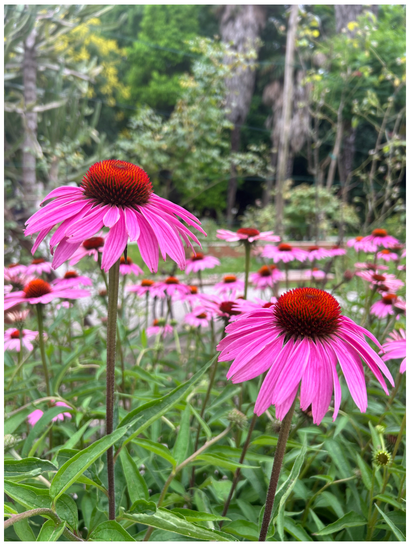
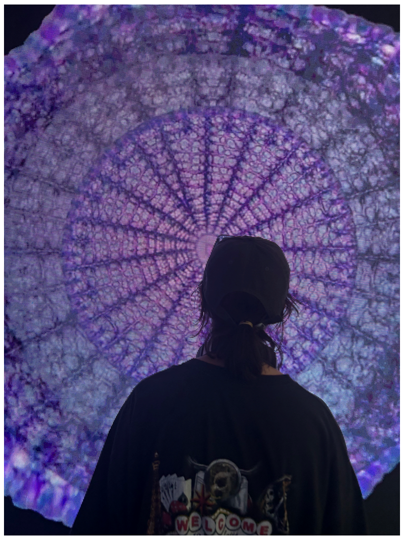
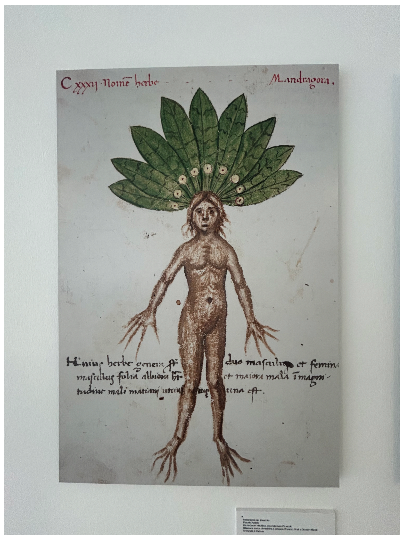
Sadie, Sam, Davis
Project 2 Write-Up
CS 4475
June 28th, 2024

