Parallel Image Filtering GUI

Final project for CSE411 Fall 2024

Matt Culbertson

Purpose

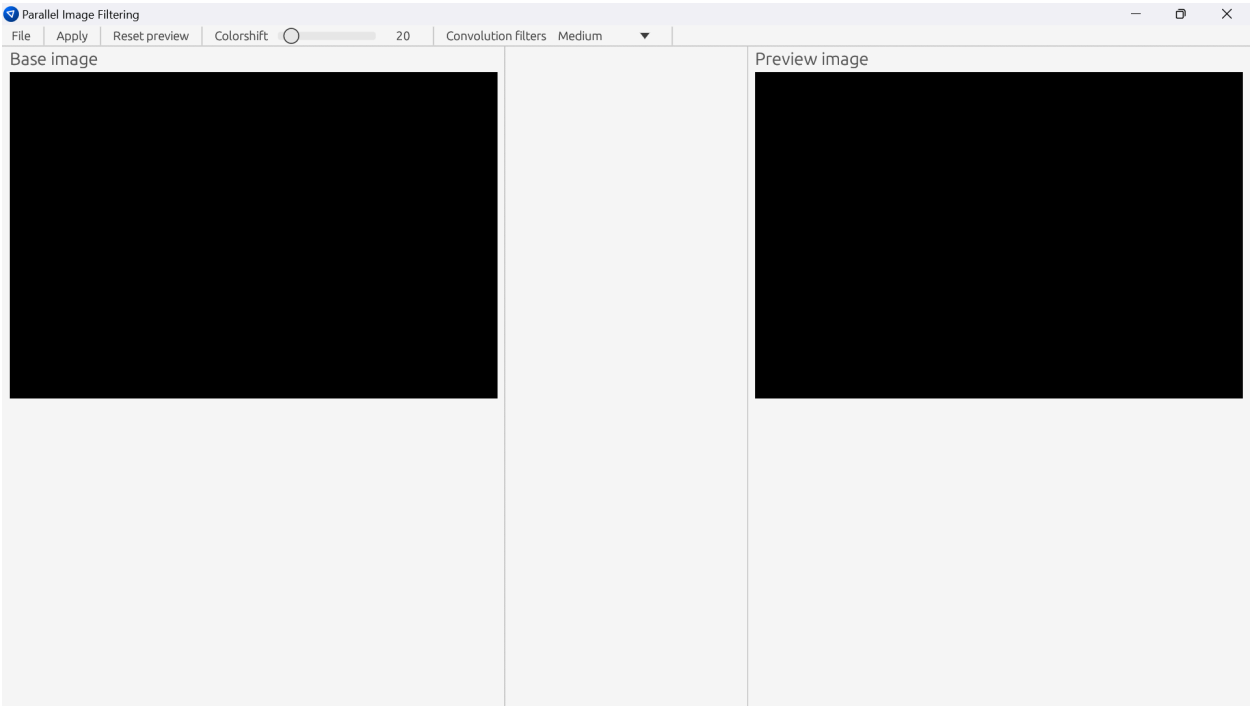
The purpose of this GUI is to apply filters to images. The GUI allows users to import and export images and apply filters one at a time to the uploaded image. To improve the speed of the filtering process, the filtering operations are done in multiple parallel threads. This application is targeted at users who want a simple and lightweight GUI for simple image filtering.

Installation and Startup

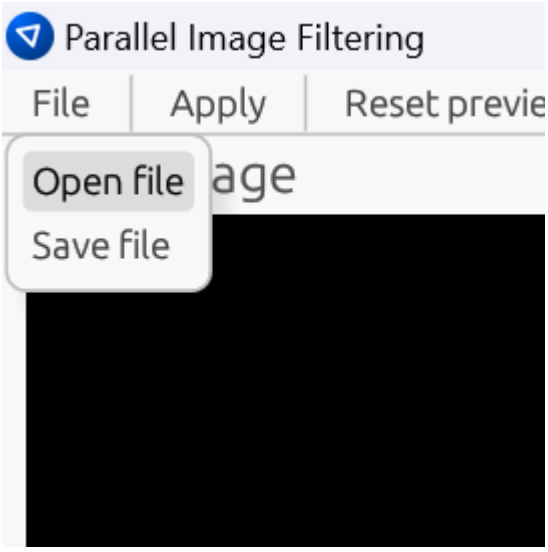
The executable file (GUI.exe in the root directory) is compiled for Windows x86 machines. For those machines, no setup or installation is needed, only to run the executable file. To compile the source code for other architectures or to make changes, Rust and Cargo will need to be installed. Version 1.79 of Rust and Cargo was used.

