ECE 9524: Digital Image Processing

Final Project Progress Report

Andrew Powell

May 4, 2015

Introduction

1.1 Motivation

Inpainting is the process of recreating portions of an image from known portions of the same image. The image's known portions could be from the image itself, or even from neighboring frames if the image in question is also a frame from the same film. Still the challenge is always to remove the unwanted areas of an image in such a way that the final image appears "visually complete" [2].

The discussion of inpainting in this proposal refers to the methods developed to perform inpainting on digital images. However, inpainting is really an ancient art, which artists have been applying to damaged photographs, painted works of art, and many other applications [1].

The goal, however, is always to fill in unwanted portions of an image with something that appears correct. Today, digital implementations of inpainting are becoming more and more relevant so as to minimize resources, such as human involvement and time, and maximize the quality of the results, which is primarily visual "correctness" [1].

Modern applications can potentially include special effects in film production, virtual reality simulation, digital cameras, and even *mobile phones.* For instance, an image is transmitted between one phone to another, and, for some reason, small portions of the image's data becomes lost during the transmission. Digital inpainting techniques could be another approach to regenerating the lost information [4].

1.2 Goal

For this project, however, the overall goal is to devise and implement an inpainting method practical enough to run on a mobile phone for the "typical" user. For instance, let's assume the user is someone who regularly uses social media services such as Facebook and Instagram. With the goal of satisfying this hypothetical person, the inpainting method would demand both quality results and a *"reasonable"* computation time.

Another goal includes comparing the performance of the proposed inpainting method with that of several alternative methods.

1.3 Inpainting

As already mentioned, inpainting is simply the process of reconstructing unwanted portions of an image from the known portions. In the context of this final report, the unwanted portions are referred to as the *occlusion* or the *inpainting region*. An occlusion refers to something that blocks the part of the image that needs to be seen. Hence, the removal of the occlusion is equivalent to image inpainting. The portion of the image from which every inpainting method acquires its information is simply referred to as the *known portion of the image*, in the context of this report. Finally, the image produced by the end of an inpainting method is referred to as the *inpainted image*.

Digital inpainting techniques are "generally" divided into three categories of algorithms. Their names and explanations slightly vary, depending on the source. The three categories are *structural*, *textural* synthesis, and a combination of the two. Structural techniques tend to perform relatively better on small occlusions but excel at the reconstruction of the image's structure. An image's structure refers to contour lines, or *isophotes*, and the objects visible in an image.

On the other hand, textural synthesis refers to the techniques excellent at recreating relatively larger, uniform occlusions with texture found in the image itself. The disadvantage of purely texture-based techniques, of course, is the loss of structural information. Obviously, the ideal solution is the third category, in which advantages of both structural- and textural-based techniques are combined [2].

Alternative Methods

Before the proposed method to inpainting is explained, it is necessary to first describe three alternative methods, the first two of which rely heavily upon solving discretized *partial derivative equations (PDE)* to recreate structural information and the third of which relies on the simple principal of *exemplar*-based inpainting to recreate textures. The three alternative methods are, in essence, uniquely combined to result in the proposed method so as to recreate textures and structure, while also focusing on reducing computation time. Section 2.1 focuses on PDE inpainting, whereas Section 2.2 concentrates on the exemplar inpainting.

2.1 Structural Inpainting

2.1.1 "PDE" method / Bertalmio

Bertalmio Et. Al. first introduced inpainting to digital image processing, deriving their method through observing and consulting artists who manually inpainted damaged paintings and other hand drawn pictures. Bertalmio Et. Al. learned, in order to recreate an inpainted region, the artist first had to extend structural information from the boundary of the inpainted region and then, once contours were connected, textural information determined from the image's known portions followed.

