MAN MLX_WINDOW

SYNOPSYS

```
void *
mlx_new_window ( void *mlx_ptr, int size_x, int size_y, char *title );

int
mlx_clear_window ( void *mlx_ptr, void *win_ptr );

int
mlx_destroy_window ( void *mlx_ptr, void *win_ptr );
```

DESCRIPTION

The mlx_new_window () function creates a new window on the screen, using the size_x and size_y parameters to determine its size, and title as the text that should be displayed in the window's title bar. The mlx_ptr parameter is the connection identifier returned by mlx_init () (see the mlx man page). mlx_new_window () returns a void * window identifier that can be used by other MiniLibX calls. Note that the MiniLibX can handle an arbitrary number of separate windows.

mlx_clear_window () and mlx_destroy_window () respectively clear (in black) and destroy the given window. They both have the same parameters: mlx_ptr is the screen connection identifier, and win_ptr is a window identifier.

RETURN VALUES

If mlx_new_window() fails to create a new window (for wathever reason), it will return NULL, otherwise a non-null pointer is returned as a window identifier. mlx_clear_window and mlx_destroy_window right now return nothing.

MAN MLX_PIXEL

SYNOPSYS

```
int
```

mlx_pixel_put (void *mlx_ptr, void *win_ptr, int x, int y, int color);

int
mlx_string_put (void *mlx_ptr, void *win_ptr, int x, int y, int color, char *string);

DESCRIPTION

The mlx_pixel_put () function draws a defined pixel in the window win_ptr using the (x , y) coordinates, and the specified color . The origin (0,0) is the upper left corner of the window,
the x and y axis respectively pointing right and down. The connection identifier, mlx_ptr , is needed (see the mlx man page).

Parameters for mlx_string_put () have the same meaning. Instead of a simple pixel, the specified string will be displayed at (x , y).

In both functions, it is impossible to display anything outside the specified window, nor display in another window in front of the selected one.

COLOR MANAGEMENT

The color parameter has an integer type. The displayed color needs to be encoded in this integer, following a defined scheme. All displayable colors can be split in 3 basic colors: red, green
and blue. Three associated values, in the 0-255 range, represent how much of each color is mixed up to create the original color. Theses three values must be set inside the integer to display
the right color. The three least significant bytes of this integer are filled as shown in the picture below:

```
| 0 | R | G | B |   color integer
+---+---+---+---+
```

While filling the integer, make sure you avoid endian problems. Remember that the "blue" byte should always be the least significant one.

MLX_NEW_IMAGE

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 man-
ual). This must be translated to fit the bits_per_pixel requirement of the image, and make the color understandable to the graphical system. 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 for remote X11 display)
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.

MLX_LOOP

SYNOPSYS

```
int
mlx_loop ( void *mlx_ptr );

int
mlx_key_hook ( void *win_ptr, int (*funct_ptr)(), void *param );

int
mlx_mouse_hook ( void *win_ptr, int (*funct_ptr)(), void *param );

int
mlx_expose_hook ( void *win_ptr, int (*funct_ptr)(), void *param );

int
mlx_loop_hook ( void *mlx_ptr, int (*funct_ptr)(), void *param );
```

EVENTS

Both X-Window and MacOSX graphical systems are bi-directionnal. On one hand, the program sends orders to the screen to display pixels, images, and so on. On the other hand, it can get infor-
mation from the keyboard and mouse associated to the screen. To do so, the program receives "events" from the keyboard or the mouse.

DESCRIPTION

To receive events, you must use mlx_loop (). This function never returns. It is an infinite loop that waits for an event, and then calls a user-defined function associated with this event. A
single parameter is needed, the connection identifier mlx_ptr (see the mlx manual).

You can assign different functions to the three following events:

- A key is pressed
- The mouse button is pressed
- A part of the window should be re-drawn (this is called an "expose" event, and it is your program's job to handle it).

Each window can define a different function for the same event.

The three functions mlx_key_hook (), mlx_mouse_hook () and mlx_expose_hook () work exactly the same way. funct_ptr is a pointer to the function you want to be called when an event occurs.
This assignment is specific to the window defined by the win_ptr identifier. The param adress will be passed to the function everytime it is called, and should be used to store the parameters
it might need.

The syntax for the mlx_loop_hook () function is identical to the previous ones, but the given function will be called when no event occurs.

When it catches an event, the MiniLibX calls the corresponding function with fixed parameters:

```
expose_hook(void *param);
key_hook(int keycode,void *param);
mouse_hook(int button,int x,int y,void *param);
loop_hook(void *param);
```

These function names are arbitrary. They here are used to distinguish parameters according to the event. These functions are NOT part of the MiniLibX.

param is the address specified in the mlx_*_hook calls. This address is never used nor modified by the MiniLibX. On key and mouse events, additional information is passed: keycode tells you
which key is pressed (X11 : look for the include file "keysymdef.h", MacOS : create a small software and find out by yourself), (x , y) are the coordinates of the mouse click in the window,
and button tells you which mouse button was pressed.

GOING FURTHER WITH EVENTS

The MiniLibX provides a much generic access to all type of events. The mlx.h include define mlx_hook() in the same manner mlx_*_hook functions work. The event and mask values will be taken
from the X11 include file "X.h" (even for MacOSX, for compatibility purposes)

See source code of mlx_int_param_event.c to find out how the MiniLibX will call your own function for a specific event.