Mosaic Image

TNM097 - Image Reproduction and Image Quality

Lars Bergman[*]

June, 2017

Abstract

This report includes a brief summary of the photographic mosaic technique and how it was applied to reproduce a input image with the help of a image database. The application consists of creating the image database, using euclidean distance for matching, furthermore a error diffusion was implemented.

1. Introduction

This paper will cover how the image mosaic was implemented using matlab, the result and a discussion. The strengths and weaknesses of the implemented method will be discussed. The goal with this project was to make an application that could reproduce a good image. The "good" here is dependent on the run time and how close it can resemble the input image.

The program takes a input image and then recreates the image with images from the given database. The image seen in Figure 1 below was the one mostly used, however any image could be used.

2. Implementation

The implementation consists of mainly two scripts. The first script is to create the database and the second one is to create the mosaic image. In section 2.1 and section 2.2 a brief explanation of the two scripts are given.

[*]Student in Media Technology at Linköpings University, Sweden, Campus Norrköping,
email: larbe444@student.liu.se

Figure 1: *The input image used the most during development, size of 640x640.*

2.1. The Database

The database ended up with 21 images which were all scaled and cropped to a certain patch size. This patch size was tested and will be described further down in the discussion section of this report. The patches of each corresponding images were stored in a array as RGB as well as the mean value of the XYZ in another array.

2.2. Creating the Mosaic

Creating the mosaic image was done by looping through each row and column in the scaled input image to get each pixel value. Every pixel value in the input image was then compared to all images in the database. The matching was then made by taking the image with the closest euclidean distance to replace that pixel. This comparison was made in the CIELAB color space for both the pixel and the database patches.

To further enhance the visual resemblance of the mosaic image, error diffusion was implemented. Error diffusion takes the difference between the mean XYZ value for the pixel and the patch and then spreads that error to adjacent pixels that has not yet been processed. By doing this for each channel of the pixel a greater result will be produced. This was accomplished with the "normal" weights discovered by Floyd and Steinberg.

3. Result

The result of the image mosaic implementation can be seen in the figures below. Figure 2 shows the implementation without error diffusion with a downscale to 0.2 of the original size.

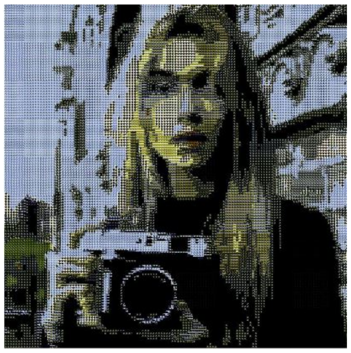

Figure 2: *Downscale to 1/5 of the original image, without error diffusion applied.*

To see the difference between the downscaling Figure 3 shows the same image but only downscaled to half of the input image. This results to a larger image and longer run time but it recreates the image better.

Figure 4 shows the implementation with error diffusion applied and a downscale to 0.2 of the original.

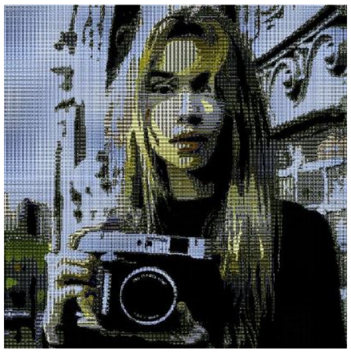

Figure 3: *Downscale to 1/2 of the original image, without error diffusion applied.*

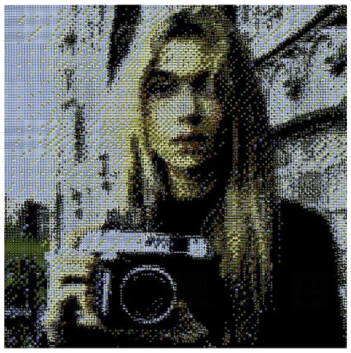

Figure 4: *Downscale to 1/5 of the original image, error diffusion applied.*

Figure 5 shows the implementation with error diffusion applied and a downscale to 0.5 of the original.

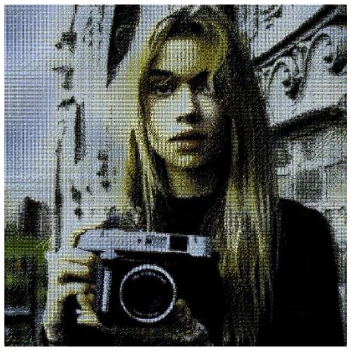

Figure 5: *Downscale to 1/2 of the original image, with error diffusion applied.*

In Figure 6 a zoom into the output mosaic image can be seen, here it is easy to see the small database images that makes up the image.

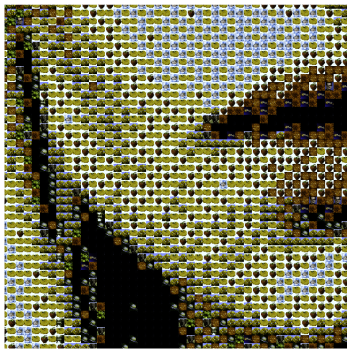

Figure 6: *A zoom in to the image to see that the image is actually made of smaller patches from the database.*

4. Discussion

The mosaic images created by the application resembles the input image fairly good most of the time. However if the input image is really bright and the database lacks bright images it will probably not look a lot alike. Nevertheless this is a problem that can easily be avoided by choosing a good variety of images for the database and having more images in it. That being said the more images in the database will require more comparisons resulting in a longer run time.

An improvement could be to try an optimize the database such that it will rate each image in the database based on the number of times it is chosen.

Another thing that increases the run time drastically is the chosen patch size and the down sampling of the input image. Down sampling the image to about 1/5 of the original size for a 640x640 image and having the patch size set to 15x15 was considered to be appropriate. This was good due to the fact that the run time was about one minute and the result was promising.

The error diffusion that was implemented really improved the visuals of the output image since it got rid of the homogeneous areas which can be seen by comparing Figure 3 with Figure 5.

Overall the result is promising considering both the run time of the application and how close it can resemble the input image. For further work the database could be improved as well as some other techniques such as lighting compensation.