42 Docs

Prototypes

TABLE OF CONTENTS

Introduction

MiniLibX is a tiny graphics library which allows you to do the most basic things for rendering something. This can vary from making a copy of Wolfenstein, to presenting complicated data in a simple form.

It is truly recommended to catch up on bitwise operands if you have no clue what they are.

Initialization functions

These are the standard functions that are almost always required to even start using MiniLibX.

mlx_init

Initializes the MLX library. Must be called before ANY other function. Will return `NULL` if initialization failed.

```
/*
** Initialize mlx.
**
** @return void*      the mlx instance
*/
void *mlx_init();
```

mlx_new_window

Creates a new window instance. It will return a window instance pointer. This should be saved for future reference.

```
/*
** Create a new window.
**
** @param  void *mlx_ptr   the mlx instance pointer;
** @param  int  size_x          the width of the window;
** @param  int  size_y          the height of the window;
** @param  char *title          the title of the window;
** @return void*                the window instance pointer.
*/
void *mlx_new_window(void *mlx_ptr, int size_x, int size_y, char *title);
```

mlx_clear_window

Clears the current window. This is not a recommended function to use. Instead it is recommended to use the function `mlx_put_image_to_window` with a recycled image that you have cleared.

```
/*
** Clear the provided window.
**
** @param  void *mlx_ptr   the mlx instance pointer;
** @param  void *win_ptr   the window instance pointer;
** @return int                  has no return value (bc).
*/
int        mlx_clear_window(void *mlx_ptr, void *win_ptr);
```

mlx_destroy_window

Destroys a window instance accordingly.

```
/*
** Destroy a window instance.
**
** @param  void *mlx_ptr   the mlx instance;
```

```
** @param  void *win_ptr   the window instance;
** @return int                 has no return value (bc).
*/
int        mlx_destroy_window(void *mlx_ptr, void *win_ptr);
```

Util functions

These are functions that can help you with conversions and pixel writing.

mlx_get_color_value

Get the color value accordingly from a int. This is useful for converting a self-declared int before writing it to certain bits.

```
/*
** Get the color value.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  int  color      the int color (0xTTRRGGBB);
** @return uint            the converted color.
*/
uint  mlx_get_color_value(void *mlx_ptr, int color);
```

mlx_pixel_put

Puts a pixel on the screen. This function is NOT recommended for use. It will lock the window output, force a refresh and a recalculation. It is therefore suggested to render a image and push that using the `mlx_put_image_to_window` function. You can find more about that in the Getting Started chapter.

```
/*
** Put a pixel on the screen.
**
** @param  void *mlx_ptr   the mlx instance pointer;
** @param  void *win_ptr   the window instance pointer;
** @param  int  x              the x coordinate of the pixel to draw;
** @param  int  y              the y coordinate of the pixel to draw;
** @param  int  color      the color of the pixel to draw (0xTTRRGGBB);
```

```
** @return int                  has no return value (bc).
*/
int        mlx_pixel_put(void *mlx_ptr, void *win_ptr, int x, int y, int color);
```

mlx_string_put

Puts a string on the location (x,y) in the given window.

```
/*
** Put a string in the window.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  int  x               the x location;
** @param  int  y               the y location;
** @param  int  color      the font color;
** @param  char *string    the text to write;
** @return int                  has no return value (bc).
*/
int        mlx_string_put(void *mlx_ptr, void *win_ptr, int x, int y, int color, char *string)
```

Image functions

Image object related functions. These will provide effective methods to mutate frames one-by-one.

mlx_new_image

Creates a new MLX compatible image. This is the recommended way to buffer the image you are rendering. It will accept a pointer to your MLX instance and requires a width and height. Will return a reference pointer to the image.

```
/*
** Create a new MLX compatible image.
**
** @param  void *mlx_ptr   the mlx instance pointer;
** @param  int  width      the width of the image to be created;
** @param  int  height          the height of the image to be created;
** @return void*                the image instance reference.
```

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

mlx_get_data_addr

Gets the memory address of the given image. Memory of images is weird. It will set the line size in your given pointer. To get or set the value of the pixel (5, 100) in an image size of (500, 500), we would need to locate the position as follows:

```
int pos = (y * size_line + x * (bits_per_pixel / 8));
```

Here we multiply size_line by `y` as we need to skip `y` lines (and yes, line size is not equal to the amount of pixels in a line). We then add the remaining `x` units multiplied by `bits_per_pixel / 8` to align with the final location.

