FAST INPAINTING ALGORITHM FOR REAL-TIME VIDEO INPAINTING PROBLEM

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Muxi Chen

Dr. Jeffrey, Uhlmann

May 2016

ACKNOWLEDGMENTS

I would love to take this opportunity to express my deep sense of thanks and gratitude to my academic advisor, Dr. Jeffrey Uhlmann for the continuous support of my graduate study and research. His patience, enthusiasm, and dedication have encouraged me in every step of my academic study here at MIZZOU. His guidance has helped me overcome many difficulties of my research and the writing of this paper. It is my great fortune to have such a wonderful advisor in the graduate study. Thank you very much! Professor!

I am grateful to my parents who have been supporting me since the beginning of my graduate study. Thank you for giving me the chance to study abroad and having such a beautiful life from my birth till now.

Last but not the least, I want to thank my girlfriend Jiajia Huang for her company and patience during my graduate study. Her concern and support were always encouraging me throughout my study.

Contents

ABSTRACT

The paper examines a simple and efficient method to solve the digital inpainting problem with a reasonale result by processing the information locally around the painting area. The method is based on a unique matrix transformation algorithm. It can guarantee transforming a non-negative matrix without rows and columns of all zero elements into another matrix with the same size but having both its column and row products equal to 1. The method is time and memory efficient so it can be used in many real time systems like video stream which may have protential inpainting problems.

Chapter 1

Introduction

1.1 Inpainting Problem

The process of reconstructing the damaged or corrupted parts of an image is called inpainting. Historically inpainting was performed by artists who would manually repair damage to paintings by painting over the damaged areas[34]. The goal is to fill in the damaged area in a manner that is consistent with the surrounding region of the painting or to achieve a restoration as similar as possible to the original picture so that the inpainted region cannot be detected by a casual observer[21]. Because digital images can be damaged during shipment or unwanted elements can be added such as commercial tags, effective and efficient digital inpainting algorithms are always needed by many different industries[21].

The term "digital inpainting" was first introduced by M. Bertalmío[20], who also presented a novel algorithm to the problem at the same time. Figure 1.1 is an example of of an image inpainting problem. The left image is an oil painting, and the right one is the image which was inpainted by professional artists.

There are many different applications of the inpainting problem. In the case of digital inpainting, applications include removing red-eye, unwanted objects or stamped date in an image, etc. For the film industry, the inpainting application is used for film restoration which may include recovering the damaged parts of film frames like cracks or irrelevant parts in images like hanging wires in action movie frames[34]. Many of the digital inpainting processes can be completely finished by computers automatically. However, different inpainting algorithms may need some human interaction to improve the inpainting results. For many inpainting problems, the algorithms can

provide very natural inpainting results. However, the digital inpainting process is still very limited when the missing or damaged parst in an image are enormous under some circumstances. This is especially true when the missing area's surrounding geometric structure is very complicated. When filling large missing areas, the digital inpainting algorithms may generate distracting artifacts in the image and lead to a bad inpainting result. Figure 1.2 shows an example of having a damaged part of a digital image. Black square in the image represents the missing block. The mission for the digital inpainting algorithm is to reconstruct the damaged or missing parts of the corrupted image.

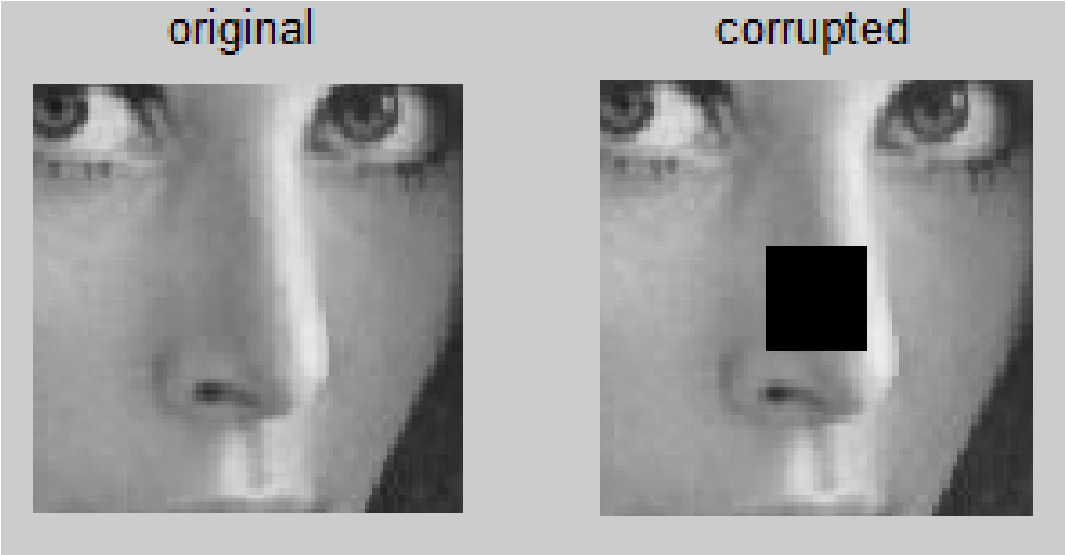


FIGURE 1.1: An example of digital inpainting problem

1.2 Related works

Many different types of inpainting algorithms have been developed. Both this section and the following section briefly introduces structural inpainting algorithms, texture synthesis inpainting algorithms and fast inpainting algorithms. For other types of inpainting algorithms, detailed information is in Chapter 2.

For structural inpainting algorithms, the main idea is to use information from the surrounding pixels of the damaged part to reconstruct the missing area. By using diffusion, structural inpainting algorithms extend lines of equal luminance values from the known area into the missing area, so as to preserve the consistency of the geometric structure of the missing area. The main drawback of this type of inpainting algorithms is that the newly created parts often contain blurring artifacts, which can become a huge distraction to viewers. Examples of structural inpainting algorithms include Chan and Shen's TV (total variation model model)[19] model which extended to the CDD (curvature driven diffusion)[4] model later. These methods can provide reasonable results when the missing area in the image is small. However, the structural inpainting algorithms will not reproduce the large textured regions well[21]. Moreover, the inpainting process for this kind of algorithms is solving the partial differential equations(PDE) generated from the missing pixels and their surrounding known pixels. Therefore, the algorithms will generate blurring artifacts within the inpainting areas as shown in Fig 1.3.

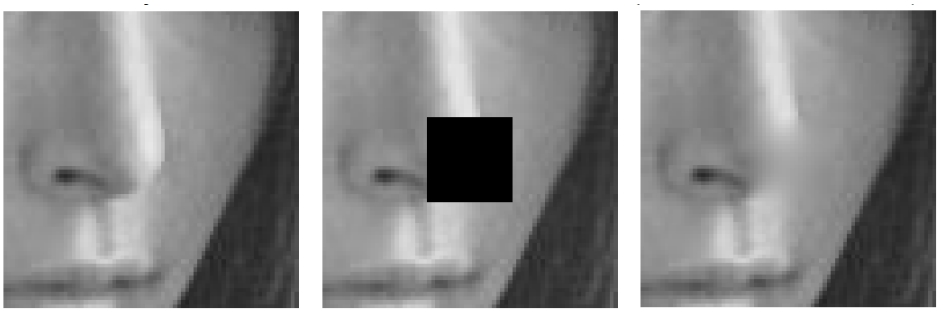


FIGURE 1.2: The above images are from left to right: the original image, the corrupted image, and the inpainting result of the structural inpainting algorithm. We can see the blurring artifacts within the target area after the inpainting process.

Texture synthesis inpainting algorithms try to fill in the damaged parts with information extracted from the whole image. By detecting and identifying the similar texture with the missing area in the entire image, texture synthesis inpainting algorithms can solve some inpainting problems with large missing areas and provide plausible results at the same time. However, the algorithms can not handle the natural images when they are composed of structures with edges. For the images which have complex textures, the algorithm may require the user to specify what texture should be used during the inpainting process[21]. Also, the algorithms are very time-consuming and therefore, they are not suitable for solving the inpainting problems that need to be addressed in a very short amount of time.

1.3 Fast Digital Inpainting

Due to time constraints, the structural and textural inpainting algorithms are not suitable for inpainting problems that need to be solved in seconds or even milliseconds, such as inpainting a video frame from a real-time video chat. For solving the inpainting problem that needs to be addressed in a limited time, fast inpainting algorithms were developed. These include Manuel M. Oliveira, Brian Bowen, Richard McKenna and Yu-sung Chang's fast digital inpainting algorithm [23] as well as Alexandru Telea's fast marching inpainting algorithm[29]. When inpainting the corrupted real-time video frames, these algorithms may provide results that are not as good as those made possible by more sophisticated algorithms such as the TV model inpainting algorithm, but they can finish the inpainting task quickly enough to avoid creating latency or interruptions for the viewer of the video. At the same time, the fast inpainting algorithms are reliable enough to not produce terrible results which contain lots of artifacts that may distract the users. This thesis presents a fast digital inpainting algorithm based on interpolating the damaged parts' surrounding pixels' horizontal and vertical structural information. The algorithm's computational time complexity is linear to the number of missing pixels in the target area. Meanwhile, the algorithm can provide almost the same or even better inpainting results for some cases when compared to the PDE based structural inpainting algorithm.

1.4 Thesis Outline

Chapter 2: In this chapter, I will provide more detailed information about the approaches for solving inpainting problem. We will introduce the general ideas of those inpainting algorithms and meanwhile, discuss the advantages and the drawbacks of the algorithms.

Chapter 3: This chapter illustrates the need for fast inpainting algorithms which may not provide perfect results but are sufficient for real-time applications such as block interpolation for corrupted frames in video streams. Finally, a description is given of the matrix scaling problem and its solution. The chapter ends with a full description of specific algorithms created in this research. This chapter will also provide examples showing the algorithm solutions.

Chapter 4 gives more examples of this research's novel algorithm's inpainting results. The chapter shows how to inpaint the missing areas with different textures and demonstrates the algorithm's performance and its processing time. Then I will compare the results of my algorithm with the results of existing algorithms.

Chapter 5 summarizes the problem addressed, the solutions developed, the experimental results obtained, and discusses what kind of future work can be pursued to possibly improve the novel algorithm presented herein.

Chapter 2

Background

This chapter gives the detailed information about the digital inpainting algorithms, which includes an overview and describes the process or the methodology for inpainting algorithms. The discussion will cover situations where particular algorithms can produce natural inpainting results and situations where the algorithms do not handle the inpainting problem well. The algorithms are categorized by the following section titles:

- 1: Structural Inpainting
- 2: Texture Synthesis Based Inpainting
- 3: Exemplar Based Inpainting
- 4: Hybrid Inpainting
- 5: Fast Inpainting

Each section includes a discussion and will refer to figures from several papers which illustrates the general idea and the process of the algorithm. Some sections provide examples of the algorithms' inpainting results. Some algorithms will receive a more detailed analysis than others. The original algorithm featured in this thesis solves inpainting problems with small missing areas at a high speed. Furthermore, it only uses information from nearby pixels around the missing area, which is similar to structural and fast inpainting algorithms. Therefore, this thesis will introduce the structural and fast inpainting algorithm detailly and compare the algorithm's inpainting results with structural and fast inpainting algorithms later in Chapter 4 later.

2.1 Structural Inpainting

The structural inpainting algorithms are also called partial differential equation (PDE)-based inpainting algorithms because their inpainting processes are solving the PDEs generated from the missing pixels and their surrounding known pixels. The ba-sic idea of this kind of algorithm is to preserve the geometric structure's consistency of the missing area by propagating the linear structures (edges) from the known area called isophotes to the missing area [22].

Bertalmio et.al [20] has developed the first PDE based inpainting algorithm. The algorithm iteratively updates the output image by adding rational information gained from the surrounding geometric structure. The following equation demonstrates the general idea of Bertalmio's [20] algorithm.

$$I^{n+1}(i,j) = I^{n}(i,j) + \Delta t I_{t}^{n}(i,j), \forall (i,j) \in \Omega \tag{2.1}$$

(i,j) are the coordinates of the missing pixel. Ω represents the set of all missing pix-els. And the $I^{n}(i,j)$ denotes the corresponding output pixel's intensity value at the it-eration n. The Δt represents the improvement rate and the $I_{t}^{n}(i,j)$ is the improvement upon the image. Then as the n increases, the algorithm will produce a better inpainting result as long as the $I_{t}^{n}(i,j)$ is an improvement instead of the wrong information which may make the inpainting result worse. The algorithm defined the $I_{t}^{n}(i,j)$ as follows:

$$I_{t}^{n}(i,j) = \bigtriangledown(\triangle I^{n}(i,j)) \cdot |\bigtriangledown I^{n}(i,j)| \tag{2.2}$$

The $\bigtriangledown(\triangle I^{n}(i,j))$ represents the Laplacian smoothness of the gradient and the no-tion $|\bigtriangledown I^{n}(i,j)|$ denotes the isophote direction. In conclusion, the algorithm is to propagate information from the surrounding areas in the isophote direction smoothly. But this algorithm is not suitable for filling large missing areas. And due to the blurring artifacts generated by the diffusion process, the algorithm will cause distractions to the viewers of inpainting results. Also, the algorithm lacks explicit treatment of the pixels on edges[22].

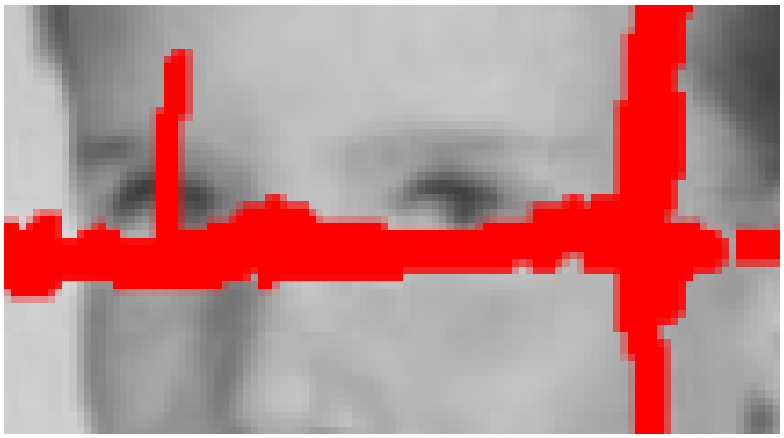


FIGURE 2.1: Images referenced from Caselles et al.[20]. The red parts in the image are missing area.

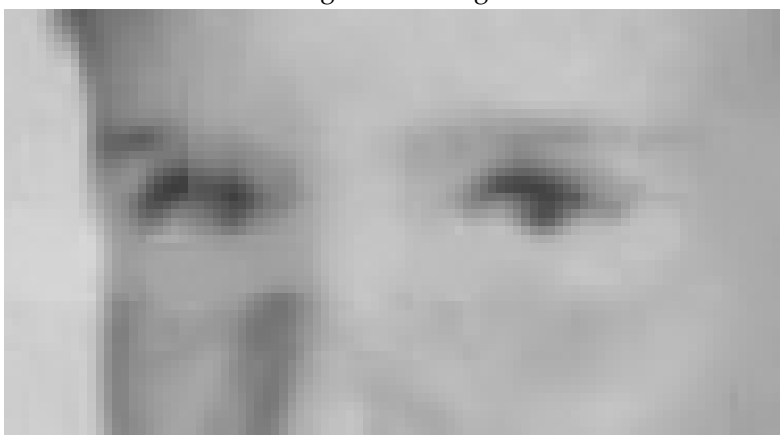


FIGURE 2.2: The inpainted version of the Figure 2.1, as we can see there are certain blurring artifacts in the inpainted area.

Later on, inspired by Bertalmio's algorithm, Chan and Shen proposed the Total Variational (TV) inpainting model[3] which is closely connected to the classic total variation (TV) denoising model of Rudin, Osher, and Fatemi [19]. In Chan and Shen's paper, they have defined three principles for a practical inpainting scheme based on the fact that Images (even non-texture ones) are deterministically not smooth functions since they contain edges and discontinuities. Furthermore, images are often statistically corrupted by noise. The three principles require that 1) the model is local, 2) the model is able to restore narrow and broken smooth edges and 3) that the model is robust to noise. Based on these principles, they developed the algorithm using the Euler-Lagrange equation and anisotropic diffusion based on the strength of the isophotes.

The TV model defines D as an inpainting domain and E as any fixed closed surrounding domain of D. The boundary lies between the two domains. The algorithm is trying to find a function u on the extended inpainting domain such that it minimizes an appropriate $E \cup D$ regularity function.

$$R[u] = \int_{E \cup D} r\left(|\bigtriangledown u|\right) dxdy \tag{2.3}$$

Here the r is an appropriate real function which is non-negative for non-negative inputs. The algorithm works well on fixing small corrupted areas, and it is robust to noise. But it has some drawbacks including an inability to handle large textured missing areas well because the algorithm only consider the surrounding information instead of the global information. Meanwhile, the algorithm may fail to connect edges or objects under some circumstances. The following figure is referenced from Chan and Shen[3] which illustrates the situation.

