NAME

glReadPixels – read a block of pixels from the frame buffer

C SPECIFICATION

void **glReadPixels**(GLint *x*,
GLint *y*,
GLsizei *width*,
GLsizei *height*,
GLenum *format*,
GLenum *type*,
GLvoid **pixels*)

delim $$

PARAMETERS

x, *y*
Specify the window coordinates of the first pixel that is read from the frame buffer. This location is the lower left corner of a rectangular block of pixels.

width, *height*
Specify the dimensions of the pixel rectangle. *width* and *height* of one correspond to a single pixel.

format
Specifies the format of the pixel data. The following symbolic values are accepted: **GL_COLOR_INDEX**, **GL_STENCIL_INDEX**, **GL_DEPTH_COMPONENT**, **GL_RED**, **GL_GREEN**, **GL_BLUE**, **GL_ALPHA**, **GL_RGB**, **GL_RGBA**, **GL_LUMINANCE**, and **GL_LUMINANCE_ALPHA**.

type
Specifies the data type of the pixel data. Must be one of **GL_UNSIGNED_BYTE**, **GL_BYTE**, **GL_BITMAP**, **GL_UNSIGNED_SHORT**, **GL_SHORT**, **GL_UNSIGNED_INT**, **GL_INT**, or **GL_FLOAT**.

pixels
Returns the pixel data.

DESCRIPTION

glReadPixels returns pixel data from the frame buffer, starting with the pixel whose lower left corner is at location (*x*, *y*), into client memory starting at location *pixels*. Several parameters control the processing of the pixel data before it is placed into client memory. These parameters are set with three commands: **glPixelStore**, **glPixelTransfer**, and **glPixelMap**. This reference page describes the effects on **glReadPixels** of most, but not all of the parameters specified by these three commands.

glReadPixels returns values from each pixel with lower left corner at (*x* + i, *y* + j) for 0≤i<*width* and 0≤j<*height*. This pixel is said to be the ith pixel in the jth row. Pixels are returned in row order from the lowest to the highest row, left to right in each row.

format specifies the format for the returned pixel values; accepted values are:

GL_COLOR_INDEX
Color indices are read from the color buffer selected by **glReadBuffer**. Each index is converted to fixed point, shifted left or right depending on the value and sign of **GL_INDEX_SHIFT**, and added to **GL_INDEX_OFFSET**. If **GL_MAP_COLOR** is **GL_TRUE**, indices are replaced by their mappings in the table **GL_PIXEL_MAP_I_TO_I**.

GL_STENCIL_INDEX
Stencil values are read from the stencil buffer. Each index is converted to fixed point, shifted left or right depending on the value and sign of **GL_INDEX_SHIFT**, and added to

GL_INDEX_OFFSET. If **GL_MAP_STENCIL** is **GL_TRUE**, indices are replaced by their mappings in the table **GL_PIXEL_MAP_S_TO_S**.

GL_DEPTH_COMPONENT

Depth values are read from the depth buffer. Each component is converted to floating point such that the minimum depth value maps to 0 and the maximum value maps to 1. Each component is then multiplied by **GL_DEPTH_SCALE**, added to **GL_DEPTH_BIAS**, and finally clamped to the range [0,1].

GL_RED

GL_GREEN

GL_BLUE

GL_ALPHA

GL_RGB

GL_RGBA

GL_LUMINANCE

GL_LUMINANCE_ALPHA

Processing differs depending on whether color buffers store color indices or RGBA color components. If color indices are stored, they are read from the color buffer selected by **glReadBuffer**. Each index is converted to fixed point, shifted left or right depending on the value and sign of **GL_INDEX_SHIFT**, and added to **GL_INDEX_OFFSET**. Indices are then replaced by the red, green, blue, and alpha values obtained by indexing the tables **GL_PIXEL_MAP_I_TO_R**, **GL_PIXEL_MAP_I_TO_G**, **GL_PIXEL_MAP_I_TO_B**, and **GL_PIXEL_MAP_I_TO_A**. Each table must be of size 2^n, but n may be different for different tables. Before an index is used to look up a value in a table of size 2^n, it must be masked against 2^n-1.