To modify each pixel with the correct color, we need to do some more fancy stuff. As we need to align the bits before writing, we need to do the following for the best result:

```
char *mlx_data_addr = mlx_get_data_addr();
*(unsigned int *)mlx_data_addr = color;
```

The function prototype is as follows:

```
/*
** Gets the data address of the current image.
**
** @param  void *img_ptr             the image instance;
** @param  int  *bits_per_pixel a pointer to where the bpp is written;
** @param  int  *size_line      a pointer to where the line is written;
** @param  int  *endian         a pointer to where the endian is written;
** @return char*                     the memory address of the image.
*/
char *mlx_get_data_addr(void *img_ptr, int *bits_per_pixel, int *size_line, int *endian);
```

mlx_put_image_to_window

Puts an image to the given window instance at location (x,y). This is the recommended way to write large amounts of graphical data in one go. Do mind that when changing the memory of

the locations, it will be displayed directly on the window.

```
/*
** Put an image to the given window.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  void *win_ptr   the window instance;
** @param  int  x               the x location of where the image ought to be placed;
** @param  int  y               the y location of where the image ought to be placed;
** @return int                  has no return value (bc).
*/
int        mlx_put_image_to_window(void *mlx_ptr, void *win_ptr, void *img_ptr, int x, int y);
```

mlx_destroy_image

Destroys an image instance accordingly.

```
/*
** Destroy an image instance.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  void *img_ptr   the image instance;
** @return int                  has no return value (bc).
*/
int        mlx_destroy_image(void *mlx_ptr, void *img_ptr);
```

Hooks

These functions will allow you to hook into the MiniLibX functions. This is core functionality and is required to use the library effectively. Please look at the Hooks chapter if you have no clue what this means.

mlx_mouse_hook

Hook into mouse events. This will trigger every time you click somewhere in the given screen. Do mind that currently these mouse events barely work, it is therefore suggested to not use them.

```
/*
** Hook into mouse events.
**
** @param  void *win_ptr   the window instance;
** @param  int  (*f)()          the handler function, will be called as follows:
**                                   (*f)(int button, int x, int y, void *param);
** @param  void *param          the parameter to give on each event;
** @return int                  has no return value (bc).
*/
int        mlx_mouse_hook(void *win_ptr, int (*f)(), void *param);
```

mlx_key_hook

Hook into key events. This will trigger every time a key is pressed in a focused window. Unfocused windows will not register any key events.

```
/*
** Hook into key events.
**
** @param  void *win_ptr   the window instance;
** @param  int  (*f)()          the handler function, will be called as follows:
**                                   (*f)(int key_code, void *param);
** @param  void *param          the parameter to give on each event;
** @return int                  has no return value (bc).
*/
int        mlx_key_hook(void *win_ptr, int (*f)(), void *param);
```

mlx_expose_hook

Has no defined behaviour.

mlx_loop_hook

Hook into the loop.

```
/*
** Hook into each loop.
**
```

```
** @param  void *mlx_ptr   the mlx instance;
** @param  int  (*f)()          the handler function, will be called as follows:
**                                   (*f)(void *param);
** @param  void *param          the parameter to give on each event;
** @return int                  has no return value (bc).
*/
int        mlx_loop_hook(void *mlx_ptr, int (*f)(), void *param);
```

mlx_loop

Loop over the given MLX pointer. Each hook that was registered prior to this will be called accordingly by order of registration.

```
/*
** Loop over the given mlx pointer.
**
** @param  void *mlx_ptr   the mlx instance;
** @return int                  has no return value (bc).
*/
int        mlx_loop(void *mlx_ptr);
```