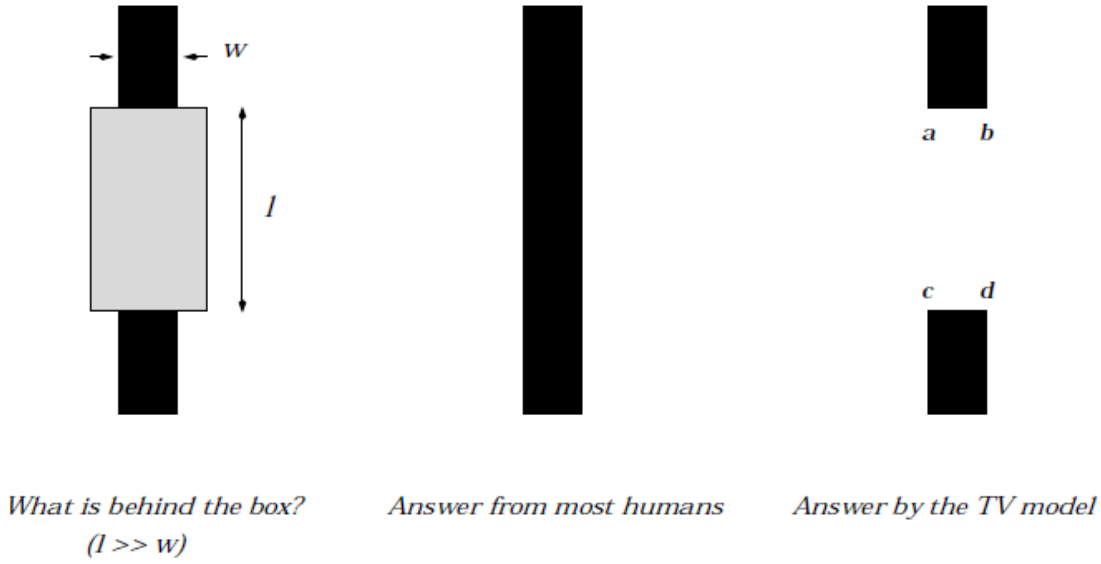


FIGURE 2.3: Images referenced from Chan and Shen's paper [3]. From this image we can see that under this situation the TV model is not providing the preferable result.

To solve the above problem, Chan and Shen have extended the TV model into the CDD (Curvature-Driven Diffusions) model[4]. The algorithm applies information from the curvature of the isophotes into the original TV model and thus can handle the curved structures better[21]. The Euler–Lagrange equation for the original TV model's regularity function $R[u]$ is shown below:

$$-\nabla.(\frac{\nabla u}{|\nabla u|}) = 0 \tag{2.4}$$

The above equation has been modified since the original TV model also considers the noise effect on the input image. However, this part is not relevant to our topics, so we are not going to analyze with detailed information there.

Chan and Shen changed the TV model by rewriting the equation (2.4) as:

$$-\nabla.(G(k,x)\frac{\nabla u}{|\nabla u|}) = 0 \tag{2.5}$$

where k represents the $\nabla.(\frac{\nabla u}{|\nabla u|})$ which is the scalar curvature at pixel x. And the $G(k,x)$ equals to 1 when x belongs to D. Otherwise the $G(k,x)$ equals to $g(|k|)$. The function $g(s)$ in this case is similar to the function r mentioned aboved, which is non-negative for non-negative inputs. The original paper introduced an example and the $g(s) = s^a$ for some $a >= 1$. With this improvement, the CDD model can extend the bent level lines inside the inpainting domain. The CDD model can also output connected objects, thereby fullfilling the connectivity principle[4].

Other models that based on the TV model, include Pascal Getreuer's TV model[12] which uses the split Bregman. In addition, other PDE-based inpainting algorithms, which are different from the TV models includes Tschumperle, and Deriche's algorithm presented in [30].The majority of these PDE-based inpainting algorithms are very time consuming and can not handle large or texture-missing areas well. Meanwhile, the blurring artifacts generated during the inpainting process can be a distraction to users. Therefore, the algorithms may not produce a good inpainting result even when the missing area is small.

2.2 Texture-synthesis-based Inpainting

The general idea behind texture-synthesis-based inpainting methods is to use the similar neighborhoods of the damaged pixels to inpaint the target regions. The algorithm first chooses an initial seed and then synthesizes the damaged image pixels according to the seed [21]. Then the algorithm tries to preserve the geo-structure between the missing area and its surrounding area.

Earlier inpainting techniques utilized these methods to fill the missing region by sampling and copying pixels from the neighboring area. For example, Efros and Leung used the Markov random field (MRF) to model the local distribution of the pixel in [9]. Then they synthesized the new texture by querying existing texture and finding all similar neighbourhoods. Later, Efros, Alexei A., and Freeman, William T. developed the technique into fast synthesizing algorithm [10] which is called image quilting. This method works by stitching together small patches of existing images, hence the term "quilting."

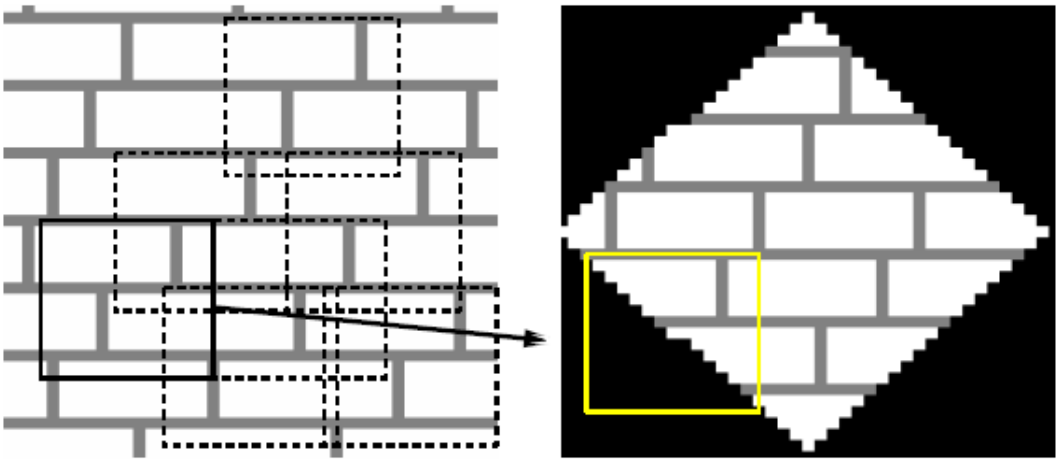


FIGURE 2.4: Images referenced from [9]. They are showing the general idea about Efros, and Leung's algorithm. Given a sample texture image(left), a new image is being synthesized one pixel at a time (right). To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image (box on the left) that are similar to the pixel's neighborhood (box on the right) and then randomly chooses one neighborhood and takes its center to be the newly synthesized pixel.

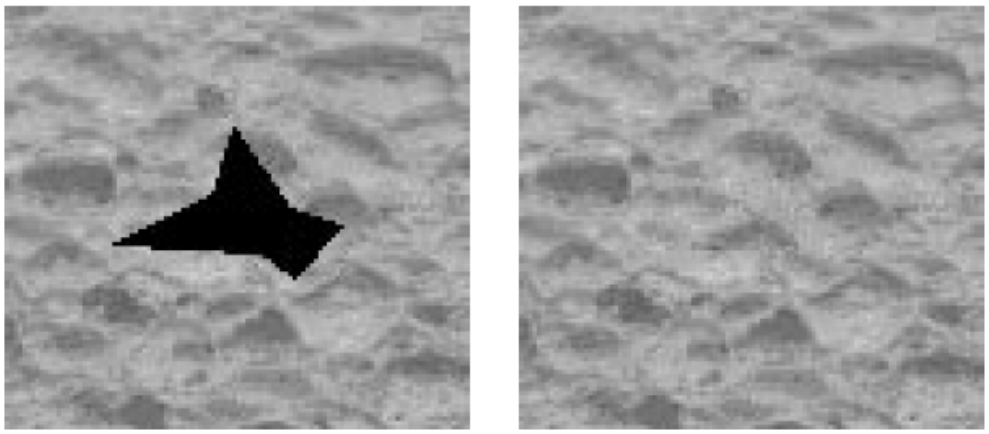


FIGURE 2.5: Images referenced from Efros, and Leung's paper[9]. This is an inpainting result of the texture-synthesis-based inpainting algorithm

Other examples of such algorithms included [13],[15],[1]. The differences between these algorithms mainly depend on the definition of continuity and how to process with the continuity between the missing area and its surrounding pixels.Texture synthesis based inpainting algorithms only work well with a certain set of images. This kind of images has homogenous texture information around the missing area. Without this premise, using texture synthesis based inpainting algorithm may not generate a desirable result.

The general idea of texture synthesis based inpainting algorithms is to use the texture information from the image and fitting the information into the missing area. It's not simply patching particular part of the image into the area. This kind of algorithm can be divided into three different categories: 1) statistical (parametric), 2) pixel-based, and 3) patch-based (non-parametric)[21]. Unlike the PDE-based methods, texture-synthesis-based inpainting algorithm can handle large missing areas in an image. They work well with approximating textures but having difficulty processing natural images which may have lots of edges and complex interaction between structure and texture boundaries. Also, the algorithms are time-consuming and as a result, they can only deal with certain kinds of inpainting problems and are not applicable to the real-time streaming video inpainting.

2.3 Exemplar-based Inpainting

The basic idea behind exemplar-based inpainting algorithms is to assign the priority of the missing pixels' inpainting order, then select the best matching patch among the known regions, and finally paste the matching patches into the missing region. The similarity between the missing region and the known regions is determined by a certain metric. This kind of algorithms has proven to be effective and has become an important member of the inpainting algorithm categories[21].

In general, the exemplar based inpainting algorithms iteratively update the pixels in the missing area according to the previously assigned priority. While filling the target region, the algorithms will consider the geo-structure between the target region and its neighbouring regions. Many different exemplar based inpainting algorithms have been developed so far, and they can often achieve good results in solving inpainting problems with large missing areas. Drori, et al. [8] presented an algorithm that iteratively updates the missing region with fragments generated by searching and approximating the known regions. However, this algorithm is very time-consuming. The computational time is quadratically related to the number of pixels in the missing regions. Criminisi et al. [5], proposed an algorithm which tries to combine the advantages of the texture and PDE methods. The algorithm has the ability to repeat two-dimensional patterns with some stochasticity and the ability to handle lines and boundaries of objects. The filling order or the priority of the missing pixels is determined by the pixels' positions in the image. The inpainting process starts at the places where the strength of nearby isophote is strong. These places are often the boundaries of the missing areas. The algorithm will then select the best matching patch among the candidate source patches by comparing the sum of squared differences (SSD).

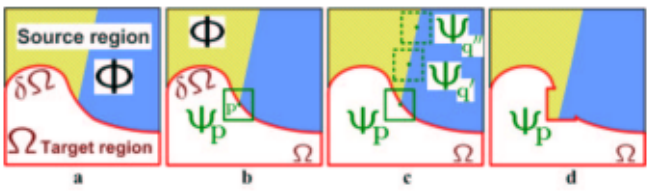


FIGURE 2.6: Images referenced from [5]. Demonstrating the filling process of Criminisi et al.'s algorithm

Exemplar based inpainting algorithms' performance is often affected by the mechanism which decides the pixels' priority. A wrong or inappropriate filling order may lead to an unacceptable inpainting result. Also, the patch selection is crucial in such algorithms, and researchers have made great progress in this area. Wong [37] proposed a way of reconstructing the filling patch by using a weight similarity function among several known patches instead of using only one source patch.

With the same basic exemplar-based inpainting algorithms' frame, which sets the priority and then selects the similar patch to paste, many exemplar inpainting algorithm differences mainly lie in two areas. The first area is the definition of the filling order and the second area is the method used to search and reconstruct the exemplars. Komodakis[18] defined a global objective function, which attempts to minimize the cost. This method can solve certain inpainting problem effectively, but it is relatively time-consuming. Wu[17] presented a cross isophotes exemplar-based model. Wu's algorithm uses the cross-isophote diffusion data and the local texture information to decide the dynamic size of the exemplars. Sun [28] proposed a method that needs human interaction. First, the algorithm needs to draw main curves manually to inpaint the missing structure. After that, it can fill the missing area using texture propagation. Fang [11] developed a rapid image inpainting system that adopts a multiresolution approach, which improves the convergence rate of the synthesis process and enables the algorithm to handle large missing regions well.

For finishing the inpainting task, all of the above algorithms try to find useful information from the known regions within the original image. Hays et al. [14], introduced the idea that inpainting algorithms can fill the missing area in a particular image with information gathered from many other different but similar images. The algorithm first matches the similar images with the target image, then uses the Poisson image editing[24] to mix the missing part with a certain part of the best match image.

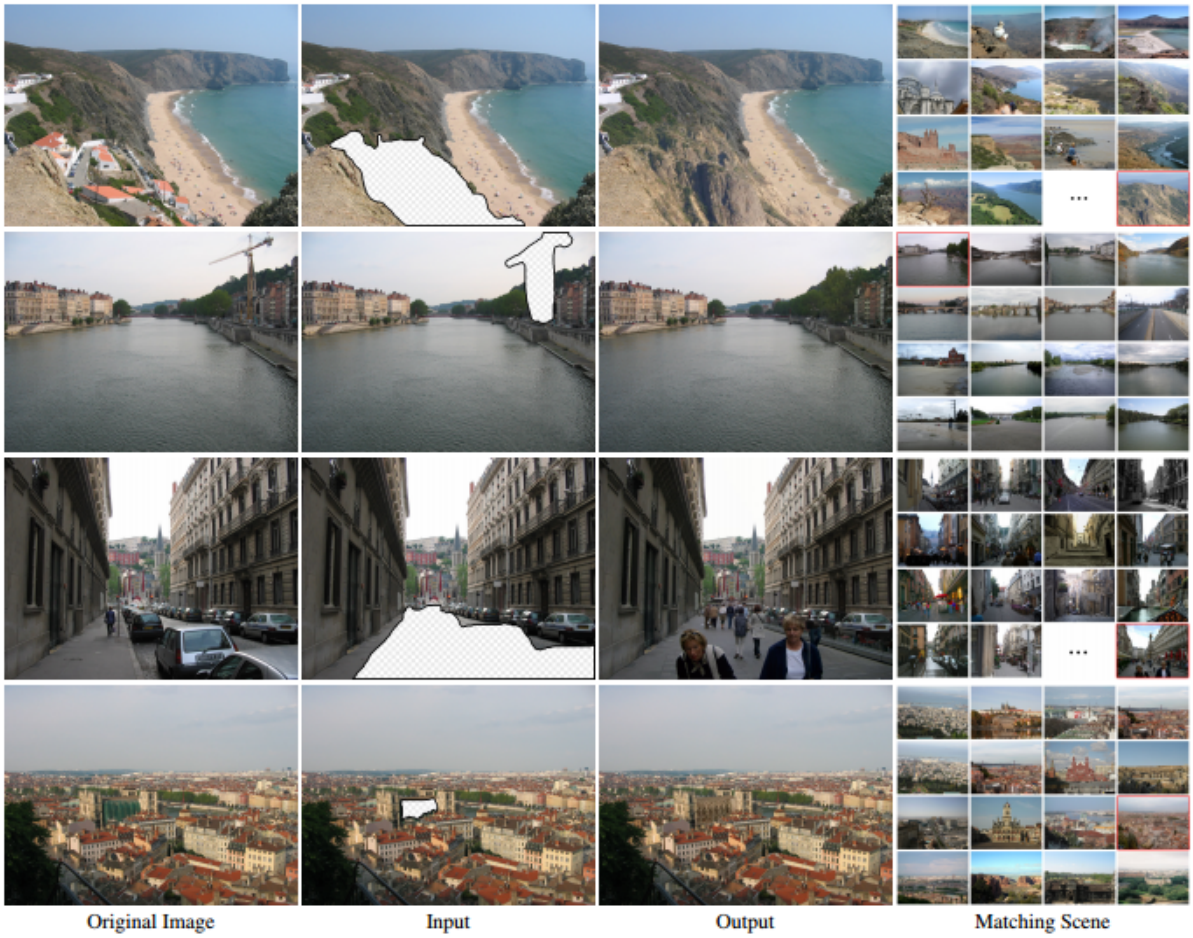


FIGURE 2.7: Images referenced from [14]. The images are demonstrating how the algorithm uses many different images as the information source to inpaint the target image.

The exemplar based inpainting algorithms can perform pretty well in filling large missing areas, and they can generate natural results without many distracting artifacts. However, different algorithms of such kind may only perform well in particular kind of images. The filling order of the algorithms and the patch selecting mechanism play critical roles in the inpainting process. For example, choosing a bad patch from the target image can produce an unacceptable result. Also, this condition can occur possibly because the patch generating mechanism cannot handle every situation of the image inpainting problems. Also, the exemplar inpainting algorithms are very time-consuming. Therefore we can not apply this kind of algorithm to the inpainting problems that need to be addressed in seconds or even milliseconds.

2.4 Hybrid Inpainting

Hybrid inpainting algorithms refer to the kind of algorithms which combine two or even more different algorithms to solve the inpainting problem. The idea is to try to utilize and combine the advantages of various inpainting algorithms to generate better results.

Hybrid inpainting algorithms often combine the PDE based methods and the texture synthesis methods. They can preserve both geo-structure and texture information and make the target image visually acceptable especially when filling a large missing area. These algorithms decompose the image into two separate parts, structure region and texture regions[26]. While filling the structure region, the hybrid algorithm may use the edge propagating algorithms. On the other hand, while filling the texture regions, texture synthesis techniques will be applied.

2.5 Fast Inpainting

Fast inpainting algorithms can be applied to inpainting problems with small missing regions. These algorithms can achieve good results within a few seconds while solving particular inpainting problems. Fast inpainting algorithms use the local information from the surrounding pixels of the missing areas to finish the inpainting task. When inpainting small areas, local pixels are often enough to provide the needed information.

Oliviera et al.[23] presented a fast digital inpainting algorithm. The algorithm considers the missing area as Ω and its boundary as $\alpha\Omega$. The inpainting procedure of the algorithm is approximated by an isotropic diffusion process that propagates information from $\alpha\Omega$ into Ω. The isotropic diffusion process is done by a diffusion kernel function which is a weighted average kernel function that only considers information from the surrounding pixels(zero weight at the center of the kernel). The following images can well demonstrate the inpainting process and the kernel function.

```
initialize Ω;
for (iter =0; iter < num_iteration; iter++)
  convolve masked regions with kernel;
```

a	b	a
b	0	b
a	b	a

c	c	c
c	0	c
c	c	c

FIGURE 2.8: Images from [23]. The image on top is the procedure of the algorithm. The image below shows the two diffusion kernels used with the algorithm. a = 0.073235, b = 0.176765, c = 0.125.

