Image Mosaic

1st Anthony Salim
Electrical Engineering
VUB
Brussels, Belgium
Anthony.Salim@vub.be

2nd Bao Han
Electrical Engineering
VUB
Brussels, Belgium
han.bao@vub.be

3rd Mouhi Al Din Mahmalji
Electrical Engineering
VUB
Brussels, Belgium
Mouhi.Al.Din.Mahmalji@vub.be

4thTekalegn Mezemr Yihune
Electrical Engineering
VUB
Brussels, Belgium
Tekalegn.Mezemr.Yihune@vub.be

***Abstract*—This paper proposes a framework for an image mosaic process. During the work, the target image is divided into a set of small tiles, and then two ways of choosing the tiles are presented, with or without color correlation. Moreover, in order to make the mosaic image more coherent and visually pleasing, the impact of using different color-averaging techniques to replace the small tiles with appropriate images is shown. Finally, a color correction technique is applied to improve the quality of the final mosaic image.**

I. INTRODUCTION

Image mosaic is a technique used to create a single image by combining multiple smaller ones, often with the aim of creating a larger, more detailed picture. In this project, the implemented image mosaic method in MATLAB is done by using a collection of images and a target image. The proposed method first divides the target image into small rectangular sections and matches each section to an appropriate image from the collection. This matching process is based on the color features of the images, and a similarity metric is used to find the best match for each section. The final image is then created by replacing each section of the target image with the corresponding matched image from the collection. The implemented method was tested on a variety of target images and the results show that it can effectively create image mosaics that resemble the original target image while incorporating the characteristics of the matched images. The proposed method can be used in a variety of applications, including image editing, graphic design, and art.

II. ALGORITHM DESCRIPTION

A. Creating Tiles for Image Mosaic

In order to create a mosaic image, we have to first divide the target image into a series of small tiles shown as Fig. 1.

For simplicity, we assume that each tile has a fixed length and width. However, one of the difficulties is that the length and width of the target image are unknown, which means that it is impossible to always divide the target image into an integer number of tiles. Therefore, a special algorithm is implemented to solve this problem. We first assume that the target image is always divided into a fixed number (100) small tiles horizontally, and the size of the original image is $M \times N$, then the following equation can be created:

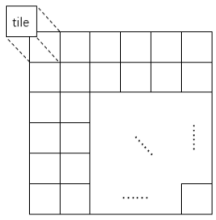

Fig. 1. Tiles

$$\frac{100}{P} = \frac{N}{M}$$

where P denotes the number of vertical tiles. After obtaining the value of p and assuming that the size of the tile image is $A \times B$, then the size of the final processed image will be

$$(P \times A) \times (100 \times B)$$

Moreover, it is worth mentioning, that rounding approaches are adopted in order to always obtain an integer number of tiles and pixels

B. Selecting Tiles

1) Color correlated tiles: To represent the target image with different tiles, the color matching between them has to be done.Which means image pixels must directly be used [3].The first step is calculating the average color of each tile of the target image, and saving them in a 3-D matrix $array$, the first matrix of $array$ saves the average value of red color and the second saves the average value of green while the third saves the blue, as shown in Fig. 2

Then in order to calculate the average color of the tile image, we create another matrix called avg matrix, whose size is $n \times 3$. Each column of this matrix saves the average value of red, green, and blue, respectively. The matrix can be expressed as below:

$$avg = \begin{bmatrix} \bar{R}_{image1} & \bar{G}_{image1} & \bar{B}_{image1} \\ \bar{R}_{image2} & \bar{G}_{image2} & \bar{B}_{image2} \\ \vdots & \vdots & \vdots \\ \bar{R}_{imagen} & \bar{G}_{imagen} & \bar{B}_{imagen} \end{bmatrix}$$

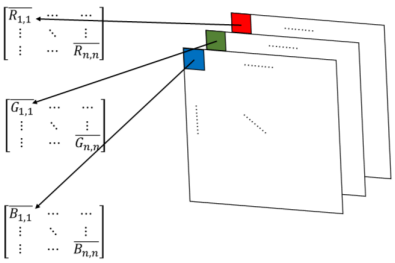

Fig. 2. RGB Matrix

The next step is to find the best-matched image from the database and to implement the color matching, the $L2-norm$ will be used, which is shown in Fig. 3. In this RGB space, the original point in the coordinate is the average RGB value of the target tile, and the blue points are the average RGB value of the images in the database.