If RGBA color components are stored in the color buffers, they are read from the color buffer selected by **glReadBuffer**. Each color component is converted to floating point such that zero intensity maps to 0.0 and full intensity maps to 1.0. Each component is then multiplied by **GL_c_SCALE** and added to **GL_c_BIAS**, where *c* is RED, GREEN, BLUE, or ALPHA. Finally, if **GL_MAP_COLOR** is **GL_TRUE**, each component is clamped to the range [0,1], scaled to the size of its corresponding table, and is then replaced by its mapping in the table **GL_PIXEL_MAP_c_TO_c**, where *c* is R, G, B, or A.

Unneeded data is then discarded. For example, **GL_RED** discards the green, blue, and alpha components, while **GL_RGB** discards only the alpha component. **GL_LUMINANCE** computes a single-component value as the sum of the red, green, and blue components, and **GL_LUMINANCE_ALPHA** does the same, while keeping alpha as a second value. The final values are clamped to the range [0,1].

The shift, scale, bias, and lookup factors just described are all specified by **glPixelTransfer**. The lookup table contents themselves are specified by **glPixelMap**.

Finally, the indices or components are converted to the proper format, as specified by *type*. If *format* is **GL_COLOR_INDEX** or **GL_STENCIL_INDEX** and *type* is not **GL_FLOAT**, each index is masked with the mask value given in the following table. If *type* is **GL_FLOAT**, then each integer index is converted to single-precision floating-point format.

If *format* is **GL_RED**, **GL_GREEN**, **GL_BLUE**, **GL_ALPHA**, **GL_RGB**, **GL_RGBA**, **GL_LUMINANCE**, or **GL_LUMINANCE_ALPHA** and *type* is not **GL_FLOAT**, each component is multiplied by the multiplier shown in the following table. If type is **GL_FLOAT**, then each component is passed as is (or converted to the client's single-precision floating-point format if it is different from the one used by the GL).

type	*index mask*	*component conversion*
GL_UNSIGNED_BYTE	$2"^"8 - 1$	$(2"^"8 - 1) c$
GL_BYTE	$2"^"7 - 1$	$[(2"^"8 - 1) c - 1] / 2$
GL_BITMAP	1	1
GL_UNSIGNED_SHORT	$2"^"16 - 1$	$(2"^"16 - 1) c$
GL_SHORT	$2"^"15 - 1$	$[(2"^"16 - 1) c - 1] / 2$
GL_UNSIGNED_INT	$2"^"32 - 1$	$(2"^"32 - 1) c$
GL_INT	$2"^"31 - 1$	$[(2"^"32 - 1) c - 1] / 2$
GL_FLOAT	none	c

Return values are placed in memory as follows. If *format* is **GL_COLOR_INDEX**, **GL_STENCIL_INDEX**, **GL_DEPTH_COMPONENT**, **GL_RED**, **GL_GREEN**, **GL_BLUE**, **GL_ALPHA**, or **GL_LUMINANCE**, a single value is returned and the data for the ith pixel in the jth row is placed in location $(j)~"width"~+~i$. **GL_RGB** returns three values, **GL_RGBA** returns four values, and **GL_LUMINANCE_ALPHA** returns two values for each pixel, with all values corresponding to a single pixel occupying contiguous space in *pixels*. Storage parameters set by **glPixelStore**, such as **GL_PACK_LSB_FIRST** and **GL_PACK_SWAP_BYTES**, affect the way that data is written into memory. See **glPixelStore** for a description.

NOTES

Values for pixels that lie outside the window connected to the current GL context are undefined.

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

ERRORS

GL_INVALID_ENUM is generated if *format* or *type* is not an accepted value.

GL_INVALID_ENUM is generated if *type* is **GL_BITMAP** and *format* is not **GL_COLOR_INDEX** or **GL_STENCIL_INDEX**.

GL_INVALID_VALUE is generated if either *width* or *height* is negative.

GL_INVALID_OPERATION is generated if *format* is **GL_COLOR_INDEX** and the color buffers store RGBA color components.

GL_INVALID_OPERATION is generated if *format* is **GL_STENCIL_INDEX** and there is no stencil buffer.

GL_INVALID_OPERATION is generated if *format* is **GL_DEPTH_COMPONENT** and there is no depth buffer.

GL_INVALID_OPERATION is generated if **glReadPixels** is executed between the execution of **glBegin** and the corresponding execution of **glEnd**.

ASSOCIATED GETS

glGet with argument **GL_INDEX_MODE**

SEE ALSO

glCopyPixels, **glDrawPixels**, **glPixelMap**, **glPixelStore**, **glPixelTransfer**, **glReadBuffer**