Oliviera et al.'s inpainting algorithm may cause blurring artifacts when the inpainting area is somewhere between certain high contrast edges. This is evitable by adding human interactions in the algorithm that define the diffusion barriers, which are the boundaries for the diffusion process inside Ω. The diffusion barrier is a two-pixel wide line segment inside Ω. As the diffusion process reaches a barrier, the process stops and sets the current pixel with a particular color.

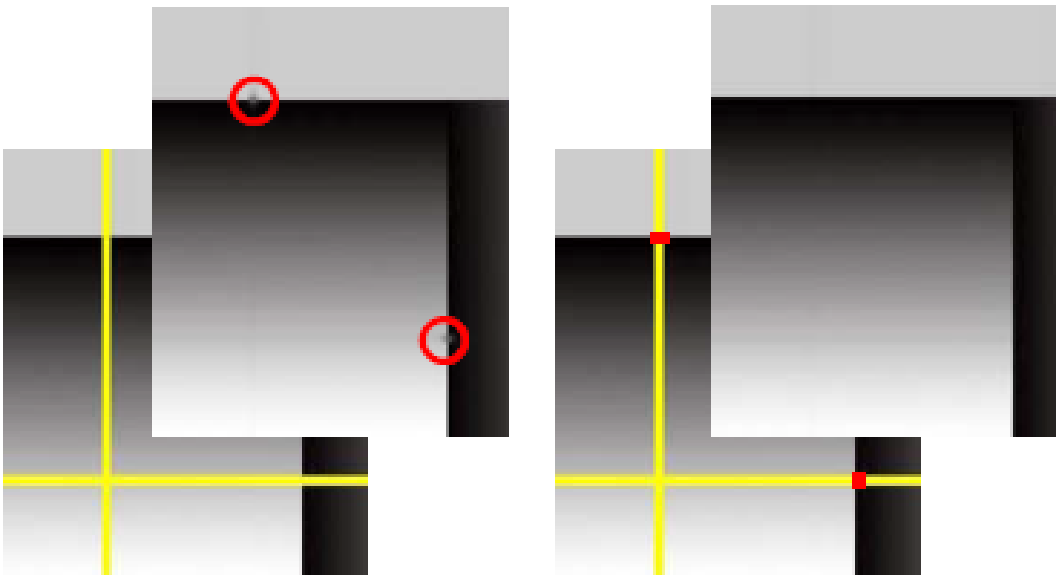


FIGURE 2.9: Images from [23]. We can see the blurred spots on the front left image produced by the simple diffusion based inpainting algorithm. After applying the diffusion barriers(front right), the diffusion process stops mixing information from both sides of the mask at the boundary.

Another fast digital inpainting algorithm was developed by Telea[29]. The method treats the missing regions as level sets and uses the fast marching method (FMM) to propagate image information. Telea believes that by continuing the isophotes (lines of equal gray value) as smoothly as possible from outside of the inpainting area into the inside of the area, one can get visually acceptable results. The following image will show the basic concept of Telea's algorithm.

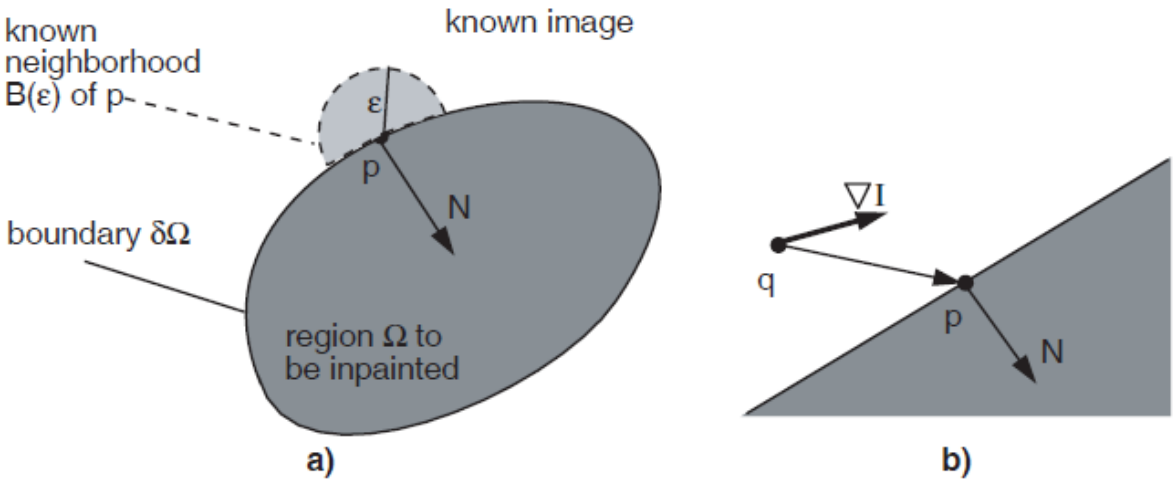


FIGURE 2.10: Illustration of Telea's algorithm

The mentioned two fast inpainting algorithms can process the corrupted image within a few seconds, and generate acceptable results at the same time. However, they all have drawbacks under certain conditions. As noted, the algorithm developed by Oliviera et.al may cause blurring artifacts while the missing area reaches high contrast edges without human interaction. The chance of this situation occuring will become much more likely when the missing area's size grows. Sometimes even a small missing area may encounter many edges in the image. Frequently asking for human interaction is not desirable for the inpainting algorithms. Since both algorithms are targeting small missing areas, they can not handle large missing areas. Moreover, when facing the missing area with complex texture, even the missing area itself is small, the blurring artifacts will be introduced into the inpainting results and become obvious distractions to the viewer.

Chapter 3

Line-Product Transformation for Real-time Video Inpainting

In this chapter, we will explain the problem of real-time digital video inpainting. Then we will show the need for an algorithm that can solve a problem fast enough under real time conditions to produce acceptable results without very distracting artifacts.

Later we will illustrate detailed information about how to apply line product constraints to solve the inpainting algorithm. We will discuss the time complexity of the algorithm which indicates the reason this algorithm is suitable for real-time video inpainting. This algorithm is targeting those inpainting problems with small missing areas, and it uses only local information from the surrounding area of the missing area to complete the inpainting task.

The general purpose of this algorithm is to extract the vertical and horizontal scaling information from the surrounding region of the missing area in the image, after which the algorithm can apply a simple inpainting method to infill the missing area and use the scaling information we extracted to recover the missing area. Three different exmaples of the simple filling method during the progress are given and the inpainting results are shown for each one. The best simple filling method will be used in the overall inpainting algorithm and produces the results to be discussed in Chapter 4.

3.1 Real-time Digitial Video Inpainting Problem

3.1.1 Bit Error

In telecommunications, the transmission process task is to send and transport an analogue or digital information signal. The transmission medium can be physical point-to-point or point-to-multipoint, including wired, optical fiber or wireless media[36].

The bit error problem is that during the transmission process of either analogue or digital signal, affected by unknown factors, the bit sequence on the receiver side may be altered and particular bits in the sequence can be different from the original sequence of the sender. The factors that may cause the bit error problem include transmission channel noise, interference, distortion, bit synchronization problems, attenuation, wireless multipath fading, etc.

Bit error rate (BER) is the number of received bits of a data stream over a communication channel that has been altered due to noise, interference, distortion or bit synchronization errors[32].

For example, consider the following bit sequence before transmission:

0 1 1 0 1 0 0 1 1 1 0 0

The next is the received bit sequence after transmission:

0 1 1 0 0 0 0 0 1 1 0 1

From the above example, we can know that during the transmission, the 5th, 8th and 12th bits are altered and the BER for this example is 25%. The BER shows that the frequency of a packet or other data unit has to be retransmitted due to bit error. While the bit error rate is high, packet lost often occurs and may cause a delay in the receiver. By then it depends on the network protocol of the transmission to decide whether to ask the sender to resend the data or just ignore the corrupted data just received.

3.1.2 Communication Protocols for Streaming Media

A communication protocol is a set of rules that determine how two or more devices transmit information between each other. The protocols define many different aspects of the transmission process including the syntax, semantics and synchronization of communication and error recovery methods. Furthermore, the protocols can be implemented by software, hardware or both[33].

For streaming media, if the data is lost or corrupted during transmission, communication protocols without error correction mechanisms may suffer from dropout, which is an instant loss of signal usually caused by noise, propagation anomalies, or system malfunctions. Under the situation, the receiving application can choose to detect loss or corruption and recover data using error correction techniques [35].

On the other hand, reliable protocols, which have the function of guaranteeing correct delivery of each bit in the media stream, often implement a system of timeouts and retries. The systems using such protocols are more complex to implement. When data are lost during the transmission process, the media stream will pause for the protocol to detect the lost packets and wait for the server to retransmit the packets, which may cause a lag in the video transmission. Clients can try to solve this by buffering data for a particular length of time and then show the buffered part of the video. The buffering mechanism has a drawback that it will cause the users of interactive applications such as real-time video chat to experience a loss of fidelity if the delay is over 200 ms[35].

The problem of missing data in the live video stream caused by bit error may generate distracting artifacts in the received video frames. Meanwhile, for many real-time streaming media transmissions, the applied communication protocols including real-time transport protocol (RTP) usually do not have any system of timeouts and retries as an error correction mechanism. Therefore, to solve the packet loss problem in a real-time environment, when there is no option for the receiver to wait for the server to retransmit the lost packet, only the receiver application can use some methods to complete the error correction tasks or it will simply use the corrupted data.

3.1.3 Data Compression and Real-time Inpainting Problems

Before transmission, streaming video data is often compressed by using a video codec such as H.264 or VP8. After compression, the transmission will require less bandwidth. However, this technology has its disadvantages in that data after compression are sensitive to errors and only a few bits error can lower the quality of the video significantly. Studies have been done to show that even a tiny bit error rate (BER) can lead to a bad result for a transmission. It is suitable for the transmission when the BER is smaller or equal to 10^{-5}. For video application which has high temporal information, this level of bit error will not provide acceptable transmission results[16].

FIGURE 3.1: Image referenced from [6]. The first image is the original image on the server side and the second image came from the receiver. During the transmission process, the bit error rate was 0.0001 (equivalently 0.01%). The result shows that even with a very small BER, the received image can still have a low quality.

Transmission results also depend on the compression algorithm, hence, only a single bit error can cause the video frames or images to generate very noticeable artifacts. When performing block-based coding for quantization in the compression process, as in JPEG-compressed images, packet loss during the transmission will not cause artifacts on the entire image. However, it will ruin blocks of pixels in a particular area. For example, the following image shows that a few bit errors can cause a large block of pixels to display in a very different way.

FIGURE 3.2: This image from [2] shows that several bit errors can lead to obvious artifacts in the image if the image data are compressed before the transmission process.

For real-time transmission protocols which guarantee delivery of correct data, they can solve the problem of artifacts existing in video frames which are generated by packet loss. These protocols can wait for a certain amount of time and buffer the data until it is correctly transmitted. However, as mentioned in the previous section, if the waiting time is over 200ms, the user will experience a loss of fidelity.

Meanwhile for those real-time transmission protocols without any error correction mechanism, the problem can only be solved by the corresponding applications' error correction methods. One option for the error correction is that applications can use the received image information to handle the artifacts in the video frames. When the corrupted parts in the video frame are relatively small and distributed separately, it is possible for the application to recover the frames from the artifacts in a very short time. Therefore, the presented problem has become an image inpainting problem, where the artifacts generated by the loss packets or bit errors are the target areas in need of inpainting.

Since this inpainting problem happens during the transmission process of a video stream, the algorithm for solving the problem must be very efficient and able to address the problem in linear time complexity. Meanwhile, the algorithm should handle small inpainting areas well and not generate new artifacts that may potentially cause the inpainting area to be more noticeable to the viewer.

The algorithm presented in our thesis can solve the mentioned inpainting problem in linear time complexity, which means the time complexity only depends on the number of corrupted pixels in the video frames or images. More detailed information about the algorithm is in the following sections. The image below is an example of applying our algorithm to the image from [2].

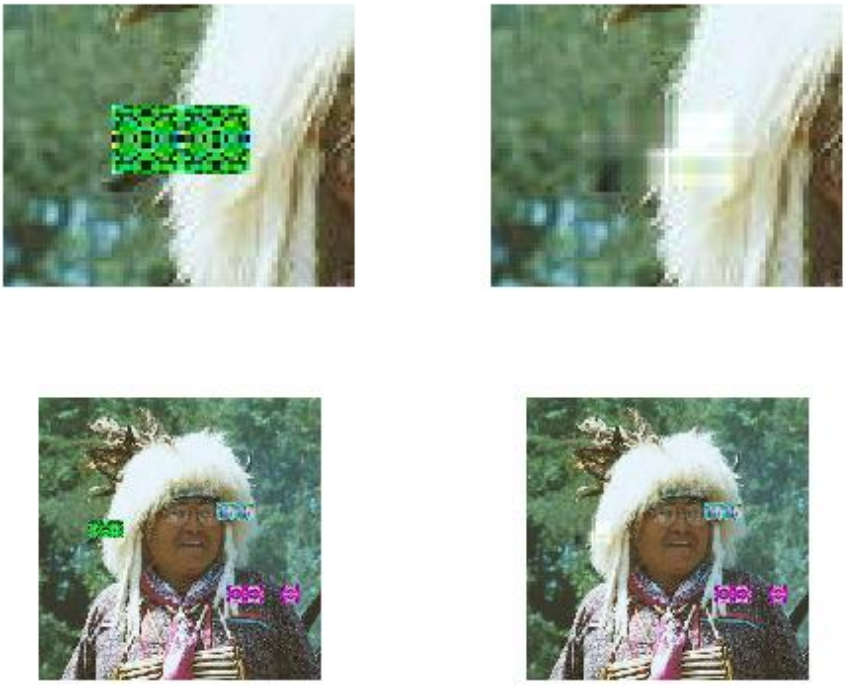


FIGURE 3.3: This image is an example of this thesis's algorithm solving the inpainting problem generated by bit error in the data sequence. The images on the left are damaged, and the images on the right are the result of recovery. The whole process was finished within 0.15s running in MATLAB.

3.2 Mathematical Model

In the thesis, we used notations similar to that utilized in the inpainting literature. The $target$ region is the region to be filled by the algorithm, represented as Ω. The boundary of the target region is indicated by $\delta\Omega$. The surrounding area of the target region is the information source of the algorithm with the $source$ region notation written as Φ, which is the same as in[5]. The source region is defined as the area between the extended bounding box of the target region and the target region. Therefore, the source region and the target region combined should be a rectangle and its size should be larger than the original bounding box of the target region.

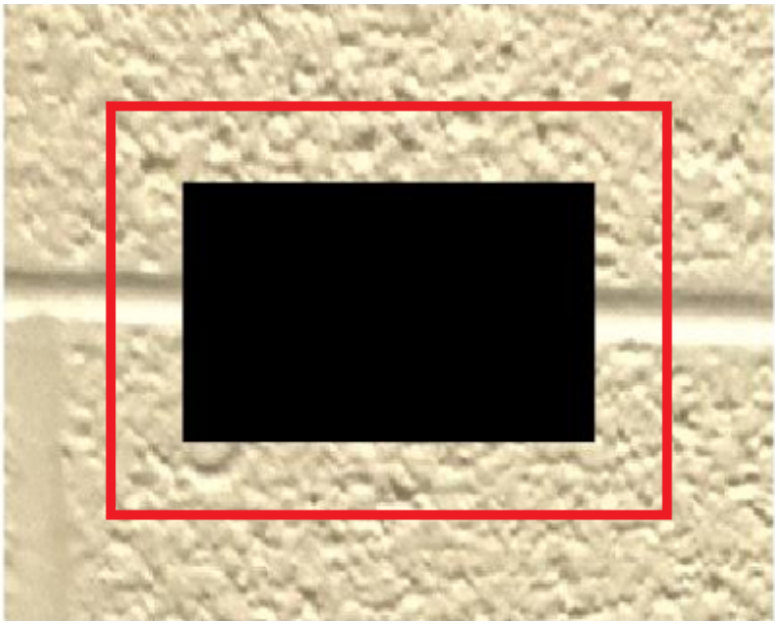


FIGURE 3.4: The black region in the image represents the target region. The area between the red rectangular and the target region is the source region provides all the necessary information for our algorithm. We assume the red rectangle's size is $m \times n$ which is the bounding box of the source region.

The size of target region will determine the size of the source region. Since the size of the target region is small, there is no reason to include too much information from the pixels far from the target region. Meanwhile, if the size of the source region is too small, the algorithm may not produce good results due to the lack of information. In our thesis, we set the size of the source region at 1.5 times the size of the target region.

