University of Manchester

School of Computer Science

Third Year Project Report – April 2018

Image Mosaics

BSc in Computer Science

Author: Qingshuai Feng

Supervisor: Jim Miles

Abstract

Image mosaics are a kind of art as well as a kind of technique. They are the images composed by thousands of small images according to the colour information. This report presents algorithms to generate equal-size block image mosaics and unequal-size block ones. The algorithms contain three main steps, initialisation, matching and combination. Feature used for matching is extracted from library images while the master image is cut into blocks during initializing. Both algorithms share the same matching method with a repetition-avoid mechanism. The feature used for matching is an 8*8 fingerprint under L*a*b* colour mode and the difference between images is calculated with CIE94 formulas. Source image overlay is provided to achieve a better effect. A data structure is designed to store the distribution of various-size blocks for unequal-size block part. The desired effects of result images prove the correctness and success of the algorithms.

Acknowledgements

I would like to thank my project supervisor, Jim Miles for his encouragement and inspiration of the project as well as his enthusiastic care and help for my study.

I am also grateful to my parents for supporting my life and study.

I would also like to thank my girlfriend, Monica for her help with testing and evaluating my project.

Thank you all.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Like many forms of art, mosaic as a very ancient visual art has a very long history and can be traced back to 3rd millennium BC in Ancient Greece and Ancient Rome [1]. At the beginning, the mosaics are made artificially from stone or glass tiles in small approximate squares or other roughly identical shapes [1]. They are used to compose pictures as a floor or a wall in the religious field. Figure 1 shows a Greek mosaic from a Palace in Greece [2].

Figure 1 A Greek mosaic of a nymph riding on a creature, 2nd century BC [2]

In the 1990s, with the development of technology and the innovation of art, the tiles of traditional mosaics are replaced by equal-size rectangular images. Photographic mosaic, also known as image mosaic, is such an image that has been cut into thousands of tiles and each of them is replaced by an image according to the colour information [3]. It shows a whole large image when watching from a distance while you can see thousands of small images when watching at close range [3]. This form of art has been developed many different types for now. Such as, unequal-size tile image mosaics, irregular-shape tile image mosaics or even video mosaics. For the first versions, the result images are believed to be made artificially piece by piece. One of the famous works is the poster of the movie, The Truman Show, which was released in 1998. The poster is the head of protagonist which consists of thousands of the stills from the movie as shown in Figure 2 [4]. In 1995, Robert Silvers, the CEO of Runaway Technology company, developed the first

algorithm to create a photomosaic and got the patent for it two years later [3].

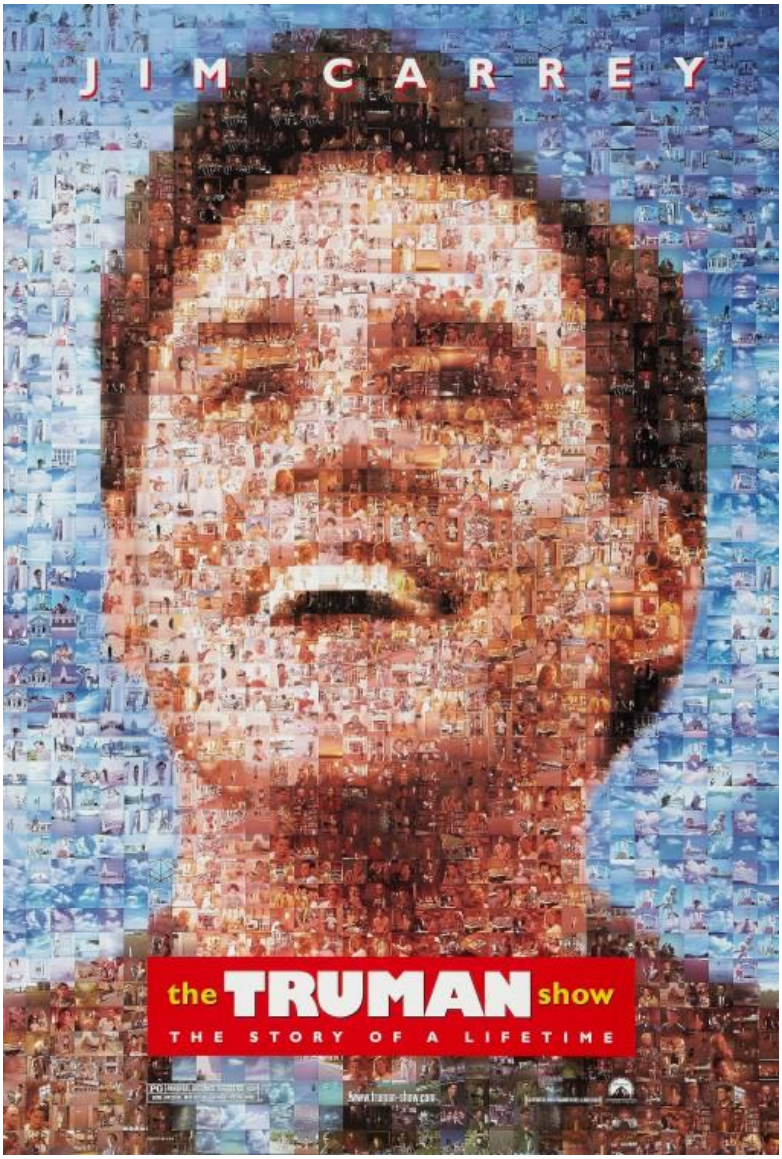


Figure 2 The poster of The Truman Show [4]

1.1 Project aims

This project is an innovative one whose goal is to build a generator for creating image mosaics. There are three main aspects to be considered for this project. The first one is the “quality” which means how similar can the result image and original image be. The second one is the “performance” which refers to the speed of the program and the consumption of the hardware resources. The last one is the “innovation” which refers to the novel approaches to the mosaic composition. Except for the first two aspects, the last one should be

decided before starting the project. For the standard approaches, it should at least generate the equal-size rectangular tile image mosaics. With respect to the novel approaches, the plan is to generate unequal-size rectangular tile image mosaics which can emphasize the foreground from the background in the image. The main target is to generate an aesthetic image mosaic which has a good rendering and mosaic effect but does not lose the profile of the original image. Therefore, plenty of problems need to be considered for this project.

1.2 Application area

Image mosaics can be used in entertainment, publicity, commemoration and other fields. A poster of a movie or an event created with image mosaic may look shocking and impressive. The author can use the stills from the movie or the pictures from previous events as the image database to compose a large poster image. Audience can enjoy the creative poster while they can learn more about the movie or the event when watching the small tile images. In addition to this, it is also a good choice to use image mosaics in ceremonies. Groom and Bride can create an image mosaic of their wedding photo by their own photos. Graduates can use their graduation photos to compose a class or school logo in image mosaic. Each tile is a piece of memory or one participant which is very memorable.

Chapter 2 Background and Context

This project covers the fields of computer vision and image processing. Most parts of the project demand plenty of manipulations to images and the analysis of them to create the final mosaic effect. With the help of more and much more functional tools of computer vision and image processing, many powerful and convenient functions are created for designers to use in their projects. And computers can achieve a more accurate mathematical and digital representation of human vision as the development of the algorithms and the hardware.

With more technical support, more algorithms have been developed nowadays to generate image mosaics. However, there is not a standard one and they are various as different authors. Every existing algorithm shares the similar steps, but the difference lies in what features and technologies are used. Before introducing the algorithms, it is necessary to define some terminologies. As the Figure 3 shows, the image that is prepared to be created to image mosaic is called *master image*. Then the master image will be divided into thousands of small blocks, each of them is called a *block* or a *tile*. The images that will be used as tiles to compose the master image are called *library images*. Thus, the user will provide the master image and library images and the program will output a result image.

