Eduardo Verde

EVERDE@Rollins.edu

Dr. Summet

Computational Photography

Pixelated Selfie

Honor Code:

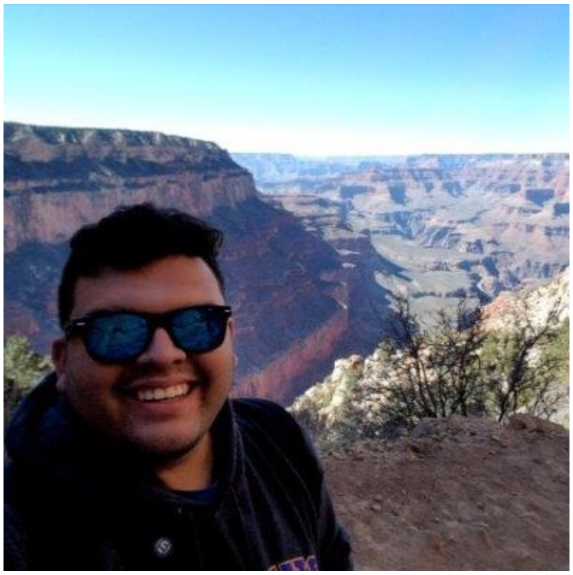
"On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work."

High Level Description:

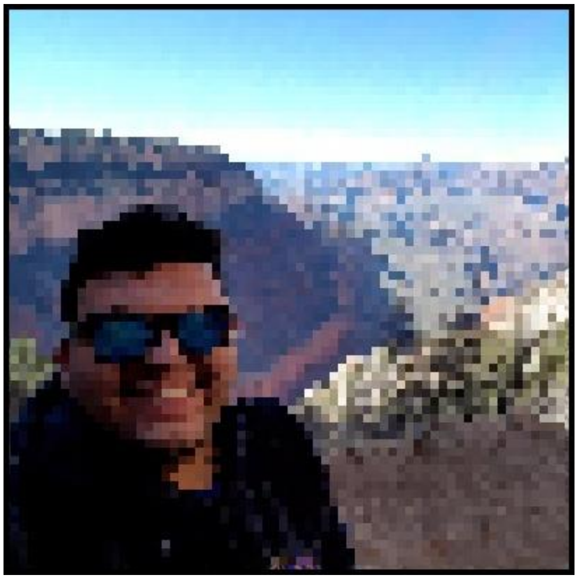
For this project, I tried to create a program that makes any image look like a retro 8-bit pixelated image. I came up with this idea while playing Mario Kart with some friends. I started the process by reading about Dither. I eventually wrote Dither algorithm but it came out as more noise than pixelated. Therefore, with some influence of Dither's Algorithm, I wrote a program that met most of my expectations. As you can see in the images below I took the original selfie and made it pixelated.

Images:

Original Image:



Pixelated Image:

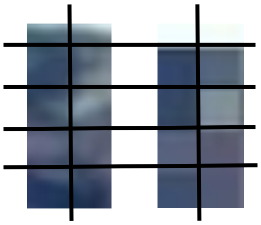


Low Level Description:

Below is a work flow of how the program works to create the pixelated image above. The majority of the processing happens between steps 4 and 5. This can be slightly confusing when looking at the flow chart therefore below is a more detailed description and example of these steps

Step 4 and 5 Description

1) Starting with the pixel at (0, 0) the program will iterate to our limit which is calculated by taking our starting point and adding the pixel_size. This is done for our x and y axis.
   a. The pixel_size is calculated in step 3, it is found by taking the length of the image and dividing it by 64. I picked 64 as a number to divide against, because between multiple images 64 gave us a result closer to 8. 8 also seemed to be a good range to create the pixelated area. When doing the division we used the double / to get the floor value.
2) During the process the program will collect the color and coordinates of each pixel within this range.
   a. The color collection is done with a dictionary where the key is the color (RGB) and value is the number of times that color appeared.
3) Once that iteration has finished, the program will get the most common color by getting the max count in the dictionary.
4) Then it will iterate over each pixel that it just visited once again and assign each pixel that most common color found in the dictionary.
5) Once completed, it will go back to the parent loop which will go to the next pixel iterating by the length of pixel_size, for example from 0 to 8