If the source region's size in the image is $m \times n$, we can generate a $m \times n$ matrix **S** which is the corresponding pixels' intensity values of the source region and the target region combined. Before applying the algorithm to the matrix **S**, we need to distinguish the pixels in the target region from the pixels in the source region. In the algorithm, the first task is to set the elements' values in the target region to zeros. The second task is to add a small positive number q to the elements in the source region. Since **S** is a non-negative matrix, adding q will make sure its elements in the source region are positive numbers. Now the elements in the target region Ω will be zero and the elements in source region Φ will be positive numbers. After this process the matrix **S** becomes a new $m \times n$ matrix **A**. With the non-negative $m \times n$ matrix **A**, the algorithm will decomposes it into the product of three matrices shown as:

$$\mathbf{A} = \mathbf{U} \times \mathbf{B} \times \mathbf{V} \tag{3.1}$$

The matrix **U** is a $m \times m$ diagonal matrix and the matrix **V** is a $n \times n$ diagonal matrix. The matrix **B** is a $m \times n$ matrix which satisfies both the prescribed property shown as below:

$$\prod_{i=1}^{m} \mathbf{B}_{i,j} = 1, \quad \text{for } j = 1, 2, ...n \text{ and } \mathbf{B}_{i,j} \in \Phi \tag{3.2}$$

$$\prod_{j=1}^{n} \mathbf{B}_{i,j} = 1, \quad \text{for } i = 1, 2, ...m \text{ and } \mathbf{B}_{i,j} \in \Phi \tag{3.3}$$

The decomposition process is finished by the algorithm presented by Rothblum and Zenios[25]. They developed an efficient algorithm to solve the problem within O(p) times. And p is the number of elements of the input matrix which should equal to $m \times n$. Meanwhile, they have proved that if a solution exists to the decomposition process, which is finding the matrix **B** with prescribed property while given matrix **A**, then the solution is unique. The detailed process of the algorithm[25] used to solve the above decomposition problem is shown below.

Assuming $\Omega_i(\mathbf{A})$ represents the set of the elements which belong to row i in the source region Ω of matrix **A**. The $\Omega^j(\mathbf{A})$ represents the set of the elements which belong to column j in the source region Ω of the matrix **A**. s_i, t_j are the prescribed row and column products for row i and column j. λ denotes a relaxation parameter.

$$a_{i,j} = \ln \mathbf{A}_{i,j}, \quad \text{for } (i,j) \in \Omega \tag{3.4}$$

$$a_{i,j} = 0\ , \qquad \text{for } (i,j) \notin \Omega \tag{3.5}$$

Step 0. Set $k = 0$. Select $u^0 \in \mathbf{R}^m$ and $v^0 \in \mathbf{R}^n$. Define $a^0 = \mathbf{R}^{m \times n}$ by:

$$a^0_{i,j} = a_{i,j} - u^0_i - v^0_j, \quad \text{for } (i,j) \in \Omega \tag{3.6}$$

$$a_{i,j} = 0\ , \qquad \text{for } (i,j) \notin \Omega \tag{3.7}$$

Step 1. For $i = 1, 2, 3..., m$ let

$$p_i = \lambda \left[\|\Omega_i(\mathbf{A})\|\right]^{-1} \left[s_i \ - \sum_{j \in \Omega_i(\mathbf{A})} a^k_{i,j} \right] \tag{3.8}$$

$$a^{k+1/2}_{i,j} = a^k_{i,j} + p_i, \quad \text{for } j \in \Omega_i(\mathbf{A}) \tag{3.9}$$

$$u^{k+1}_i = u^k_i - p_i \tag{3.10}$$

Step 2. For $j = 1, 2, 3..., n$ let

$$p_j = \lambda \left[\left\|\Omega^j(\mathbf{A})\right\|\right]^{-1} \left[t_j \ - \sum_{i \in \Omega^j(\mathbf{A})} a^{k+1/2}_{i,j} \right] \tag{3.11}$$

$$a^{k+1}_{i,j} = a^{k+1/2}_{i,j} + p_j, \quad \text{for } i \in \Omega^j(\mathbf{A}) \tag{3.12}$$

$$v^{k+1}_j = v^k_j - p_j \tag{3.13}$$

Step 3. Replace $k = k + 1$ and return to *Step*1.

The iteration process will converge once the p_i and p_j are very closed to 0. Then, we can generate the matrices **U, B, V** by applying exponentiation to the converged $u^k, a^k_{i,j}, v^k$.

Actually, the decomposition process cannot guarantee transforming the non-negative matrix **A** into **UBV** for all prescribed row and column products of **B**. But for usage in this thesis, I can proof that the algorithm will always transform the matrix **A** into the form **UBV** if we set all the row and column products of **B** to 1. It has been proved by Uhlmann[31] that a solution always exists in the special case in which every row and column product is the same (e.g., 1). Then we can conclude that the transformation of Equation 3.1 will always solve the decomposition and provide the unique matrix **B** for any given non-negative matrix **A**, when we set all the row and column products of **B** to 1.

Equation of 3.1 shows the scale extraction process of the algorithm. The purpose of this process is to extract the vertical and horizontal structural information from the source region in the image. After the extraction, the scaling information of matrix **A** is stored at the diagonal matrices **U,V**. Meanwhile, each element of matrix **A** can be represented as:

$$\mathbf{A}_{i,j} = \mathbf{U}_{i,i} \times \mathbf{B}_{i,j} \times \mathbf{V}_{j,j} \tag{3.14}$$

For elements which belong to the missing area in matrix **A**, we have set their values to zeros in the beginning. Because matrix **A** is a non-negative matrix, and according to the Eq. 3.14, the corresponding elements in matrix **B** is zero too. Since the elements of the source region in matrix **A** are all positive, we can know that the corresponding elements of the source region in matrix **B** are also all positive numbers.

In the algorithm, a simple filling method can be used replace all zero elements in matrix **B** and generate a new matrix **B'**. The simple filling method is discussed in the next section. Next, matrix **B'** is used to calculate matrix **A'** by applying Equation 3.15:

$$\mathbf{A'}_{i,j} = \mathbf{U}_{i,i} \times \mathbf{B'}_{i,j} \times \mathbf{V}_{j,j} \tag{3.15}$$

Finally with matrix **A'**, the small positive number **q** must be substracted from matrix **A'** for our final inpainting result.

Matrix **A'** is an $m \times n$ non-negative matrix and for elements in the source region, they are the same as the corresponding elements in matrix **A**. Because the algorithm does not change any elements in the source region of matrix **B** to generate the matrix **B'**, the inpainting algorithm will not introduce new artifacts into the known parts of the video frames or images after inpainting.

For explaining the inpainting algorithm's time complexity, the time-complexity of the algorithm for the decomposition process[25] mentioned that the decomposition algorithm is a special case of Algorithm 2.5 from Zenios and Censor's research[27]. [27] stated that to guarantee asymptotic convergence of the algorithm, it must be restricted to the interval where $\epsilon \leq \lambda \leq 1$ for some arbitrarily chosen real $\epsilon > 0$. Rothblum and Zenios[25], they, set the λ to 1 and therefore, our decomposition process guarantees asymptotic convergence.

Meanwhile, Zenios and Censor[27] showed that the algorithm is well suited for parallel computing. Moreover, the decomposition algorithm can be implemented to exploit matrix sparsity. Consider the Equations 3.8, 3.9, 3.11 and 3.12, which are the main time-consuming parts among the entire decomposition process. These equations only calculate the elements within the set $\Omega^j(\mathbf{A})$ and $\Omega_i(\mathbf{A})$. Therefore, we can implement a link list data structure where each node of the link list will store four pointers, which are the pointers to the elements on the left, right, top, and bottom side of the current element in the image. By implementing the above data structure, we can simplify the decomposition algorithm's time-complexity to O(p) time, where p is the number of elements in the source region.

For the simple filling method of matrix **B**, which is the process of generating matrix **B'**, the time complexity is O(w) and w is the number of elements in the target region. Therefore, the algorithm's time complexity will not exceed O(s), where s is the number of elements in both the source region and the target region combined.

3.3 Methods for Filling Missing Elements in Matrix B′

In this section, I introduce three different filling methods to fill in the missing elements which are the zero elements in matrix **B′**. Then I show examples of the three methods and choose the algorithm that produces the best results as the filling methods for the inpainting algorithm.

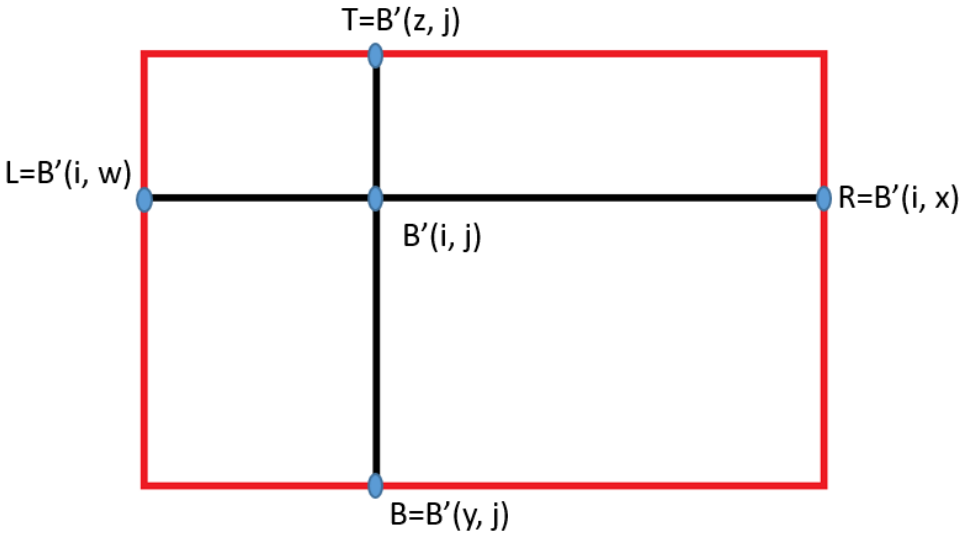


FIGURE 3.5: The above image shows the four pixels' position on the boundary of the target region. The intensity values of these four pixels are used to calculate the missing pixel's intensity in the center.

Assuming the four pixels' intensities are L, R, B and T in the above image. We define the distance from the four pixels as **B′**(i,w), **B′**(i,x), **B′**(y,j), **B′**(z,j) to the missing pixel **B′**(i,j) as L', R', B', T'. Then for the first filling method, it can be represented as:

$$\mathbf{B'}_{i,j} = (L + R + B + T)/4, \quad \text{for } \mathbf{B'}_{i,j} \in \Omega \tag{3.16}$$

In this method, we set the intensity value of the missing pixel according to the average intensity values of the corresponding four pixels L, R, B and T. The experiment result of this filling method has noticeable blocking artifacts. The inpainted pixels close to the target region's boundary are contrasting their intensity against the boundary's surrounding pixels' intensity, which can cause the artifacts to appear as lines on the boundary of the target region. The following images show the inpainted result of the first filling method.

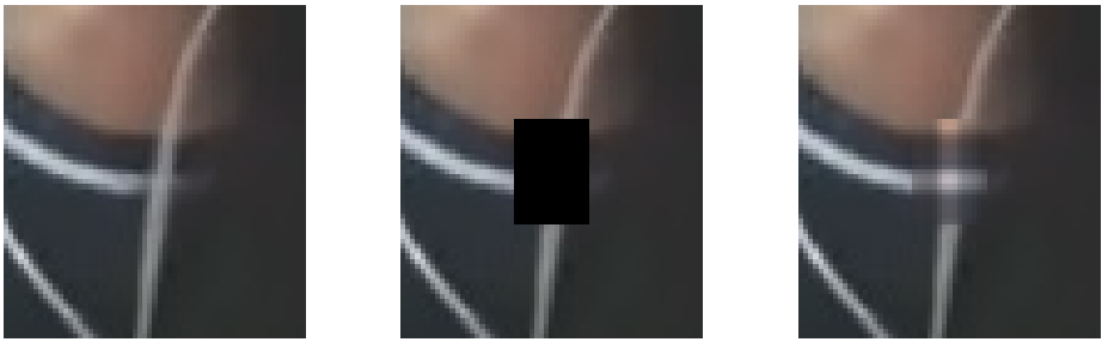


FIGURE 3.6: The images above are the original image, the corrupted one and the inpainting result of using the first filling method for matrix **B**.

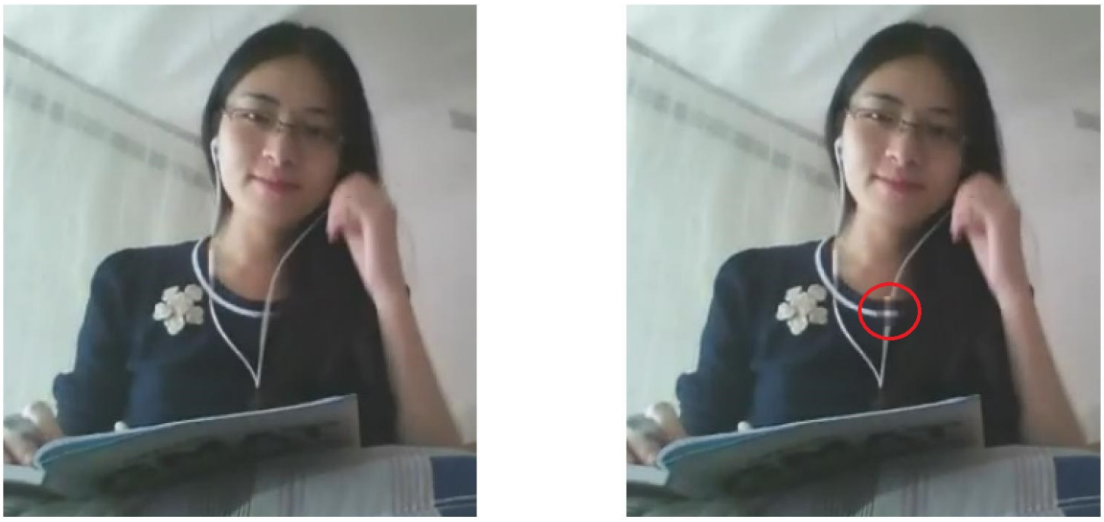


FIGURE 3.7: These images are the original image and the image after inpainting. The red circle indicates the inpainting result.

From the above images, we can see the blocking artifacts around the boundary of the target region. The difference between the intensity of the pixels on both sides of the boundary generates artifact like lines in the image. Meanwhile, the elements on the same row or column tend to have a similar filling result. Because these elements share the same L, R or B, T during the calculation. Therefore, some line artifacts appear inside the target region. When we look at the entire image, the artifacts are noticeable and distracting.

To address the above problem and under Dr. Uhlmann's supervision, I developed the second and third methods, which can calculate the missing pixel's intensity while considering its position in the target region. The goal was to to make the intensity of the pixels in the target region similar to its closest elements in the source region.

For the second filling method, we first set the initial values of the zero elements which are the elements in the target region in **B** to the values of their closest elements in the source region. For example, if the missing pixel is closest to the top side of the boundary of the target region, we will set the initial value of the pixel to T.

After replacing the zero elements in matrix **B** with initial values, we still need to apply a blurring process to the **B** matrix. The reason is that if we use the matrix **B** to generate the inpainting result directly, the inpainting result will contain some distracting line artifacts. The artifacts are generated due to the high contrast of the different pixel values in matrix **B**. Moreover, the artifacts mainly lie on the diagonals of our target region which is a block of missing pixels. Therefore, we apply a blurring process to make the artifacts less distracting. The following images will demonstrate the problem and our solution.

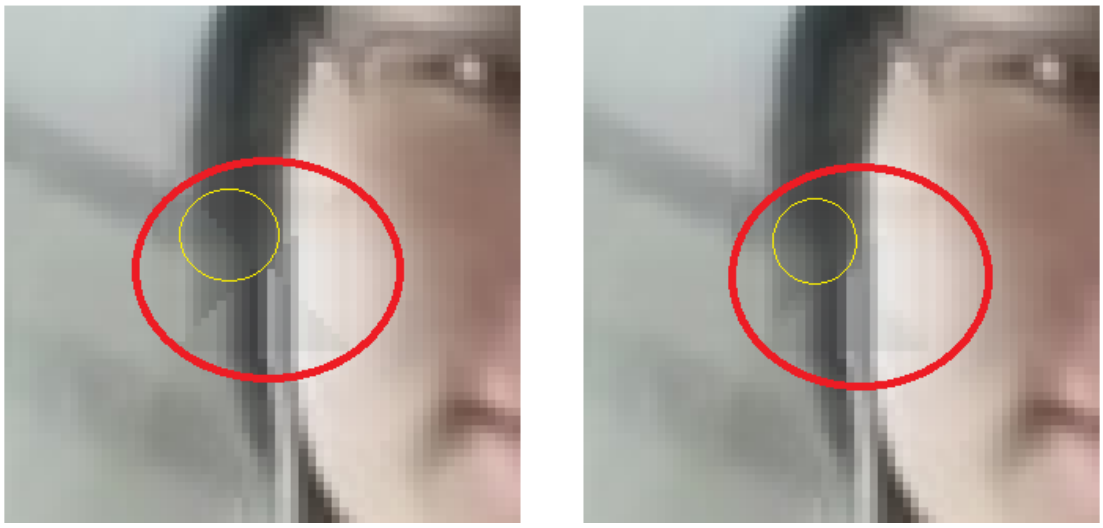


FIGURE 3.8: The above images are the inpainting results of applying the second filling method for matrix **B**. The image on the left is the result without the blurring process. The image on the right is the result after applying the blurring process.

In the algorithm, we use the following equation to blur to line artifacts.

$$\mathbf{B'}_{i,j} = (\mathbf{B}_{i,j} + \mathbf{B}_{i+1,j} + \mathbf{B}_{i,j+1} + \mathbf{B}_{i-1,j} + \mathbf{B}_{i,j-1})/5 \tag{3.17}$$

After the blurring process, the matrix **B'** can be used to generate the inpainting result **A'** by applying the Equation 3.15. The following images show the inpainted result of the seond filling method. The same target region can then be used as the first filling method for comparison.