1. Project Brief

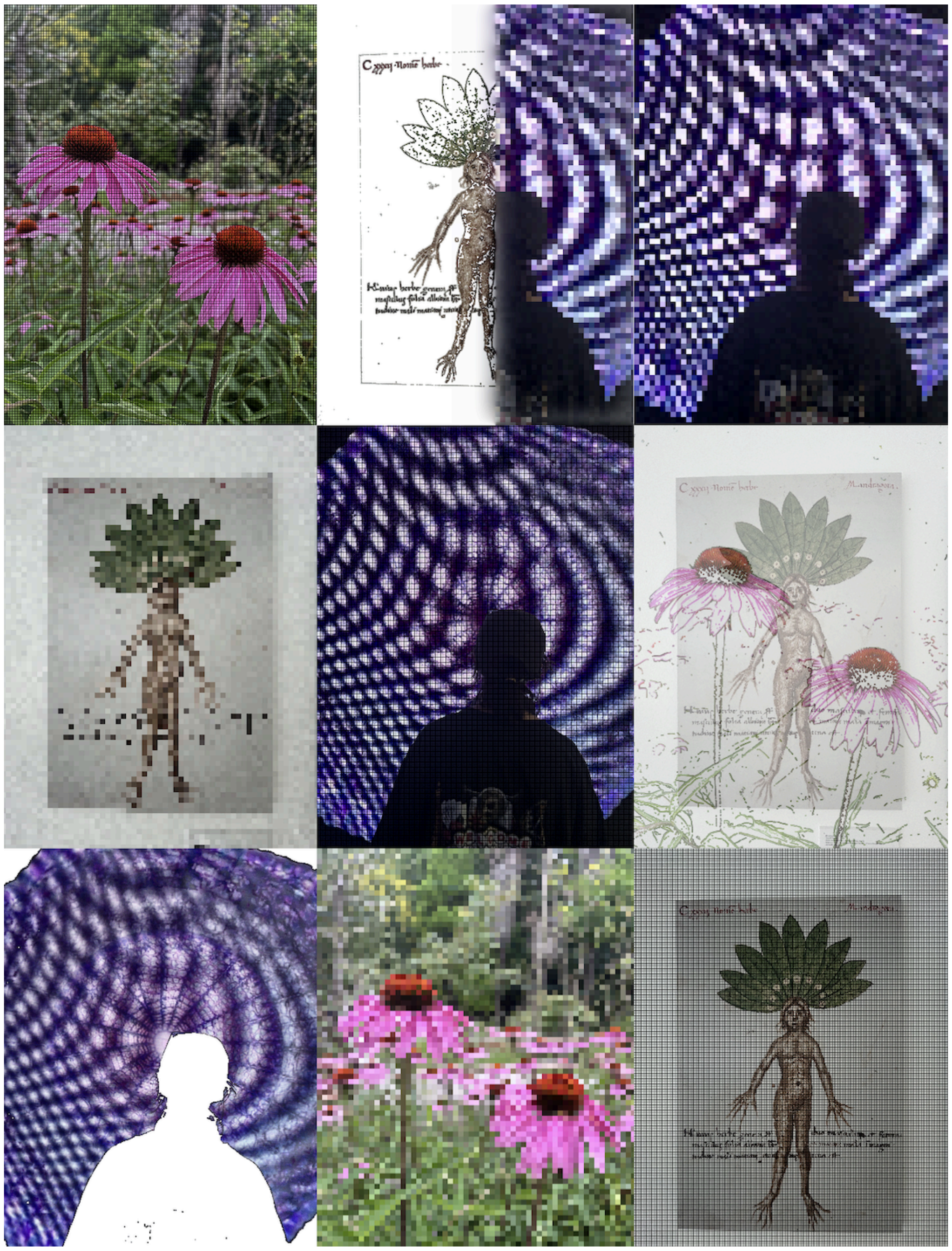
In this project we wanted to assemble a quilt with 9 unique patches. In order to do this we are using three different images and running each of them through three different functions, giving them effects to make them unique. Just as a quilt overlaps on the edge of the patches, we use a blend function to blend the edges of each image. In the end, the result is a completely new, quilt-like image.

The code runs eighteen separate functions. Each group member wrote a function that creates their own filter that is applied over each image, these functions include a glitch, an overlay, and a pixelation. The next function takes these three filters, runs them over each image, and then stitches the nine output images together to make the quilt. To blend the horizontal edges we imported the code from HW4 and modified it to match the code previously mentioned. To run the code, the user can input their own three photographs in cv2.read line at the bottom of the code, outside the functions. The code will then open a window thread to show the user their quilt.

Input images: (images self owned): chosen for their distinct colors and shapes.

Output image:


Mandragora

2. **Glitch function**: The function takes in one the three images, intensifies the RGB values, and then puts a black border around every 10th pixel in the image. The result is “glitched” picture, where black dots are seen across the picture and the colors of the pixels are intensified.

Pixelate Function: This function iterates over one of the three images in blocks of size pixelation_factor x pixelation_factor. For each block, we get the color of the top-left pixel. We then fill in the entire block with that color, creating a pixelated effect. The result is a pixelated image, where each block of pixelation_factor x pixelation_factor pixels has the same color.

Overlay Edges Function:

To write the function overlay_edges(image), I started by applying the Canny edge detection algorithm to the input image to detect edges, using thresholds of 150 and 200. Next, I converted the edges from grayscale to a BGR color format so they could be processed in color. I then dilated the edges to make them bigger and used bitwise inversion to use for the overlay. In the nested loops, I checked each pixel, and if it was black (indicating an edge), I replaced it with the corresponding pixel from the original image. Finally, I applied a Gaussian blur to the resulting image to smooth it out before returning it.

