MATLAB Image Stacker App

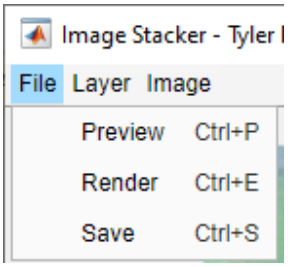
Tyler Pigott

This MATLAB app allows you change the background of an image of an item of a specific color. It provides options to combine an arbitrary number of images with segmented colors together, allowing for many cool effects. The app requires that the Image Stacker and Image Stacker Settings apps are in the same directory to run. More of the specific features of the app are outlined below.

Features

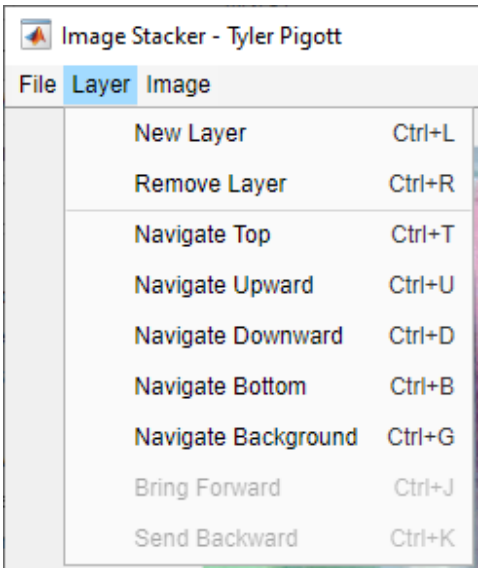
This app comes with several features. Some of the features may be difficult to understand at first, but a few simple examples will be included to show how they could be used. Most features are accesses through the menu bar at the top of the screen, and many features involve opening a second, connected app via the menu bar. Below I will walk through each of the menu bar options.

File



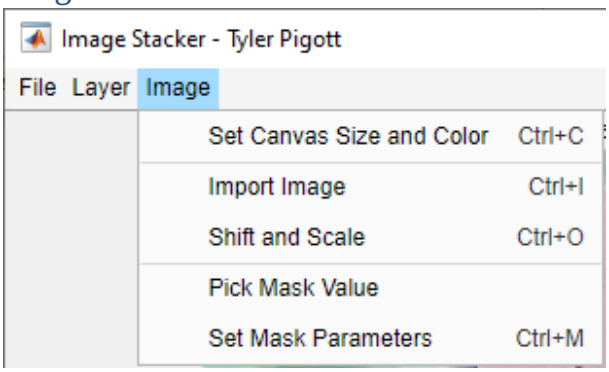
Under the file section there are a few options for the entire app. Preview will combine all the layers of your image to give a preview of the final version. All previews in the app have dimensions no greater than 300 pixels to make rendering quicker. Render will combine the layers at the full size and typically takes longer than Preview. Save renders the image and saves it under a user-specified file name.