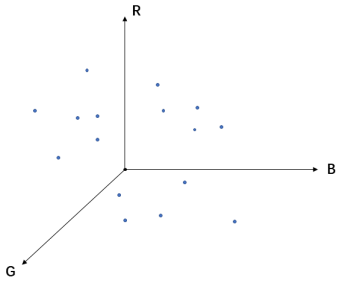

Fig. 3. L2-norm

In order to find the best-matched image, we only need to find a blue point that is closest to the original point, i.e. get the minimum Euclidean distance:

$$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

In this case, we can ensure that the image that we find has a similar color to the target tiles.

Based on the comparison on [1], Apart from using Euclidean distance for color matching, we can use Manhattan distance, which is known as

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n}|p_i - q_i|$$

or Chebyshev distance, which is known as

$$d(\mathbf{p},\mathbf{q}) = \max_{i=1}^{n}|p_i - q_i|$$

For Manhattan distance, it considers the absolute differences between color vectors and calculates the distance by summing the vertical line segments along the coordinate axes. Therefore, it ignores the correlations and directions between colors, focusing only on the differences between dimensions, which may result in some loss of information. However, the good side of using the Manhattan distance instead of the Euclidean distance is that the Manhattan distance is not sensitive to outliers in color space since even if there are outliers, their contribution to the overall distance is limited, and they do not excessively affect the computed distance by calculating the absolute differences between color vectors in each dimension. Also, it is suitable for handling color vectors with discrete or distinctly segmented features.

As for the Chebyshev distance, it considers the maximum dimensional difference between color vectors, i.e., the maximum coordinate axis difference. Since it only focuses on the maximum difference, it may overlook important information. For example, if two color vectors have small but correlated differences in one or more dimensions, the Chebyshev distance cannot capture this correlation. However, it can better capture boundary cases between colors and is not sensitive to outliers. The result of different distance is shown in Fig.4

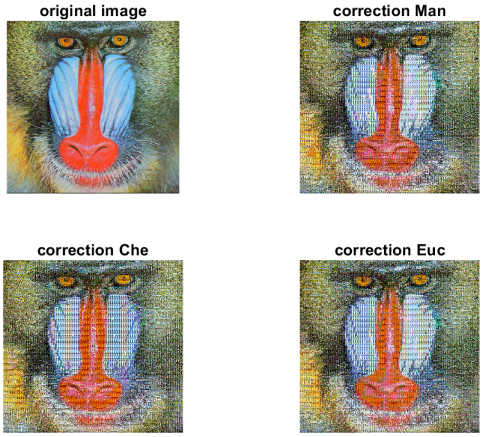

Fig. 4. Color Correlation

2) Random tiles: So far, we have implemented the image mosaic by using a fixed and pre-defined size of tiles and choosing the best image from the set that fits a specific tile depending on one of the three different color-averaging methods. However, by applying the previous methods it is possible to use one image for a number of tiles, also some images may not appear at all in the final mosaic image due to their average color. The random tiles scheme will ensure that the images from that data set will be randomly distributed on the tiles. In Figure 5 we can see that more images from the database are used instead of some of them. However, it is obvious that the final quality of the image mosaic decreases a lot.

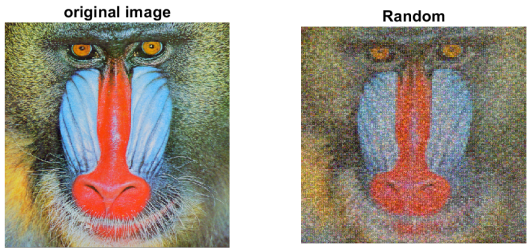

Fig. 5. Random Tiles

C. Color Correction

By finishing Step A and Step B, we have already derived the mosaic image with a low quality, seen as Fig. 6

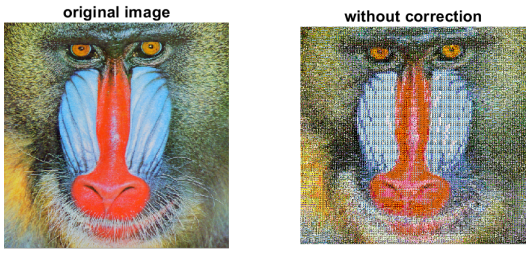

Fig. 6. mosaic without color correction

To further improve the quality of the mosaic image, color correction can be done. we make use of a correction rule, which takes as input an image tile and a desired average color a, and generates a correction function that maps a color x in the image tile to a color F(x) in the final mosaic such that the region of the mosaic covered by the image tile will have the average color a [2] . And one of the easiest ways to achieve this is to perform the opacity operation shown in Fig. 7.