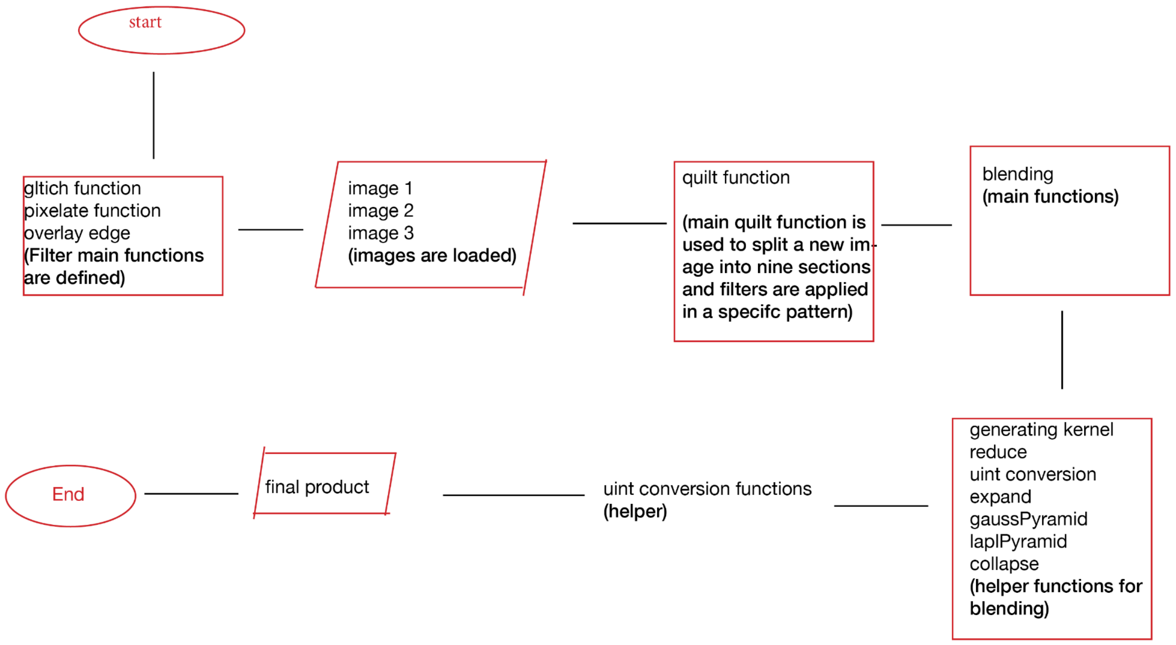
Creating the “Quilt” Format:

To write the function Quilt(image1, image2, image3), we began by creating a blank image that would serve as the canvas for the quilt, with dimensions three times the height and width of the input images. We defined a pattern for applying three different filters—Glitch, overlay_edges, and pixelateImage—in a 3x3 grid. We organized these filters into a list of patterns corresponding to each row of the quilt. For each cell, we selected the appropriate filter based on the current pattern and applied it to one of the three input images. For specific positions in the grid, we added additional logic: blending images at certain positions to create a smooth transition effect. Specifically, in the first row's middle column, we blended the left and right halves of the filtered image with adjacent images. In the second row's last column, we blended the filtered image with another image using cv2.addWeighted. After doing each cell, we placed the filtered image into the corresponding position on the quilt canvas. The final quilted image is returned as the output.

Blend function: Imported from HW 4, we used the same conditions stated in HW 4 to create the pyramids and blend the horizontal edges of the image's nine boxes. We used the mask from HW 4 for slightly blend the horizontal edges, making the line between the rows softer.