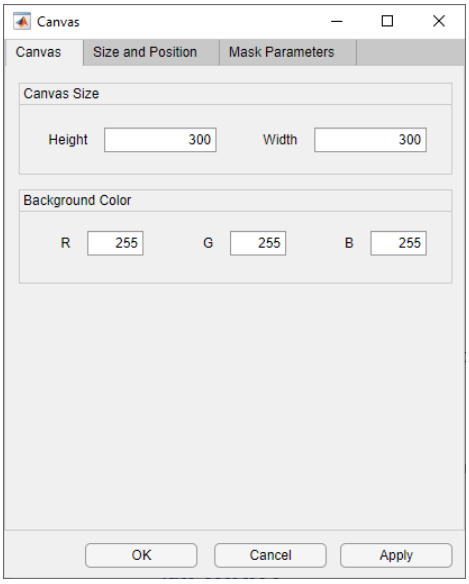
Layer



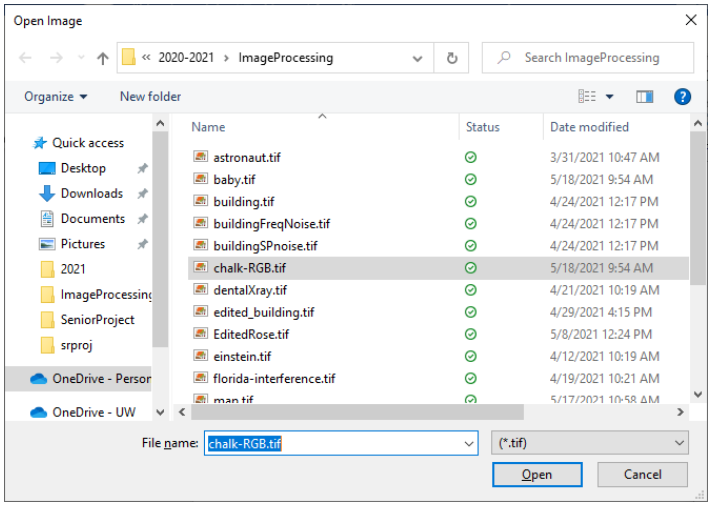
The layer menu allows you to manipulate the layers of your project. Most of these options are pretty self-explanatory. New Layer makes a new blank layer. The placeholder image for a new layer is always 100% transparent, so it will just look like the background of the canvas. Remove Layer will delete the current layer. And move you to the next layer down. Navigate Top takes you to the layer that renders on top of all the other layers, the highest number layer. Navigate up moves you one layer up, e.g., layer 3 to layer 4. Navigate down does the opposite. Navigate Bottom takes you to the bottom layer, or layer 1. Navigate background takes you to the background layer where only the canvas color is visible. You can also move layers up or down using the Bring Forward and Send Backward buttons. Some of these options will be disabled if there aren't enough layers.

Image

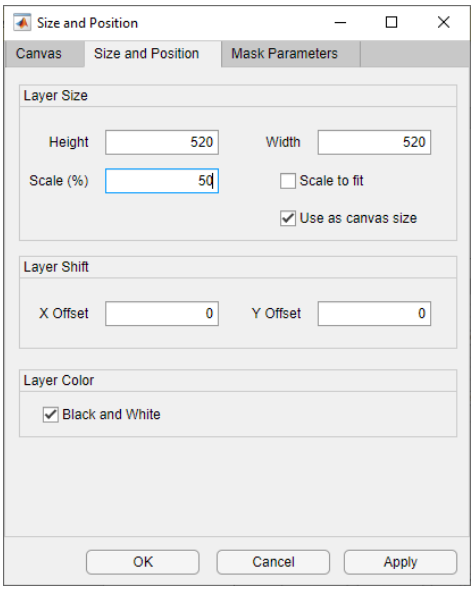




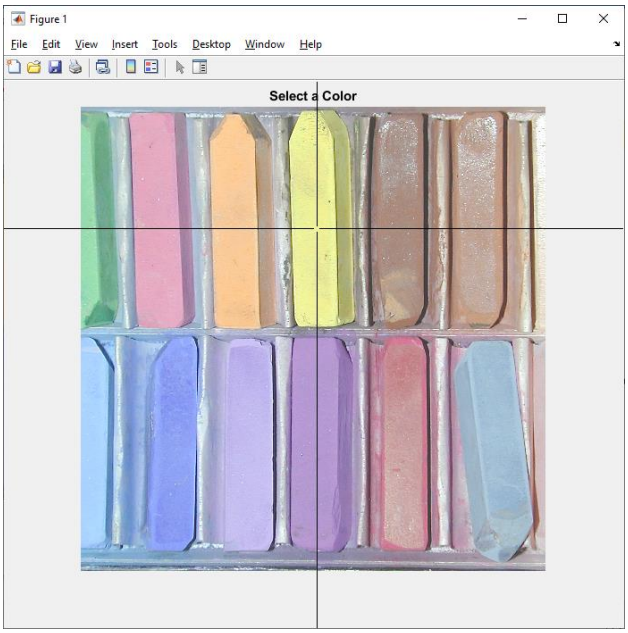
The last option on the menu bar is the Image menu. This menu contains the most interesting options. Firstly, you can set the canvas size and color in “Set Canvas Size and Color.” This opens up a new window with input fields for the canvas options.



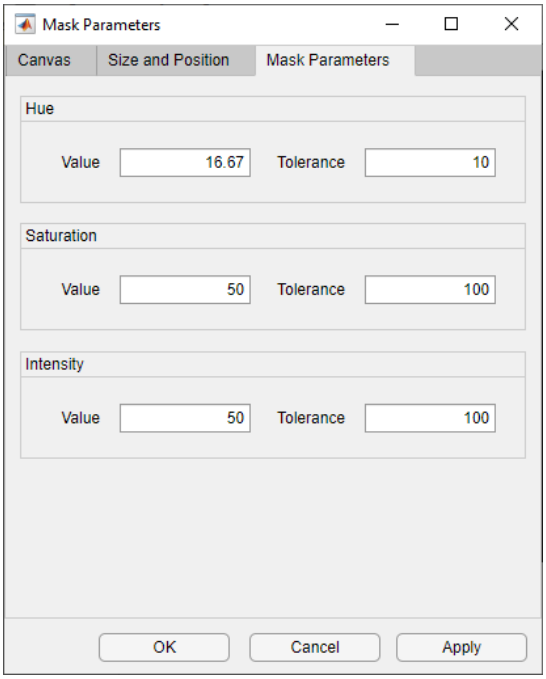
Import Image opens a system window to load an image to a layer.