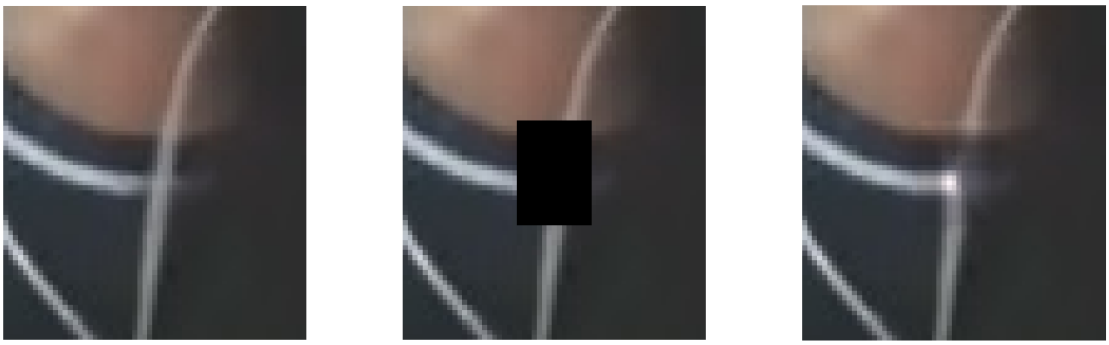


FIGURE 3.9: The images above are from left to right: the original image, the corrupted one and the inpainting result of using the second filling method for matrix **B**.

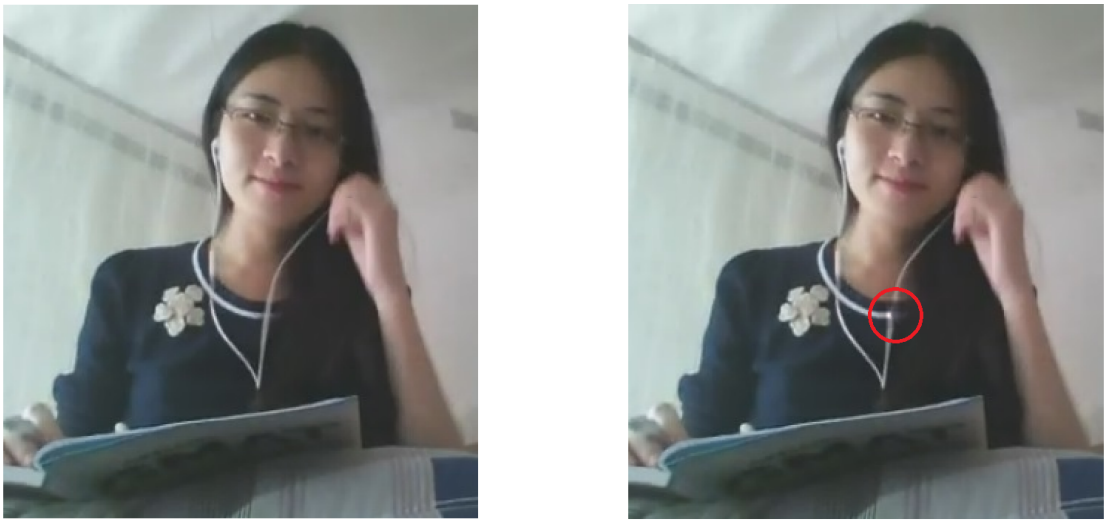


FIGURE 3.10: These images are from left to right: the original image and the image after inpainting. The red circle indicates the inpainting result.

For the third filling method, the objective is to calculate the intensity of the missing pixel based on its distance from the boundary of the target region. We calculate the pixels in the target region by applying the following equations.

$$H = 1/L'^2 + 1/R'^2 + 1/B'^2 + 1/T'^2 \tag{3.18}$$

$$\mathbf{B'}_{i,j} = (L/L'^2 + R/R'^2 + B/B'^2 + T/T'^2)/H \tag{3.19}$$

In the third filling method for the matrix **B**, when the missing pixel is closest to the top boundary of the target region, the T' will have the smallest value among the L', R', B', T'. Hence the T will have the biggest weight during the calculation according to the Equation 3.19. The following images show the result of the third filling method.

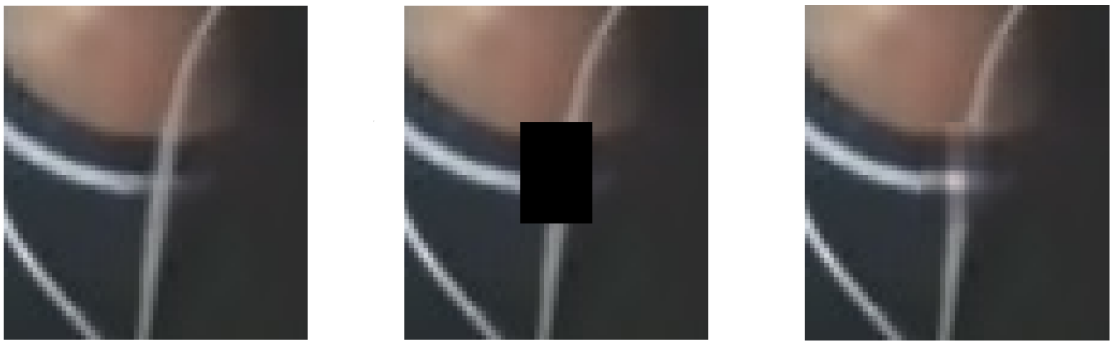


FIGURE 3.11: The images above are from left to right: the original image, the corrupted one and the inpainting result of using the third filling method for matrix **B**.

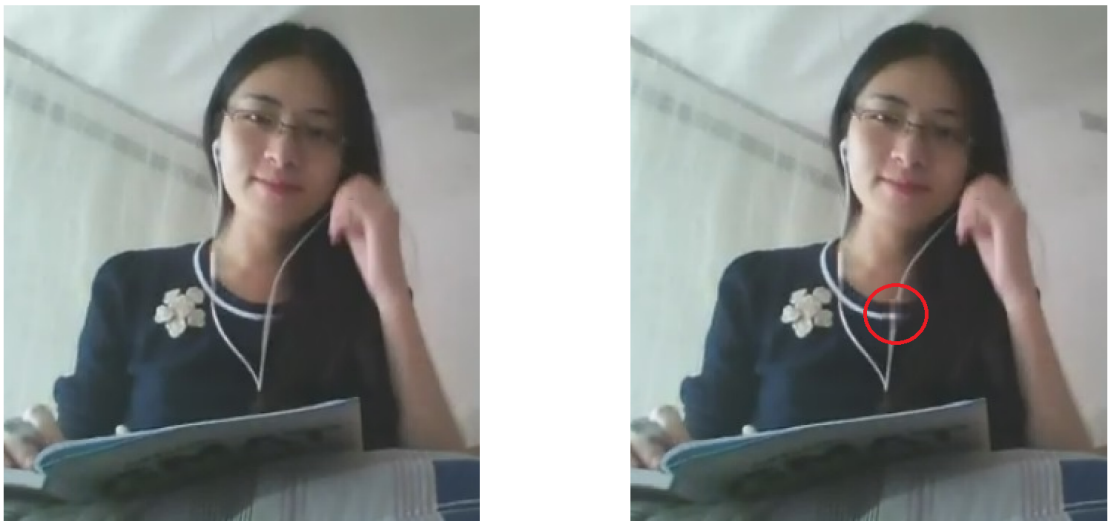


FIGURE 3.12: These images are from left to right: the original image and the image after inpainting. The red circle indicates the inpainting result.

The following images are the original image, the corrupted image and the three different filling methods' inpainting results.

FIGURE 3.13: The images above are from left to right and then top to bottom: the original image, the corrupted image and the results of the first, second and the third filling methods for the matrix **B** when finishing the same inpainting task.

From the above images, it is obvious that the second and third filling methods are better than the first filling method. The artifacts generated by the first method are more noticeable on the boundary of the target region. When comparing the second and third methods' results, we can see that inside the target region and on the boundary of the target region, the second method is producing less line artifacts than the third method. As a result, we choose the second method to fill the missing pixels of the matrix **B** for the algorithm to generate the experiment results in the following chapter.

Chapter 4

Experiment Results

This chapter will present more experiment results of applying the featured algorithm to different images and show the processing time of the whole inpainting process. The target regions in the images are shown as the blocks of missing pixels for simulating the video inpainting problem. Different regions within the same images were selected as the target regions. The chapter will then present the experiment results which our algorithm can not handle well. Finally in this chapter, the results of our algorithm will be compared with the results of the PDE inpainting algorithm and the fast marching algorithm. I will also discuss the conclusions based on the experiments and comparisons at the very end of this chapter.

our algorithm was implemented in MATLAB, and codes were run on a Windows desktop with an i5-4440 CPU and 12 GB RAM. I have also implemented the C++ codes to exploit matrix sparsity, and lower the processing time of our algorithm. However, only the MATLAB codes were used to generate the experiment results. For generating the results of PDE based algorithm, we used the codes from the website[7]. For generating the results of fast marching algorithm, the codes from [29] was used.

4.1 Inpainting Simple Texture

In this section, we will apply our algorithm to the missing areas with a simple surrounding texture. The missing area in our examples will be represented as a block of black pixels in all the following sections in chapter 4.

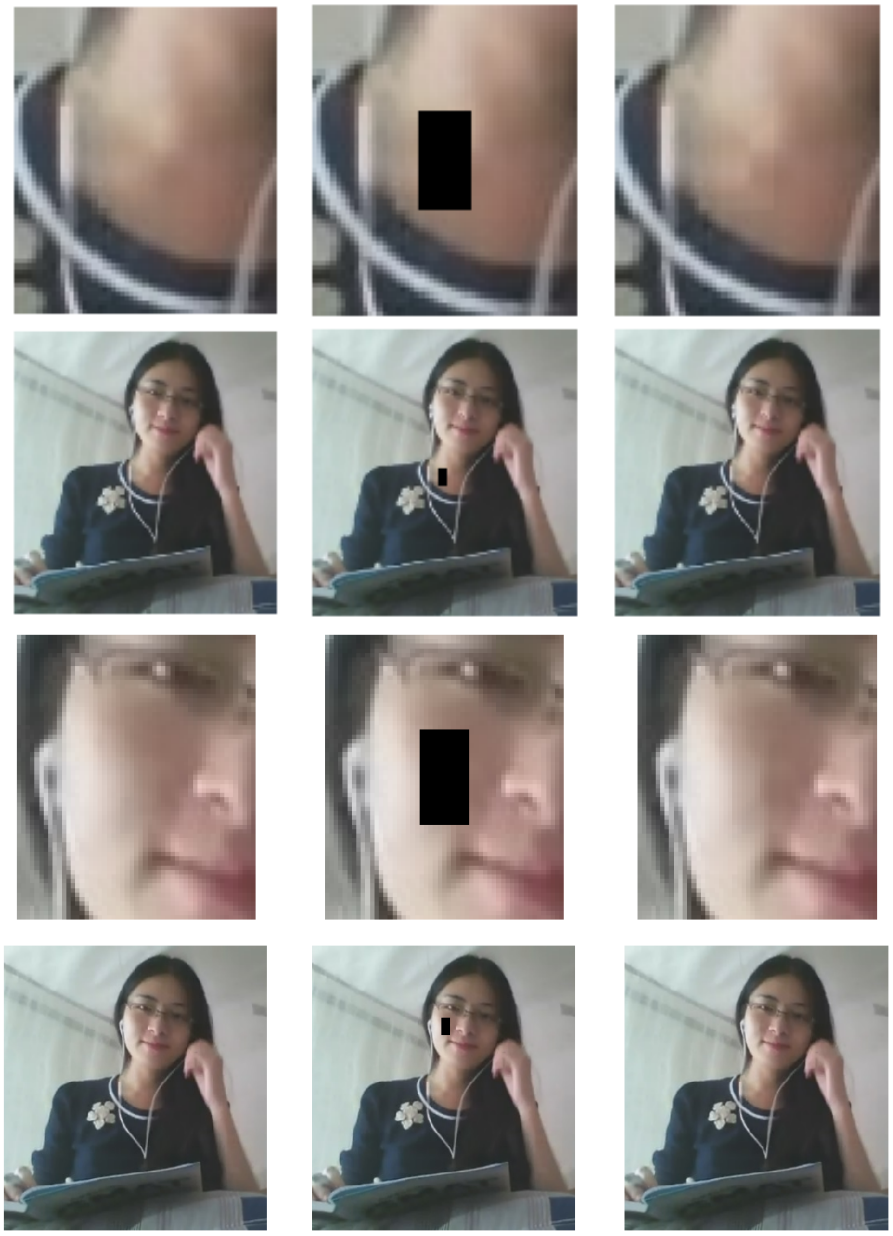


FIGURE 4.1: The above images and those that follow will be displayed in a left to right, top to bottom order, which include the the original images, the corrupted image, and the inpainting results. In this section, we apply our algorithm to the low-resolution images, and the target regions and their surrounding areas have simple textures in the images.

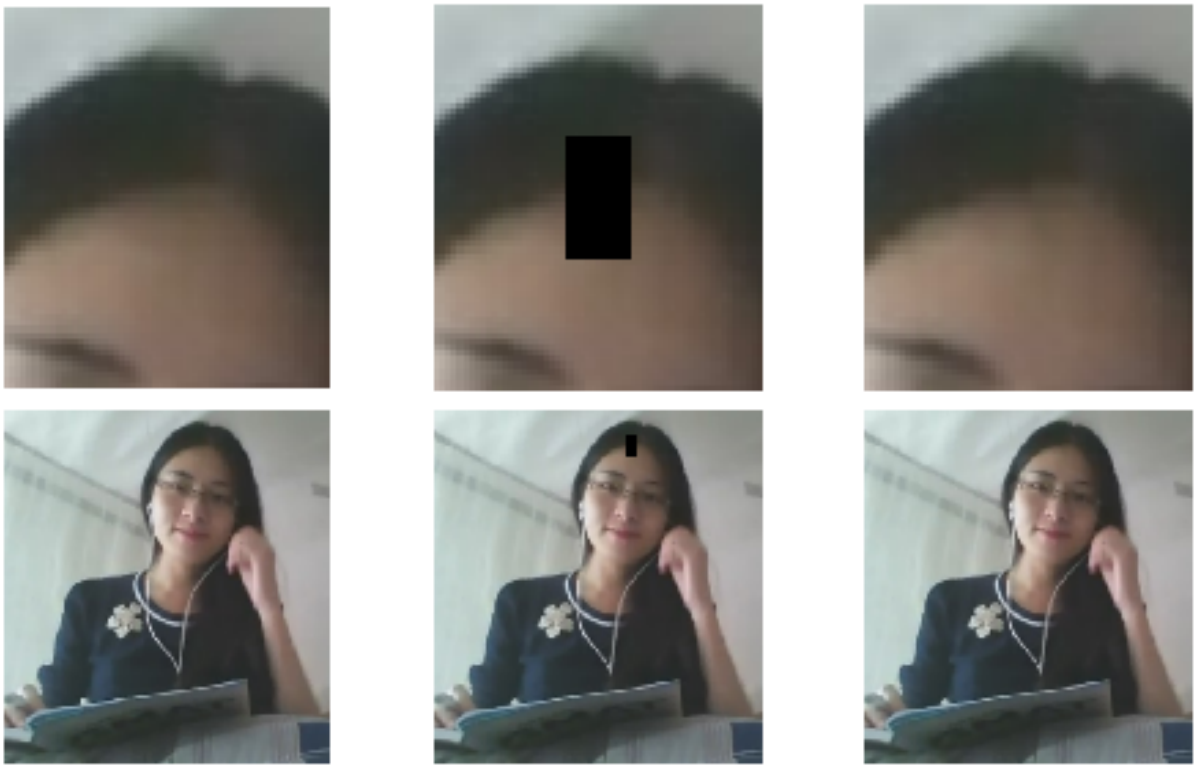


FIGURE 4.2: In this image, we selected the target region with a slightly more complex texture than the previous example.

FIGURE 4.3: The algorithm created in this research was applied to this image from [2] by focusing on one of the blocks of the damaged pixels. We selected target region with a simple texture in this example.

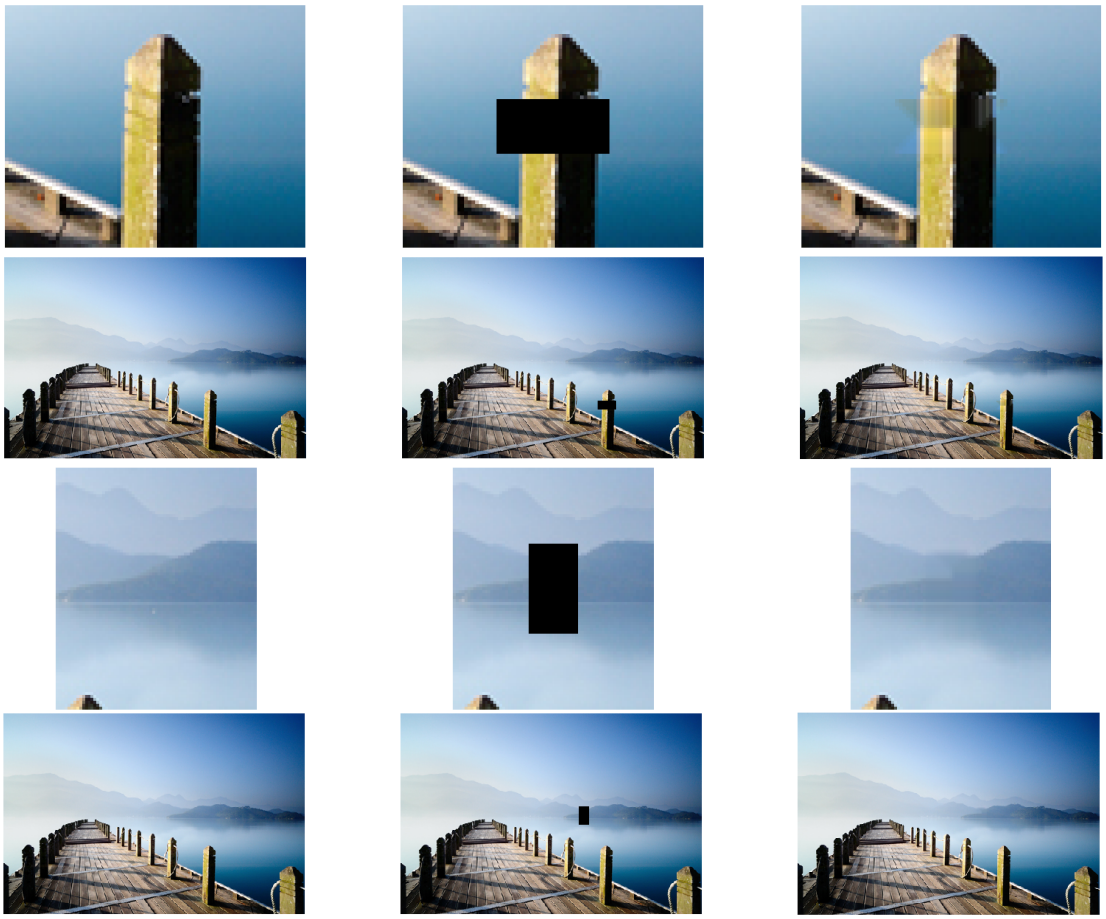


FIGURE 4.4: More examples of inpainting featuring target regions with simple textures.

