Assignment 04

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to give you hands-on practice with the following concepts related to binary image processing:

- Thresholding and binarization
- Otsu's method for automatic thresholding
- Moments of shapes
- Morphological operations
- Using moments to identify and localize shapes regardless of their size and orientation
- Combining all of these together into a single tool

Part 1 of 1: Shape Finder

As we saw in class, binary image processing begins to give us a handle on some core vision tasks like detecting, localizing, and identifying simple shapes. In this part of the assignment, you will put this into practice by building a tool that can respond to queries like "how many small green rectangles are there in this image, and where are they?" During development, you'll use the `shapes.png` image given to you here:

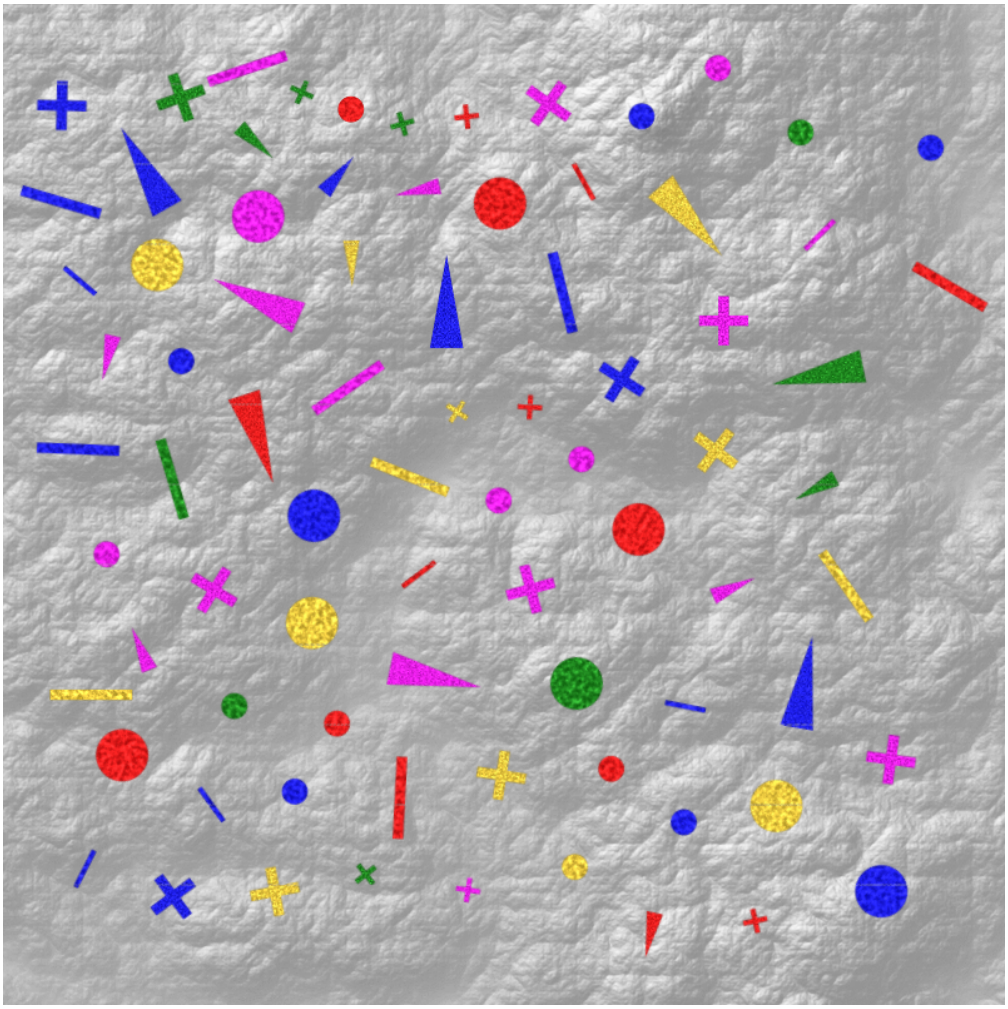

Figure 1: shapes.png

We'll then be testing your code with this image plus one 'held out' image. Here's what you can assume to be true about any of the images you'll be tested on:

- The shapes will not overlap with each other or occlude each other.
- The shapes will always be one of "rectangle", "circle", "wedge", or "cross", and they will always have the same *relative* proportions as those you're seeing here (i.e. we will not test you on equilateral triangles - 'wedges' will always be the same isosceles triangle you see in the example above, and rectangles will always have this long and narrow aspect ratio).
- The shapes will otherwise be randomly located, oriented, and sized.
- Each shape will come in roughly one of two sizes - small or large. However, the exact threshold for what counts as 'small' or 'large' will vary from image to image and shape to shape.
- The background will be some shade(s) of gray, and the shapes will always have some decently

saturated color.