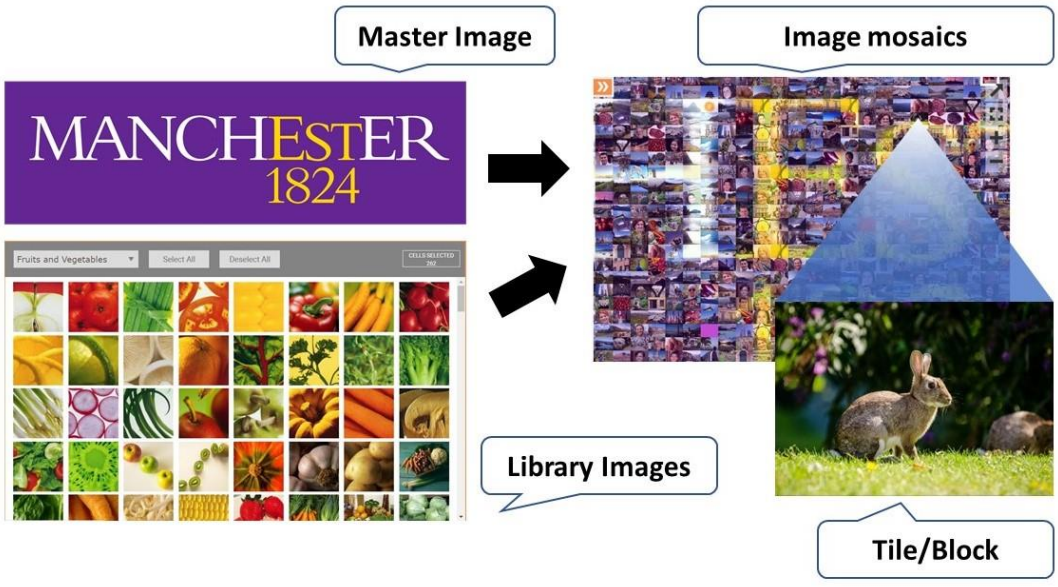


Figure 3 Terminologies of image mosaics

2.1 Algorithm to generate equal-size block image mosaics

Most algorithms that can be found consist of two main sections, initialisation and matching. During the initialisation process, master image and library images will be initialised in parallel. Master image is cut into thousands of equal-size blocks followed by extracting the features from each of them, normally about the colour, for the further matching. Meanwhile, the same features are extracted from each of the library images as well. During the matching process, each block from the master image will be matched with and replaced by a library image using the features extracted. Then the result image is generated by copying the chosen library images in order. The core of the algorithm is the features extracted from images. This chapter will also discuss the developed features and the background information to be understood of my project.

2.2 Algorithm to generate unequal-size block image mosaics

There is currently no published algorithm to generate unequal-size block image mosaics. But some paid websites provide such image generation service [6]. The matching process should be the same as the equal-size part, but the difference is how to decide the size of blocks of the master image. Since the target of unequal-size part is to emphasize the foreground from the background. In this chapter, a function of extracting foreground of an image will be introduced. And in the development chapter, the details of its design and implementation will be discussed.

2.3 Colour mode and colour difference

Human colour vision can be represented by a mathematical way on computer. This can be realized by combining several channels and the colour mode determines such combination [7]. Different colour modes have different colour ranges and various applications. These are the basic and crucial for understanding the significance and usefulness of the features used for matching.

2.3.1 HSV colour mode

HSV colour mode consists of three channels, hue, saturation and value. The colour space is shown in Figure 4 [8] where the hue channel has the range from 0 to 360, saturation and value have the range from 0 to 1 [9]. The colour can be expressed by only using hue channel and the full boundaries of colours are shown in Table 1. Thus, only one channel is enough to represent colours. However, black, gray and white are still missing. The other two channels, saturation and value, can fill this vacancy. The higher the saturation, the purer the colour. And the colour becomes white when saturation values 0. Value channel is also known as the brightness and when it values 0, the colour is black [9].

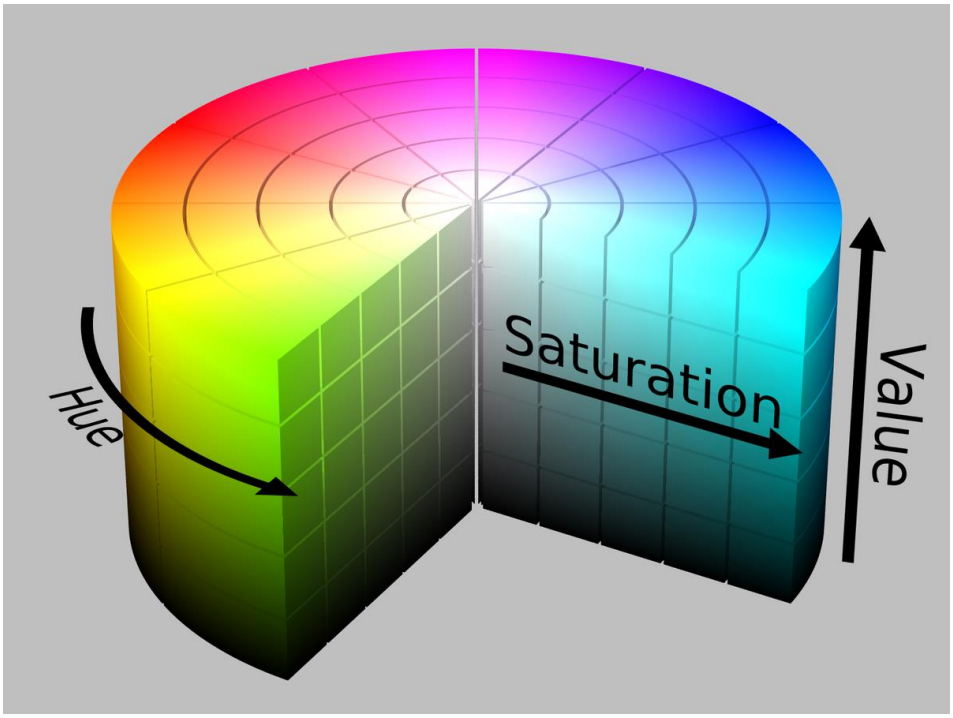


Figure 4 HSV colour mode represented in a cylinder [8]

Colour Boundaries			Grayscale Boundaries				
H value lower limit	H value upper limit	Colour	S value lower limit	S value upper limit	V value lower limit	V value lower limit	Colour
311	360	red	0	100	0	18	black
0	20	red	0	17	18	86	gray
21	50	orange	0	17	80	100	white
51	68	yellow	When H takes any value				
69	154	green					
155	198	cyan					
199	248	blue					
249	310	purple					
When S is in [17,100], V is in [18,100]							

Table 1 Boundaries of colours and grayscales in HSV colour mode

2.3.2 L*a*b* colour mode and colour difference

L*a*b* colour mode, also called CIELAB here, is designed to be much closer to human perception. The “L” channel represents the brightness of pixels in range between 0 and 100 corresponding to the brightness from black to white. The “a” and “b” channels have the value from 127 to -128 which represents the colour from red to green and the colour from yellow to blue respectively. This colour mode has a much larger colour space than computer displays or even human vision [7].