The image size in Fig 4.1 and Fig 4.2 are both 350×324, and the image in Fig 4.3 has a size 250×250. The target regions' sizes are 20×10 for the first two examples and 16×40 for the last example. The time-complexity of the algorithm is only related to the size of the target region. Therefore, the first two examples share the same processing time 0.086s due to the same target region's size. For the third result, the processing time is 0.21s. In Fig 4.4, we present more examples to show the results of inpainting.

For the examples shown in Fig 4.1 and 4.2, our algorithm has produced excellent inpainting results that look very natural, which do not cause any distractions to the viewer. However, for the inpainting results of Fig 4.3, our algorithm has created some line artifacts within the missing area due to the large missing area size, which is 16×40. Although the inpainting result in Fig 4.3 has some noticeable artifacts, the missing area is still not very distracting when we look at the entire image after inpainting.

4.2 Inpainting Complex Texture

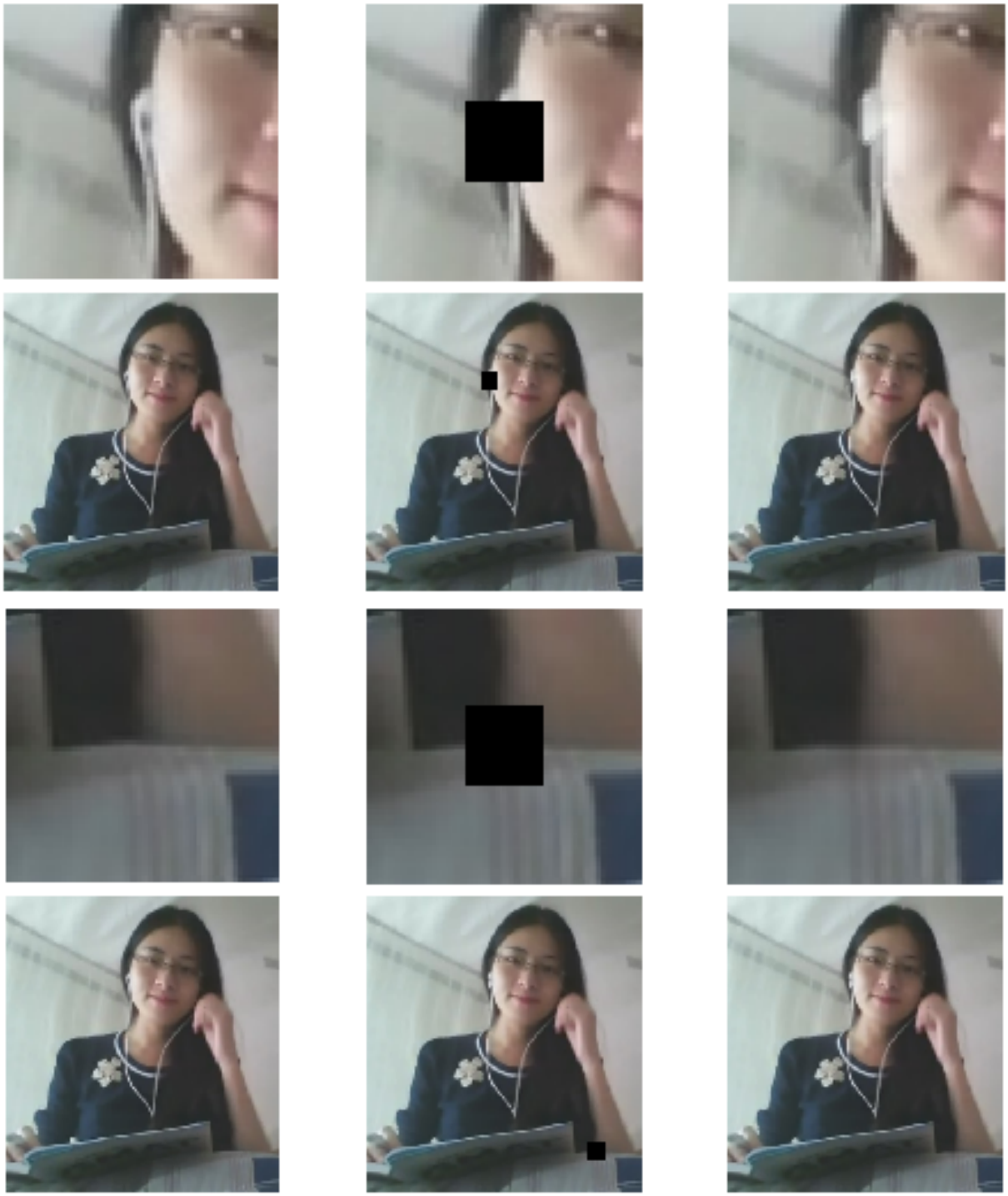


FIGURE 4.5: The above and the following images are from left to right: the original image, the corrupted image, and the inpainting result. In this section, we apply our algorithm to the low-resolution images, and the target regions and their surrounding areas have complex textures in the images. The size of the missing area is 20×20

FIGURE 4.6: The target region of the above example has a more complex texture than the previous example generated from the same image. The inpainting result has noticeable line artifacts on the boundary. The size of the missing area is 16×42 in this example.

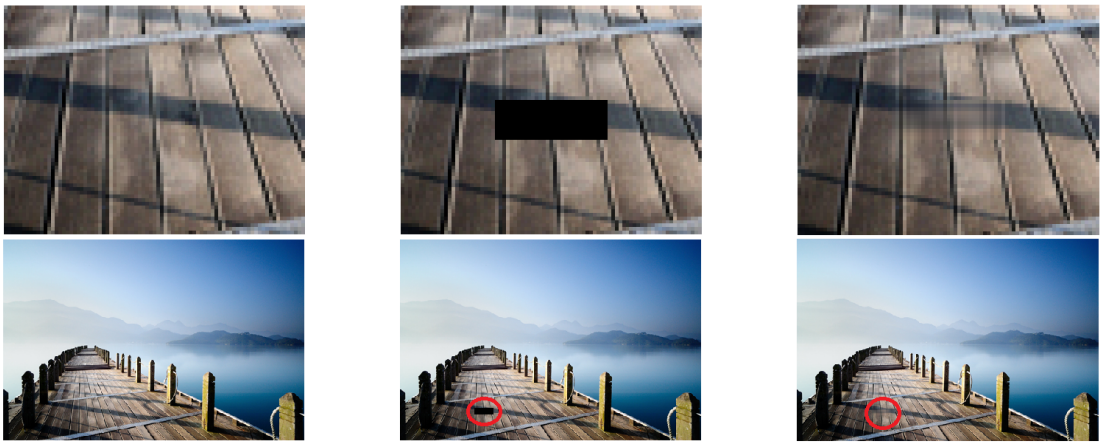


FIGURE 4.7: The size of the above image is 330×500, and the target region's size is 5×15. The inpainting result shows that the algorithm does not recover the shadow very well in the image, but it reconstructs the lines in the vertical direction.

From the above examples, we can know that the algorithm may not produce a natural result when the target region and its surrounding area have complex textures. However, the artifacts generated by the algorithm are not very distracting when we look at the entire image after inpainting. Meanwhile, some of our experiment results are very natural even when the missing area has a complex texture. Especially for the results shown on the bottom side of Fig 4.5, our algorithm almost perfectly inpaints the missing area. The processing time for the above examples are 0.13s, 0.12s, 0.21s, 0.085s.

4.3 Failing Inpainting Results

One issue of the algorithm is that, although the algorithm can interpolate the horizontal and the vertical structural information from the surrounding area of the missing region, it can not extract the diagonal structural information. For example, the algorithm can not extend the line structure from the known regions to the missing area well if the lines or edges are oriented in the diagonal direction. When inpainting the above situation, the algorithm will generate very distracting artifacts in its inpainting result. The following images are the examples of the above situation.

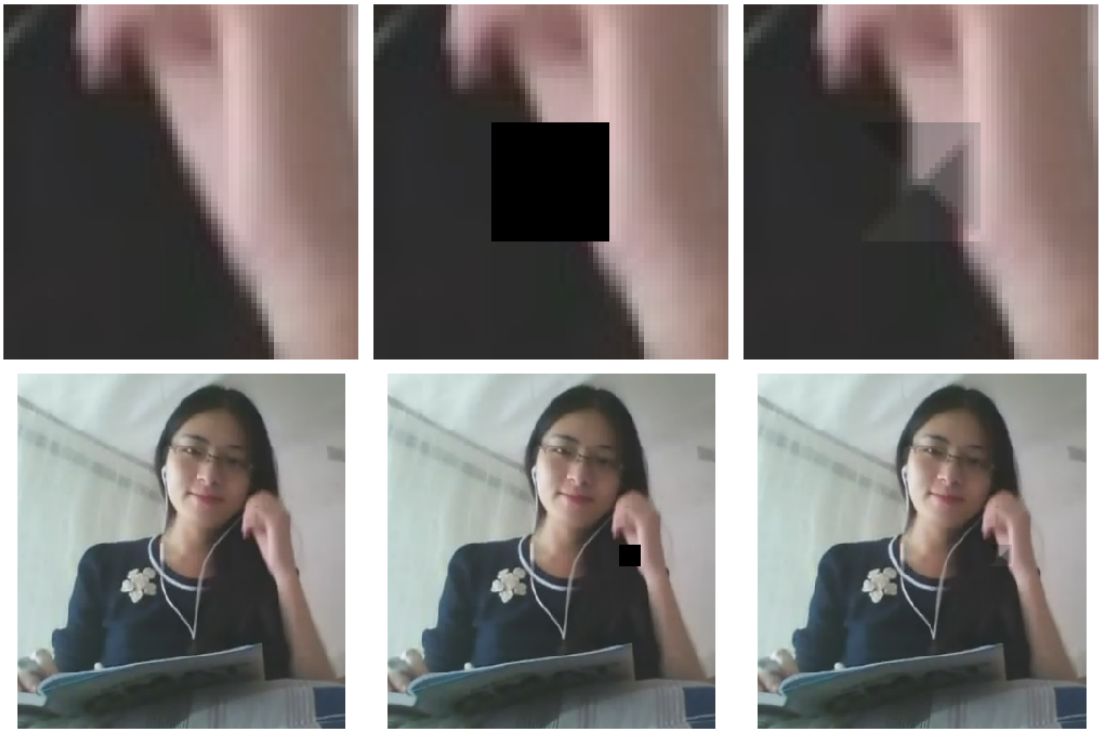


FIGURE 4.8: The above images are the original image, the corrupted image and the inpainting result of our algorithm. In this example, we choose the particular missing area with a simple texture. As we can see the algorithm has produced an inpainting result with very distracting artifacts in the image.

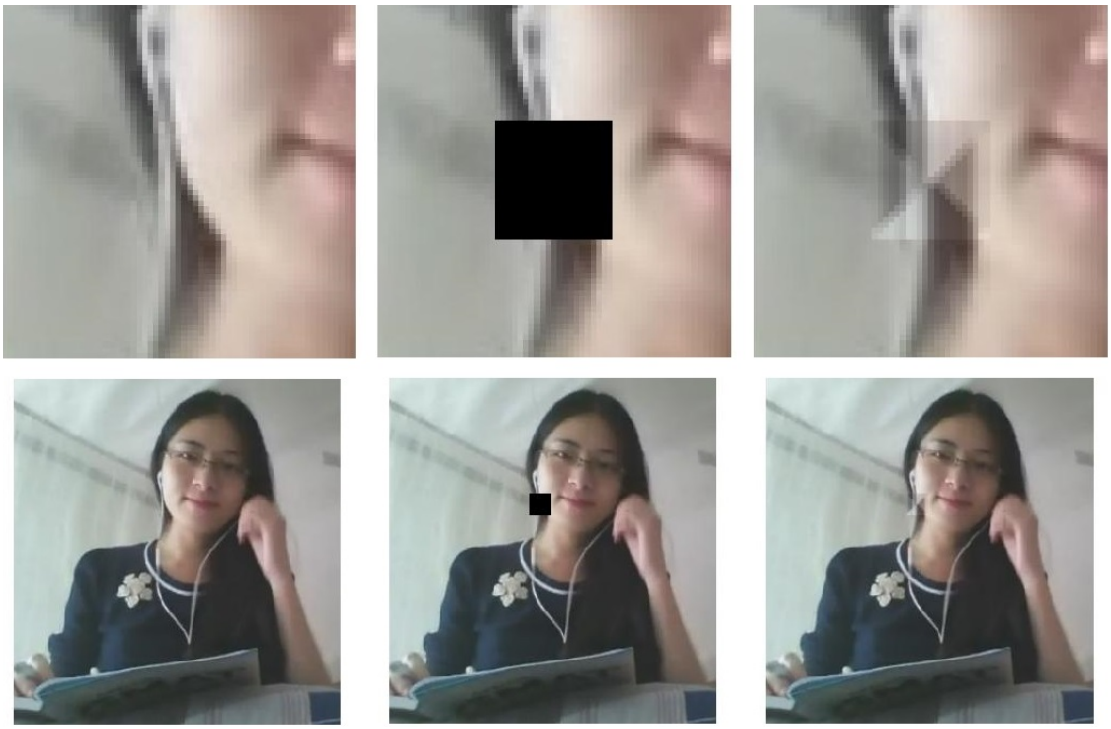


FIGURE 4.9: The above images are the original image, the corrupted image and the inpainting result of our algorithm. In this example, we choose the particular missing area with a more complex texture. As we can see the algorithm has failed to comprehend to structural information of the surrounding area. And the line artifacts are very distracting in the inpainting results.

As shown in the above images, our algorithm does not inpaint the particular missing area in a decent manner, and has produced an inpainting result with very distracting line artifacts. To address the above problem, we must improve the algorithm and make it have the ability to extract the diagonal structural information from the surrounding area of the target region. This is the main issue we have to address. And I am confident that further research will produce a more natural inpainting result when facing the above situations in the future.

4.4 Comparing with the PDE Algorithm

This section compares the inpainting results of our algorithm with the results of the PDE inpainting algorithm. We use the PDE algorithm code from [7] to generate the inpainting results for comparison. The reason for comparing our algorithm with the PDE algorithm is that both algorithms are targeting inpainting problems with the small missing areas. The task here was to compare the two inpainting algorithms' performance on the different missing areas.

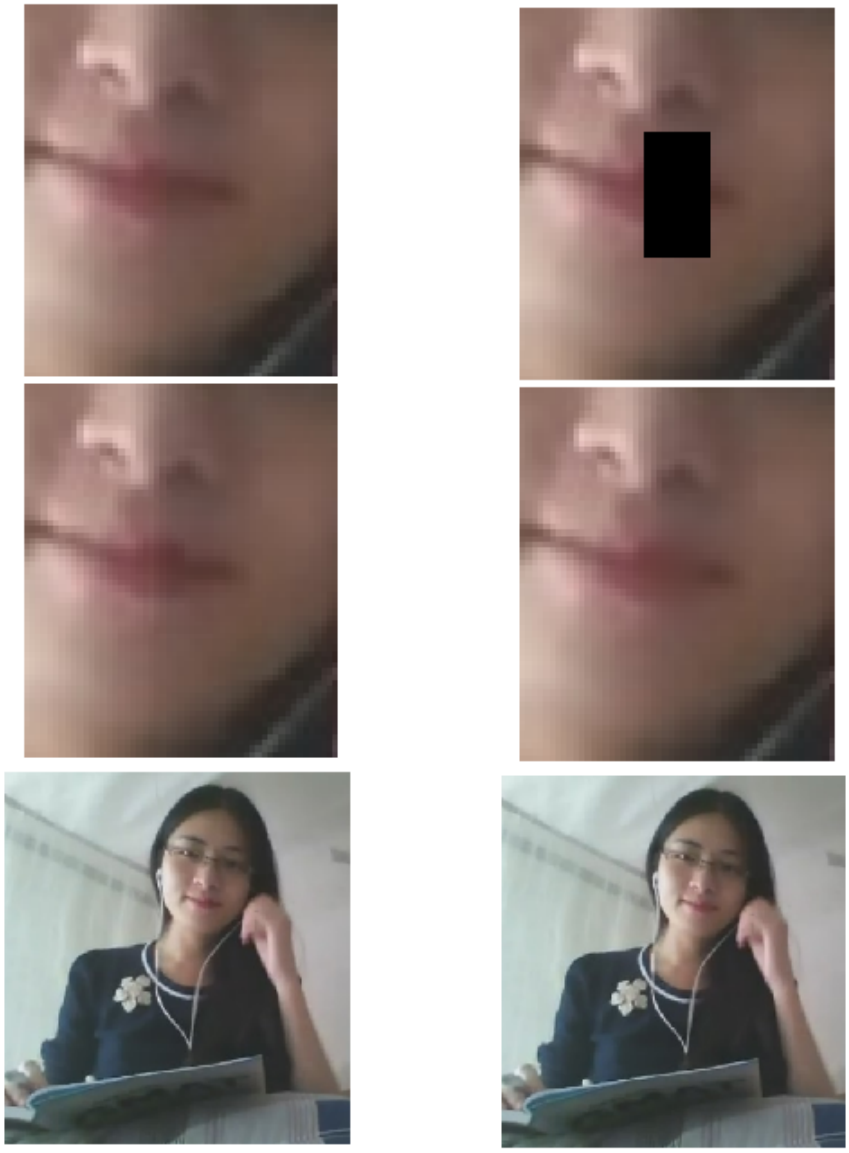


FIGURE 4.10: The above images are from left to right and from top to bottom: the original image, the corrupted image, the inpainting result of our algorithm and the inpainting result of the PDE algorithm. The results of both algorithms are very natural and similar to each other.

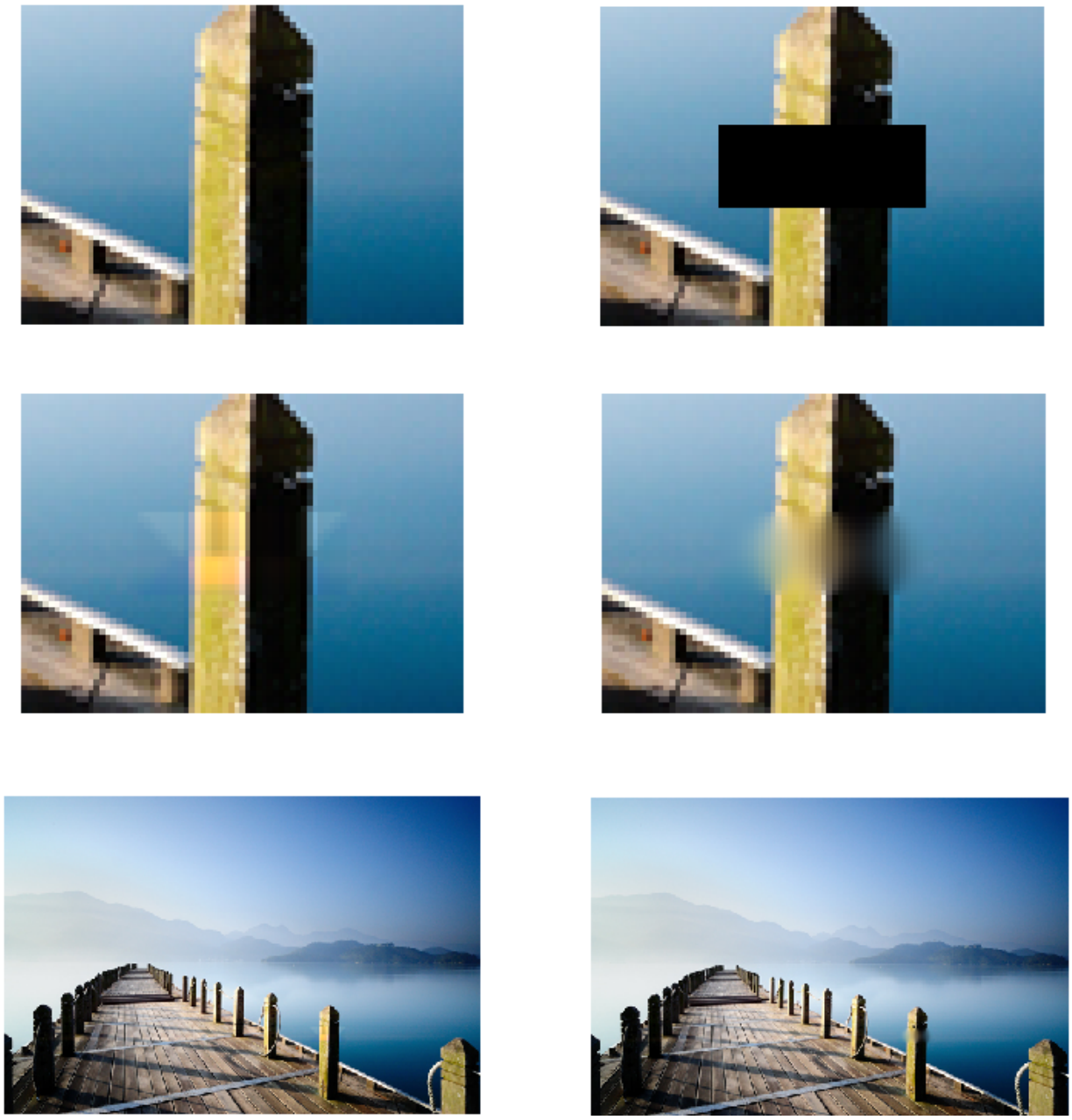


FIGURE 4.11: The above images are from left to right and top to bottom: the original image, the corrupted image, the inpainting result of our algorithm and the inpainting result of the PDE algorithm. In this example, we chose the target region from one of our previous inpainting examples, which shows how the PDE algorithm can generate blurring artifacts in the inpainting results if there exists high contrast edges within the target region.

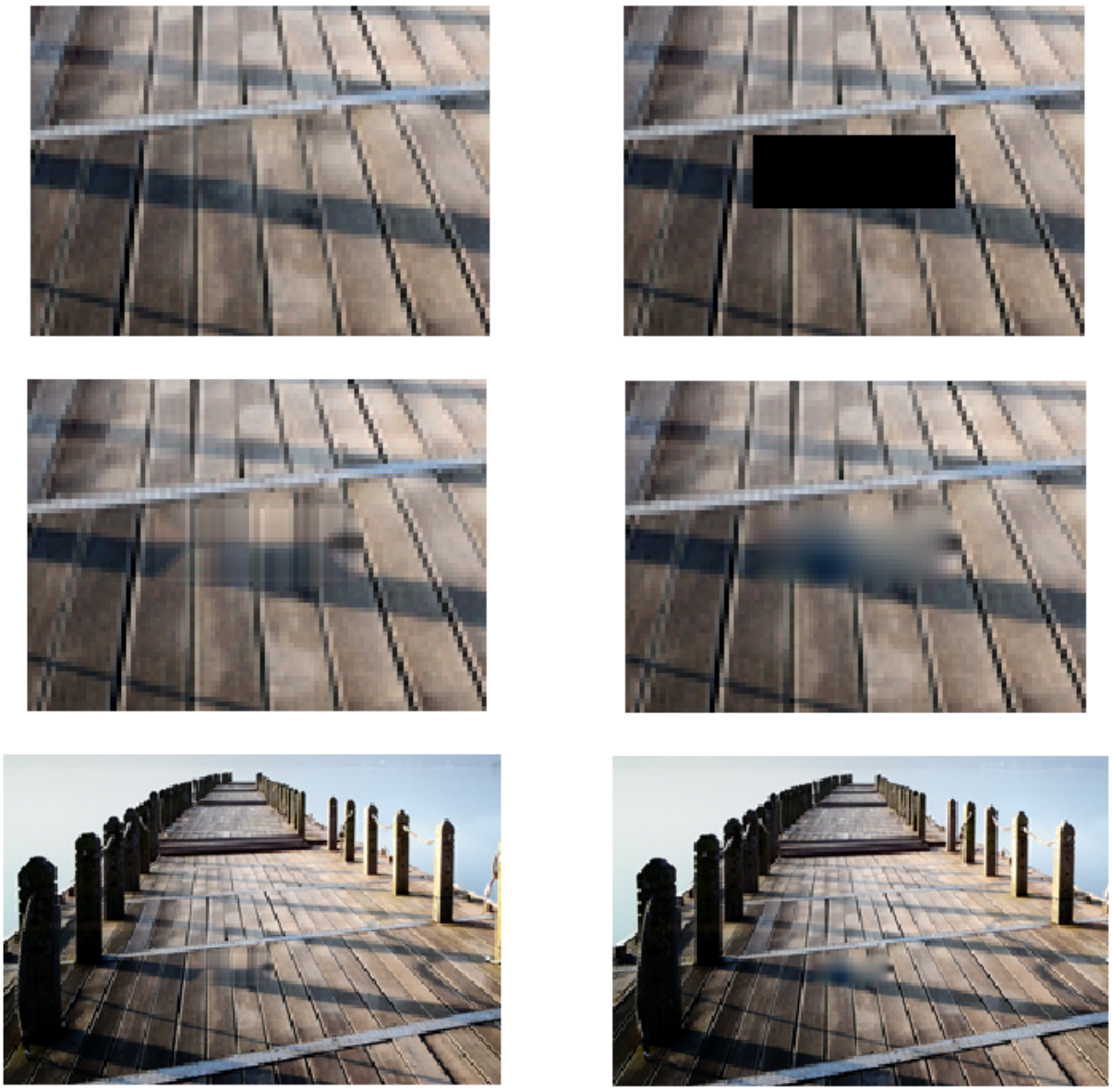


FIGURE 4.12: The above images are from left to right and top to bottom: the original image, the corrupted image, the inpainting result of our algorithm and the inpainting result of the PDE algorithm. For the target region with the complex texture, both algorithms were able to produce results with very distracting artifacts.

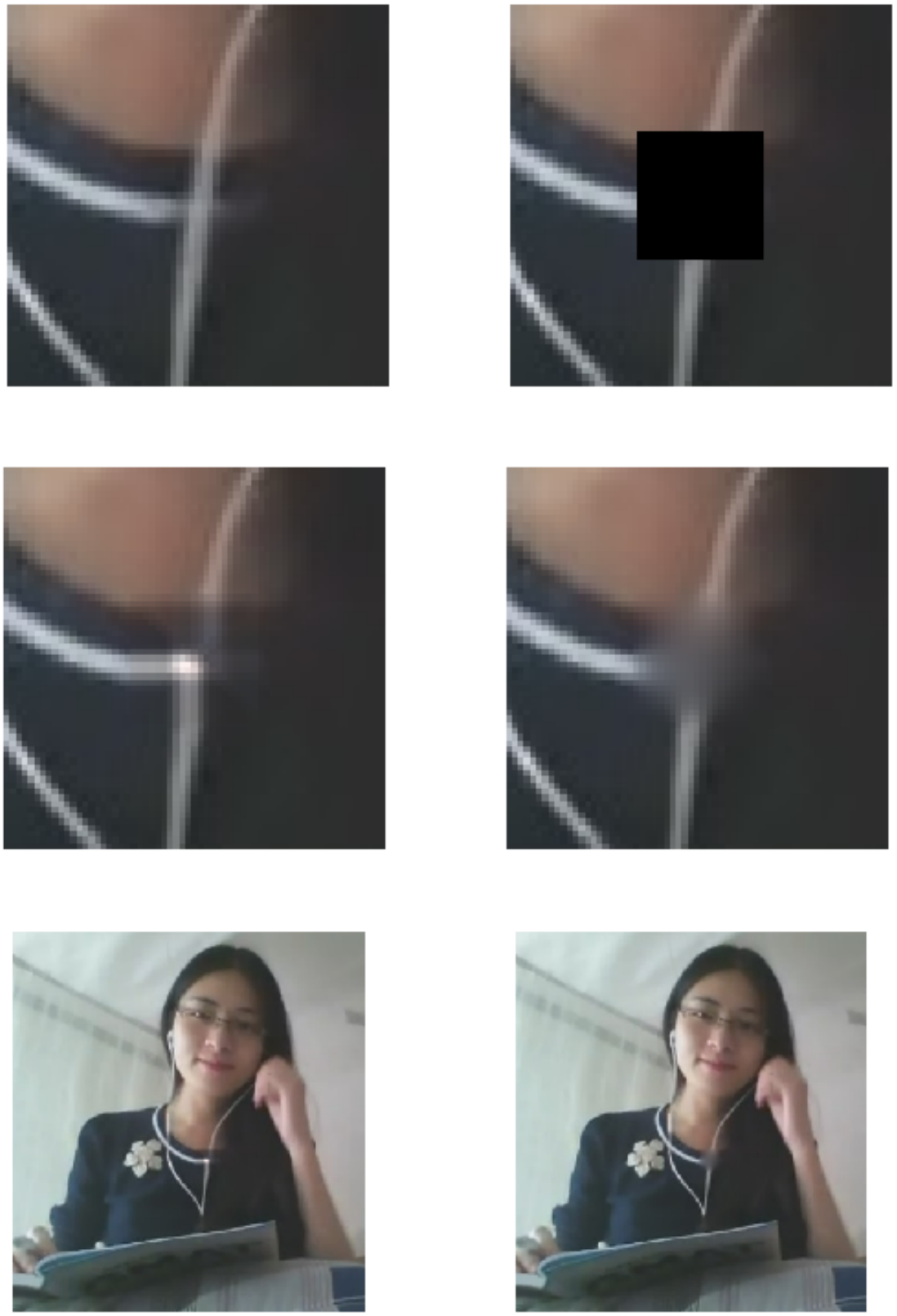


FIGURE 4.13: The above images are from left to right and top to bottom: the original image, the corrupted image, the inpainting result of our algorithm and the inpainting result of the PDE algorithm. In this example, our algorithm has produced a better result than the result of the PDE inpainting algorithm. The blurring artifacts is noticeable and distracting in the result of the PDE method.

For the comparison between our algorithm and PDE based algorithm, the experiment results show that when inpainting the missing area with a simple texture, both algorithm may provide acceptable results, such as the examples shown in Fig 4.10 and 4.11. However, when the missing area's texture is complex, the blurring artifacts generated by the PDE based algorithm will become much more noticeable and distracting. The experiment results of the above situations are shown in Fig 4.12 and 4.13. Meanwhile, our algorithm has produced some better inpainting results in the above inpainting examples, especially the example shown in Fig 4.13.

4.5 Comparing with the Fast Marching Algorithm

This section shows a comparison of this thesis's featured algorithm results with the results of the fast marching algorithm developed by Telea[29]. The two algorithms have the same time complexity which is linear to the size of the missing area. Comparing Telea's fast marching algorithm with the featured algorithm allowed me to determine if the featured algorithm can produce similar or even better results than the results of the other algorithm, which has the same time complexity and is also targeting small inpainting area.

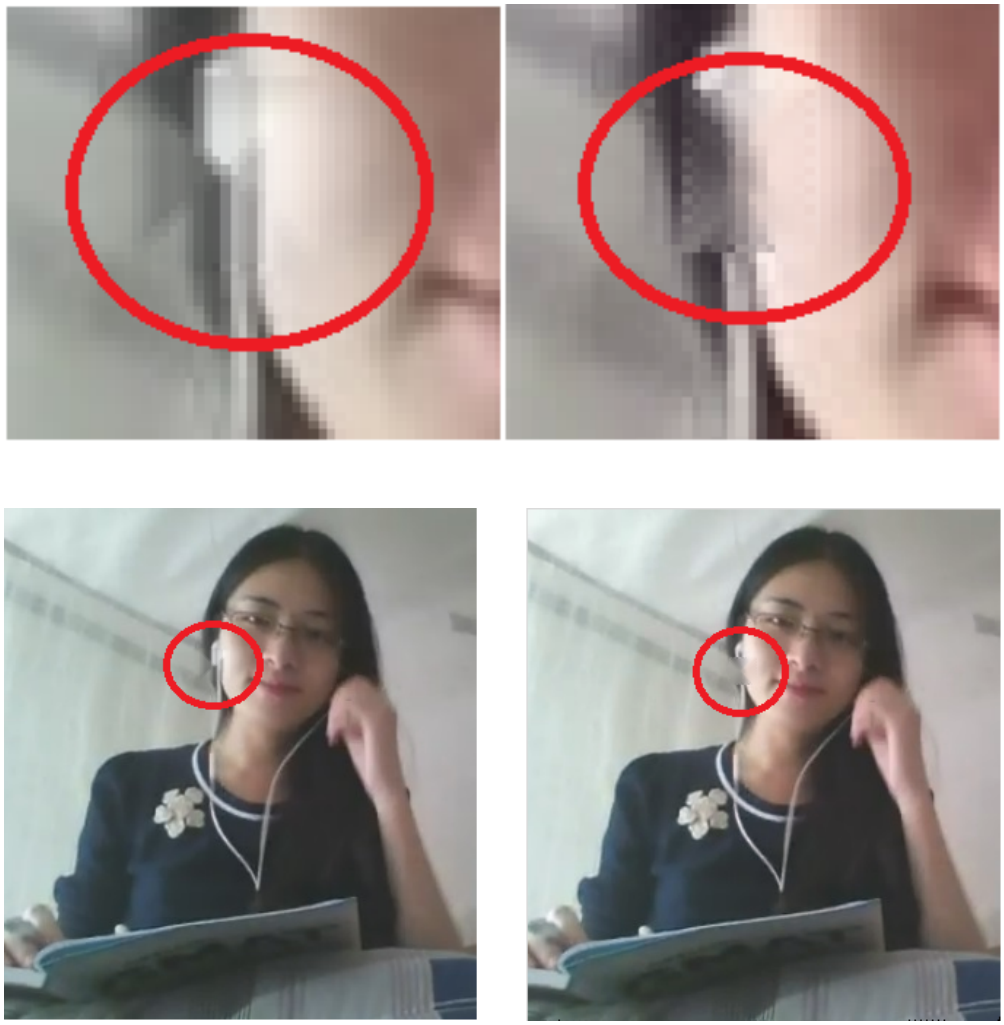


FIGURE 4.14: The above images are the inpainting result of our algorithm and the inpainting result of Telea's fast marching method algorithm. In the example, we can see that the fast marching algorithm generated the blurring artifacts within the target region and proved distracting to some degree. In this case, our algorithm produced a more natural result which has less noticeable artifacts in the missing area.

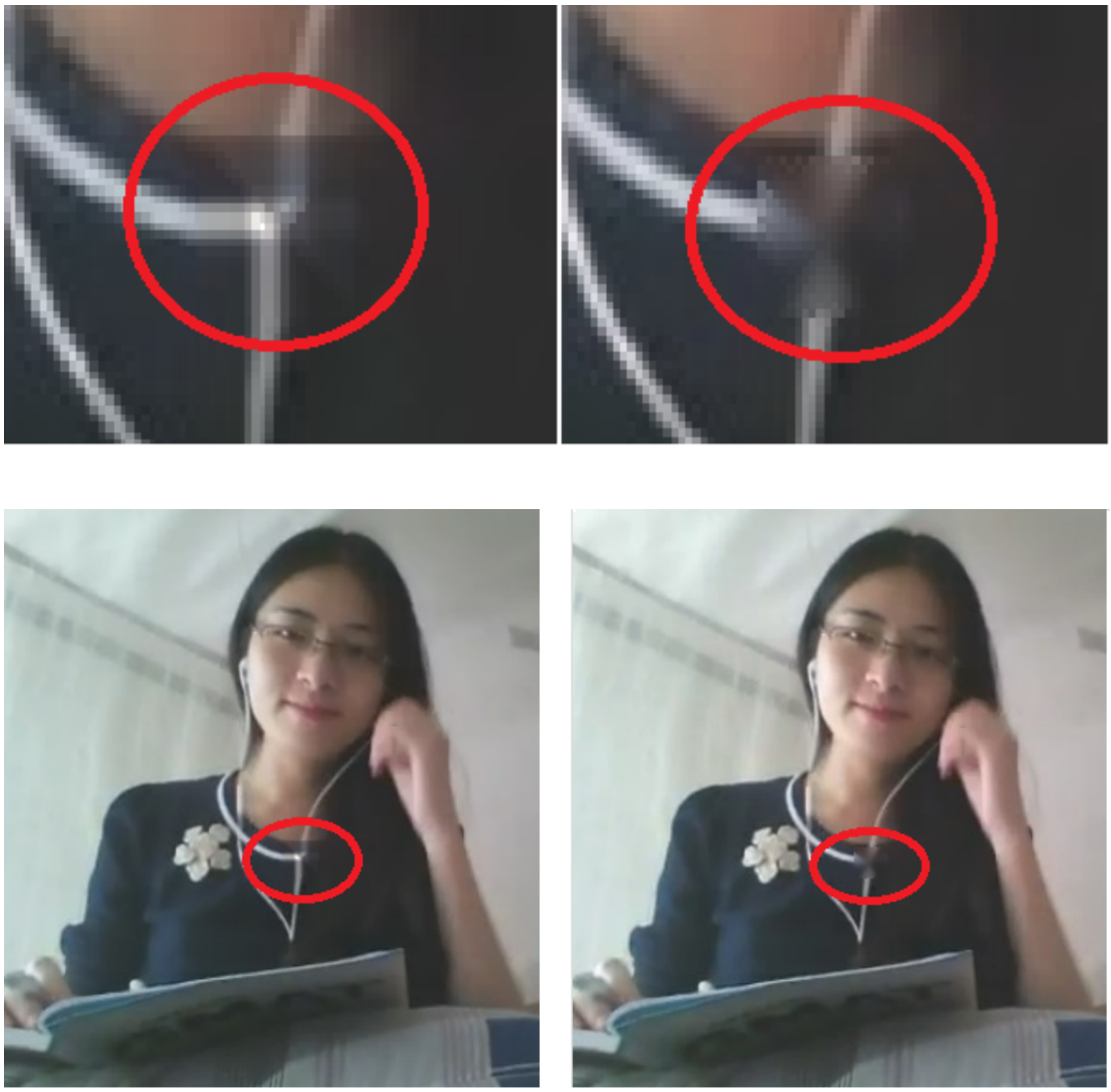


FIGURE 4.15: The above images are the inpainting result of our algorithm and the inpainting result of the fast marching algorithm. In the example, we can see that Telea's fast marching algorithm generates the blurring artifacts within the target region, and it is distracting in some degree.

In this section, the experiment results show that our algorithm has produced more natural results in the experiments. Similar to the PDE based inpainting algorithm, the fast marching algorithm will generate some blurring artifacts when inpainting a missing area with a complex texture as shown in Fig 4.14 and 4.15. The blurring artifacts generated by the fast marching algorithm shown in Fig 4.14 are not very distracting when we look at the whole image. However, in Fig 4.15, the blurring artifacts in the

result of the fast marching algorithm are distracting to some degree. Meanwhile, for the examples shown in Fig 4.15, our algorithm almost recovered the line structures in the missing area and produced a more natural result.

In chapter 4, we have discussed the results of applying our algorithm to inpaint many different image examples, and the comparison of our algorithm with the PDE based algorithm and the fast marching algorithm.

In section 4.1 and 4.2, the experiment results have well demonstrated that our algorithm can extract the horizontal and vertical structural information from the surrounding region of the missing area. However, when the surrounding area's texture is relatively complex, our algorithm may fail to reconstruct the missing area. Meanwhile, we have provided the processing time of all the experiment results. In section 4.3, we have discussed the situation that our algorithm can not provide an acceptable result.

In section 4.4 and 4.5, the experiments results have demonstrated that our algorithm can produce comparable or even better results when comparing with the fast marching algorithm and the PDE based inpainting algorithm. The former has a similar time complexity with our algorithm, and the latter has a more complex time complexity.

Chapter 5

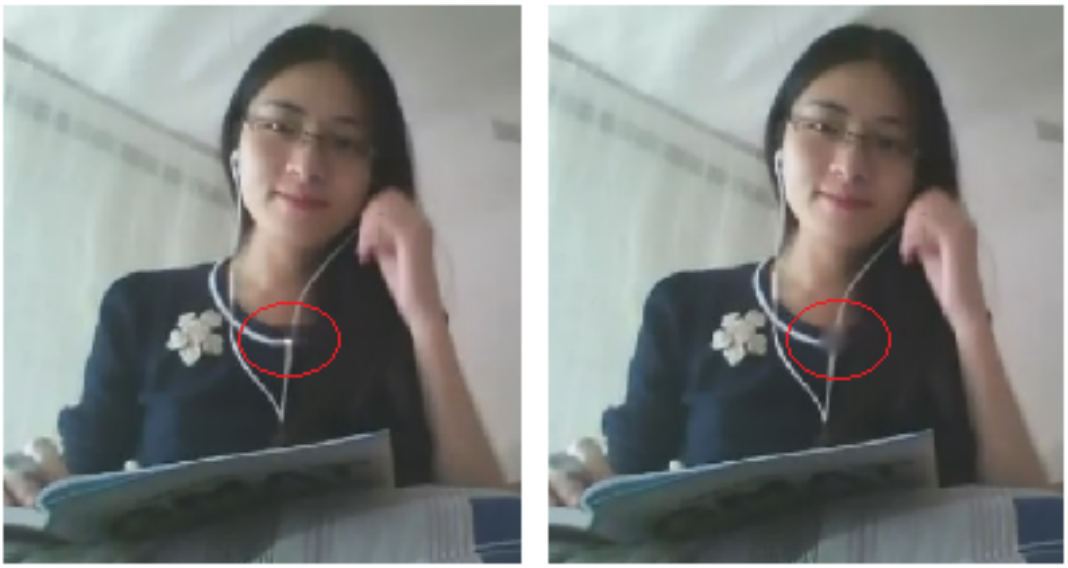
Conclusions

The thesis examines a fast and efficient algorithm developed by the author under the supervision of Dr. Jeffrey Uhlman to solve the digital inpainting problem. The algorithm is targeting the inpainting problem with small missing areas, and it can address the inpainting problem with the time complexity that is linear to the number of the missing pixels. The general idea of the algorithm is that after extracting the vertical and horizontal scaling information from the surrounding area of the target area, the algorithm will apply simple filling algorithms to the pixels within the missing area. Then finally, the algorithm will use the extracted scaling information to recover the missing area.

Our algorithm is well suitable for solving the real-time steam video inpainting problem. During the transmission process of the real-time stream video, bit error and the loss packets may affect the received video frames. Since the data of the video is block-based encoded, the damaged parts of the video frames can be blocks of missing or damaged pixels. For inpainting the mentioned blocks of pixels, the algorithm must address the problem within milliseconds. Meanwhile, the algorithm cannot introduce very distracting artifacts into the missing area. Since our algorithm may provide acceptable results, and the time-complexity is only linear to the number of the missing pixels. We are confident that the algorithm can be applied to the real-time steam video inpainting problem.

In chapter 4, we have presented many experiment results of applying our algorithm to different types of missing areas. The examples demonstrate that our algorithm can extract the structural information oriented horizontally and vertically from the known regions well. However, the algorithm may fail to catch the structural information from the diagonal direction as examples shown in section 4.3.

Later in chapter 4, we have compared the inpainting results of our algorithm with the results of the PDE based inpainting algorithm and the fast marching inpainting algorithm. The experiment results have shown that for many cases, our algorithm can produce better inpainting results even when comparing with the PDE based algorithm which has a more complex time-complexity. The following image is an example that our algorithm has produced a better result. The image on the left is the result of our algorithm and the image on the right is the result of the PDE based algorithm.



For the future work, we need to address the problem that our algorithm cannot extract the structural information oriented diagonal direction as mentioned in section 4.3. Another issue is that our algorithm can not inpaint large texture areas. For the time limitation of the real-time video inpainting, we can not extend the algorithm to have a very complicate inpainting process, which will increase the time-complexity. However, we expect to develop a new simple filling method for matrix **B** in the future. And with the new simple filling method, the original inpainting algorithm may handle the missing areas with complex texture better.

Bibliography

References

[1] Michael Ashikhmin. "Synthesizing Natural Textures". In: *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. New York, NY, USA: ACM, 2001, pp. 217–226. ISBN: 1-58113-292-1. DOI: 10.1145/364338.364405. URL: http://doi.acm.org/10.1145/364338.364405.

[2] Mr Shanawaz A. Basith and Mr Stephen R. Done. "Digital Video, MPEG and Associated Artifacts". In: 4 (1996). URL: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/sab/report.html.

[3] Tony F. Chan and Jianhong Shen. "Mathematical Models for Local Nontexture Inpaintings". In: *SIAM Journal on Applied Mathematics* 62.3 (Dec. 2001), pp. 1019–1043. URL: http://www.jstor.org/stable/3061798.

[4] Tony F. Chan and Jianhong Shen. "Non-Texture Inpainting by Curvature-Driven Diffusions (CDD)". In: *J. Visual Comm. Image Rep* 12 (2001), pp. 436–449.

[5] A. Criminisi, P. Perez, and K. Toyama. "Object removal by exemplar-based inpainting". In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 2. 2003, II–721–II–728 vol.2. DOI: 10.1109/CVPR.2003.1211538.

[6] Multimedia Communications Laboratory University of Texas at Dallas. *Source-Channel Coding of Progressive Sources*. http://www.utdallas.edu/~aria/mcl/joint/. Accessed Feb, 2002.

[7] John D'Errico. *Inpaint-nans, MATLAB Central File Exchange*. [(Updated 13 Aug 2012)]. 29 Feb 2004. URL: http://www.mathworks.com/matlabcentral/fileexchange/4551-inpaint-nans.

[8] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. "Fragment-based Image Completion". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 303–312. ISSN: 0730-0301. DOI: 10.1145/882262.882267. URL: http://doi.acm.org/10.1145/882262.882267.

[9] A.A. Efros and T.K. Leung. "Texture synthesis by non-parametric sampling". In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 2. 1999, 1033–1038 vol.2. DOI: 10.1109/ICCV.1999.790383.

[10] Alexei A. Efros and William T. Freeman. "Image Quilting for Texture Synthesis and Transfer". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 341–346. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383296. URL: http://doi.acm.org/10.1145/383259.383296.

[11] Chih-Wei Fang and J.-J.J. Lien. "Rapid Image Completion System Using Multiresolution Patch-Based Directional and Nondirectional Approaches". In: *Image Processing, IEEE Transactions on* 18.12 (2009), pp. 2769–2779. ISSN: 1057-7149. DOI: 10.1109/TIP.2009.2027635.

[12] Pascal Getreuer. "Total Variation Inpainting using Split Bregman". In: *Image Processing On Line* 2 (2012), pp. 147–157.

[13] Paul Harrison. "A Non-Hierarchical Procedure for Re-Synthesis of Complex Textures". In: *In WSCG '2001 Conference proceedings*. 2001, pp. 190–197.

[14] James Hays and Alexei A. Efros. "Scene Completion Using Millions of Photographs". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301. DOI: 10.1145/1276377.1276382. URL: http://doi.acm.org/10.1145/1276377.1276382.

[15] David J. Heeger and James R. Bergen. "Pyramid-based Texture Analysis/Synthesis". In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 229–238. ISBN: 0-89791-701-4. DOI: 10.1145/218380.218446. URL: http://doi.acm.org/10.1145/218380.218446.

[16] Hendrawan and N. I. Yusuf. "Impact analysis of bit error transmission on quality of H.264/AVC video codec". In: *Telecommunication Systems, Services, and Applications (TSSA), 2012 7th International Conference on*. 2012, pp. 314–317. DOI: 10.1109/TSSA.2012.6366074.

[17] Jason C. Hung et al. "Exemplar-based Image Inpainting base on Structure Construction". In: *Journal of Software* 3.8 (2008). URL: http://ojs.academypublisher.com/index.php/jsw/article/view/03085764.

[18] N. Komodakis and G. Tziritas. "Image Completion Using Efficient Belief Propagation Via Priority Scheduling and Dynamic Pruning". In: *Image Processing, IEEE Transactions on* 16.11 (2007), pp. 2649–2661. ISSN: 1057-7149. DOI: 10.1109/TIP.2007.906269.

[19] Stanley Osher Leonid I. Rudin 1 and Emad Fatemi 2. "Nonlinear total variation based noise removal algorithms". In: *Physica D* 60.60 (Nov. 1992), pp. 259–268.

[20] V. Caselles M. Bertalmío G. Sapiro and C. Ballester. "Image Inpainting". In: *Proceedings of SIGGRAPH 2000* (July 2000).

[21] Prof. M. B. Vaidya2 Komal s Mahajan1. "Image in Painting Techniques: A survey". In: *IOSR Journal of Computer Engineering (IOSRJCE)* 5 (2012), pp. 45–49.

[22] Vijay Venkatesh Mahalingam. "DIGITAL INPAINTING ALGORITHMS AND EVALUATION". Doctoral Dissertations. Paper 55. PhD thesis. University of Kentucky, 2010.

[23] Manuel M. Oliveira et al. "Fast Digital Image Inpainting". In: *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VISUALIZATION, IMAGING AND IMAGE PROCESSING (VIIP 2001*. ACTA Press, 2001, pp. 261–266.

[24] Patrick Pérez, Michel Gangnet, and Andrew Blake. "Poisson Image Editing". In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 313–318. ISSN: 0730-0301. DOI: 10.1145/882262.882269. URL: http://doi.acm.org/10.1145/882262.882269.

[25] Uriel G. Rothblum and Stavros A. Zenios. "Scalings of matrices satisfying line-product constraints and generalizations". In: *Linear Algebra and its Applications* 175 (1992), pp. 159 –175. ISSN: 0024-3795. DOI: http://dx.doi.org/10.

1016/0024-3795(92)90307-V. URL: http://www.sciencedirect.com/science/article/pii/002437959290307V.

[26] J.-L. Starck, M. Elad, and D.L. Donoho. "Image decomposition via the combination of sparse representations and a variational approach". In: *Image Processing, IEEE Transactions on* 14.10 (2005), pp. 1570–1582. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.852206.

[27] Yair Censor Stavros A. Zenios. "Massively Parallel Row-Action Algorithms for Some Nonlinear Transportation Problems". In: *Linear Algebra and its Applications* 1 (1991), pp. 373–400. ISSN: DOI: 10.1137/0801024. URL: https://www.researchgate.net/publication/230663688_Massively_Parallel_Row-Action_Algorithms_for_Some_Nonlinear_Transportation_Problems.

[28] Jian Sun et al. "Image Completion with Structure Propagation". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 861–868. ISSN: 0730-0301. DOI: 10.1145/1073204.1073274. URL: http://doi.acm.org/10.1145/1073204.1073274.

[29] Alexandru Telea. "An Image Inpainting Technique Based on the Fast Marching Method". In: *Default journal* (2004). Relation: http://www.rug.nl/informatica/organisatie/overc Rights: University of Groningen. Research Institute for Mathematics and Computing Science (IWI).

[30] D. Tschumperle and R. Deriche. "Vector-valued image regularization with PDEs: a common framework for different applications". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 27.4 (2005), pp. 506–517. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.87.

[31] Jeffrey Uhlmann. "Unit Consistency, Generalized Inverses, and Effective System Design Methods". In: (2015).

[32] Wikipedia. *Bit error rate — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-March-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Bit_error_rate&oldid=703334494.

[33] Wikipedia. *Communications protocol — Wikipedia, The Free Encyclopedia*. [Online; accessed 13-March-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Communications_protocol&oldid=709288525.

[34] Wikipedia. *Inpainting — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-September-2015]. 2015. URL: \url{https://en.wikipedia.org/w/index.php?title=Inpainting&oldid=667321097}.

[35] Wikipedia. *Streaming media — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-March-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Streaming_media&oldid=705082174.

[36] Wikipedia. *Transmission (telecommunications) — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-March-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Transmission_(telecommunications)&oldid=708047336.

[37] A. Wong and J. Orchard. "A nonlocal-means approach to exemplar-based inpainting". In: *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*. 2008, pp. 2600–2603. DOI: 10.1109/ICIP.2008.4712326.