The Chinese University of Hong Kong, Shenzhen

MAT 3007

Optimization

Midterm Project Report

Author:
Guo Chengxi
Yan Tianshuo
Cui Yuncong

Student Number:
118010080
118010363
118010045

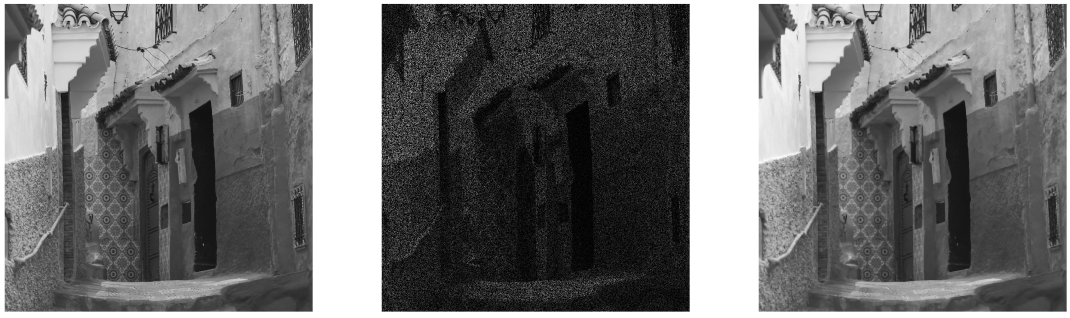
July 17, 2020

Contents

1 Introduction

Inpainting, the technique of modifying an image in an undetectable form, is as ancient as art itself. The goals and applications of inpainting are numerous, from the restoration of damaged paintings and photographs to the removal/replacement of selected objects.

This report introduces a approach to image inpainting that optimizes the shape of masked regions given by users. In image inpainting, which restores damaged regions in images, users draw masks to specify the regions. However, it is widely known that the users typically need to adjust the masked region by trial and error until they obtain the desired natural inpainting result, because inpainting quality is significantly affected by even a slight change in the mask. This manual masking takes a great deal of users' working time and requires considerable input. To reduce the human labor required, we practice a method for masked region optimization so that good inpainting results can be automatically obtained. To this end, our approach estimates damaged inpainting for all undamaged pixels in inpainted images and reforms an original mask on a undamaged pixel basis, so that the naturalness of the inpainting result is improved. The efficacy of this approach does not depend on inpainting algorithms, thus it can be applied for every inpainting method as a plug-in. To demonstrate the effectiveness of our approach, we test our algorithm with varied images and show that it performs well with relatively high PSNR value.



(a) Original Image (b) Damaged Image (c) Reconstructed Image

Figure 1: An Example of Inpainting a Damaged Picture

2 Objectives

1) Apply different methods to damaged pictures to inpaint them;

2) Compare the effectiveness and efficiency of the different algorithm and models;

3) Generate mosaics for pictures;

4) Discuss the drawbacks of each method and the further application.

3 Theory

3.1 Total Variation: Adjacent Distance Sum

Adjacent distance sum, as its name suggests, is a sum of color distance or the difference between color value of two adjacent points, and we call this difference "little edge" later on. Our next step is the same as the former one, generating a matrix that derives little edges from original stacked vector. Each row of D corresponds to one little edges, for example, $\left\|x_{11} - x_{12}\right\|$, corresponds to $\begin{pmatrix} 1 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \end{pmatrix}$ in which -1 appears at the position (m+1).

3.2 PSNR

Peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a very wide dynamic range, PSNR is usually expressed in terms of the logarithmic decibel scale. PSNR value is defined by:

$$\mathbf{PSNR} := 10 \cdot \log \frac{mn}{\left\|x - u^*\right\|^2}$$

The higher the PSNR value is, the better we have done in inpainting a damaged picture.

3.3 Sparse Reconstruction: Block-DCT

Using sparse reconstitution, we group all the pixels into blocks and using DCT to simplify these blocks. A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.The use of cosine rather than sine functions is critical for compression, since it turns out (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions. In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers.

With the help of Block-DCT, we can reconstruct the pixels of pictures by reducing the repeated information thus decrease the running time of the program.

3.4 Interior Point

Interior point methods are a type of algorithm that are used in solving both linear and nonlinear convex optimization problems that contain inequalities as constraints. The LP Interior-Point method relies on having a linear programming model with the objective function and all constraints being continuous and twice continuously differentiable. In general, a problem is assumed to be strictly feasible, and will have a dual optimal that will satisfy Karush-Kuhn-Tucker (KKT) constraints described below. The problem is solved (assuming there IS a solution) either by iteratively solving for KKT conditions or to the original problem with equality instead of inequality constraints, and then applying Newton's method to these conditions. Now we introduce the Barrier method: For the barrier method algorithm, there a few approximations that must be made. Given a problem in the form of

$$Minimize\ f_0(x)$$
$$s.t.\ f_i(x) \leq 0$$
$$Ax = b$$

We must reformulate it to implicitly include the inequalities in the objective function. We can do this by creating a function that greatly increases the objective if a constraint is not met. Our conditions then changes to

$$Minimize\ f_0(x) + \sum_i^m I_-(f_i(x))$$
$$s.t.\ Ax = b$$
$$where\ I_-(x) = \left| \begin{array}{ll} 0 & x \leq 0 \\ \infty & x > 0 \end{array} \right|$$

This problem, however, is not continuous. A modification can be made by approximating $I_-(x)$ as a logarithm log(-x), which approaches infinity when x approaches 0 as we want, and makes all functions twice differentiable. We then put the logarithm over a variable that sets a level of accuracy for the approximation we make. Here we will call that variable t. We define

$$\phi(x) = -\sum_i^m log(-f_i(x))$$

which blows up if any of our constraints are violated. Our LP problem now becomes

$$minimize\ f_0(x) + \frac{1}{t}\phi(x)$$
$$s.t.\ Ax = b$$

This allows us to use Newton's method to follow what is called a Central Path, which is a series of points we iterate through that all satisfy the equality constraints Ax=b from the original problem, but give increasingly more optimized values for the objective function, with the inequality constraints $f_i(x)$ not necessarily equal to 0. Then the algorithm starts:

Given strictly feasible x, $t := t^0 > 0, \mu > 1, \epsilon < 0$

Repeat:
1. Compute $x^*(t)$ by minimizing $tf_0 + \phi subject to Ax = b$, starting at x.
2. Update $x := x^*(t)$.
3. Quit if $\frac{m}{t} \leq \epsilon$, else
4. Increase $t := \mu t$

3.5 Simplex Dual

The dual simplex algorithm is an attractive alternative method for solving linear programming problems. Since the addition of new constraints to a problem typically breaks primal feasibility but not dual feasibility, the dual simplex can be deployed for rapid reoptimization, without the need of finding new primal basic feasible solutions. This is especially useful in integer programming, where the use of cutting plane techniques require the introduction of new constraints at various stages of the branch-and-bound/cut/price algorithms.

This method is derived from simplex table. The only that changes is that this algorithm used the dual problem to get the optimal solution.

3.6 Derive Undamaged Points

To get the set of undamaged points, we first define a assistant matrix, say, $\mathbf{A}$, which should consist of all the position information of undamaged pixels. To further construct, each row of $\mathbf{A}$ represents position information of one undamaged pixel, for instance, if the position of the first undamaged pixel is $(2, 1)$ and in the stacked vector that of it is 2, then the first row of A should be $010\cdots0$, and then we finished our construction.

Suppose the set of all pixels with their position and color information consists a stacked vector $\mathbf{u}$, and the color information of all undamaged points is $\mathbf{Au}$, represented as vector $\mathbf{b}$.

The strict definition of $\mathbf{A}$ is as below:

$$\mathbf{A} = \begin{pmatrix} \mathbf{e}_{q_1}^T \\ \vdots \\ \mathbf{e}_{q_s}^T \end{pmatrix} \in \mathbb{R}^{s\times mn}, where \mathbf{e}_j \in \mathbb{R}^{mn},\ [\mathbf{e}_j]_i = \begin{cases} 1, & \text{i = j} \\ 0, & \text{otherwise} \end{cases}$$

$$where\ q_i := i^{th}\ position\ information\ of\ undamaged\ pixel.$$

4 Part I

4.1 Problem Description

Image inpainting describes the task of recovering an image U from partial data and observations. In particular, we assume that parts of the image U are missing or damaged, (e.g., due to scratches, stains, or compression), and the aim is to reconstruct this missing or damaged information via solving a suitable inpainting or optimization problem.

4.2 Part I-1: Total Variation Minimization

4.2.1 Reformulate Problems

Reformulate problem (2) as a linear program and derive the associated dual of the linear optimization formulation.

Primal:

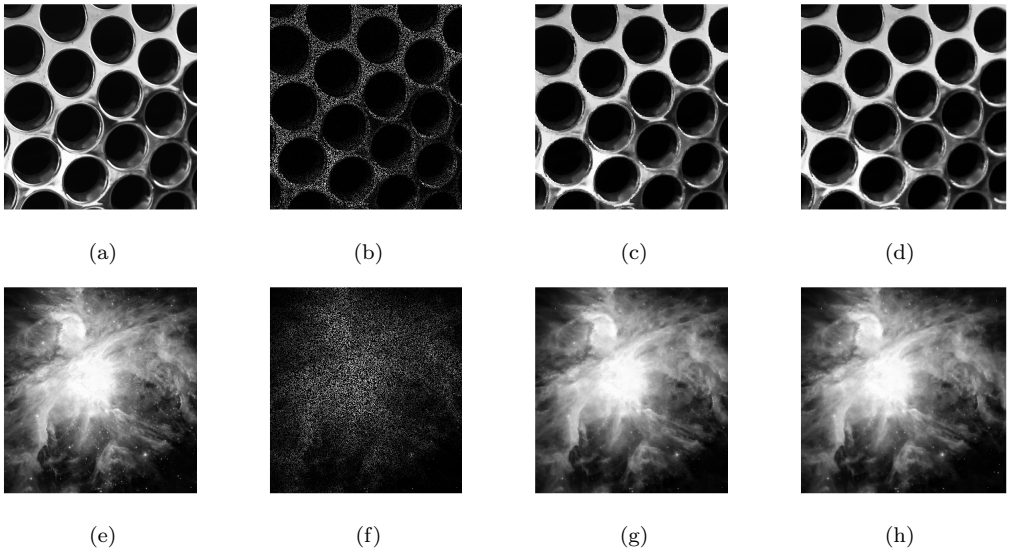
$$\min_{t\in\mathbb{R}^\kappa} \mathbf{t}$$
$$s.t. - \begin{bmatrix} -\mathbf{D} & \mathbf{I} \\ \mathbf{D} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix} \leq \mathbf{0}$$
$$\mathbf{Ax} = b$$
$$\mathbf{x} \geq 0$$

Dual:

$$\min_{x\in\mathbb{R}^{mn}, t\in\mathbb{R}^\kappa} (O^T \ \mathbf{1}^T) \begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix}$$
$$s.t. \begin{bmatrix} \mathbf{D} & \mathbf{I} \\ -\mathbf{D} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{t} \end{bmatrix} \geq 0$$
$$\mathbf{Ax} = \mathbf{b}$$

4.2.2 Reconstructed Image Samples

In Part I-2, we implement the Interior-Point and Dual-Simplex. Figure 2 is two samples of image reconstruction. The original image file (a) is "512_512_circles.png". The original image file (e) is "512_512_stars_01.png". The mask file for both is "512_512_random70.png". Image (b)(f) is the original images with mask. Image (d)(h) is the reconstructed images by using interior-point. Image (c)(g) is the reconstructed images by using Dual-Simplex.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 2: (a): Original Image(circles) (b): Damaged Image(circles) (c): Reconstructed Image(circles) by Dual Simplex (d): Reconstructed Image(circles) by interior point (e): Original Image(stars) (f): Damaged Image(stars) (g): Reconstructed Image(stars) by Dual Simplex (h): Reconstructed Image(stars) by interior point

Intuitively speaking, it can be easily seen the performance of interior-point is better than Dual-Simplex. The time of interior-point and Dual-Simplex are respectively 25.075077 and 331.882814s, which indicates that interior-point is faster.

4.2.3 Comparison of the Performance of Interior Point and Dual Simplex

A suitable image quality measure is the so-called PSNR value. Figure 3 is a table of the PSNR of the image reconstruction. In the chosen three samples, the PSNR of interior-point are respectively higher than the PSNR Dual-Simplex, which indicates that interior-point is more accurate. From the perspective of quantified standard, it can be easily seen the performance of interior-point is better than Dual-Simplex.

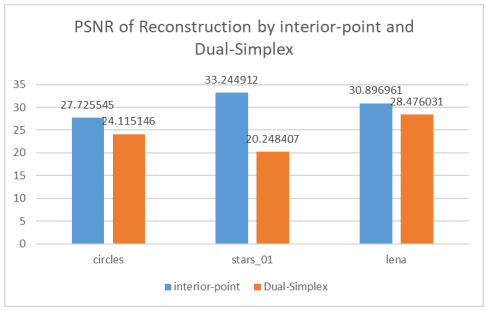


Figure 3: PSNR of Reconstruction by interior-point and Dual-Simplex (The size of images is 512*512. The mask is "512_512_random70.png".)

Another important standard for algorithms is running time. Figure 4 is a table of the running time of the image reconstruction. In the chosen three samples, the running time of interior-point are respectively much less than the PSNR Dual-Simplex, which indicates that interior-point is faster. From the perspective of quantified standard, it can be easily seen the performance of interior-point is better than Dual-Simplex.

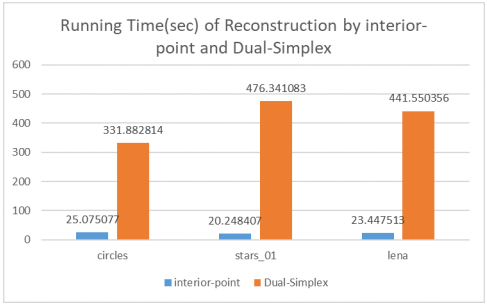


Figure 4: Running Time(sec) of Reconstruction by interior-point and Dual-Simplex (The size of images is 512*512. The mask is "512_512_random70.png".)

The number of iterations shows the efficiency and complexity of algorithms. Figure 5 is a table of the iteration times of the image reconstruction. In the chosen three samples, the iteration times of interior-point are respectively much less than the PSNR Dual-Simplex, which indicates that interior-point is faster. From the perspective of quantified standard, it can be easily seen the performance of interior-point is much more efficient than Dual-Simplex.

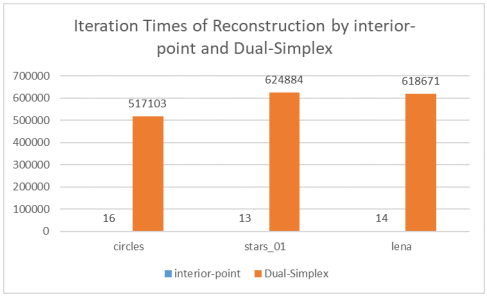


Figure 5: Iteration Times of Reconstruction by interior-point and Dual-Simplex (The size of images is 512*512. The mask is "512_512_random70.png".)

Figure 6 is a table of the performance of interior-point on images with different pixels. In the chosen samples, the running time of interior-point are increasing gradually with the ascent of pixel. However, there is no great fluctuation in the PSNR and iterations, which indicates that the pixel does not affect the performance of interior-point.

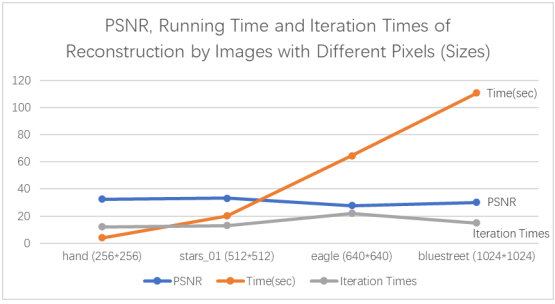


Figure 6: PSNR, Running Time and Iteration Times of Reconstruction by Images with Different Pixels (Sizes) (Masks are all "random_70" type in each pixel.)

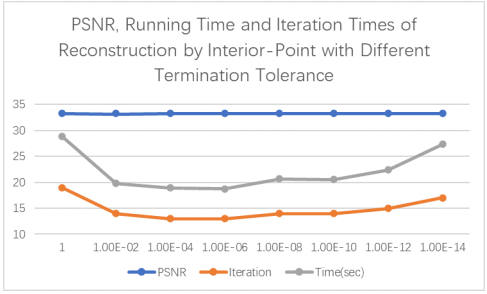


Figure 7: PSNR, Running Time and Iteration Times of Reconstruction by Interior-Point with Different Termination Tolerance (Image: "512_512_stars_01.png". Mask: "512_512_random70.png")

Figure 7 is a table of the performance of interior-point with different termination tolerance (Optimality-Tolerance). Obviously, there is little difference between distinct values of termination tolerance. However,

no matter the termination tolerance is too large or too small, the running time and numbers of iterations can increase to cause unnecessary demands to CPU. According to this table, the most suitable termination tolerance should be set to around 1E-06.

4.2.4 Conclusion

- Interior Point is better than Dual-Simplex. The PSNR demonstrates that the same picture with same mask get higher PSNR by processing with Interior Point than Dual-Simplex.
- Interior Point is more efficiency than Dual-Simplex. The numbers of iterations demonstrates that the same picture with same mask needs less iterations by processing with Interior Point than Dual-Simplex.
- Interior Point is faster than Dual-Simplex. The running time demonstrates that the same picture with same mask needs less time by processing with Interior Point than Dual-Simplex.
- The Termination Tolerance of Interior-Point should be set around 1E-06. The performance demonstrates that the same picture with same mask needs less time and iterations by processing with Termination Tolerance of 1E-06 than that of larger or smaller while the PSNR remains the same..

4.2.5 Limitation

The Total Variation Minimization used the distances between one pixel and other four pixels from up, down, left and right. However, it fails to consider about other four pixels from upleft, up right, downleft and downright. Figure 8 is a 3*3 image to show that the possible mistake that TV model could make. When the original image misses these two points, the TV will recover this two point by pixels from up, down, left and right and get the opposite result from the original picture.

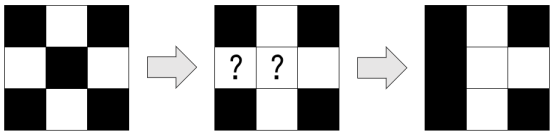


Figure 8: A 3*3 pixel image. Left is the original image. Middle is the image that is lack of two pixels. Right is the reconstructed image.

Thus, this drawback also proves the standpoint in the conclusion that the TV model is suitable for processing the pictures with regular pattern or streams, but not for the images with random points. Certain pattern can easily help a pixel to recognize pixels around it, while a image with random points cannot. For example, TV model is impossible to recover a damaged QR-code because there is no organized pattern of stream in the code. The possible solution for solving this can be a method add the distances between one pixel and other four pixels from upleft, up right, downleft and downright.

4.3 Part I-2: Sparse Reconstruction

4.3.1 Reformulate Problems

Reformulate problem (3) as a linear program. Derive the associated dual of the linear optimization formulation and try to simplify the dual problem (e.g., by reducing the number of variables).

Original Linear Programing:

$$min_{x\in\mathbb{R}^{mn}} \left\|\mathbf{\Phi x}\right\|$$
$$s.t.\ \left\|\mathbf{Ax}-b\right\|_\infty \leq \delta$$

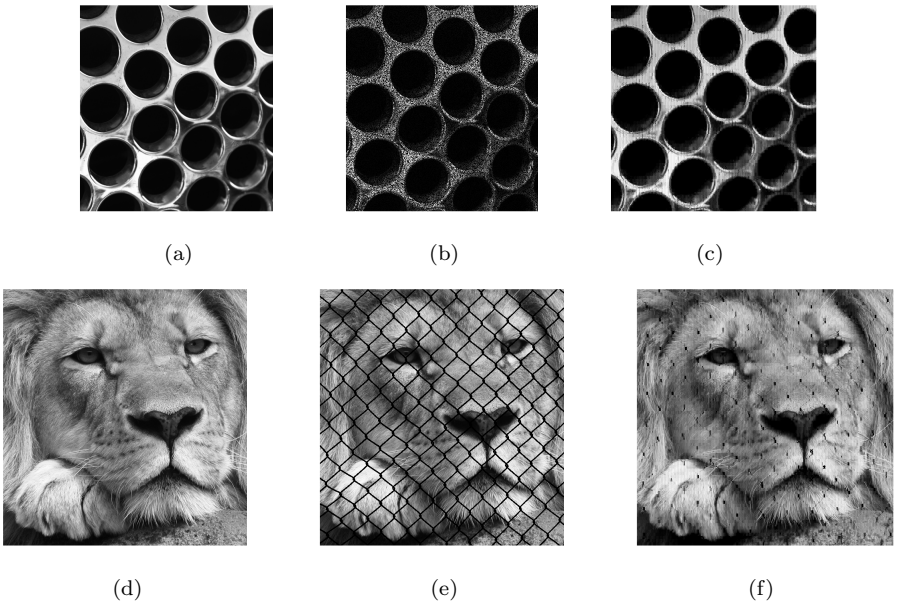
let $t_i = \left|\mathbf{\Phi}_i\mathbf{x}\right|$ for $i \in \{1,\cdots,mn\}$. Then we have the LP:

$$\min_{x,t\in\mathbb{R}^{mn}} \mathbf{1}^T t$$
$$s.t. \begin{bmatrix} \mathbf{\Phi} & -\mathbf{I} \\ -\mathbf{\Phi} & -\mathbf{I} \\ \mathbf{A} & \mathbf{0} \\ -\mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \leq \begin{bmatrix} \mathbf{0} \\ b+\delta\mathbf{1} \\ -b+\delta\mathbf{1} \end{bmatrix}$$

Dual:

$$\max_{y\in\mathbf{R}^{2mn+2s}} \begin{bmatrix} \mathbf{0}^T & \mathbf{b}^T+\delta\mathbf{1}^T & -\mathbf{b}^T+\delta\mathbf{1}^T \end{bmatrix} \mathbf{y}$$
$$s.t. \begin{bmatrix} \mathbf{\Phi}^T & -\mathbf{\Phi}^T & \mathbf{A}^T & -\mathbf{A}^T-\mathbf{I} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{y} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$
$$\mathbf{y} \leq 0$$

4.3.2 Reconstructed Image Samples



(a) (b) (c)

(d) (e) (f)

Figure 9: (a): Original Image(circles) (b): Damaged Image(circles) (c) Reconstructed Image(circles) by interior point (d): Original Image(lion) (e): Damaged Image(lion) (f): Reconstructed Image(lion) by interior point

In Part I-2, we only implement the Dual-Simplex. Figure 9 is two samples of image reconstruction. The original image file (a) is "512_512_circles.png". The mask file for (a) is "512_512_random50.png". The original image file (d) is "640_640_lion.png". The mask file for (d) is "640_640_mesh.png". Image (b)(e) is the original images with mask. Image (c)(f) is the reconstructed images by using interior-point.

4.3.3 Comparison of the Performance of Interior Point with Different Parameters

Figure 10 is a diagram of the PSNR, Running Time and Iteration Times of Different Pictures with the Same Pixel and the Same Masks. According to the average data, average PSNR is 23.2598, average running time is 249s and average iteration times is 28. The performance is acceptable.

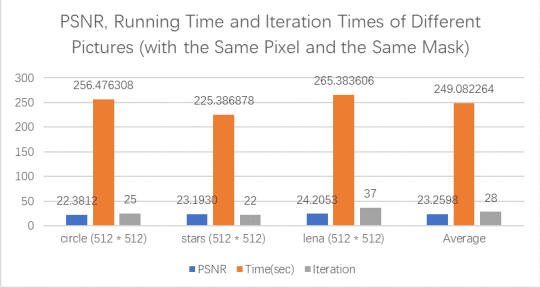


Figure 10: PSNR, Running Time and Iteration Times of Different Pictures with the Same Pixel and the Same Mask (Mask: "512_512_random50.png", Pixel: 512)

Figure 11 is a diagram of the PSNR of interior-point with different masks. We started δ with 0.01; however, the result cannot be calculated by the normal computer until δ raises to 0.04. This diagram records the PSNR from $\delta = 0.04$ to $\delta = 0.10$. From the perspective of quantified standard, it can be easily seen that the performance(PSNR) decreases with the ascent of δ. It better to choose a small δ to make the reconstructed picture clearer while this δ do not cause infinite iterations. The value will be decided as 0.06, practical and precise. Figure 11 also indicates that the clearer the damaged picture is, the faster (PSNR) decreases with the ascent of δ. When the damage is extremely serious, the change of δ cannot affect the PSNR. When the damage is extremely slight, the change of δ is the decisive parameter to the PSNR.

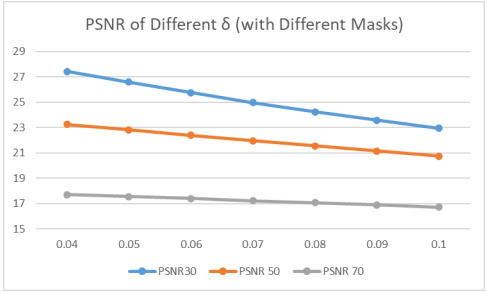


Figure 11: PSNR of Different with Different Masks ("PSNR30" refers to "512_512_random30.png", "PSNR50" refers to "512_512_random50.png", "PSNR70" refers to "512_512_random70.png")

Figure 12 is a diagarm of the performance of interior-point with different termination tolerance (OptimalityTolerance). Obviously, there is no difference between distinct values of termination tolerance. However, no matter the termination tolerance is too large or too small, the running time and numbers of iterations can increase to cause unnecessary demands to CPU. According to this table, the most suitable termination tolerance should be set to around 1E-03.

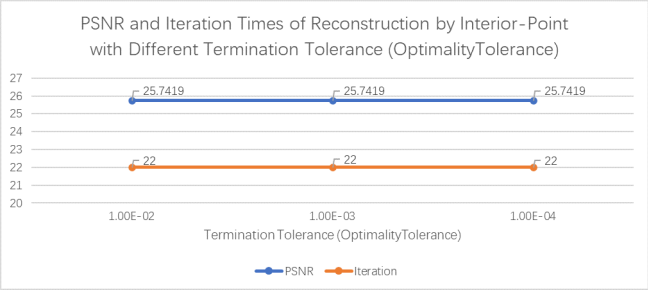


Figure 12: PSNR and Iteration Times of Reconstruction by Interior-Point with Different Termination Tolerance (OptimalityTolerance) (Image: "512_512_stars_01.png". Mask: "512_512_random30.png")

According to Figure 13, the two algorithms, *Total Variation Minimization* and *Sparse Reconstruction*, both achieve acceptable PSNR. However, the PSNR in Part I-1 *Total Variation Minimization* is higher than that in Part I-2 *Sparse Reconstruction*.

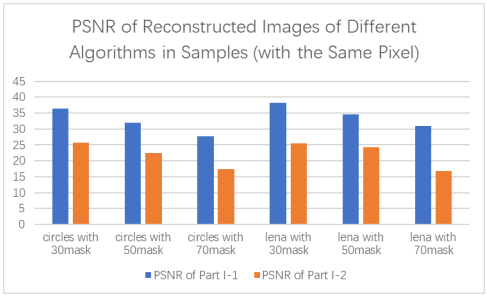


Figure 13: PSNR of Reconstructed Images of Different Algorithms in Samples ("30mask" refers to "512_512_random30.png", "50mask" refers to "512_512_random50.png", "70mask" refers to "512_512_random70.png")

4.3.4 Conclusion

- The PSNR is acceptable but less than that in Part I-1. The general performance(including runing time and the number of iterations) is not as good as Part I-1.
- δ is chosen as 0.06. It better to choose a small δ to make the reconstructed picture clearer while this δ does not cause infinite iterations. The value is decided as 0.06, practical and precise.
- The Termination Tolerance should be set around 1E-03. The performance demonstrates that the same picture with same mask needs same iterations by processing with Termination Tolerance of 1E-06 than

while the PSNR remains the same. To reduce the cost of CPU, the tolerance should be not more than 1E-02 or no less than 1E-04. Thus, it be set around 1E-03

4.4 Summary

The two algorithms, *Total Variation Minimization* and *Sparse Reconstruction*, both achieve acceptable PSNR. However, the PSNR in *Total Variation Minimization* seems higher than that in *Sparse Reconstruction*. This is because these two has own merits and characters.

The algorithm of *Sparse Reconstruction* is more suitable for the images with blocks, because it is easy to repair large part with same color. The algorithm of *Total Variation Minimization* is more suitable for the images with strikes and lines, because it is easy to repair stream part with same color. Thus, for a common picture, most damaged pictures in life should apply *Total Variation Minimization* to repair. On the contrary, damaged pictures from artificial producing (e.g. commercials, posters) should apply *Sparse Reconstruction*.

As for termination tolerance, *Total Variation Minimization* is better to choose 1E-06, while *Sparse Reconstruction* is supposed to be set as 1E-03. The standard considers both the accuracy of reconstructed image and the cost of CPU.

5 Part II

5.1 Problem Description

The last part of the project aims to generate mosaics from an image using a given set of mosaic tiles via linear optimization methods. Suppose that we have a set of r individual mosaic tiles of size $l \times l$. Our goal is to utilize these different tiles to form a mosaic of a given target image $U \in \mathbb{R}^{m \times n}$. In order to fit the tiles into the target image, we first assume that $mod(m, l) = mod(n, l) = 0$ and partition the image into $\frac{m}{l} \times \frac{n}{l}$ blocks of pixels. The desired matching is based on comparing the block brightness of the image and the tile brightness values so that the outline of the target image can be recovered as much as possible.

5.2 Reformulate Problems

To find a perfect matching between mosaic tiles and the image, our first task is to measure the brightness of tiles and images. The block brightness of the image can be calculated by taking the average of the pixel values. The same method can be applied to measure the brightness of each tile. This will result in $\frac{m}{l} \cdot \frac{n}{l}$ block brightness values $\beta_{i,j},\ i = 1, ..., \frac{m}{l},\ j = 1, ..., \frac{n}{l}$ for the image and r brightness values $c_k,\ k = 1, ..., r$ for the different tiles.

With the brightness data, we can use the matrix of block brightness values to linearly assign a unique tile to each image block. However, to ensure that the obtained mosaic is balanced, we enforce that each type of tile is used the same number of times, i.e., $[\frac{m}{l} \cdot \frac{n}{l}]/r$ many times. In this case, we again assume that the number $[\frac{m}{l} \cdot \frac{n}{l}]/r$ is an integer. Moreover, to avoid possible confusion, we only allow that each block of the target image be masked with one mosaic tile.

To formulate the described mosaic generation task as a linear optimization problem, we model the decision variables with the following constraints and objective functions (For simplicity, we denote the number of tiles per column $p = \frac{m}{l}$, the umber of tiles per row as $q = \frac{n}{l}$):

Decision variable In this task, we model the decision variable as a tensor $x_{k,i,j} \in \mathbb{R}^{r\times p\times q}$. The binary number $x_{k,i,j}$ contains the information if we place the k-th at block (i,j) or not (1 for yes, 0 for no). However, due to the limitation of algorithm implementation, we linearly transform the decision variable to vector form $x \in \mathbb{R}^{pqr}$ ($x_i \in \{0,1\}\ \forall i = 1,2,...,pqr$). We first stack the columns vertically and replace elements of each index with a selection vector $x_i \in \mathbb{R}^r$. The elements x_{ij} of x_i have nonzero value 1 means that the jth kind of tile is selected for ith block of pixel ($i = 1,2,...,pq$). That is,

$$x = [x_1^T, x_2^T, ..., x_{pq}^T]^T, x_i = [x_{i1}, x_{i2}, ..., x_{ir}]^T, \forall i = 1, ..., pq$$

Linear Constraints I As mentioned before, we exactly place $[\frac{m}{l} \cdot \frac{n}{l}]/r$ copies of each tile. Then, we have

$$\textstyle\sum_{i=0}^{pq} x_{1+ir} = \sum_{i=0}^{pq} x_{2+ir} = ... = \sum_{i=0}^{pq} x_{k+ir} = \frac{pq}{r} \quad \forall k = 1,2,...,r$$

Linear Constraints II The second constraint is that we place only one (of the r many) tiles in block (i,j). As a result,

$$\textstyle\sum_{j=0}^{r} x_j = \sum_{j=0}^{r} x_{r+j} = ... = \sum_{j=0}^{r} x_{kr+j} = 1 \quad \forall k = 1,2,...,pq$$

Objective Function The objective of this problem is to design a mosaic that resembles the target image. Specifically, suppose we place a copy of tile k in block (i,j). We know the tile k has brightness c_k, and block (i,j) has brightness $\beta_{i,j}$, the cost for placing a copy of tile k in block (i,j) can be defined as the squared error $(c_k - \beta_{i,j})^2$. But since we want to formulate a linear problem, we define the error as the difference of brightness $|c_k - \beta_{i,j}|$ ($\forall k = 1,...,r, i \subseteq [1,p], j \subseteq [1,q],\ c \in \mathbb{R}^{pq}, i,j \in \mathbb{Z}$). Therefore, the function can be written as

$$\min \quad c^T x \quad \text{ where } c = [c_1, ...c_{pq}],\ c_i = ||c_k - \beta_{i,j}||_1\ \ \forall k = 1,...,r,\ i = 1,...,pq$$

Based on the discussion above, we can derive the formula for the linear optimization of mosaic generation task.

$$\begin{gathered}\min\ c^T x \\ \text{subject to} \\ A_1 x = b_1 \\ A_2 x = b_2\end{gathered}$$

$$\text{where } A_1 = [\mathbb{I}_{r\times r}\ \mathbb{I}_{r\times r}\ ...\ \mathbb{I}_{r\times r}] \in \mathbb{R}^{r\times pqr}, b_1 = (\tfrac{pq}{r})_{r\times 1},$$

$$A_2 = \begin{bmatrix} 1_{r\times 1} & 0_{r\times 1} & \dots & 0_{r\times 1} \\ 0 & 1_{r\times 1} & \dots & 0_{r\times 1} \\ \vdots & \vdots & \ddots & \vdots \\ 0_{r\times 1} & 0_{r\times 1} & \dots & 1_{r\times 1} \end{bmatrix} \in \mathbb{R}^{pq\times pqr}, b_2 = 1_{pq\times 1}$$

$$x_i \in \{0,1\} \quad \forall i = 1,2,...,pqr, c \in \mathbb{R}^{pqr}$$

The dual formulation is

$$\max\ [b_1; b_2]^T y$$
$$\text{subject to}$$
$$[A_1^T A_2^T] y \leqslant c,\ y \in \mathbb{R}^{pq+r}$$

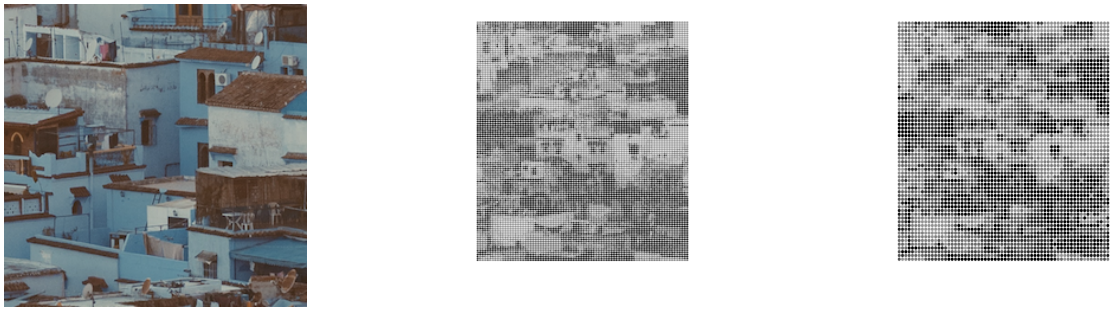
5.3 Results and Discussions

5.3.1 Matlab Implementation

To implement the linear optimization problem of mosaic generation, we use the **linprog**, the optimization toolbox in **MATLAB**, which uses the dual simplex method by default. In order to solve this problem in standard form, we first transform the pixel data of both tiles and image from matrix form to vector using the **reshape** function. For the RGB image, we use **rgb2gray** function to convert the color info in three-dimensional array to gray image data. Due to the requirements of perfect matching between tiles and images, we utilize **imresize** to truncate the images to smaller and proper sizes. When solving the optimization problem, a large scale matrix A_1 will be generated, which requires the **sparse** function to save storage space. Finally, we use **imread** and **imshow** to read the pixel data and show the image based on the values of pixel matrix.

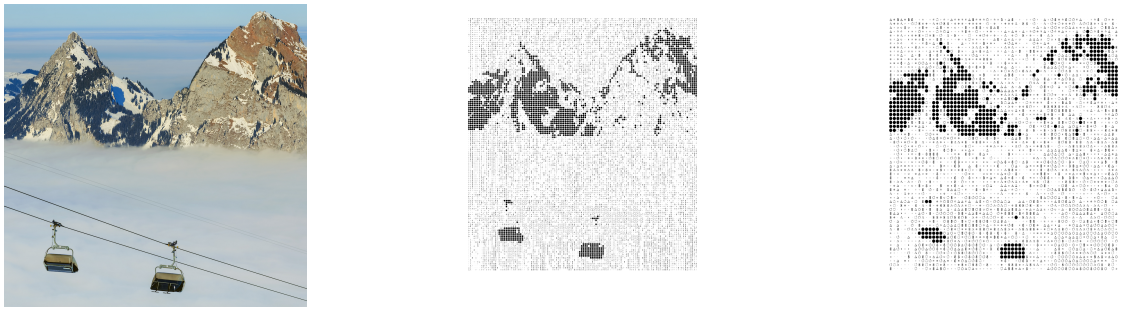
Also, it is noted that when implementing this linear program, we change the binary constraints $x \in \{0, 1\}$ to the continuous constraints $x \in [0, 1]$ due to the limitation that **linprog** does not support integer programming. However, the generated results showed that when the problem is optimal, x always is an integar (0 or 1). Besides, no degenerate solution is found in the optimal solution. It can be explained that for every block of pixel, the optimal tile is always unique, because every tile has unique value and one can always choose the most suited one. Also, since we have imposed the constraints that every block must match with one type of tile, the result of x is never degenerate for every block. Therefore, the obtained results are solutions of the original integer problem.

After implementing the codes, we test different sets of pixels and images and plotted the images. Some of them are shown in figures below and in appendix.



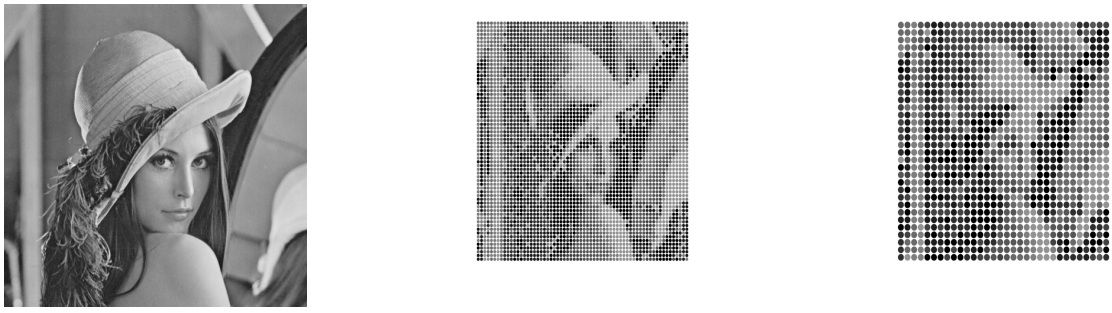
(a) (b) (c)

Figure 14: (a) truth image (256×256); (b) mosaic with 8 circle tiles of size 4×4; (c) mosaic with 8 circle tiles of size 8×8



(a) (b) (c)

Figure 15: (a) truth image (1024×1024); (b) mosaic with 8 symbol tiles of size 8×8; (c) symbol with 8 circle tiles of size 16×16



(a) (b) (c)

Figure 16: (a) truth image (512×512); (b) mosaic with 8 circle tiles of size 8×8; (c) mosaic with 8 circle tiles of size 16×16

5.3.2 Analysis I: Duality Gap in Program Runtime

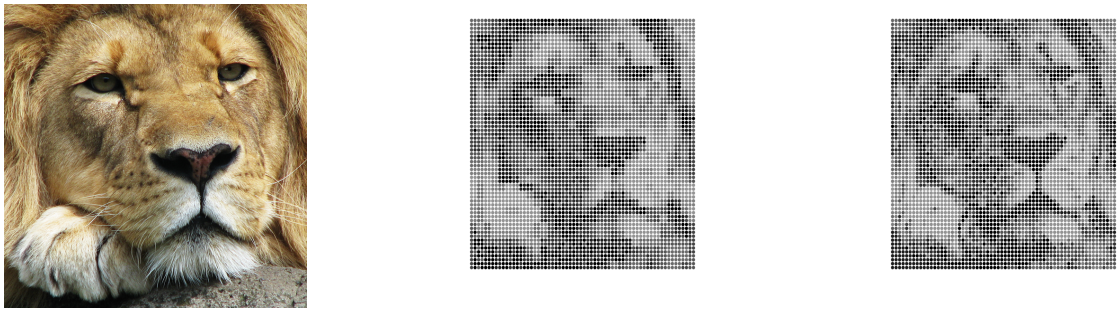
Tile Size	Primal	Dual	Duality Gap
8×8	2.2205s	7.4071s	5.1866s
16×16	0.36642s	0.45779s	0.09137s
Number of Iteration (8×8)	10407	18653	8246

Table 1: Runtime and Iteration Comparison of Figure 3(b)(c) in Primal and Dual Form

As for the runtime of the program, our important observation is that the solving the primal problem by dual simplex method is faster. And the smaller size the tiles have, the bigger the difference is between solving the problem by primal form and by dual form. We hypothesize that the runtime of the program heavily depends on the size of decision variables. For the primal problem, the variable x has $p \cdot q \cdot r$ entries, which has a very large scale. However, after we transform it to the dual form, the dimensions of the variable significantly drops, and is the same as the number of constraints $(p \cdot q + r)$ in the primal. And since the

number of dual constraints far outweighs the variable size, they become redundant and affects little on the runtime.

5.3.3 Analysis II: Comparison of Mosaic Quality between RGB and Gray Images



(a) (b) (c)

Figure 17: (a) truth RGB image (640×640); (b) mosaic generated from gray image; (c) mosaic generated from RGB image

In previous parts, we only investigate the mosaic generation problem based on gray images. In other words, we first transformed the target images into gray ones and compute their brightness and derive optimal solutions. But when we generate mosaics from RGB images, the result showed that the mosaics better recover main features of the original images. In this case, we incorporate the color information in the calculation of the brightness coefficients $\beta_{i,j}$ of U, i.e.,

$$\beta_{i,j} = \tfrac{1}{3}\times \text{ average in red dimension } + \tfrac{1}{3}\times \text{ average in green dimension}$$

$$+\tfrac{1}{3}\times\text{average in blue dimension}$$

After recomputing the problem, we found that the optimal solution changes and the images has slightly better adjustments. As shown in Figure 21, the mosaic generated from RGB image has more distinct structures and lines, while the one generated from gray image is more blurry in details.

5.4 Conclusion

In the mosaic generation task, we first formulate the task as a linear optimization problem. Then, we implement the problem in **MATLAB** by dual simplex method using both the primal form and dual form. Then we experiment with different images and tiles to test the feasibility of our algorithm. The result indicates that the generated mosaics can recover the main features of target image to a certain extent. We first observe that the efficiency of the program depends on which problem form we choose (primal and dual), and it is crucial to choose one that has smaller dimensions. Another important conclusion is that generating mosaic from RGB can better recover the target images.

5.5 Limitation

However, one serious limitation is that the quality of mosaics will vary among different tiles and images. We have presented many examples that generate suitable mosaics for the target images, but for certain images, the outcome is less satisfactory (one exmaple is in Figure 21). We infer that the brightness information will affect the quality of mosaics. When the target images are too dark or light, it is difficult to distinguish the features of the mosaic images. Also, using different tiles will generate mosaic of different quality. In general, the circle tiles generates mosaics of higher quality than the symbol tiles (one example is in Figure 20).

To improve the mosaic quality, we suggest formulating the task into nonlinear problem, i.e., minimize $(c_k - \beta i, j)^2$. In this case, the tile brightness will match more precisely with the image brightness. But the efficiency of the program is a great concern. In our attempt, **cvx** package in **MATLAB** is utilized to solve this nonlinear program but the long runtime is beyond our expectation. Besides, RGB mosaic tiles can be explored, because they retain the color information and will outweight gray tiles in recovering distinct features of color images.

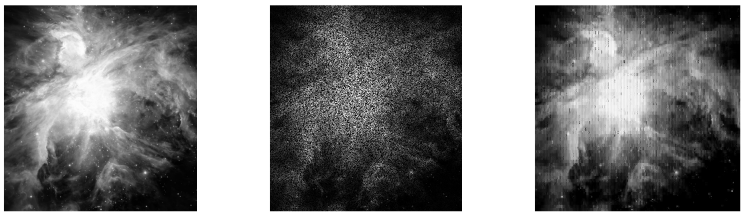
6 Appendix

6.1 Examples of Total Variation Minimization

(a) (b) (c)

Figure 18: (a): Original Image(eagle) (b): Damaged Image(eagle) (c): Reconstructed Image(eagle) by interior point in Total Variation Minimization

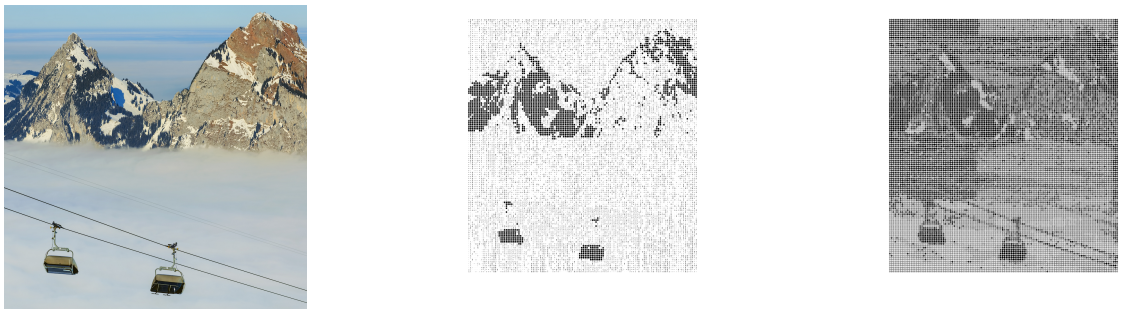
6.2 Examples of Sparse Reconstruction



(a) (b) (c)

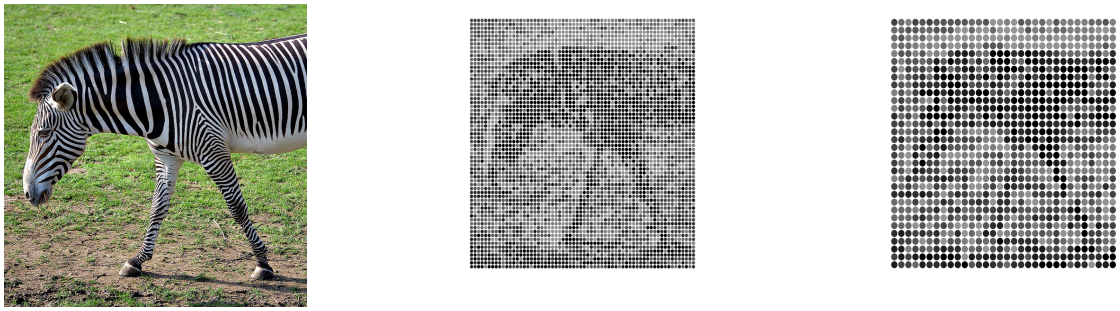
Figure 19: (a): Original Image(stars_01) (b): Damaged Image(stars_01) (c): Reconstructed Image(stars_01) by interior point in Sparse Reconstruction

6.3 Examples of Mosaic Generations



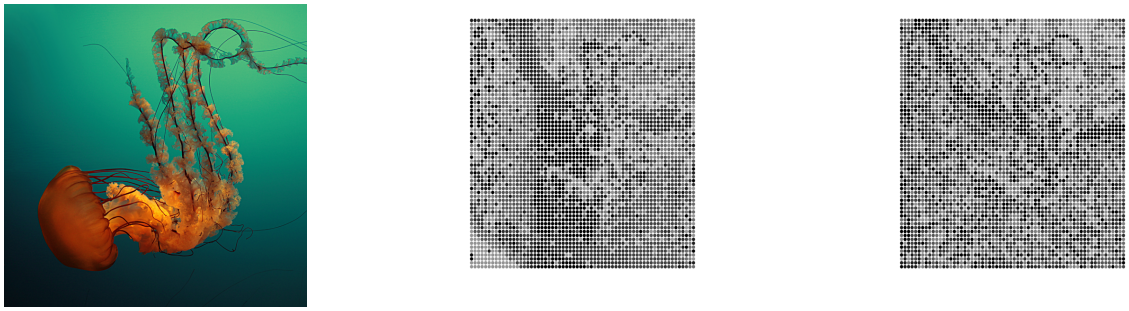
(a) (b) (c)

Figure 20: (a) truth image (1024×1024); (b) mosaic with 8 symbol tiles of size 8×8; (c) mosaic with 8 circle tiles of size 8×8



(a) (b) (c)

Figure 21: (a) truth image (640×640); (b) mosaic with 8 circle tiles of size 10×10; (c) mosaic with 8 circle tiles of size 20×20



(a) (b) (c)

Figure 22: (a) truth image (640×640); (b) mosaic with 8 symbol tiles of size 10×10 genereated from gray image; (c) mosaic with 8 symbol tiles of size 10×10 genereated from RGB image

6.4 Contribution

Yuncong, Cui (118010045): Part I-1 code, Part I-2 code, Part I report(50%)
Chengxi, Guo (11801080): Part II code, Part II report
Tianshuo, Yan (118010363): Part I report(50%)

6.5 Appreciation

Guanhua, Shao (118010252)
Chenxun, Zhao (118010436)
Appreciate the help of students above.