- The colors of the shapes will be close to the ones you see here, but they might not be exactly the same. In general, they will be one of the 6 primary hues (red, yellow, green, cyan, blue, magenta), plus or minus some slight hue variation and texture-like noise (as in the example).
- Each image will have at least one 'small' and one 'large' version of each shape, which will allow you to automatically determine the threshold for what counts as 'small' or 'large' for each shape.

As in Assignment 02, you are given code that runs and is already commented and factored in a way that suggests how it *should* work, but contains a number of subtle bugs and omissions. Your job is to find and fix all the bugs. Unlike Assignment 02, however, we're making this a bit more difficult by using integration tests rather than unit tests. This means that you'll primarily get coarse-grained feedback on whether your code is working, but you'll have to figure out for yourself where the problems are. Use the debugger. Visualize intermediate results. It might be a good idea for you to write your own unit tests that will isolate potential problems in each function!

There are no bugs in `annotate_locations` or in the `if __name__ == "__main__":` block. Focus on the other functions in the `shape_finder.py` file.

Partial credit is possible. If you are unable to fix all bugs, you may still receive partial credit by detailing your thought process and debugging strategy in the `debug-process.txt` file. For any failing tests, we will look in the `debug-process.txt` file to see your thought process and strategy for that function. We may award partial points based on the quality of your debugging process, even if you were unable to fix the bug. If you write your own test code and include it in the `src` directory, make a note of it in the `debug-process.txt` file, and we will take that into account when grading.

Some examples of correct outputs:

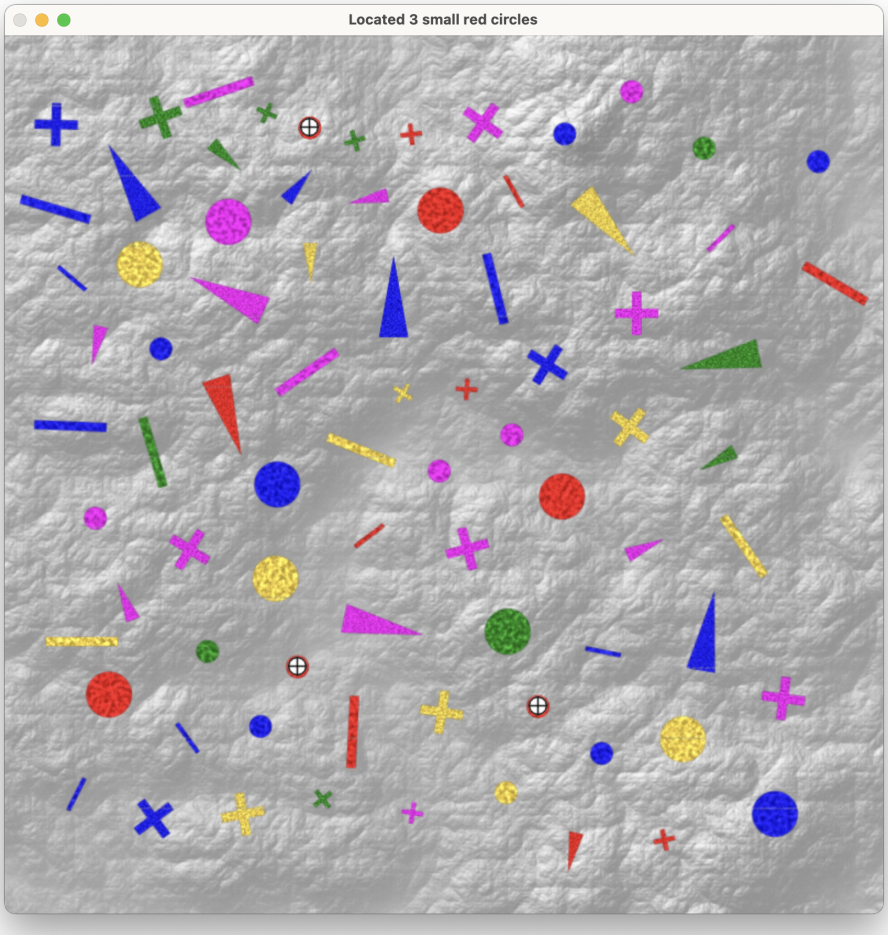

Figure 2: Example output of `python shape_finder.py shapes.png small red circle`

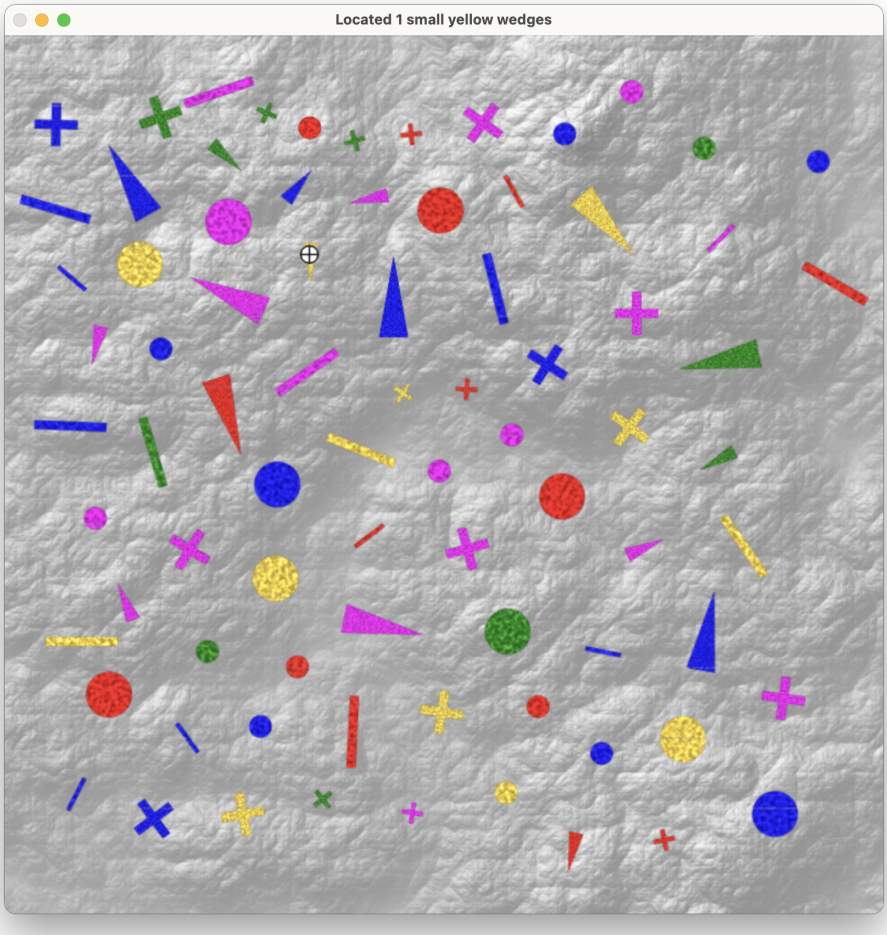

Figure 3: Example output of `python shape_finder.py shapes.png small yellow wedge`