Image conversions

These are functions that are useful for loading sprites or even saving images.

mlx_xpm_to_image

Converts xpm data to a new image instance.

```
/*
** Converts xpm data to a new image instance.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  char **xpm_data the xpm data in a 2 dimensional char array;
** @param  int  *width          a pointer to where the width ought to be written;
** @param  int  *height    a pointer to where the height ought to be written;
** @return void*                the image instance, and NULL in case of error.
*/
```

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

mlx_xpm_file_to_image

Converts an xpm file to a new image instance.

```
/*
** Convert an xpm file to a new image instance.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  char *filename  the file to load;
** @param  int  *width          a pointer to where the width ought to be written;
** @param  int  *height    a pointer to where the height ought to be written;
** @return void*                the image instance, and NULL in case of error.
*/
void *mlx_xpm_file_to_image(void *mlx_ptr, char *filename, int *width, int *height);
```

mlx_png_file_to_image

Converts a png file to a new image instance.

```
/*
** Convert a png file to a new image instance.
**
** @param  void *mlx_ptr   the mlx instance;
** @param  char *filename  the file to load;
** @param  int  *width          a pointer to where the width ought to be written;
** @param  int  *height    a pointer to where the height ought to be written;
** @warn   mem_leak        this function has a memory leak, try using xpm
**                                 instead;
** @return void*                the image instance.
*/
void *mlx_png_file_to_image(void *mlx_ptr, char *filename, int *width, int *height);
```

Mouse functions

These functions will allow you to hide, show, move or get the mouse position.

mlx_mouse_hide

Hides the mouse.

```
/*
** Hide the mouse.
**
** @return int       has no return value (bc).
*/
int        mlx_mouse_hide();
```

mlx_mouse_show

Shows the mouse.

```
/*
** Show the mouse.
**
** @return int       has no return value (bc).
*/
int        mlx_mouse_show();
```

mlx_mouse_move

Moves the cursor to the given location.

```
/*
** Move the cursor to the given location.
**
** @param  void *win_ptr   the window instance;
** @param  int  x               the x location to move to;
** @param  int  y               the y location to move to;
** @return int                  has no return value (bc).
*/
int        mlx_mouse_move(void *win_ptr, int x, int y);
```

mlx_mouse_get_pos

Gets the current mouse position on the window.

```
/*
** Get the current mouse position on the window.
**
** @param  void *win_ptr   the window instance;
** @param  int  *x              the pointer to write the x location to;
** @param  int  *y              the pointer to write the y location to;
** @return int                  has no return value (bc).
*/
int        mlx_mouse_get_pos(void *win_ptr, int *x, int *y);
```

Key auto repeat

These functions will allow you to either enable or disable key autorepeat.

mlx_do_key_autorepeatoff

Disable key auto repeat.

```
/*
** Disable key auto repeat.
**
** @param  void *mlx_ptr   the mlx instance;
** @return int                  has no return value (bc).
*/
int        mlx_do_key_autorepeatoff(void *mlx_ptr);
```

mlx_do_key_autorepeaton

Enable key auto repeat.

```
/*
** Enable key auto repeat.
**
** @param  void *mlx_ptr   the mlx instance;
** @return int                  has no return value (bc).
*/
int        mlx_do_key_autorepeaton(void *mlx_ptr);
```

Un-categorized

mlx_do_sync

```
/*
** Synchronize frames of all windows in MLX.
**
** @param  void *mlx_ptr   the mlx instance;
** @return int                  has no return value (bc).
*/
int        mlx_do_sync(void *mlx_ptr);
```

mlx_get_screen_size

```
/*
** Get the current screen size (because macOS is sheit)
**
** @param  void *mlx_ptr   the mlx instance;
** @param  int  *sizex          the screen width;
** @param  int  *sizey          the screen height
** @return int                  has no return value (bc).
*/
int        mlx_get_screen_size(void *mlx_ptr, int *sizex, int *sizey);
```