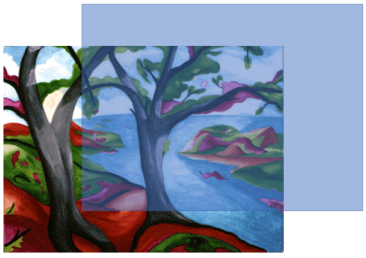

Fig. 7. opacity

In this step, we adjust the ratio between the original image and the mask via factor α :

$$Output = \alpha \times input + (1 - \alpha) * mask$$

where the mask is the main RGB color of the corresponding tiles. The result is shown in Fig.8

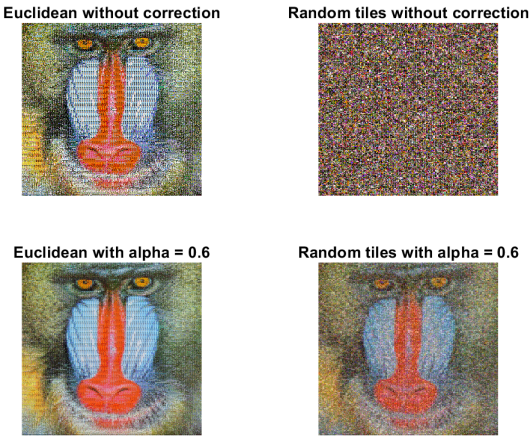

Fig. 8. Color Correction

Furthermore, we can change the α values to adjust the strength of correction, where $\alpha = 0$ represents zero correction as shown in Fig.9

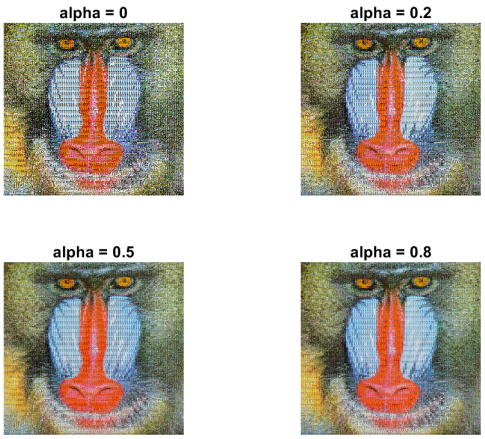

Fig. 9. Different α

D. Pseudo Step of Image Mosaic

In this section, the pseudo step of image mosaic is illustrated:

- Inputs:
  - Target image
  - Set of collected images(tiles)
- Output:
  - Mosaic image
- Step 1: Read the tile images from a directory.
  - Set the directory path where the tile images are stored.
  - Read all the images from the directory using the *dir* function.

  - Loop through each image file, read it in using the *imread* function, and store it in a cell array.
- Step 2: Calculate the color average of each tile image.
  - Loop through each tile image in the cell array.
  - Calculate the average color of the image using the *mean2* function and store it in a matrix.
- Step 3: Divide the target image into cells.
  - Determine the cell size.
  - Divide the target image into non-overlapping cells of the determined size.
- Step 4: Calculate the color average of each cell.
  - Loop through each cell in the target image cell array
  - Calculate the average color of the cell using the *mean2* function and Store the average color in a matrix.
- Step 5: Find the collected image with the closest color average for each tile.
  - Loop through each cell in the target image cell array.
  - Loop through each tile image in the collected image cell array.
  - Calculate the color difference between the average color of a cell and all tile images and keep the difference in a matrix.
  - Keep track of the collected image with the smallest color difference.
- Step 6: Replace each target tile with the corresponding tile image.
  - Loop through each cell in the target image cell array.
  - Replace the cell with the correct tile.
- Step 7: Combine tiles to create the mosaic image.
  - Combine the modified tiles in the correct order to create the final mosaic image using the cell2mat function.
  - Do the color correction according to the original image.
  - Save the mosaic image to a file using the *imwrite* function.

III. EXPERIMENTAL RESULTS

By now the image mosaic has been successfully implemented and three different ways of fulfilling the color correlation are illustrated, which are known as Euclidean distance, Manhattan distance, and Chebyshev distance, respectively. The result of color matching is shown in Fig.4.

