Constructing an Image Mosaic With Digital Image Processing

Emma Bradley, Ashley Kim, Isabella Rodriguez, and Cate Whitehouse

***Abstract*—We describe the implementation of a procedure to create a photo mosaic through MATLAB code by utilizing image processing features. The procedure began with obtaining two sets of images; in our case we used a set of 10 full-size images related to UCLA and a set of 500 random images with various color scopes and intensity values obtained from the Unsplash website. We utilized the first set of images as input images to be converted into mosaics and the second set of images as photo tiles that were placed at specific locations that matched color and spatial features. Our code then converted all of the input images into accurate photo mosaics.**

***Index Terms*—Image Mosaic, Tiling, RGB Intensity**

I. Introduction

IMAGE MOSAICING, in the context of image processing, refers to the process of replacing an input image with a specific combination of multiple tile images such that the display of the output image, a mosaic of the tile images, accurately resembles the original image when viewed from a far distance. To achieve this result, the procedure uses comparison, scaling, and averaging manipulations of the input image and tiling images.

The goal of this project is to enable image mosaicing through MATLAB script. To accomplish this, we employed our knowledge from the lectures on the colored images and image transformations, such as image scaling and arithmetic computations, based on intensity values. Moreover, we conducted individual research on the process of image mosaicing to fully understand each step of this image processing. Combining this knowledge, the script that we generated followed the following steps:

1) Read in one input image to tile and a set of tile images to be used to construct a mosaic of the input image.
2) Resize all tile images in the set so that they are of identical size.
3) Resize the input image so that the tile images fit evenly into the mosaic.
4) Divide the input image into tiles of equal area such that each tile can be replaced by images from the tile image set based on color intensity.
5) Compute the average RGB intensity values of each tile in the input image as well as each tile image in the set.
6) For each tile in the input image, find the tile image with the most similar RGB intensity values.
7) Ultimately, create the mosaic.

By writing a script that followed these steps, we were able to successfully generate image mosaics with varying input images. When viewed from a distance, each mosaic accurately resembles the original image's color divisions and intensities.

II. Background

The image mosaicing process has been implemented by computer scientists using compuer software or code segments for decades. For instance, Joseph Francis is known as the modern day generator of photomosaics [1]. Francis specifically developed a software program that can convert a single image into a mosaic given sufficient tiling images. One of his projects called Live From Bell Labs, which was generated in 1993, effectively shows how an input image that contains a human face is converted into an image divided into numerous tiles in which each tile representing different images is accurately matched based on color and intensity. Furthermore, Robert Silvers is known for developing a programming algorithm that produces photo mosaics and further received a patent for his algorithm [4].

Our project builds upon previously implemented methods [3] of image mosaicing by writing an original MATLAB script to create a photo mosaic. While this project is not the first to implement an algorithm to create a photo mosaic, we have combined multiple sources in order to showcase our own understanding of the image mosaicing process.

III. Methods

The processes we used to carry out our mosaic-construction objective is divided into seven subsections, each of which outlines the major steps used to implement our image mosaicing MATLAB script. The key mathematical tools used in our methods include tiling and minimizing distance using the three-dimensional distance formula.

A. Reading in Images

We first imported the primary image that we want to make a mosaic of. We elected to use a photo of the iconic Royce Hall building at UCLA. Although we ran several trials with varying primary images, the Royce Hall building mosaic will be the highlighted figure in our results. So, our anticipated output will be a compilation of images ordered in a grid such that when viewed as a whole, they will bear a resemblance to the building in the original image. We then imported the random images to be used to construct the mosaic. We compiled 500 random images from Unsplash[5], a website with free stock photos available to the public.

B. Resizing Tile Images

We ran into issues with the dimensions of the random stock photos not being consistent, so we chose to resize all of them

to be identical dimensions. In our code, the user is able to specify the tile dimensions they wish to use to construct the mosaic.

C. Resizing the Input Image

The next lines of code resized the input image so that an integer number of tiles can be used to construct the mosaic. The number of tiles across the mosaic was computed by taking the floor of the width of the input image divided by the width of a tile image (recall that all tile images are the same dimensions at this point). A similar process was performed to compute the number of tiles down in the mosaic: the floor of the height of the input image divided by the height of a tile image. The input image was subsequently resized based on these new dimensions.

D. The Tiling Process

Tiling, a method that segments areas of an image, was chosen for our process, as it is an effective tool to group pixels into more primitive regions that are more perceptually meaningful than single pixels [2]. Seen below, Fig. 1 highlights a particular example as to how pixels could be grouped in order to serve a purpose in digital imaging manipulation.

Fig. 1. An example of an image satisfying required segmentation axioms. Credit: Digital Image Processing

This tiling process is imperative to our project as it dictates the positions where our stock images are placed in relation to the original image. We constructed a grid within the input image in order to segment the image and ensured that our tiling satisfied the axioms of image segmentation. We did so by ensuring there is no overlap and accounting for all areas of the image. We then stored these tiles in a cell array - a data type in which each cell of the array can contain any data type. In the case of this project, each cell contained an image represented by its RGB intensity values in matrix form.

E. Calculating RGB Intensity Values

The next step in our project was calculating the average RGB intensity values of each newly formed tile in the original image as well those in the set of tiling images. We opted for the RGB color model [2] to measure the levels of red, green and blue pixels (utilizing a for-loop) within every individual image and then took the mean values of them, giving us an average value for the color of each respective tile. We then implemented an identical process for each of the tiles in the main input image and determined their average RGB intensity value.

F. Finding the Best Fit Tiling Image

With the average RGB intensity values of both the input image tiles and the set of tiling images stored, we could now proceed with finding the best fit tiling image for each of the input image's tiles. We did so by comparing and minimizing their average RGB intensity values using the distance formula. For each input image tile, we looped through the full set of 500 tiling images and computed the distance. Then we selected the image tile that was most similar and that minimized the distance via the three-dimensional distance formula:

$$distance = \sqrt{(x_1 - x_2) + (y_1 - y_2) + (z_1 - z_2)}$$

where x_1, y_1, z_1 are the average intensities of the red, green and blue components respectively for the input image tile and x_2, y_2, z_2 are the average intensities of the red, green and blue components respectively for the tile image.

G. Constructing the Mosaic

Lastly, after finding the best fit image for each tile on the image and saving this information into our cell array, we constructed our mosaic image. Using the cell2mat function, we concatenated the images into our singular final image.

IV. Results

As discussed, we chose a photo of the well-known Royce Hall at UCLA to be our input image in this project (Fig. 2).

Fig. 2. A photo of Royce Hall, one of our chosen input images.

Upon examining our initial results, one thing we realized we had to be mindful of while writing the script was choosing the

correct tile size. As shown in Figure 3, the 45x50 tiles proved to be too large. While the general shapes and colors resemble those of the original image of Royce Hall, the output image didn't quite have the clarity we had hoped for.

Fig. 3. One result of image mosaicing with large tiles–45x30.

With this in mind, we made an adjustment to our code so that the tiles would be smaller, with the intention of making a clearer image. The following image in Figure 4 is our final result; our adjustments to the tile size proved to be effective, as the final result now captured a sharper image, bearing a more distinct resemblance to the input image.

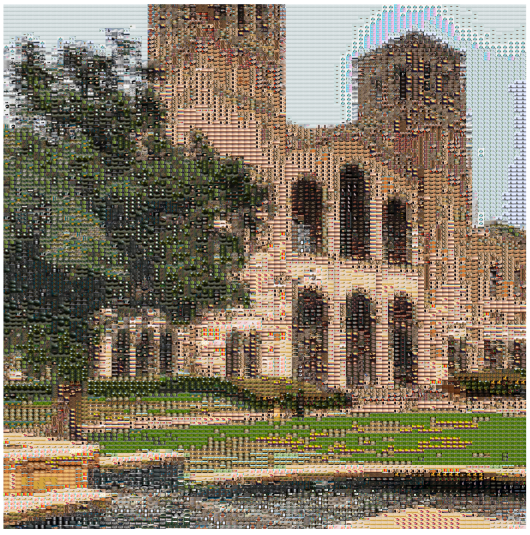

Fig. 4. The final output image.

V. Conclusion and Discussion

In conclusion, our group successfully carried out our project objectives, and in the process we gained a better understanding of the capabilities of mathematical imaging and honed our skills in MATLAB. Throughout the completion of the project, we overcame unexpected challenges and learned to adapt to these problems, resulting in a new accumulation of expertise within the mathematical imaging field.

In lecture, we spent time learning about RBG, HSI, and CMY color models. This project allowed us to practice our new knowledge of the RGB color model, as the image mosaic construction process requires averaging each tile's RGB intensity values. Additionally, we were able to extend our knowledge of image segmentation to the scope of this project when creating the tiles of the mosaic.

One unexpected challenge that we had to work on was the resizing of the original image as well as tile images. Before resizing the original image, the program did not run because the tile images did not fit into the specified dimensions. The tile images also had to be resized so that they would fit evenly into the input image, using any of the 500 random images that were an option. If the input image was, for example, 1000x1000 and the tile images were 30x25, there would not be a way for the tiles to evenly fit and fill the desired square image. So we resized our input image to take an integer number of tiles, and resized all 500 tiles to be of identical size. Then, with any choice of input image and no matter which tile images were used, the mosaic was constructed.

While we were able to work through all areas of challenge and deliver a successful output, there are certainly ways in which we could improve the project in the future. Here we offer a few ideas for potential future work. One of these is making the program more user-friendly. To do this, we could package it as a web app. While you are able to change input image(s) in MATLAB, a web app would allow a smoother way to change the input image and/or tile images. Another more complex project could create a final output image with varying tile sizes; either leaving tile images in their original size and use them, or potentially using some sort of optimization algorithm to calculate how to crop or scale them to best fit the input image. A final idea for an improved project is filtering the original image before creating the image mosaic; in lecture we learned a number of methods of how to improve an image's appearance including noise reduction, smoothing and sharpening, and color balancing (to name a few). We could include code to modify and correct the original image as needed before creating the end result mosaic.

VI. Code Repository

https://github.com/emmakbradley/155FinalProject

Acknowledgements

We would like to thank Professor Tymochko for her support on this project as well as throughout the quarter in MATH 155.

VII. Group Contributions

Emma Bradley: Code; Methods section

Ashley Kim: Code; Abstract, Introduction, and Background sections

Isabella Rodriguez: Methods and Results sections

Cate Whitehouse: Conclusion section; Writeup and Presentation editing

REFERENCES

[1] Joseph Francis. History of photo mosaics, Jan 2017.
[2] Rafael C. Gonzalez and Richard E. Woods. Digital image processing, 4th ed. 2018.
[3] Mdcanham. Matlab photo mosaic creator. github.com/mdcanham/matlab-photo-mosaic-creator/blob/master/CreateMosaic.m, Sep 2014.
[4] Robert Silvers. Photomosaics: Bio. https://www.photomosaic.com/about/.
[5] Unsplash. Beautiful free images and pictures. unsplash.com/, June 2023.