LAB 02:

Mode 3 Drawing Functions & VSync

Provided Files

- `Makefile`
- `print.h`
- `print.c`
- `Example.gba`

Files to Edit/Add

- `.vscode`
    - `tasks.json`
- `gba.c`
- `gba.h`
- `main.c`

Instructions

In this lab, you will be implementing functions to draw different shapes in mode 3. The included `Example.gba` file is a reference that roughly demonstrates what the lab should look like once you have completed each TODO.

TODO 1 – Header Files and Debug Setup

In this TODO, you will be linking additional files to `main.c` so that the functions, variables, and other tools from those files can be used.

TODO 1.0

- At the top of `main.c`, include `gba.h` and `print.h` using the `#include` preprocessor command.

TODO 1.1

- In the `initialize` function in `main.c`, initialize debug logs with the `mgba_open` command.

Build and run. Navigate to **Tools > View logs** in the emulator menu to display the debug logs, and ensure that the **Warning** option is checked. If everything from this TODO is complete, you should see a message that reads "Debug logs initialized!".

TODO 2 – Drawing Functions

In this TODO, you will be writing and testing four functions, each designed to draw a specific shape.

TODO 2.0

- Complete the `drawRectangle` function in `gba.c`, where `x` and `y` control the coordinates for the top-left corner of the rectangle, `width` and `height` control the size of the rectangle, and `color` controls the color of the rectangle.
    - A filled-in rectangle can be drawn by stacking horizontal lines on top of each other.
    - The `setPixel` macro is defined in `gba.h`.
    - Be sure not to mix up height and width!

TODO 2.1

- Complete the `drawRightTriangle` function in `gba.c`.
    - This function is similar to `drawRectangle`, except that pixels should only be drawn on and below the diagonal on the shape (where $\Delta x = \Delta y$).

TODO 2.2

- Complete the `drawParallelogram` function in `gba.c`.
    - This function is similar to `drawRectangle`, except that each row of pixels in the shape should be shifted to the right by 1 pixel from the row above it.

TODO 2.3

- Complete the `drawCircle` function in `gba.c`.
    - NOTE: On each of the other drawing functions, `x` and `y` represent the position of the top-left corner of the shape. For this function, `x` and `y` will represent the middle of the circle, and the `radius` argument will represent the distance from the middle of the circle to its edges.
    - Remember that the formula for a circle is $x^2 + y^2 = r^2$. Pixels on or inside this boundary should be drawn, while pixels outside this boundary should not be drawn.

TODO 2.4

- In the `main` method of `main.c`, call each of these functions to ensure they are working correctly (you will delete them from this section later).

Build and run. You should be able to see each shape generated by the drawing functions. If there are any issues with how one or more of the shapes look, revisit those TODOs before moving forward.

TODO 3 – waitForVBlank & Game Loop

In this TODO, you will be writing and utilizing the `waitForVBlank` function to give your program a consistent frame rate. Additionally, you will write code in the while loop in the main function, allowing for the implementation of images that change over time!

TODO 3.0

- In `gba.h`, complete the `REG_VCOUNT` macro. It should have the same format as the `REG_DISPCTL` macro, only using a different memory address.

TODO 3.1

- Complete the `waitForVBlank` function in `gba.c`.

TODO 3.2

- In the while loop in the main function, call `waitForVBlank`, `updateGame`, and `drawGame` in the correct order.

TODO 3.3

- Complete the `COLOR1`, `COLOR2`, and `FRAME_DELAY` macros in `main.c`.
    - `COLOR1` and `COLOR2` should be given a color value.
    - `FRAME_DELAY` should be given a value between 20 and 60 for the number of frames elapsed between color flickers (given the emulator is running at 60 FPS, 60 frames would be about 1 flicker per second, and 20 frames would be about 3 flickers per second).

TODO 3.4

- In the `updateGame` function in `main.c`, print the current value of the `frameCount` variable on every frame.
    - NOTE: While you don't have to write the update code that causes the colors to flicker for this lab, take a look at it and try to figure out how it works.

TODO 3.5

- Call each of your drawing functions in the `drawGame` function, using the `flickeringColor` variable as the `color` argument for each. If you still have any leftover function calls from TODO 2.4, remove them during this step.

You will know your lab runs correctly if:

- A rectangle, right triangle, parallelogram, and circle are drawn on the screen
- Each shape is switching between two colors at a consistent rate

- The debug logs are activated and printing an output on every frame

---

Submission Instructions:

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of previous assignments for instructions on how to perform a "clean" command.**

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission Lab02_LastnameFirstname, for example:

"Lab02_NollMichael.zip"

It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.