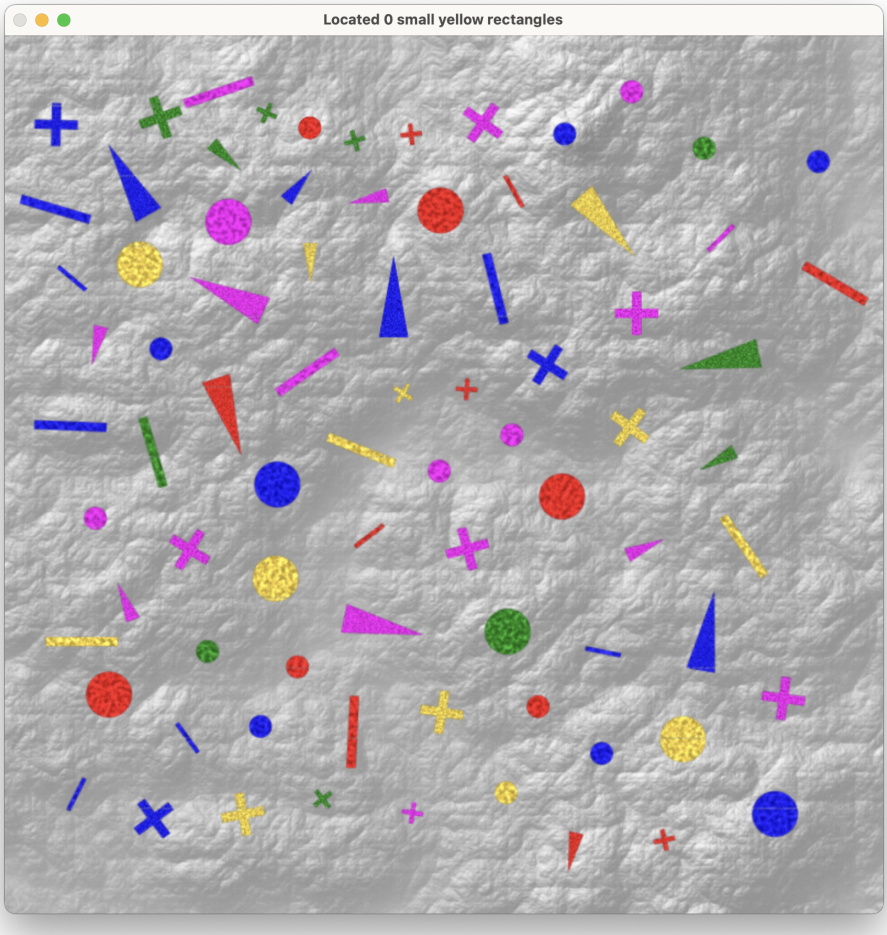

Figure 4: Example output of `python shape_finder.py shapes.png small yellow rectangle`

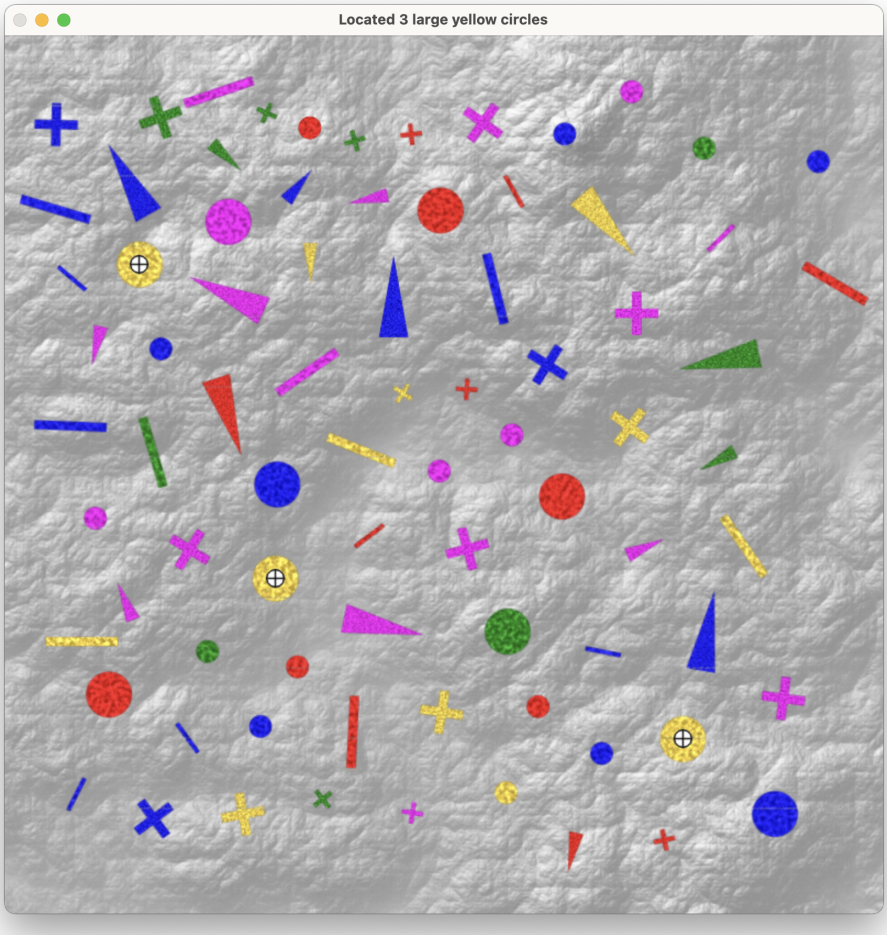

Figure 5: Example output of `python shape_finder.py shapes.png large yellow circle`

Collaboration and Generative AI disclosure and Assignment Reflection

Did you collaborate with anyone? Did you use any Generative AI tools? How did the assignment go? Briefly explain what you did in the `collaboration-disclosure.txt` file. **Completing the collaboration disclosure for each assignment is required.** Completing the reflection part of the file is optional, but it is a good way to give feedback to the instructor and grader about how the course is going.

Submitting

As described above, use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline.