Spark Session: MLX42

updated: 18/05/2022

Session description:

> Learn the basics of working with MLX42

This tutorial was written by de la Hamette Leon Jean Laurenti, click here for the documentation regarding MLX42.

Hint: Most question you will have will most likely be answered by simply reading the documentation or the MLX42.h file!

> VERY IMPORTANT! PEOPLE REALLY DON'T LIKE READING!

Topics

1. Window Management
2. Images & Pixel Putting
3. More Pixels
4. Hooks
5. Bonus!

Window Management

Our first step will be to open up a window! (30 mins)

1. In the set-up instructions, Some code was given to you for your `main.c` that included a call to `mlx_init`.
   But what does it do and what is its prototype? What does it return? (5 mins)
   - What are the variables we are passing to `mlx_init` ?
   - Study and understand the return value type and its layout!
   - Change the parameters to create a window with a width of **800** and a height of **400**, a title of your choice and make it so that we can **not** resize our window.

   > mlx_init creates a connection to the graphics API as well as opening a window using GLFW.
   > The pointer returns basically the applications context such as width and height and title.

2. What happens if you compile and run the program at this point? Your window should have only popped up for a moment.
   To make it stay longer, we need to use `mlx_loop`. (15 mins)

- What does it do and what is its prototype?
- Once you understand that, add `mlx_loop` to your code.
- Do you now get a window that stays open?

> mlx_loop, as the name suggests, makes the program loop until the application receives a request to exit.
> To request an exit `mlx_close_window` can be called which causes the program loop to break.

3. How do we properly exit our application? When our application closes, we needs clean up its resources.
   To make this possible use `mlx_terminate` (2 mins)

   - Understand the purpose of having this function.
   - **Important**: `mlx_loop` should be called before this function in your code. Do you know why?
     > MLX42 works a bit different to minilibx in that it cleans up after its done with its job, it deletes all images and some internal stuff. Its vital to call this as this also terminates GLFW, the windowing library.
     > Not calling this function could cause memory leaks.

Break (5 mins)

Images & Pixel Putting

Time to put something on that empty window. (60 mins)

1. Let there be colourful pixels! As of now your window is pretty much void of anything. Just like a painter we need a canvas to draw on, its time to learn about images in MLX.

   - What exactly is an image in its most basic form?
   - Use `mlx_new_image`. What is the prototype and the return value for the function?
   - Study and understand the return value!
   - Once you understand that, go ahead and initialise an image with a width of **64** and a height of **64**.

   > In short: An image is a pixel buffer that stores its dimension and data which when displayed show some form of output.
   > In long: An image is like a canvas that one can draw on an display multiple copies of. Images work with instances, that is, exact copies that can be placed throughout the window. The image itself is just the 'painting' while all its instances are the 'replicas'.

2. Now, using the `mlx_put_pixel` function, put a **white** pixel in the **middle** of your image. (10 mins)

   - How do you define colors?
   - Understand how this function works, that is, how does it modify the image itself.

   > MLX42 uses RGBA, One byte per channel and all channels must be specified e.g: 0xFF0000FF will display non-transparent red. RGBA is a standart format to display colors.

> Its vital to undertsand the internals of `mlx_put_pixel`. It works by bit shifting out each byte of the integer to retrieve the individual color channels. Each channel is a `uint8`, therfor we shift by powers of 2.

3. Our image is all ready to be shown! To do so, use `mlx_image_to_window`. MLX42 works with instances of images, that is, an image is like a 'painting' while an instance is a 'physical copy' of that painting.
    - What are the prototypes and the return value for the function?
    - What can you do with the return value?
      > As the name suggest this puts a 'replica'/copy onto the window. MLX42 dynamically updates the images for you, you can simply modify the buffer at any given time and the changes are reflected instantly.
      > Since the instances array inside the image ptr gets reallocted holding a pointer to the instance is a bad idea. Instead putting an image to the window returns the index of where the instance is in the image's instances array.

Break (5 mins)

More Pixels

Let's get fancier. Now we're gonna try drawing *lines*. (25 mins)

1. Draw a single horizontal white line running across the middle of the image. You'll need to call `mlx_put_pixel` in a loop. (15 mins)
2. Now draw a single vertical white line down the middle of the image. You should end up with what looks like a crosshair in your image. (10 mins)

Hooks

You might have noticed that we can already very easily close our window, however, just because for this exercise, we want to actually intercept the execution when we press the 'X' close button. (35 mins)

1. Hooks, are vital to making your program interactive. They allow you to intercept keyboard or mouse events and respond to them. You can think of hooks as functions that get called when an event occurs. MLX provides 2 types of hooks, specialized and generic ones.
    - What exactly are generic and specialized hooks ?
    - Here's something that might help you understand: Hooks

    > Generic: Simple hook that gets executed every frame, can have X amount of them. And execute any type of logic.
    > Specialized: Specific hook for a specific task, there can only be one hooked function for each. Have to follow a specific prototype.

2. Write a function that: (10 mins)
    - Gets executed when we click onto the 'X' close button.

- Prints the window title with a nicely formatted message.

3. Add a call to `mlx_close_hook` in your main that calls this exiting function when the 'X' button is pressed. (5 mins)
    - Does your window close and print a message when you press the 'X' close button on your window?

Bonus!

Let's get some movement on the screen: Make a rectangle move in 4 directions!

1. Make a `ft_draw_rect` function that:
    - Takes the MLX pointer.
    - Takes a width and height value.
    - Takes a color value for the rectangle.
    - Creates a new image of that width and height.
    - Sets every pixel in the image to the specified color.
    - Returns the image pointer.
2. Get a **30 x 30** blue rectangle onto your window.

Now let's hook into keyboard events!

1. Add a call to `mlx_loop_hook` in your main that calls a function `ft_on_key` so we can handle key presses.
    - You can also use `mlx_key_hook`.
2. Write the `ft_on_key` function so that:
    - It closes the window/exits when the `ESC` key is pressed.
    - Moves the rectangle, up, down, left, and right when the corresponding key is pressed.
        - You can choose to use the arrow keys or `W` - `A` - `S` - `D` keys.
3. Regarding hooking onto the keyboard, MLX provides a header file that neatly displays all the different keycodes. Which are set out
in the **GLFW** library.
4. Add a call to `mlx_loop_hook` in your main that calls a function to move the new rectangle, of course don't forget to create it first!