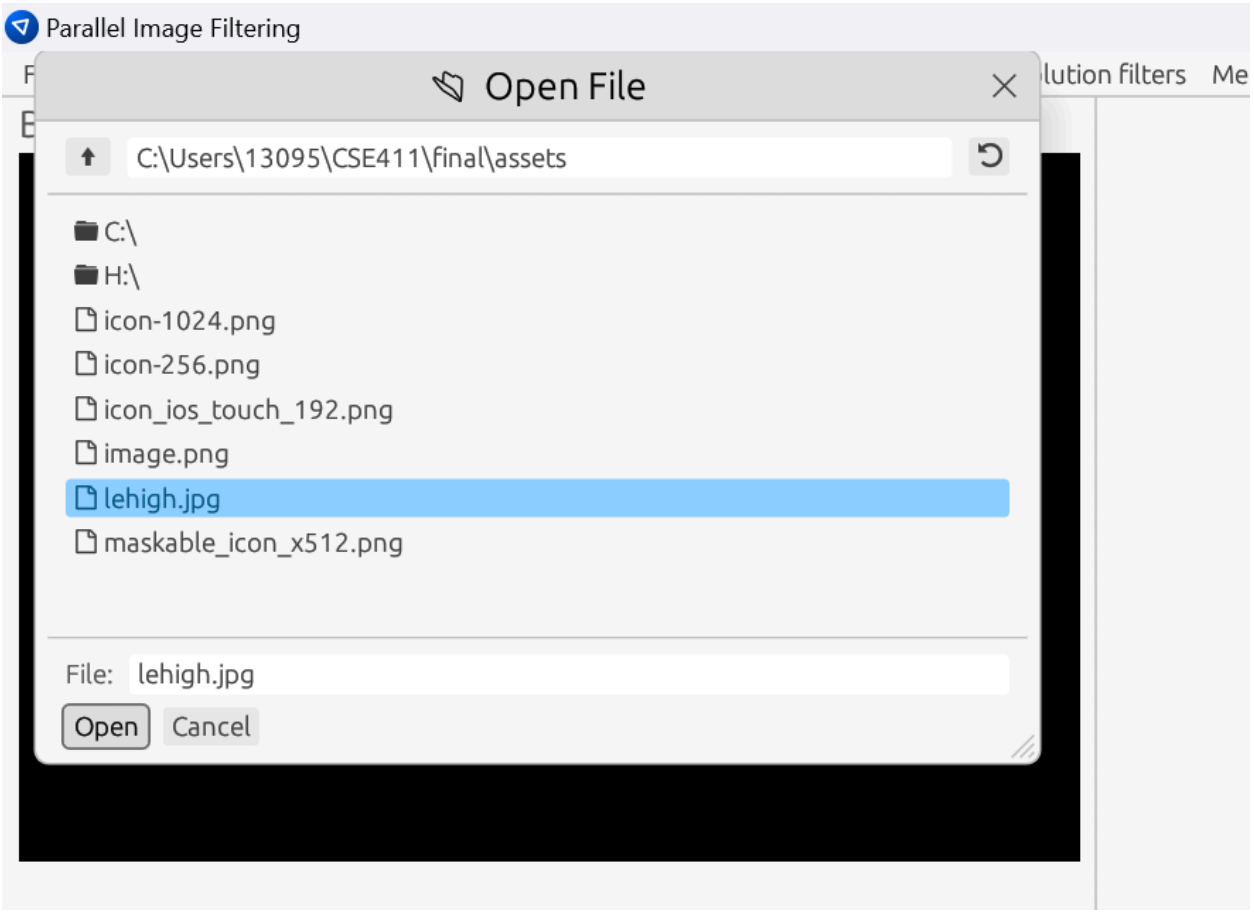
Usage

When the application is first opened, two black placeholder images will be displayed. The image on the left is the original image, and the image on the right is a preview of the filter. To make changes to an image of your choice, first you need to import an image.