Human can tell the difference between different colours and judge how similar they are. This difference, also called metric, can be quantified and calculated under L*C*h* colour mode calculated from L*a*b* colour mode by formulas defined by International Commission on Illumination (CIE). The CIE formulates their standard of colour distance as ΔE also called “Delta E” [10] [11] [12]. Three editions of formulas have been published so far and each one improved and fixed some errors than its previous edition. The earliest one is CIE76 which use L*a*b* colour mode and just calculate the Euclidean distance between two colours which is not good enough. The latest standard is CIE2000 but there are too many complicated calculations which may influence the performance. The middle edition, CIE94, even has small errors in some cases. However, it performs well enough in using a real number to tell the difference between colours. The following formulas shown in Figure 5 explain how the distance is calculated [10] [11] [12].

$$\Delta E_{94}^{*} = \sqrt{\left(\frac{\Delta L^{*}}{k_L S_L}\right)^2 + \left(\frac{\Delta C_{ab}^{*}}{k_C S_C}\right)^2 + \left(\frac{\Delta H_{ab}^{*}}{k_H S_H}\right)^2}$$

$$\Delta L^{*} = L_1^{*} - L_2^{*}$$
$$C_1^{*} = \sqrt{a_1^{*2} + b_1^{*2}}$$
$$C_2^{*} = \sqrt{a_2^{*2} + b_2^{*2}}$$
$$\Delta C_{ab}^{*} = C_1^{*} - C_2^{*}$$
$$\Delta H_{ab}^{*} = \sqrt{\Delta E_{ab}^{*\,2} - \Delta L^{*2} - \Delta C_{ab}^{*\,2}} = \sqrt{\Delta a^{*2} + \Delta b^{*2} - \Delta C_{ab}^{*\,2}}$$
$$\Delta a^{*} = a_1^{*} - a_2^{*}$$
$$\Delta b^{*} = b_1^{*} - b_2^{*}$$
$$S_L = 1$$
$$S_C = 1 + K_1 C_1^{*}$$
$$S_H = 1 + K_2 C_1^{*}$$

Figure 5 CIE94 formulas, they calculate the colour distance between two colours (L1, a1*, b1*) and (L2*, a2*, b2*), where kC, kH, kL takes 1 and K1 takes 0.045, K2 takes 0.015 here*

2.4 Colour mode conversion

Images read by computer are normally under RGB colour mode. To realise the desired implementations, these RGB values should be converted into desired colour modes.

2.4.1 RGB to HSV

Assume there is a colour in RGB is (r, g, b) where r, g and b are between 0 and 1. The “V” channel in HSV is the maximum among r, g, b. And the “H” and “S” channels are calculated as shown in Figure 6. In this figure, max means the maximum among r, g and b while min means the minimum. The “H” value in the result is equal and bigger than 0 and smaller than 360 while both “S” and “V” have the value from 0 to 1 [13].

$$h = \begin{cases} 0^\circ & \text{if } max = min \\ 60^\circ \times \frac{g-b}{max-min} + 0^\circ, & \text{if } max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{max-min} + 360^\circ, & \text{if } max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{max-min} + 120^\circ, & \text{if } max = g \\ 60^\circ \times \frac{r-g}{max-min} + 240^\circ, & \text{if } max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } max = 0 \\ \frac{max-min}{max} = 1 - \frac{min}{max}, & \text{otherwise} \end{cases}$$

Figure 6 Formulas to convert RGB colour mode to HSV colour mode [13]

2.4.2 RGB to L*a*b*

Since RGB colour mode is device-dependent which means the same RGB values may look different on different device. While L*a*b* colour mode is device-independent. There is no direct conversion between these colour modes. Another colour mode, XYZ, which is also a device-independent colour mode acts as a transfer stop between RGB and L*a*b*. RGB values will be projected to XYZ values then transferred to L*a*b* [14] [15].

Since RGB colour mode is device-dependent, a reference of the colours needs to be confirmed to convert RGB to a device-independent colour mode [15] [16]. Such reference is called reference white and it varies with different types of RGB mode. After

doing plenty of research and tests, the reference white in this project is D65 standard under sRGB colour mode. The matrix of D65 white reference and the formulas to convert RGB to XYZ are shown in Figure 7 [16].

$$D65 = \begin{matrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{matrix}$$

$$\begin{cases} R = \text{gamma}(\frac{r}{255.0}) \\ G = \text{gamma}(\frac{g}{255.0}) \\ B = \text{gamma}(\frac{b}{255.0}) \end{cases}$$

$$\text{gamma}(x) = \begin{cases} \left(\frac{x + 0.055}{1.055}\right)^{2.4} & (x > 0.04045) \\ \frac{x}{12.92} & (\text{else}) \end{cases}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figure 7 Formulas to convert RGB colour mode to XYZ colour mode [16]

The conversion from XYZ to L*a*b* has a series of normalised formulas shown in Figure 8 [14]. Due to the errors in the calculation process, the colour with the resulting L*a*b* values may be not the exact colour with the original RGB values. But this will not influence too much on the result.

$$\begin{aligned} L^* &= 116 f(Y/Y_n) - 16 \\ a^* &= 500 [f(X/X_n) - f(Y/Y_n)] \\ b^* &= 200 [f(Y/Y_n) - f(Z/Z_n)] \end{aligned}$$

$$f(t) = \begin{cases} t^{1/3} & \text{if } t > (\frac{6}{29})^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} & \text{otherwise} \end{cases}$$

Figure 8 Formulas to convert XYZ colour mode to Lab colour mode, where Xn, Yn, Zn are 95.047， 100.0， 108.883 respectively. [14]

2.5 Features

Different designers use different features from images for matching. Some are so simple that they just calculate the mean colour of the whole image and calculate the Euclidean distance between these average values [5]. This method could be the fastest one but might lead a very poor result. Another method is to compare images pixel by pixel [5]. This could achieve a good result, but it is quite resource-consuming. The most used feature is a down-sampled version of images. It is normally a 3*3 or 8*8 matrix. Each unit of the matrix is a set of RGB values which represent the average colour of the corresponding area of the image. Some of them use string matching algorithms to compare the RGB values of the matrix of each pair of images and choose the most matched one [17] [18].

Except for these features, some designers also consider smoothing the edge between each neighbour blocks. This could make the edge composed by library images smoother, but library images need to be cut to remove the undesired part. Sometimes, it is not expected by users.

2.6 Function to extract foreground from an image

To realize the unequal-size block image mosaics, the foreground needs to be extracted so that the size of blocks can be determined. Extracting objects from an image is a very broad topic and very complicated to realize. However, some build-in functions offered in OpenCV, the library used in this project, can provide a satisfactory result. The grabCut function is one of them and provides the foreground extraction with user interaction. The function is realized by using Gaussian Mixture Model of the RGB channels in an iterative way [19]. User could define a rectangle area which contains the whole foreground and then select some sample areas for foreground and background respectively. The function will only output the pixels belong to the foreground.

2.7 Expected work

With the aid of these existing technologies and functions, the desired achievements for the equal-size block image mosaics are initializing master image and library images under the suitable colour mode properly, matching blocks of master image with library images using a new method and doing some necessary operations

to make the result image better, such as avoid too many repeat selected images within a small area. For the unequal-size block part, the main desired achievement is to decide how the various-size blocks distribute and design a data structure to represent such distribution.

Chapter 3 Development

The development chapter contains two main sections. Design section is the overview of the design of the project without too many details. Implementation section discusses those interesting and difficult parts when realizing functions. After reading this chapter, readers will completely understand the whole process to generate two types of image mosaics and the methods to realize them.

3.1 Design

This section describes the design of the algorithms of two types of image mosaics. Main steps of the algorithms will be explained briefly with the aid of diagrams. Firstly, it is necessary to discuss the integrated development environment (IDE) and the tools that have been chosen.

3.1.1 Tools

3.1.1.1 IDE – Visual Studio

The environment I choose here is Visual Studio 2017, which is a powerful and functional IDE to be used to develop computer projects under Windows operating system. VS consists of code editor, debugger, designer and plenty of useful extensions [20].

The build-in code editor, Visual Studio Code, has most of the features of a common source code editor, such as syntax highlighting, autocomplete, brace matching and it has a good compatibility and synchronization with the project created under VS [20]. The debugger can satisfy most requirements of running my project. It can run a single line of the code in the main body at a time as well as each line inside a self-create function. An analyser inside the debugger which records the amount of time taken to compile and run and the usage of CPU and memory helps to evaluate the project and do optimisation.

