL_DEPTH_COMPONENT and *internalformat* is not GL_DEPTH_COMPONENT, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24, or GL_DEPTH_COMPONENT32F.

GL_INVALID_OPERATION is generated if *internalformat* is GL_DEPTH_COMPONENT, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24, or GL_DEPTH_COMPONENT32F, and *format* is not GL_DEPTH_COMPONENT.

GL_INVALID_OPERATION is generated if a non-zero buffer object name is bound to the GL_PIXEL_UNPACK_BUFFER target and the buffer object's data store is currently mapped.

GL_INVALID_OPERATION is generated if a non-zero buffer object name is bound to the GL_PIXEL_UNPACK_BUFFER target and the data would be unpacked from the buffer object such that the memory reads required would exceed the data store size.

GL_INVALID_OPERATION is generated if a non-zero buffer object name is bound to the GL_PIXEL_UNPACK_BUFFER target and *data* is not evenly divisible into the number of bytes needed to store in memory a datum indicated by *type*.

GL_INVALID_VALUE is generated if *target* is GL_TEXTURE_RECTANGLE or GL_PROXY_TEXTURE_RECTANGLE and *level* is not 0.

Associated Gets

glGetTexImage

glGet with argument GL_PIXEL_UNPACK_BUFFER_BINDING

Version Support

	OpenGL Version											
Function / Feature Name	**2.0**	**2.1**	**3.0**	**3.1**	**3.2**	**3.3**	**4.0**	**4.1**	**4.2**	**4.3**	**4.4**	**4.5**
glTexImage2D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GL_HALF_FLOAT	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

See Also

glActiveTexture, glCopyTexImage1D, glCopyTexImage2D, glCopyTexSubImage1D, glCopyTexSubImage2D, glCopyTexSubImage3D, glPixelStore, glTexImage1D, glTexImage3D, glTexSubImage1D, glTexSubImage2D, glTexSubImage3D, glTexParameter

Copyright