NAME

MiniLibX - Manipulating images

SYNOPSYS

*void **
mlx_new_image (*void *mlx_ptr, int width, int height*);

*char **
mlx_get_data_addr (*void *img_ptr, int *bits_per_pixel, int *size_line, int *endian*);

int
mlx_put_image_to_window (*void *mlx_ptr, void *win_ptr, void *img_ptr, int x, int y*);

unsigned int
mlx_get_color_value (*void *mlx_ptr, int color*);

*void **
mlx_xpm_to_image (*void *mlx_ptr, char **xpm_data, int *width, int *height*);

*void **
mlx_xpm_file_to_image (*void *mlx_ptr, char *filename, int *width, int *height*);

int
mlx_destroy_image (*void *mlx_ptr, void *img_ptr*);

DESCRIPTION

mlx_new_image () creates a new image in memory. It returns a *void ** identifier needed to manipulate this image later. It only needs the size of the image to be created, using the *width* and *height* parameters, and the *mlx_ptr* connection identifier (see the **mlx** manual).

The user can draw inside the image (see below), and can dump the image inside a specified window at any time to display it on the screen. This is done using **mlx_put_image_to_window** (). Three identifiers are needed here, for the connection to the display, the window to use, and the image (respectively *mlx_ptr* , *win_ptr* and *img_ptr*). The (*x* , *y*) coordinates define where the image should be placed in the window.

mlx_get_data_addr () returns information about the created image, allowing a user to modify it later. The *img_ptr* parameter specifies the image to use. The three next parameters should be the addresses of three different valid integers. *bits_per_pixel* will be filled with the number of bits needed to represent a pixel color (also called the depth of the image). *size_line* is the number of bytes used to store one line of the image in memory. This information is needed to move from one line to another in the image. *endian* tells you wether the pixel color in the image needs to be stored in little endian (*endian* == 0), or big endian (*endian* == 1).

mlx_get_data_addr returns a *char ** address that represents the begining of the memory area where the image is stored. From this adress, the first *bits_per_pixel* bits represent the color of the first pixel in the first line of the image. The second group of *bits_per_pixel* bits represent the second pixel of the first line, and so on. Add *size_line* to the adress to get the begining of the second line. You can reach any pixels of the image that way.

mlx_destroy_image destroys the given image (*img_ptr*).

STORING COLOR INSIDE IMAGES

Depending on the display, the number of bits used to store a pixel color can change. The user usually represents a color in RGB mode, using one byte for each component (see **mlx_pixel_put** manual). This must be translated to fit the *bits_per_pixel* requirement of the image, and make the color understandable to the X-Server. That is the purpose of the **mlx_get_color_value** () function. It takes a standard RGB *color* parameter, and returns an *unsigned int* value. The *bits_per_pixel* least significant bits of this value can be stored in the image.

Keep in mind that the least significant bits position depends on the local computer's endian. If the endian of the image (in fact the endian of the X-Server's computer) differs from the local endian, then the value should be transformed before being used.

XPM IMAGES

The **mlx_xpm_to_image** () and **mlx_xpm_file_to_image** () functions will create a new image the same way. They will fill it using the specified *xpm_data* or *filename* , depending on which function is used. Note that MiniLibX does not use the standard Xpm library to deal with xpm images. You may not be able to read all types of xpm images. It however handles transparency.

RETURN VALUES

The three functions that create images, **mlx_new_image()** , **mlx_xpm_to_image()** and **mlx_xpm_file_to_image()** , will return NULL if an error occurs. Otherwise they return a non-null pointer as an image identifier.

SEE ALSO

mlx(3), mlx_new_window(3), mlx_pixel_put(3), mlx_loop(3)

AUTHOR