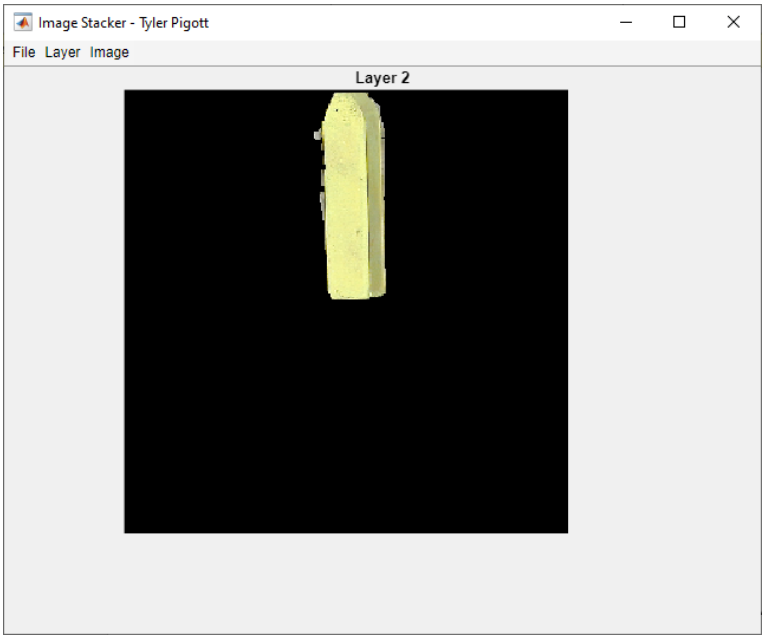
Shift and Scale allows you to move the layer relative to the canvas and change the size of the image with a new option window. This window has fields where you can input a desired height, width, or scale for an image to set its size. The aspect ratio must be maintained. There is also a “scale to fit” option that scales the image down, so it fits in the canvas and a “use as canvas size” option to reset the canvas size to fit the image at its new scale. The X any Y Offset fields shift the image left and right based off of the current image scale, and the black and white checkmark can be used to convert the layer to grayscale.



Pick Mask Value opens a new window showing the original version of the current layer and allows you to select a hue of the image to keep by clicking on the image where the color you want is. The mask parameters are then set to default values, allowing for 10% tolerance of nearby hues and 100% tolerance of saturation and intensity options.



Set Mask Parameters allows you to adjust the values and tolerances of the mask to filter out more or less of the image. The values considered are within a range the size of the tolerance centered on the value out of 100, so a value of 50 with tolerance 100 covers all values, but a value of 25 with a tolerance of 50 only covers the lower half. Hues wrap around, so a value of 100 or 0 with a tolerance of 10 would both consider hues of value 95-100 and 0-5. Any mask is sent through a quick closing filter to clean up the results a little bit. The “Apply” button is especially useful here to test some of the mask values but recalculating the mask after an adjustment takes time.

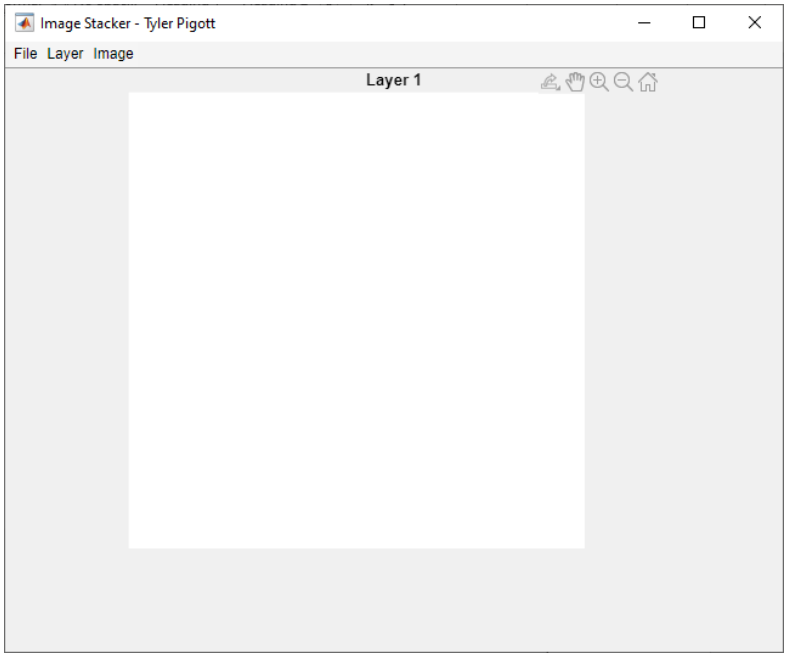


Most of the area of the app is taken up by the image preview pane. The text above the image preview should display what you are looking at, either a layer number or the preview/render state. The preview of a layer is no more than 300x300 pixels to make the previews load quickly, but a render is displayed at full size. Whenever no image sets a value for a pixel, either when a mask is applied to an image in a layer preview or in the full combination of all layers, the unassigned pixel is set to the background color.