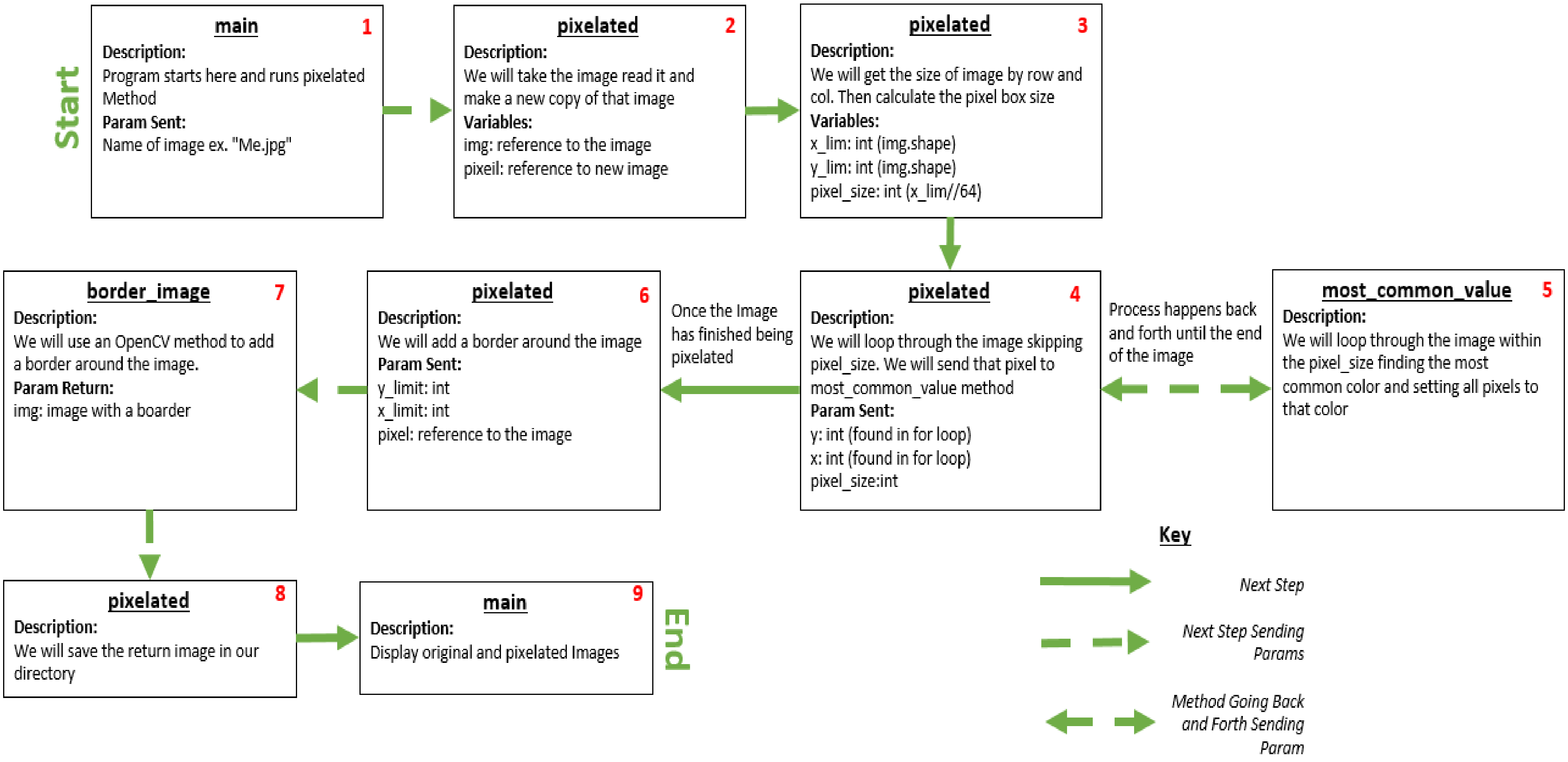
Original Pixelated

Source:

https://www.polygon.com/2015/5/13/8595963/a-pixel-artist-renounces-pixel-art

https://en.wikipedia.org/wiki/Dither

Work Flow:



Start

main 1

Description:
Program starts here and runs pixelated Method
Param Sent:
Name of image ex. "Me.jpg"

pixelated 2

Description:
We will take the image read it and make a new copy of that image
Variables:
img: reference to the image
pixeil: reference to new image

pixelated 3

Description:
We will get the size of image by row and col. Then calculate the pixel box size
Variables:
x_lim: int (img.shape)
y_lim: int (img.shape)
pixel_size: int (x_lim//64)

pixelated 4

Description:
We will loop through the image skipping pixel_size. We will send that pixel to most_common_value method
Param Sent:
y: int (found in for loop)
x: int (found in for loop)
pixel_size:int

Process happens back and forth until the end of the image

most_common_value 5

Description:
We will loop through the image within the pixel_size finding the most common color and setting all pixels to that color

Once the Image has finished being pixelated

pixelated 6

Description:
We will add a border around the image
Param Sent:
y_limit: int
x_limit: int
pixel: reference to the image

border_image 7

Description:
We will use an OpenCV method to add a border around the image.
Param Return:
img: image with a boarder

pixelated 8

Description:
We will save the return image in our directory

main 9

Description:
Display original and pixelated Images

End

Key

Next Step

Next Step Sending Params

Method Going Back and Forth Sending Param