Conversion: after blending the two functions convert the image to account for overflow

Project flowchart:



3. To accomplish a “glitch” effect, I researched different glitch photos and saw this picture online.

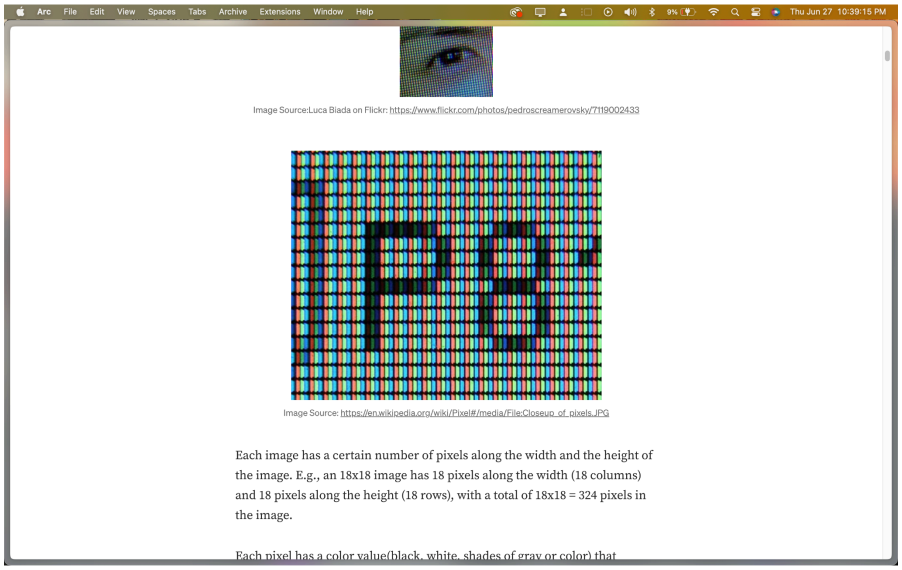

Arc File Edit View Spaces Tabs Archive Extensions Window Help
Thu Jun 27 10:39:15 PM

Image Source:Luca Biada on Flickr: https://www.flickr.com/photos/pedroscreamerovsky/7119002433

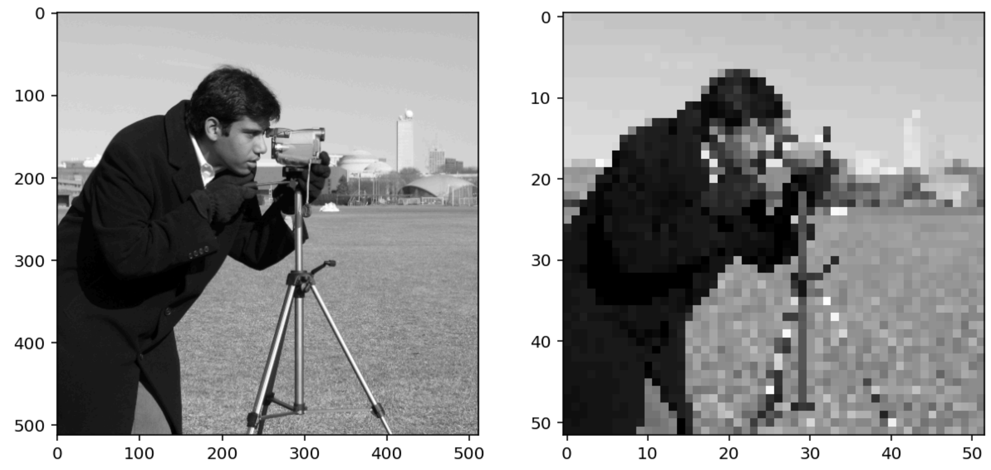
Image Source: https://en.wikipedia.org/wiki/Pixel#/media/File:Closeup_of_pixels.JPG

Each image has a certain number of pixels along the width and the height of the image. E.g., an 18x18 image has 18 pixels along the width (18 columns) and 18 pixels along the height (18 rows), with a total of 18x18 = 324 pixels in the image.

Each pixel has a color value(black, white, shades of gray or color) that

(credit: https://medium.com/@nimritakoul01/image-processing-using-opencv-python-9c9b83f4b1ca)
I thought the best way to accomplish this filter would be to intensify the RGB values. I noticed each pixel was bordered by a small black line to get that gradient pixelated effect.

The pixelated effect was inspired by this image:



https://scikit-image.org/skimage-tutorials/lectures/1_image_filters.html

4. 30 artistic 30 technical