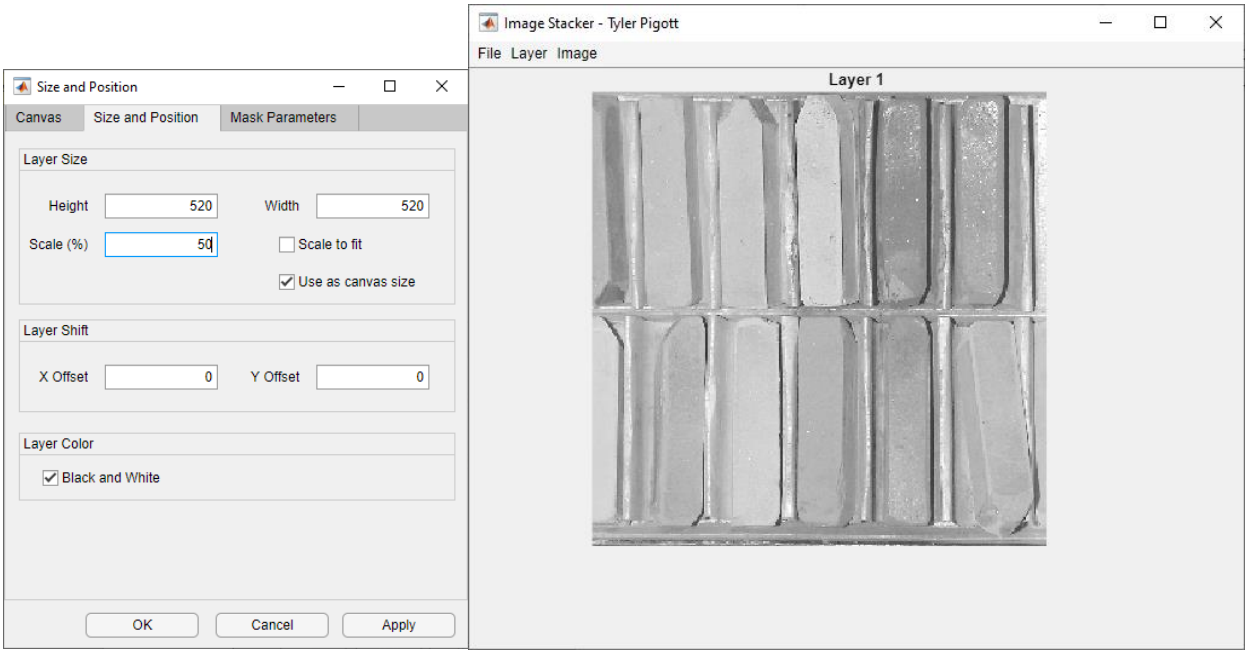
Now I will go through a few example projects to demonstrate a typical use of the software.

Example: Color Pop

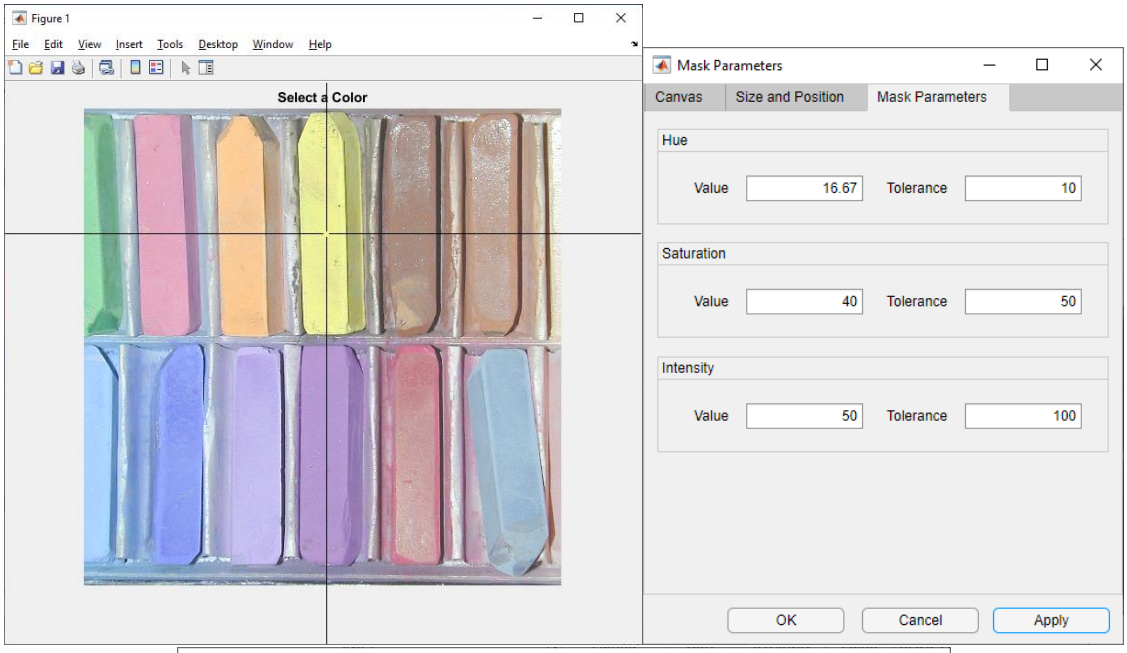
Start a new instance of the app. You should see a blank white canvas like below. We don't need to set the canvas size or color because we will set our canvas to the size of our image.