The PDE method described in [1] focuses solely on the structural inpainting by simply employing the idea that structures are created by extending the contours of the image's known portions into the inpainted region. Thus, Bertalmio Et. Al presented the following PDE which is only computed over the inpainted region.

$$\frac{\partial u}{\partial t} + \nabla^{\perp} u \cdot \Delta u = 0 \tag{2.1}$$

The idea behind Equation 2.1 is to apply a smoothing operation only in the direction of the image's contour. u is a short hand notation for the intensity $u(\vec{x}, t)$, for which $\vec{x}$ describes the location of u in the image at time equal to t. PDE methods, in general, are applied iteratively and thus the inpainted region converges to a result overtime (ideally). $\frac{\partial u}{\partial t}$ would be the partial derivative of u with respect to t. Δ refers to the Laplacian operator, whereas ∇ refers to the gradient. $\nabla^{\perp}$, specifically, refers to the pixel's isophote.

In order to prevent the PDE method from diverging, though, a smoothing operation capable of the preserving the structural information is needed. Bertalmio Et. Al. apply a few iterations of *anisotropic diffusion* for every few iterations of their PDE method, though it is assumed another smoothing operation is applicable, so long as the structures are preserved. In the context of this project, Perona-Malik anisotropic diffusion is carried out and whose operation is described by the following formulation.

$$\frac{\partial u}{\partial t} = \nabla \cdot (c \nabla u) \tag{2.2}$$

The $\nabla \cdot$ operation refers to the divergence operation and c is a shorthand notation for the diffusion coefficient $c(||\nabla u||)$, for which $|| \bullet ||$ is the magnitude of the vector array $\bullet$. c is defined such that the coefficient becomes smaller for larger values of the gradient. It is worth noting the removal of c from Equation 2.2 reduces the equation to $\nabla \cdot \nabla u$, which is the definition of the laplacian operator Δu.

[1].

2.1.2 "Fast PDE" method / Cahn / Gillette

(Unfortunately, I devoted most of my time to finishing the project but allotted little time to writing the report, so some sections will be incomplete as far as all the writing

is concerned.)

$$\begin{aligned}
\frac{\partial u}{\partial t} &= -\Delta\left(\epsilon\Delta u + \epsilon^{-1}\frac{dW(u)}{du}\right) + \lambda(\vec{x})(f-u) \\
W(u) &= u^2(u-1) \\
\lambda(\vec{x}) &= \begin{cases} 0 & \text{if } \vec{x} \in \text{ Inpainted region} \\ \lambda_0 & \text{if } \vec{x} \in \text{ Known portion of Image} \end{cases}
\end{aligned} \tag{2.3}$$

$$\begin{aligned}
E_1 &= \int_{\text{Image}} \frac{\epsilon}{2}|\nabla u|^2 + \frac{1}{\epsilon}W(u)d\vec{x} \\
&= E_{11} - E_{12} \\
E_{11} &= \int_{\text{Image}} \frac{\epsilon}{2}|\nabla u|^2 + \frac{C_1}{2}|u|^2 d\vec{x} \\
E_{12} &= \int_{\text{Image}} \frac{1}{\epsilon}W(u) + \frac{C_1}{2}|u|^2 d\vec{x}
\end{aligned} \tag{2.4}$$

$$\begin{aligned}
E_2 &= \lambda_0 \int_{\text{Known portion of Image}} (f-u)^2 d\vec{x} \\
&= E_{21} - E_{22} \\
E_{21} &= \frac{C_2}{2}|u|^2 d\vec{x} \\
E_{22} &= \int_{\text{Known portion of Image}} -\lambda_0(f-u)^2 + \frac{C_2}{2}|u|^2 d\vec{x}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial u}{\partial t} &= -\left[\nabla E_1 + \nabla E_2\right] \\
\frac{u(n+1)-u(n)}{\Delta t} &= -\left[\nabla(E_{11}(n+1) - E_{12}(n+1)) + \nabla(E_{21}(n+1) - E_{22}(n+1))\right]
\end{aligned} \tag{2.5}$$

[3]