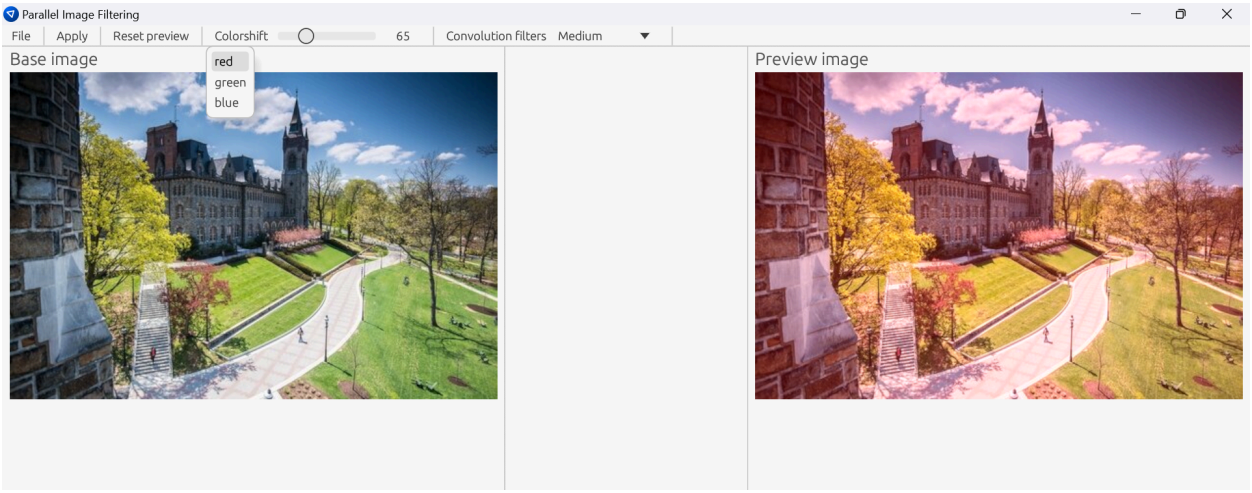


The button to open files is under the “File” option in the top right. Selecting that button will open a file picker popup as shown below. Only files with a jpg, jpeg, or png extension are supported, and so only those files will shown by the file picker.

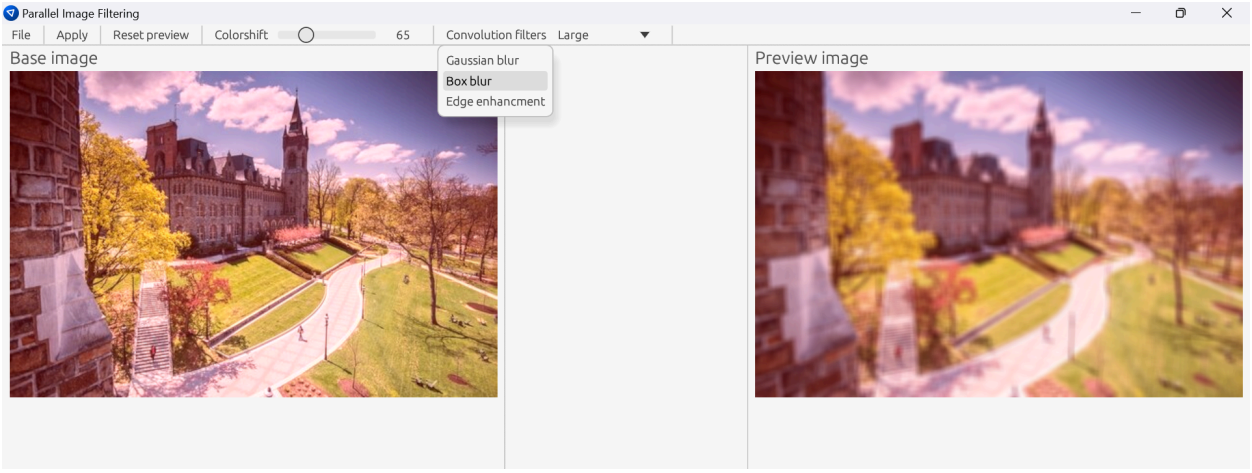




There are two main categories of filters in the application - colorshifts and convolution filters. The color shifts add a value of 1 to 255 to the existing color value of each pixel, either red, green, or blue. The amount added is configurable with a slider to the left of the color shift option in the top menu panel. The convolution filters (two blurs and an edge enhancement) are configurable by the size of the kernel. The drop-downs for the convolution filters and their configuration are also located in the top menu panel. The Gaussian blur retains more detail than the box blur. The edge enhancement filter uses a combination of a Laplacian filter on a Gaussian filter to emphasize areas where there is rapid change between neighboring pixels.



Only one filter can be applied to the preview image at once. Applying another filter while one is already present on the preview image replaces the older filter with the new one. To stack filters and save your changes, you will need to use the apply button in the upper right. The apply button copies the changes from the preview image over to the base image. Then, another filter can be applied on top. Because the “Save image” option saves the base image, it is important to apply filters before exporting the image. The example below shows that a redshift has been applied, and a subsequent large box blur. The “Reset preview” option in the menu bar will clear the filter from the preview image so that it matches the base image.



When you are satisfied with your result, make sure to save your image under the “File” option in the top right - no images will be retained on exit. This will open up a similar file picker to the “Open image” option. All files will be saved in PNG format. Saving a file will save the base image on the left side, so remember to apply changes that you want to be saved.

Design

At the core of the application is an update loop that contains instructions on how to rerender the GUI. Code from inside this loop calls other functions to handle user actions. There are two vectors of 8-bit unsigned integers that store image data in rgba format and two egui::TextureHandles that store information about how to display the images. These data structures are updated by various filtering functions to apply filters. The color shift filtering uses parallel iterators for multithreading. The convolution filters spawn crossbeam scoped threads that each process a section of the image. Because a convolution filter takes input from neighboring pixels, each threads can view the entire original image, but only modify their own section.

Sources

Extensively used the documentation for Rust libraries - https://docs.rs/.

Starter code for the GUI came from “eframe_template” on Github.
https://github.com/emilk/eframe_template.

Using textures to display vectors of color values in egui came from “noise-functions-demo” on GitHub by user bluurryy.
https://github.com/bluurryy/noise-functions-demo/tree/main