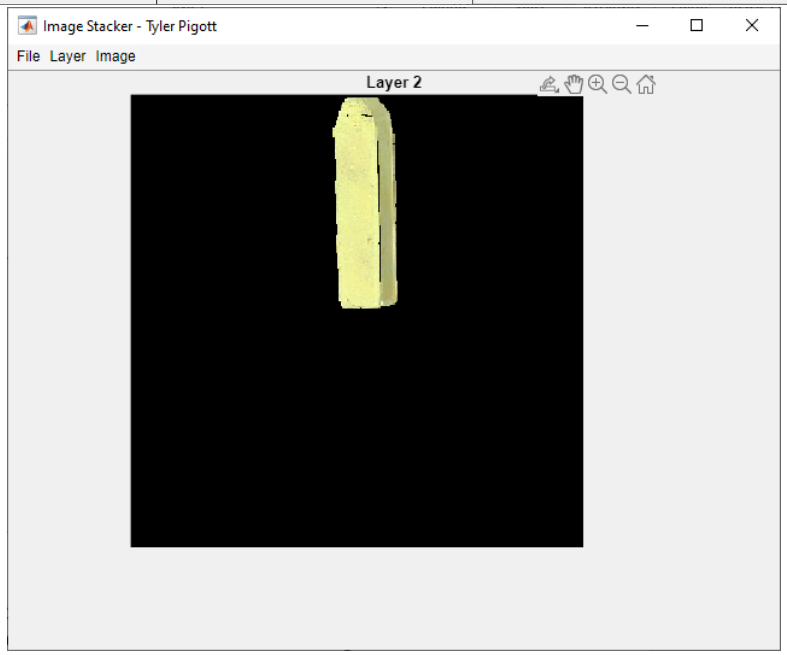


On layer one, import the chalk-RGB.tif image. Once it has loaded, go to the shift and scale menu. To make rendering the image quicker, set the scale to 50. You should see the height and width set to 520 pixels. Next, check the "use as canvas size" box and the "black and white" box so the canvas changes to 520x520 and the image is switched to grayscale. When you press OK, you should get a preview of layer 1 like shown below.