Since a window application with user interface is preferred, Windows Form Designer is a choice to build a GUI application. It is easy and convenient to add and edit buttons, textboxes, image containers and control the layout of the window using Windows Forms which is a graphical class library provided by Microsoft .NET Framework [21]. Because of the difficulty of debugging programs

with GUI, the programs can be created as a Win32 console application first and then copy to an application under Windows Forms after all the functions have been realized. All these works can be done with Visual Studio without the help of any other software or platforms.

3.1.1.2 C++ and OpenCV

After doing some research of this project, the analysis is that the key point is to find the distribution of the colour in an image and then match images using this feature. So, a functional tool of image processing and computer vision is required to manipulate images. Most people online suggest OpenCV, a function library for computer vision. With the initial learning and the project progress, it is no doubt that OpenCV is a perfect choice since it provides a great quantity of functions which support many complicate applications in the field of computer vision. Such as, using grabCut function mentioned before to extract object or foreground from an image. Moreover, plenty of data structures are also supported by OpenCV [19]. Like Mat is used to store images and it has several other cases used to store different types of images.

OpenCV is built by C++ and this efficient object-oriented programming language is the primary interface of OpenCV [22]. Since C++ is an extension of C language. C++ offers more data structures and namespace feature which are demanded by this project [23]. In most cases, we could use less code and time to realize the same algorithm under C++.

3.1.1.3 Other tools

When starting to consider the algorithm of the project, I worried about the space taken by the library images. At least hundreds of library images should be uploaded by user and they need to be stored and wait for being matched until a result image is generated. Initially, I concerned to store the original images in memory. But it may occupy a few hundred megabytes of space. So, I tried to use database, MySQL, to store these files but finally I realized it is unworthy to do that. As an efficient method of matching images which will be discussed later is created. Only few hundreds of bytes need to be stored for each library image. As a result, just C++ file stream is enough for the storage requirement without the help of other third-party tools.

3.1.2 Design of the project

The project contains two main sections and with an additional GUI design. One of the sections is the equal-size block image mosaics while the other one is the unequal-size one. Both have three stages designed in this project which are initialisation, matching and combination, but there are differences between each same stage of different sections due to the different requirements.

3.1.2.1 Equal-size block image mosaics

The idea to realize this application is similar as those existing algorithms but with an additional combination step for further rendering. Equal-size blocks are cut from master image and then matched with and replaced by one of the library images. A diagram of all the steps of three stages to generate an equal-size block image mosaics is shown in Figure 9.

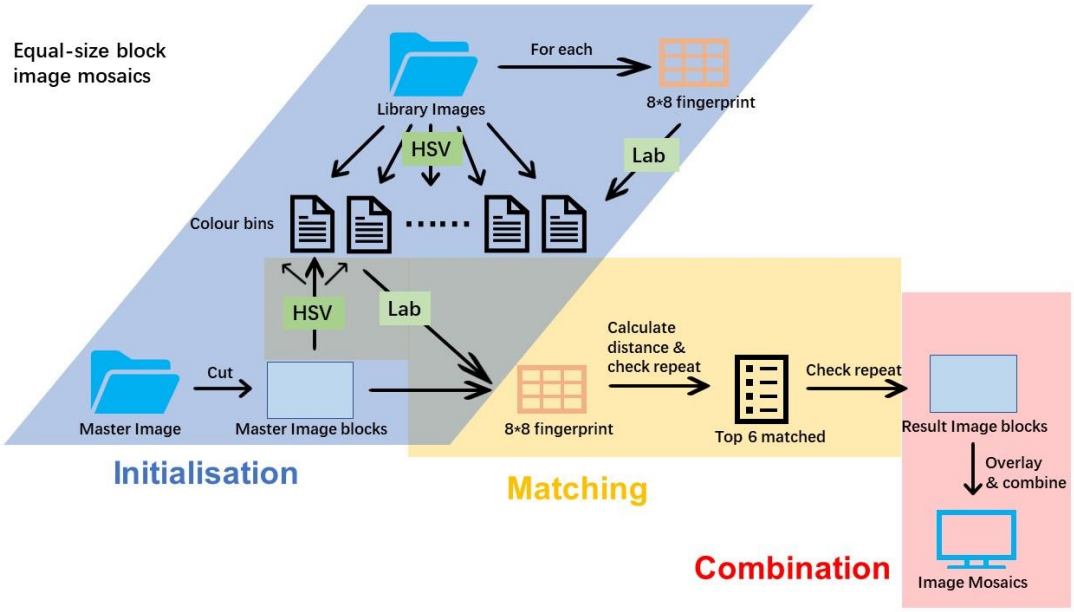


Figure 9 The algorithm of generating equal-size block image mosaics

3.1.2.1.1 Initialisation

The program starts at initializing library images and the master image. Library images are classified and stored into colour bin files according to the average colour of the whole image under HSV colour mode which has been introduced in background and context

chapter. The data stored into files is the feature used to do matching in this project which is extracted from every image. Such feature is an 8*8 matrix calculated from the pixels of images and can represent the distribution of the colour in an image. It is called fingerprint here and each unit is the average colour under L*a*b* colour mode of the corresponding area of the image. An example is shown in Figure 10. Thus, for each image, 192 real numbers are stored with the library image index.

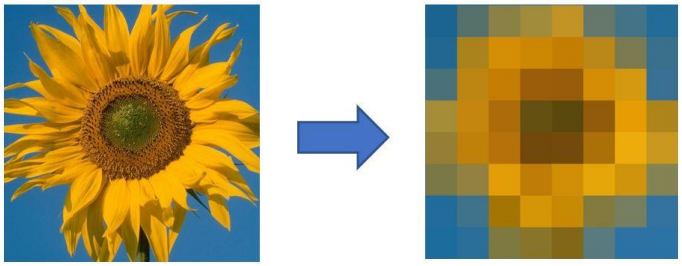


Figure 10 The fingerprint (right) of an image of sunflower (left)

At the same time, master image is cut into equal-size blocks according to the block size defined by user. The fingerprints are extracted from those blocks and stored into a separate file.

3.1.2.1.2 Matching

For each block of master image, one library image should be chosen to replace it. Here the fingerprints will be used to calculate the colour distance between the blocks and library images. The distance between fingerprints is the sum of the distances of each pair of units. And the distances are used to choose the suitable library image. This will be discussed in detail in the next section. To reduce the amount of calculation and speed up, each block will compare with the library images stored in the specific colour bin file with the same mean colour as the current block and in its neighbour colour bin files instead of all the library images. Meanwhile, this will provide more choices and decrease the error. This process is illustrated clearly in the matching process in Figure 9. A repetition-avoid mechanism is applied to make sure it will not select too many identical images in a small area with similar colours.

3.1.2.1.3 Combination

The final step is to combine the chosen library images together to

generate the result image. However, due to the quality of the library images or too much detailed contents in master image, the result image may not look like the master image or lose some indispensable details. Thus, original image overlay is necessary to be applied to the result image. The degree of the overlay is controlled by user.

3.1.2.2 Unequal-size block image mosaics

The unequal-size block effect has a more targeted aim than the equal-size one. The object or the foreground are emphasized by using more smaller blocks to reflect details while the background areas are replaced by less but larger blocks. Therefore, the first step is to extract the foreground from the master image with the help of grabCut function of OpenCV so that we can decide the size of each block. The distribution of different size blocks should be stored properly until the final stage since it is required to copy the chosen library images together during the combination stage. The main steps are shown in Figure 11 and how they are implemented will be discussed later.

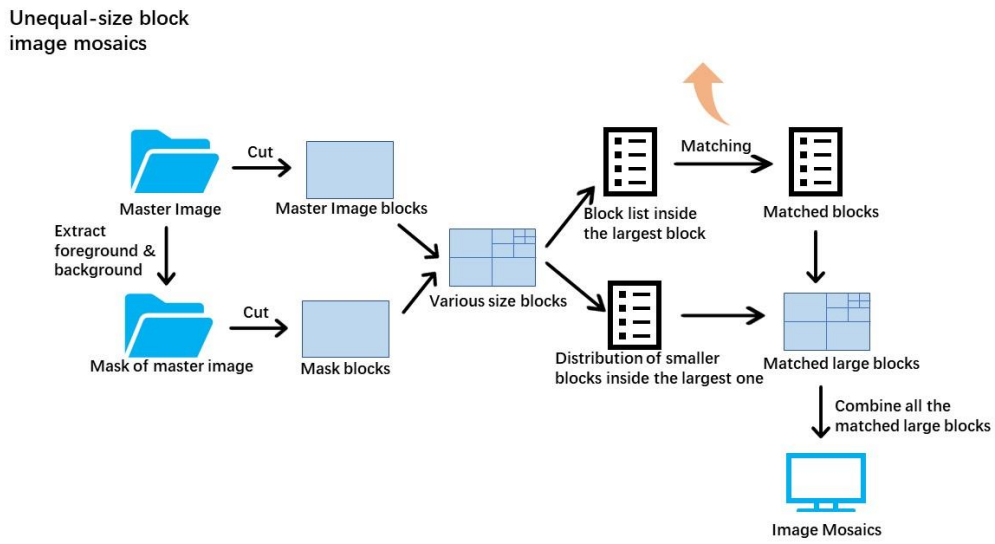


Figure 11 The algorithm of generating unequal-size block image mosaics

Except for the block size difference, original image overlay is removed because we have already use smaller enough blocks to show more information of the objects. However, the matching steps for this part are same as these in the equal-size block part.

3.1.2.3 User interface

The user interface of the project is simple but clear as shown in Figure 12. User could use the left side menu to choose the master image, the number of library images, block size, degree of overlay and the resolution and the name of the result image. The library images should be copied into the library folder first. Then after running the program, result image will be written into result folder by default.

Figure 12 The user interface of image mosaics generator

Because it requires the interaction with user to select the foreground of an image when generating unequal-size block image mosaics. This part is designed to be a terminal application. User can interact with the program from the pop-up window and enter the parameters in the terminal. The foreground image will be written into tmp folder and the result image will be written into target folder.

3.2 Implementation

This section will describe those interesting and difficult technical parts of the three stages in design section. Everything will be explained and analysed in detail. Some technologies that have tried but are not chosen in final will be still discussed here.

3.2.1 Features used for matching images

3.2.1.1 Fingerprint

Fingerprints of an image is the feature used for matching and mentioned in design section. The fingerprint is designed in such way so that the colour structure of the image can be retained while the details are removed. The fingerprint is also used for searching similar images but each unit of it is normally a set of RGB values or only the binary bit. It is the first time to use L*a*b* values as the unit of fingerprint. Here, 64 colour distances are calculated from the fingerprint of each pair of images. The Figure 13 and 14 show a group of examples that can illustrate how the calculated distance tell the difference between the images.

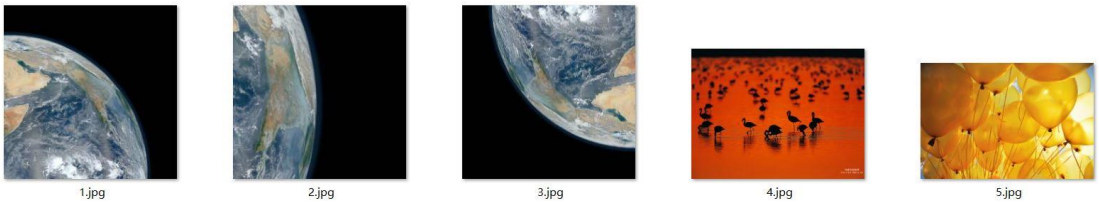


Figure 13 A group of images used for testing colour distance. 1.jpg is used as the reference, the others are used for comparing. It is obvious that 2.jpg is the most similar one.

```
Choose the first image: 1
1.jpg
mean BGR: [80.5671, 77.8316, 72.1082, 0]

mean HSV from BGR: 199, 10, 32

Choose the second image: 2
2.jpg
mean BGR: [62.1611, 59.9041, 53.2699, 0]

mean HSV from BGR: 195, 14, 24

The distance of two images is: 20.5138
**************************************************

Choose the first image: 1
1.jpg
mean BGR: [80.5671, 77.8316, 72.1082, 0]

mean HSV from BGR: 199, 10, 32

Choose the second image: 3
3.jpg
mean BGR: [78.4501, 75.5436, 69.6892, 0]

mean HSV from BGR: 200, 11, 31

The distance of two images is: 46.1504
**************************************************

Choose the first image: 1
1.jpg
mean BGR: [80.5671, 77.8316, 72.1082, 0]

mean HSV from BGR: 199, 10, 32

Choose the second image: 4
4.jpg
mean BGR: [10.2296, 53.2648, 171.983, 0]

mean HSV from BGR: 16, 94, 67

The distance of two images is: 65.1847
**************************************************

Choose the first image: 1
1.jpg
mean BGR: [80.5671, 77.8316, 72.1082, 0]

mean HSV from BGR: 199, 10, 32

Choose the second image: 5
5.jpg
mean BGR: [42.2871, 146.86, 206.293, 0]

mean HSV from BGR: 38, 80, 81

The distance of two images is: 72.7507
**************************************************
```

Figure 14 The results of colour distance. The results have been divided by 64 to get the average colour distance of each unit. Such value does not influence the comparison. The first result is the smallest with value 20.5138. The others are bigger than this value

3.2.1.2 Histogram and others

The histogram is a method to represent the distribution of the colours in an image as well. The results are revealed in the form of tables and able to reflect the main colour of the image and the distribution of the colours. However, histogram can only reflect the distribution of the amount of each colour channel but not the spatial structure. Such feature cannot satisfy the demand in the project. However, OpenCV provides build-in functions to compare the histograms of images. If the number of blocks of the master image is not large or it does not need a very precise matching, histogram is a faster choice rather than the fingerprint method which needs a plenty of calculations.

3.2.2 Repetition-avoid mechanism

Some images have a large area of background. After cutting the images into small blocks, the background blocks share the very similar colours. The library images matched for these blocks by the program may be identical and this may cause a repetition effect in a small area. To avoid the repetition, an area will be checked shown as the Figure 15. Each block refers to the library image chosen there. For every block, a top six matched list which stores the top six minimum colour distances between the current block and the library images with their indexes will be created when matching. If the colour distance of another library image and the block is smaller than the largest one in top six list. Then this library image will replace the previous largest one.

O	O	O	O
O	O	O	O
O	O	X	

Figure 15 The check repeat area. X is current block, O are the blocks to check

After matching all the potential library images, check whether the library image with the minimum distance in the list appears in a larger area shown in Figure 16. If not, then it is chosen for the current block. Or randomly choose one from the list to replace the block. This operation intends to avoid the repetition of a larger block shown

in Figure 17 rather than the single block.

@	O	O	O	O
@	O	O	O	O
@	O	O	X	

Figure 16 The larger check repeat area. @ are the blocks for second round checking.

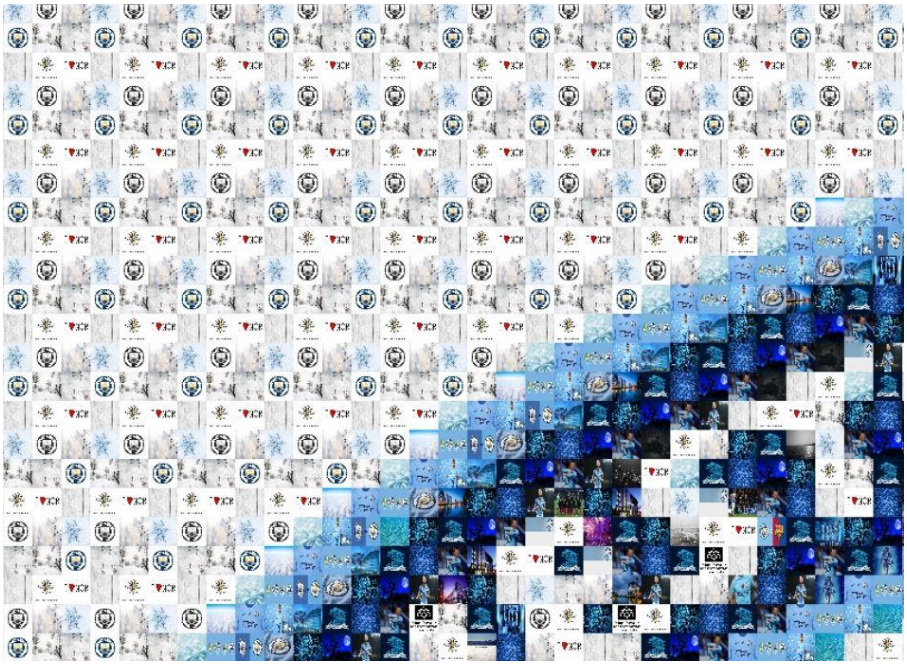


Figure 17 Large area repetition if do not consider the second-round check.

3.2.3 Original image overlay

To retain essential details and profiles of the master image in the result image, sometimes the master image overlay is demanded. Even though it is like a cheating operation, it depends on the quantity and quality of the library images too much and most time the overlay operation is inevitable. The combination of images with different transparency can be realized by the addWeighted function from OpenCV. The parameters of it define the two input images, the alpha value which represents the transparency and the output image. An example of this combination is shown in Figure 18.

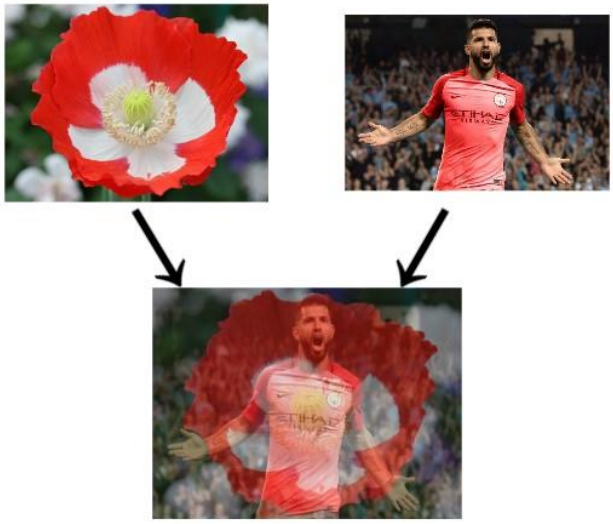


Figure 18 The combination of two images. Each of them is applied 50% transparency.

3.2.4 Method to cut image into unequal-size rectangular blocks and its data structure

The main idea for the unequal-size block image mosaics is to use more and smaller blocks to cover the foreground and less and larger blocks for the background. After applying the grabCut function, an image which is the same size as the master image and only contains the foreground pixels is generated. It is called foreground image. The master image and foreground image are cut into equal-size blocks firstly but in much less number. Then continue to cut the blocks into smaller size according to the proportion of the foreground pixel inside the blocks. There are four levels of the block size which are shown in Figure 19.

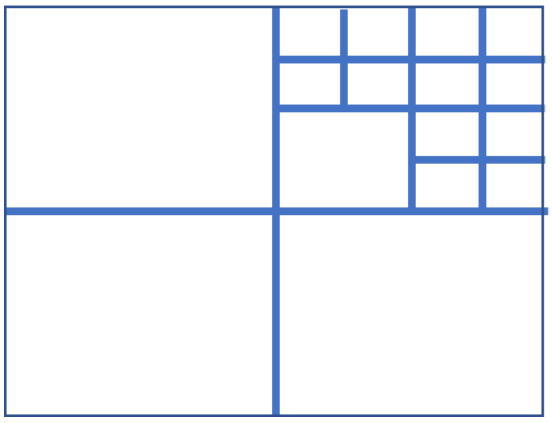
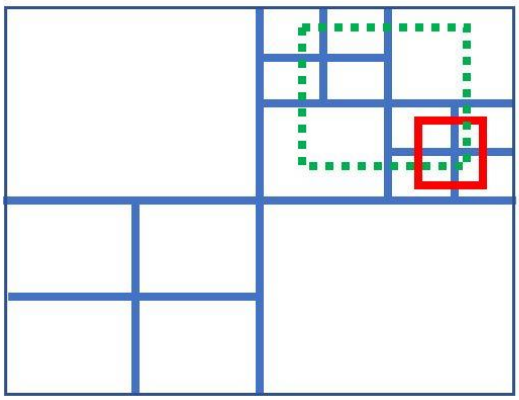


Figure 19 Four levels of block size. The size of the whole block is the first level.

Start from the largest single block, if the proportion of the foreground pixels is higher than 1%, then the largest block is cut into four equal-

size blocks. Or it will remain its size. If the largest block is cut, then do the same operation to each smaller block until the size of the smallest block is one sixty-fourth of the largest one or the current block contains less than 1% of foreground pixels. Thus, if the whole block is covered by foreground pixels, the block will be cut into 64 equal-size smaller blocks.

The data structure for storing such distribution of various-size block is a 4*4 integer matrix and shown in Figure 20 Each matrix represents for a largest single block. The row in the matrix represents for the second large block in the example. The first row corresponds to the top left block. One example is framed by green rectangles in the matrix and example block respectively. While each unit which is framed in red square represents for the block distribution of the third large block framed in red square in the example block as well.



0	0	0	0
4	1	1	4
1	1	1	1
0	0	0	0

Figure 20 The data structure used to store the distribution of the various-size blocks

Each unit has four possible values, -1, 0, 1 and 4. When one of them is -1, all the other units in the matrix must have the same value of -1. This means the largest block remains its size without further cutting. When one of them is 0, all the other units in its row must be 0. This means the area represented by this row has the second large block size without any further cutting. When one of them is 1, it means the corresponding area has the third large block size without any further cutting. And when one of them is 4, it means the corresponding area is cut into smallest blocks. All of this can be cleared observed in Figure 20.

To make the result image more beautiful, some background blocks, also called the largest single blocks, are randomly cut into four second large blocks to produce a random effect.

Chapter 4 Evaluation

This chapter will discuss the evaluation of the project in two ways. One is about the testing about the subproblems, such as the correctness of the colour conversion. Some of these subproblems has a relative objective results which can be compared with the result got from the program. The other way is to evaluate the final effect of the result images. This is very subjective since different user may have different opinions to the same result image. However, in this report, the final effect will be evaluated contrasted with the requirements of the project.

4.1 Testing of subproblems

4.1.1 Colour mode conversion

There are two colour mode conversions applied in this project, RGB to HSV and RGB to L*a*b*. Due to the existing standard formulas, it is not difficult to test the correctness of the conversion from RGB to HSV. This conversion can be tested by the online converter and be reviewed by transferring the values to the colours then checked by manual inspection. Since there is no standard formula to convert from RGB to L*a*b*, the result should be manually checked by transferring the values to the colours. Thus, some errors are inevitable, and the result cannot be evaluated precisely. However, the purpose of this project is to serve human vision. Therefore, such evaluation method could be accepted. The results are not bad even there are a few errors occurring when converting to L*a*b* colour mode. It is acceptable and may not have too much influence on the final effect.

4.1.2 Colour distance

Colour difference is a very subjective topic, the only way to test the correctness of the calculated distance between two colours or two images is human inspection. It is expected that the smaller the distance, the closer the colours or images are. The distance calculated from fingerprints of images should be able to find the best matched image from a library of images. The testing example shown in Figure 13 and 14 illustrates how the distances tell the difference among images. The result clearly explains that fingerprint is a nice choice and the implementation is correct.

4.1.3 Implementation and data structure for unequal-size blocks

This part has a fixed answer according to the cutting rules designed in this project. Firstly, to test whether the program cuts the blocks in a correct way. All the completed small blocks are written into a folder. The size of these blocks can be easily compared with each other and it is not difficult to read the content of each block then check whether the size matches. Next, for checking the correctness of the data structure. The smaller blocks of one single largest block can do the matching with library images. And then combine the chosen library images together to see whether the result image restores the original block. One example is shown in Figure 21. According to the testing result, this part is perfectly implemented.

Figure 21 The largest single block combined under the designed data structure

4.1.4 Others

Other than the testing mentioned above, each step of the project has been proved their correctness or feasibility. Such as, the file stream in C++ for writing and reading fingerprints. The content and format of these files are checked for each initialisation. There are more than ten separate testing in this project and each of them refers to a single subproblem.

4.2 Evaluation of whole effect of image mosaics

4.2.1 Evaluation of equal-size block image mosaics

There are four aspects to be considered to evaluate the result images of equal-size block image mosaics. The basic one is to check the mosaic effect. Whether the whole image can be recognized in distance and the library images are clear at close range. One result image is shown with its master image in Figure 22. Have to say that the effect is not bad.

Figure 22 Original image and its image mosaic

The second one is to check whether the blocks have been matched with a suitable library image. This is similar as the colour distance testing in 4.1.2 paragraph. The most obvious result can be achieved by choosing the blocks lie on the edges. A sample area of one of the result image is shown in Figure 23. It can be observed that the colour distribution of the chosen library images in white rectangle matches with this of the original blocks very well. Due to the repetition-avoid mechanism, not all the blocks can have the best choice.

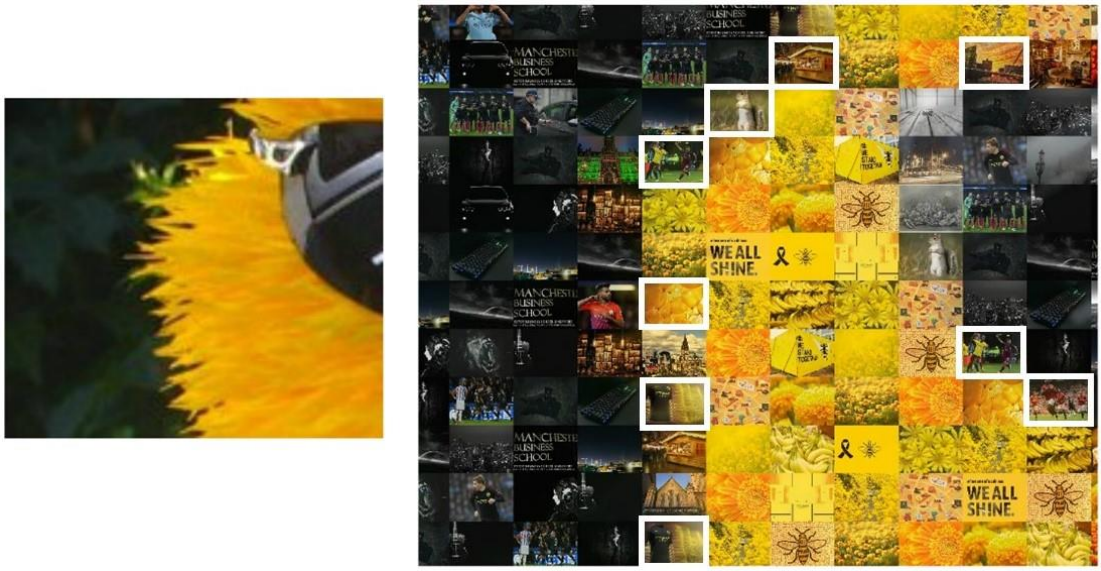


Figure 23 A part of image mosaic shows the correctness of the matching choice.

So, the third aspect is to check the duplication of library images. The Figure 24 shows a section of a result image where the master image has a large area of similar colours. The duplication of blocks does indeed exist, but they are placed in a random way so that it is hard to see any obvious duplicate area.

Figure 24 A part of image mosaic shows the correctness of the repetition-avoid mechanism

The last part is the overlay effect. The example shown in Figure 25 is a very suitable one. If there is no master image overlay applied, it is very hard to see the profile of the tower, the essential part of the master image. However, after applying a certain degree of overlay, the whole image remains the mosaic effect. In addition, the tower is much clearer to be recognised. The degree of the overlay depends on the quality of the library images and the content of the master

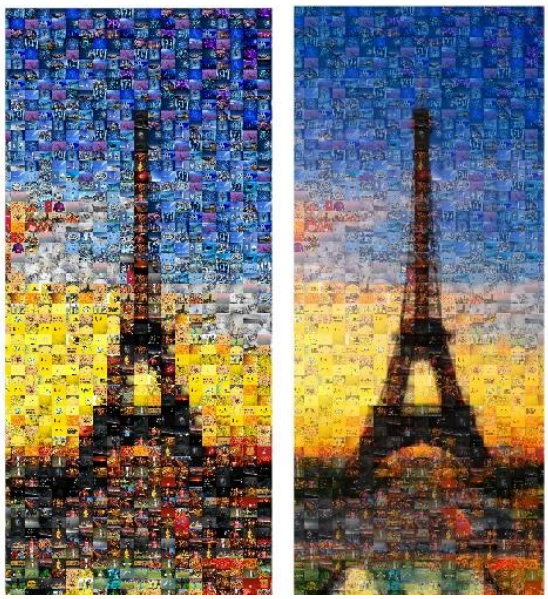


Figure 25 Comparison of the effect of overlay. The right image with overlay can clearly show the towel and its reflection in water

image. More examples of equal-size block image mosaics can be found in Appendix A.

4.2.2 Evaluation of unequal-size block image mosaics

After the testing mentioned in 4.1.3, there is not much to evaluate. The Figure 26 shows a result image of unequal-size block image mosaics and more examples can be found in Appendix B. Two aspects need to be checked in this section. One is to check the correctness of block cutting. The other one is to check whether some largest blocks are cut down to a lower level. In the example image, the logo is shown clearly with more and smaller images while the background is covered by less and larger images. Meanwhile some largest blocks have been cut into smaller blocks for aesthetic effect.

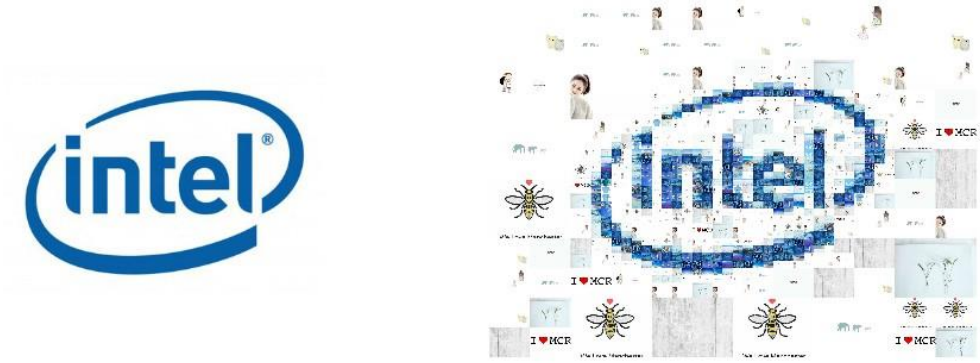


Figure 26 Example of unequal-size block image mosaics

4.2.3 Evaluation of user interface

The equal-size block part is created as a window application. The interface is clear and easy to use. However, due to lack of time, the image viewer can only show the image but have no other functions. The result images can be found in target folder. The unequal-size block part is created as a terminal application with pop-up window for interaction with user. It is not difficult to use, and the process can be tracked by the processing bars.

4.2.4 Evaluation of performance

By comparing the performance with those online generators, the speed of this project belongs to the middle level while the memory

consumes less than 300 megabytes in most cases. By analysing the features which may influence the speed. The size or the resolution of images does not affect the running time too much. The factor that increases the running time in a large amount is the number of blocks of the master image being cut. The process of initialisation and matching only spend a little time while the process of copying the images together spends most of the running time. It is guessed that the running time may decrease a lot if the project runs on a high-performance computer. However, the range of the current running time is acceptable by comparing with the existing generators.

Chapter 5 Reflection and Conclusion

This chapter will discuss the changes from the initial plan with their reasons and the knowledge gained in this project. The conclusion will summarize the achievements of this project.

5.1 Completion of the plan

The initial plan of this project contains two main sections. The standard image mosaics include the equal-size block one and parquet type. The novel image mosaics include the unequal-size block one and animation mosaics. It is expected that the standard ones can be finished before the Christmas break. But learning the colour distance and finding the solution to convert RGB to L*a*b* took a long time. These knowledge is the basic for the matching algorithm in this project. So, after developing these sections, the progress was getting better. The equal-size block image mosaics were completed at the start of the second semester. At that time, the progress was slightly behind. Because the difference between equal-size section and parquet type is just the shape of the blocks. It is not worth taking the time to do the similar work and may waste time fixing more bugs. Thus, the parquet type was given up.

For the novel part, the animation mosaics were finally determined to be beyond the context of this project. So, more attention is placed on unequal-size block image mosaics. The method to extract foreground and the idea to cut blocks into various smaller blocks were developed in time. Thus, the unequal-size block image mosaics were completed as scheduled.

The functional part can be said to be completed on time. The user interface was planned to be developed at the middle and late stages of the project. However, the structure of windows application is totally different from the terminal application. And data transfer and compatibility problems occurred during the development of windows application. Thus, the equal-size part has a user interface while the unequal-size one is still a terminal application.

5.2 New knowledge

From this project, I learnt how to create projects and windows applications under Visual Studio. With the use of C++ and OpenCV, I learnt how to manipulate and analysis images as well as write and

read data using file stream by coding. Especially a wide variety of data structures for colours and images, and a large amount of build-in functions of image processing. Except for the technical tools, some theories are also worth learning. Like different colour modes and CIE colour distance.

5.3 Conclusion

Comparing the results of the project to the desired achievements, most of the requirements were completed. The achievements of this project are listed below:

User can generate equal-size block image mosaics with the following features:

- Graphical user interface
- User-defined library images, block size, resolution and degree of original image overlay
- The blocks are matched with library images to a very precise extent
- Repetition-avoid mechanism applied for a better aesthetic effect
- Original image overlay applied for a better restore effect

User can generate unequal-size block image mosaics with the following features:

- User-defined library images, block size
- Emphasize the foreground of an image in mosaics effect
- Smart cutting images, smaller blocks for foreground and larger blocks for background
- Random reduce the size of background blocks for a better aesthetic effect

Reference

[1] https://en.wikipedia.org/wiki/Mosaic, *Mosaic*. 23th April 2018.

[2] Sailko, F.B. (2013). *Palazzo dei gran maestri di rodi, sala del cavalluccio, mosaico della ninfa sull'ippocampo, da kos, periodo romano.* [Image] Available at: https://commons.wikimedia.org/wiki/File:Palazzo_dei_gran_maestri_di_rodi,_sala_del_cavalluccio,_mosaico_della_ninfa_sull%27ippocampo,_da_kos,_periodo_romano,_02.JPG [Assessed 23th April 2018].

[3] Angela Cartwright, *Mixed Emulsions: Altered Art Techniques for Photographic Imagery*, p. 102, Quarry Books, 2007.

[4] The Truman Show Poster, (1998). *The Truman Show.* [Poster].

[5] https://en.wikipedia.org/wiki/Photographic_mosaic, *Photographic Mosaic*. 23th April 2018.

[6] https://www.picturemosaics.com, *Online image mosaics generator*. 24th April 2018.

[7] https://helpx.adobe.com/photoshop/using/color-modes.html, Colour modes. 24th April 2018.

[8] SharkD. (2010). *The HSV colour model mapped to a cylinder*. [Image] Available at: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png [Assessed 24th April 2018].

[9] https://en.wikipedia.org/wiki/HSL_and_HSV, *HSV colour mode.* 24th April 2018.

[10] http://www.colorwiki.com/wiki/Delta_E:_The_Color_Difference, *Colour difference*, 25th April 2018.

[11] https://en.wikipedia.org/wiki/Color_difference#cite_note-11, *Colour difference*, 25th April 2018.

[12] http://zschuessler.github.io/DeltaE/learn/, *Colour difference*, 25th April 2018.

[13] http://mattlockyer.github.io/iat455/documents/rgb-hsv.pdf, *Convert from RGB to HSV*, 25th April 2018.

[14] https://en.wikipedia.org/wiki/Lab_color_space, *Lab colour space*, 25th April 2018.

[15] http://hao.qinz.net/comments.php?y=08&m=07&entry=entry080727-033517, *Convert RGB to Lab*, 25th April 2018.

[16] http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html, *Convert RGB to XYZ*, 25th April 2018.

[17] Hae-Yeoun Lee, *Generation of Photo-Mosaic Images through Block Matching and Color Adjustment*, World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering, Vol:8, No:3, 2014

[18] Nicholas Tran, *GENERATING PHOTOMOSAICS: AN EMPIRICAL STUDY*

[19] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. *"GrabCut": interactive foreground extraction using iterated graph cuts.* ACM Transactions on Graphics (TOG), Volume 23 Issue 3, August 2004, Pages 309-314.

[20] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio, *Visual Studio*, 26th April 2018.

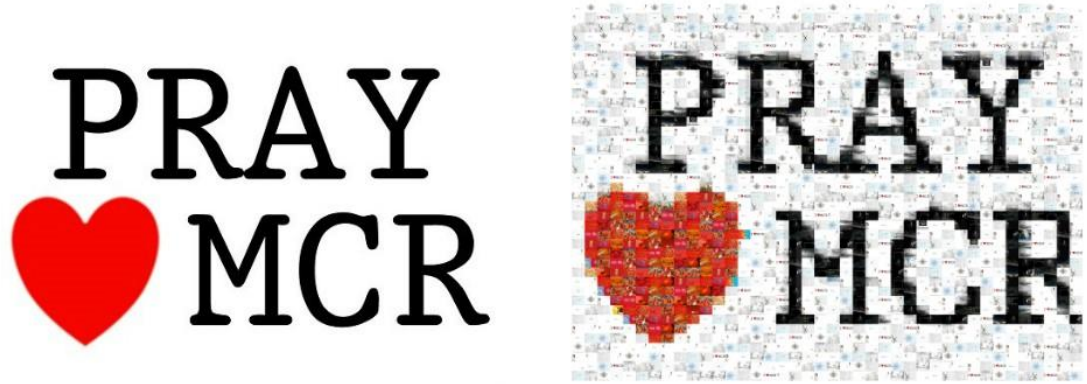
[21] https://docs.microsoft.com/en-us/dotnet/visual-basic/developing-apps/windows-forms/windows-forms-application-basics, *Windows forms application*, 26th April 2018.

[22] https://opencv.org/, *OpenCV*, 26th April 2018.

[23] https://en.wikipedia.org/wiki/C%2B%2B, C++ programming language, 26th April 2018.

Appendix A

Examples of equal-size block image mosaics



Example Image 1 Big block size, 10% overlay

Example Image 2 Big block size, 20% overlay

Example Image 3 Medium block size, 20% overlay

Example Image 4 Medium block size, 20% overlay

Example Image 5 Medium block size, 10% overlay

Appendix B

Examples of unequal-size block image mosaics

Example Image 6 Unequal-size block image mosaics

Example Image 7 Unequal-size block image mosaics

Example Image 8 Unequal-size block image mosaics

Example Image 9 Unequal-size block image mosaics

Example Image 10 Unequal-size block image mosaics