When implementing the Euclidean distance for color correlation, it can consider the differences between each dimension of the color vectors and calculates the Euclidean distance between them. However, it may be sensitive to outliers in color space. For Manhattan distance and Chebyshev distance, they may not be affected by the outliers but they will cause some information loss at the same time.

Then the color correction is implemented, here we use the original tiles as masks to achieve the opacity effect. The result of color correction is shown in Fig.8.

Moreover, we can see the differences in our results when using different α in Fig.9.

After finishing all of those functions, we further integrate them into a GUI shown in Fig.10 so that the users can change the corresponding parameters according to their requirements.

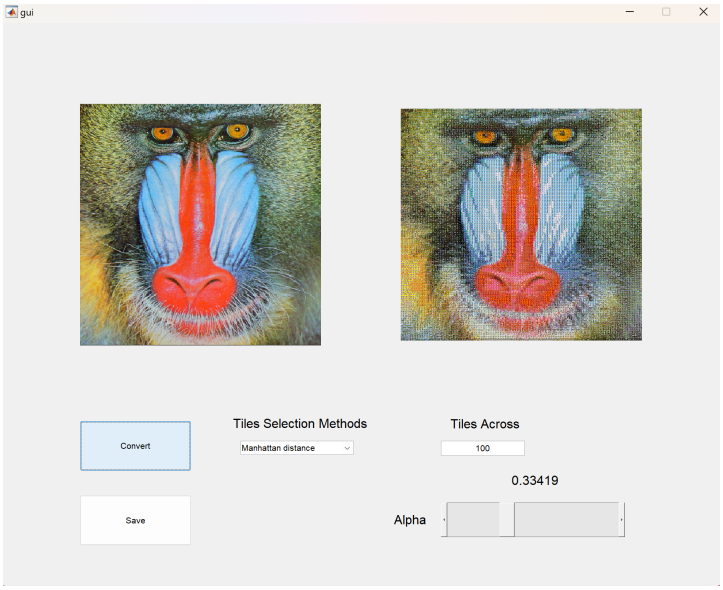

Fig. 10. GUI

IV. CONCLUSIONS

In this project, a framework for image mosaic processing was proposed and implemented in MATLAB. The process involved dividing the target image into small tiles, performing color matching between the tiles and a collection of images, and applying color correction techniques to enhance the quality of the final mosaic image.

The results of the implemented method demonstrated its effectiveness in creating image mosaics that closely resemble the original target image while incorporating the characteristics of the matched images. By dividing the target image into small rectangular sections and matching each section to an appropriate image from the collection based on color features, a visually coherent and pleasing mosaic image was achieved.

The importance of the color correlation process is shown where only a small part of the images in the database are used to build a 'higher quality' mosaic image. Moreover, another approach is applied to use all the images in the database to build a 'lower quality' mosaic image. Furthermore, several methods of calculating the correlation are illustrated, which are known as Euclidean distance, Manhattan distance, and Chebyshev distance, respectively.

After that, the color correction process is implemented, too. We use α to adjust the strength of color correction. All of these functions are integrated into a GUI so that the users can adjust the parameters according to their requirements.

In conclusion, the proposed framework for image mosaic processing presents a comprehensive method to generate visually pleasing mosaic images. The combination of tile division, color correlation, and color correction techniques contributes to the creation of high-quality mosaics that retain the characteristics of the original target image.

REFERENCES

[1] Malkauthekar, M.D., 2013, October. Analysis of Euclidean distance and Manhattan distance measure in Face recognition. In Third International Conference on Computational Intelligence and Information Technology (CIIT 2013) (pp. 503-507). IET.

[2] Finkelstein, Adam, and Marisa Range. "Image mosaics." Electronic Publishing, Artistic Imaging, and Digital Typography: 7th International Conference on Electronic Publishing, EP'98 Held Jointly with the 4th International Conference on Raster Imaging and Digital Typography, RIDT'98 St. Malo, France, March 30–April 3, 1998 Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[3] Barnea, Daniel I., and Harvey F. Silverman. "A class of algorithms for fast digital image registration." IEEE transactions on Computers 100.2 (1972): 179-186.and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

APPENDIX

A. Applied Matlab code

- https://github.com/XIAOBAOnpu/Image-Mosaic

B. Workload

Anthony, Bao, Mouhi Al Din, and Tekalegn worked together on understanding the algorithm, applying it, and doing literature research.