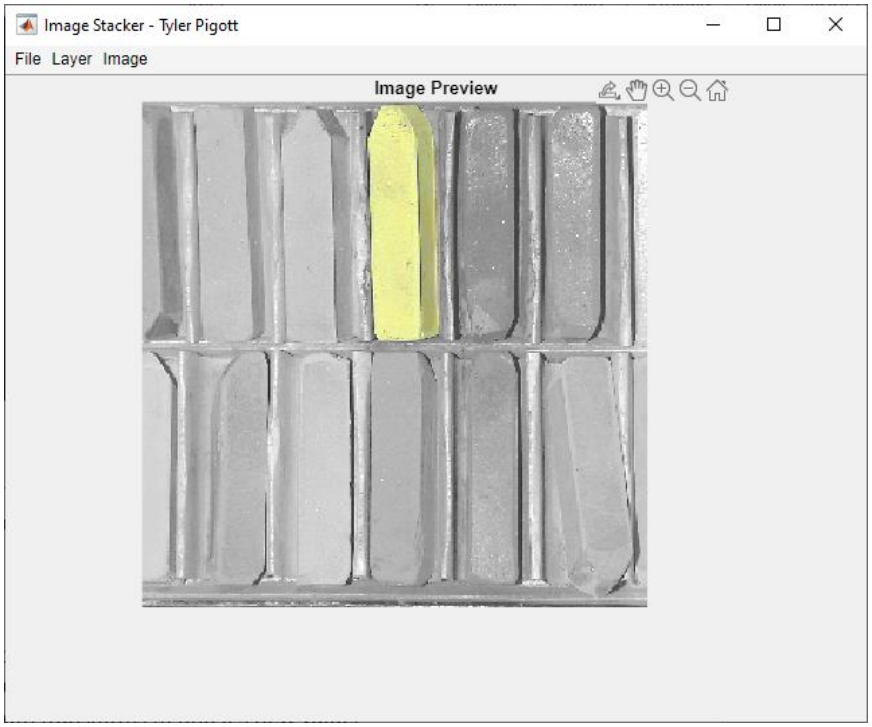


Next, make a new layer, and import the same chalk-RGB.tif image. Again open the shift and scale menu and either set the scale to 50, the height or width to 520, or select “scale to fit” to make this layer the same size as the first one and press OK. Now, use the Pick Mask Value option to select a color of chalk to pop out. I will choose Yellow because it works well. Then use the Set Mask Parameters option to adjust the mask to improve your results.

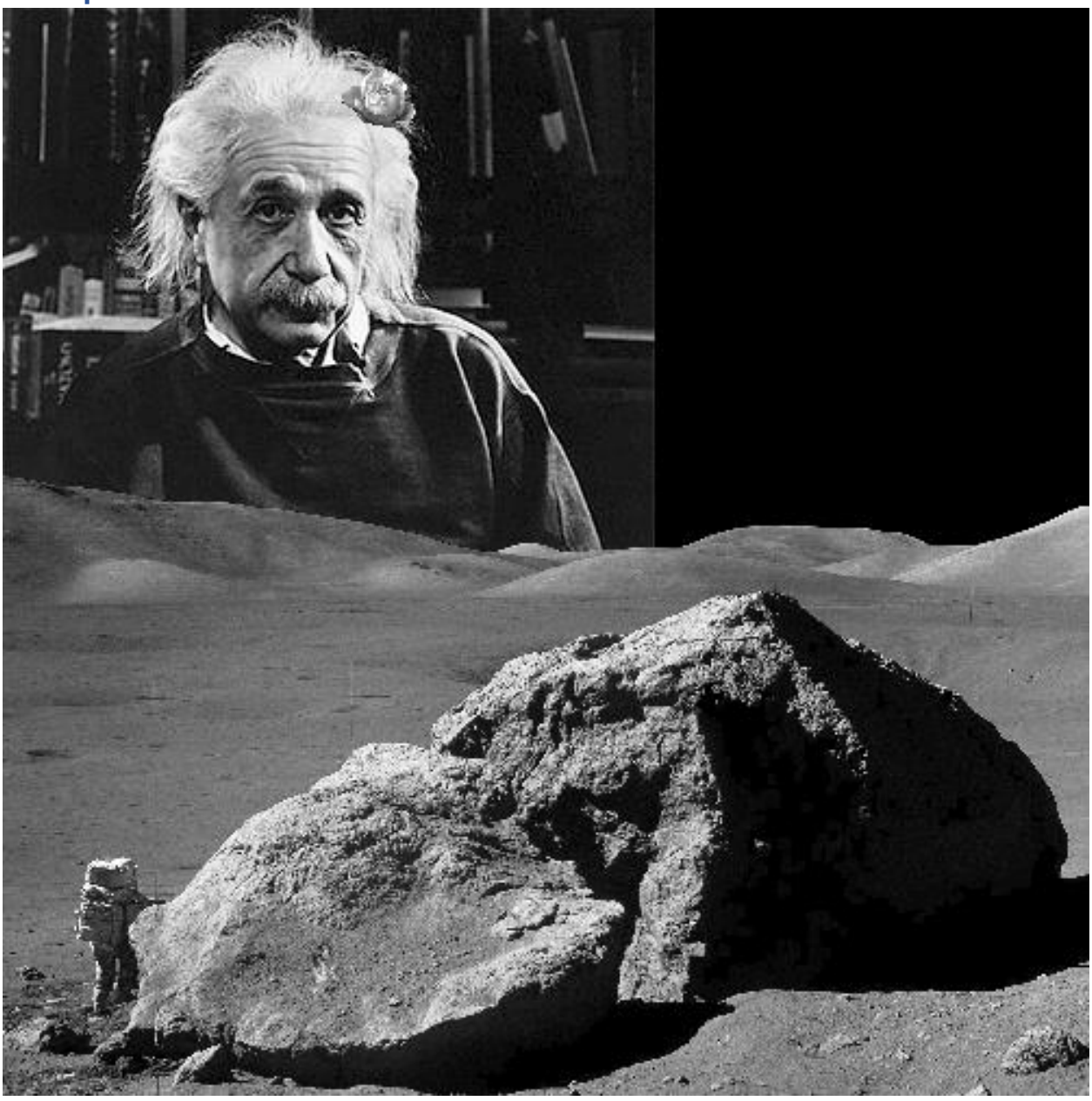




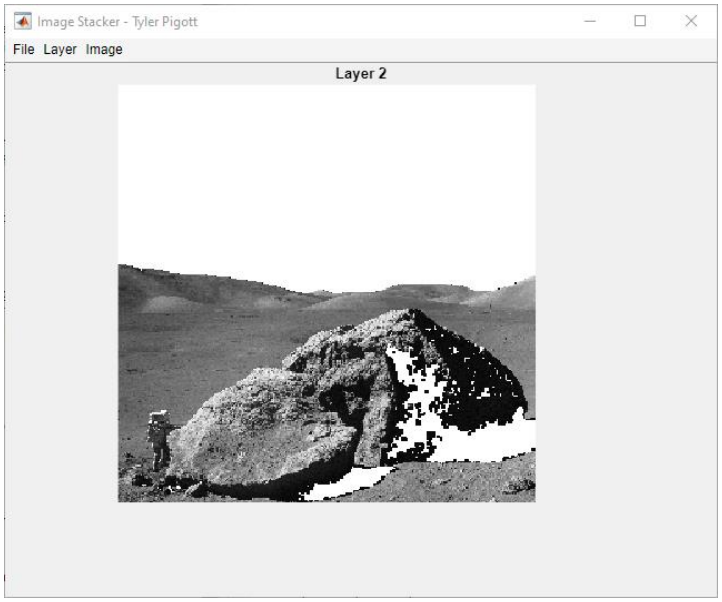
Now you can use the preview button to combine the layers at a lower resolution, and if satisfied, either render or save the image.



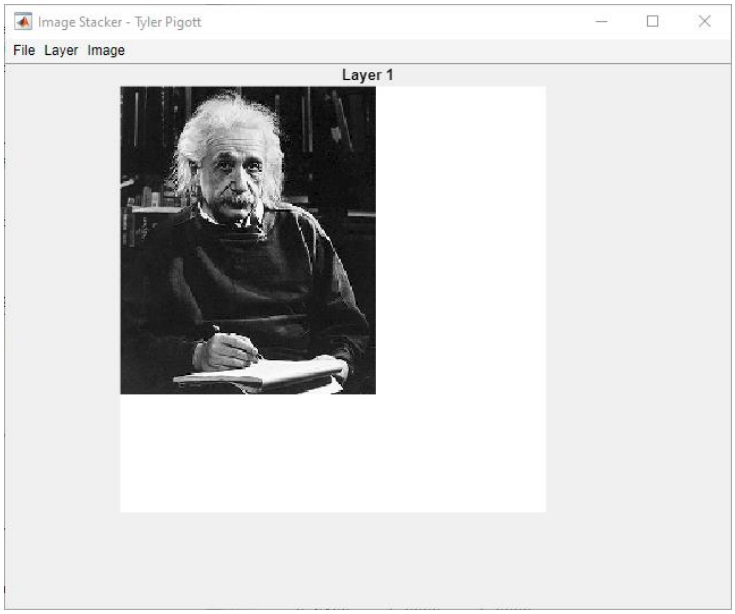
Example: Modern Einstein Art



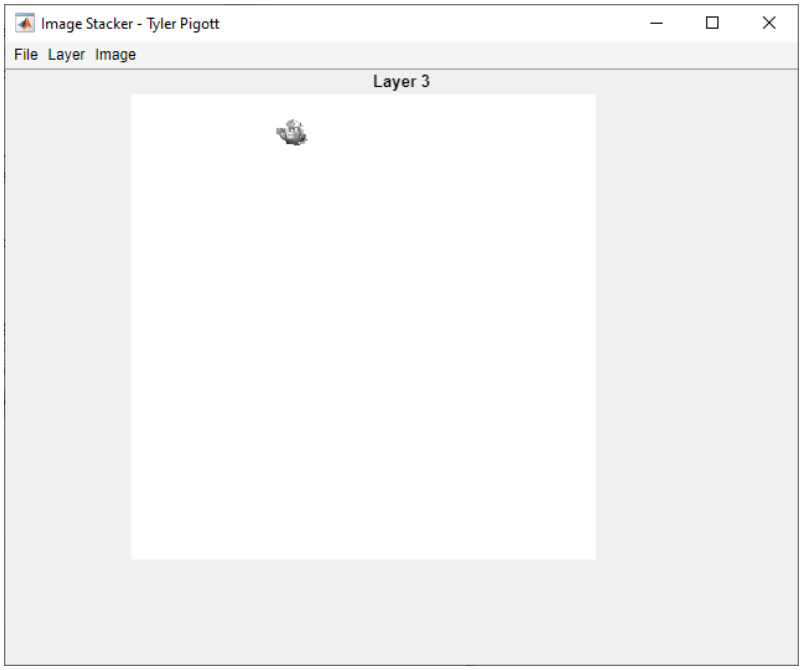
Start a new instance of the app. On layer one, import the einstien.tif image. On layer 2, import the astronaut.tif image. On layer three, import the rose-LR.tif image. Set the canvas size to 500x500 with the Set Canvas Size and Color menu. Navigate to the astronaut image on layer 2 by using the navigate downward button and open the Shift and Scale menu. Set the scale of the astronaut image to 25, and the X offset to 97.5 to scale down the image and shift it to the bottom of the canvas. Now, open up the Set Mask Parameters menu and change the intensity value to 50.1. You should see something like below:



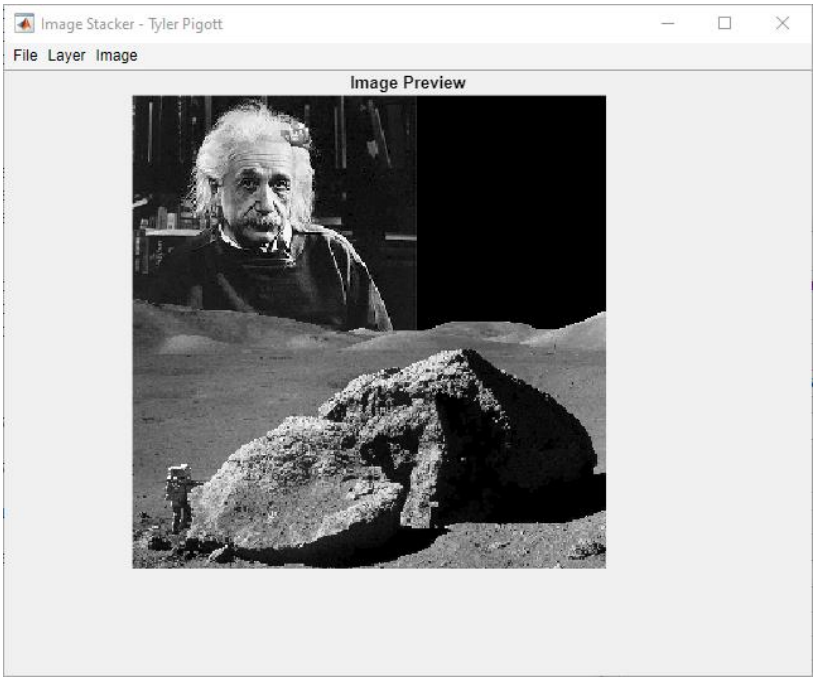
Now swap over to layer 1 using the navigate to bottom button. We need to shift Einstein further into the top left corner, so we will set the scale to 50. You should see something like this:



Now go to the top layer with the rose. Scale the image to 20% and shift it by 20 X and 150 Y. Set the intensity value to 54 and the intensity tolerance to 92 to keep only the lightest pixels. You should see something like this:



Finally, set the canvas color to [0,0,0] and preview the image. If it looks right, you can save it. You should see something like this, where Einstein with a rose in his hair is in space behind the moon:



Methods

The methods used for this project are fairly simple. Image resizing was accomplished using simple upsampling without interpolation, just checking the nearest pixel for it's value. the black and white checkbox just averages R,G, and B values and sets them all equal. The image combination is a set of nested loops, going through each row and column and checking each layer starting from the top to see if an unmasked pixel is present, then copying the values and going to the next pixel.

The most complicated and new system in the app was the color segmentation section. Using HSV parameters, similar pixels are identified, and other pixels are thrown out. Pixels are identified with an HSV version of the image, and pixels that are rejected have a mask value set to 0. This mask is stored with the original image as a fourth layer. So base_imgs(m,n,4,l) is the mask value, 1 for include and 0 for reject, for the pixel at (m,n) for layer l. the masks are run through a closing function, an erosion of the original followed by a dilation, to clean up the masks a bit, removing single lonely pixels and making straighter lines.

Using a second app for the entry boxes allowed the original app to keep nice and clean and was an interesting challenge to work with. The layer system was also a fascinating challenge, finding a way to scale the operations for any number of layers was fun.

Conclusion

Overall, I'm happy with how this turned out. I wish I could have included more color adjustments and more customizable morphological options for the masks, but the project got complicated enough by itself. Let me know if you have questions about anything. I think I fixed most of the errors, but there might still be a few here and there. I didn't run in to any following my examples.