2.2 Textural Inpainting

2.2.1 "Exemplar" method / Criminsi

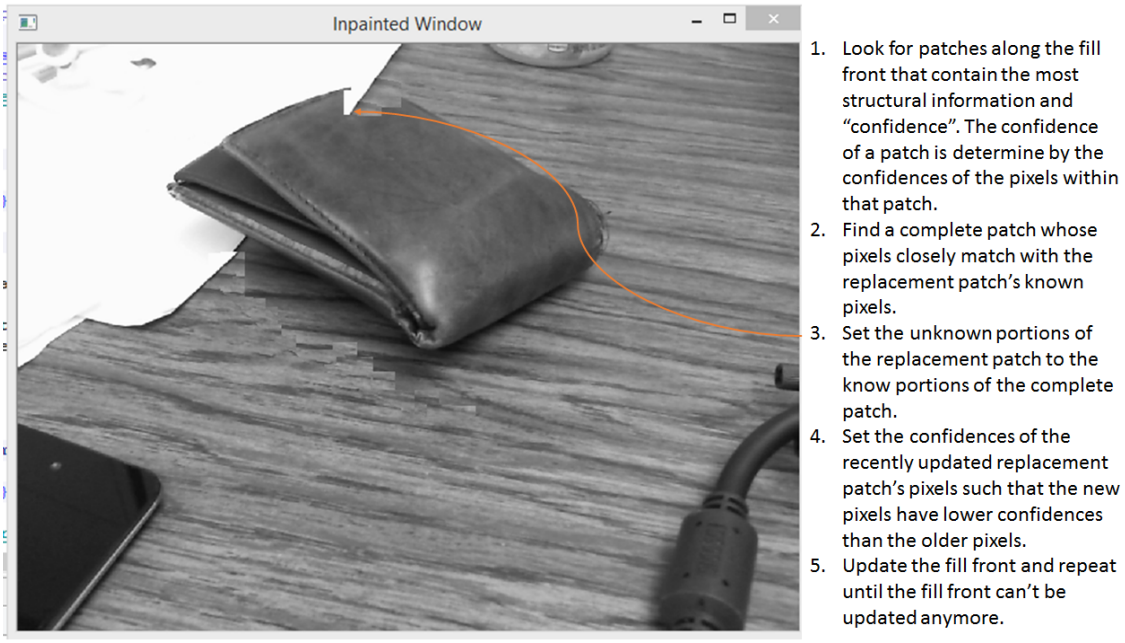

Figure 2.1

[2].

Proposed Method

The proposed method can be summarized as the application of the exemplar method explained in Section 2.2 for the recreation of the inpainted region's textures and then the application of the Fast-PDE approach described in Section 2.1.2. However, the differences in how the exemplar inpainting method is modified makes the proposed method, in comparison to the alternative methods, more practical for mobile applications. Specifically, the exemplar portion of the proposed method is modified in order consider the following information with the purpose of reducing computation time while maintaining decent quality inpainted images.

1. *Distance* — Information in an image tends to repeat, locally. In other words, having to check every pixel's patch for a potential complete patch to match with the replacement patch is likely to waste a lot of time. Instead, only nearby areas are searched and thus time is saved.

2. *Skip repeated information* — Information may repeat along a particular direction. Thus, in lieu of checking every pixel's patch, patches can be skipped along the direction where the information is redundant.

The Fast-PDE method is simply applied once the modified exemplar method is complete as a way to mend together structures in the image.

3.1 Structural and Textural Approach

Analysis / Results

The analysis of the proposed method with the alternative approaches to inpainting is composed of two sections, the first of which, Section 4.1, is comprised of how the inpainting methods are implemented and then compared over a chosen set of pictures, and the second of which, Section 4.2, discusses the development of two standalone applications created mostly for demonstration purposes. The two standalone applications are also developed to show how well the propose method operates on a desktop computer as compared to a mobile application.

It should be noted the proposed method and each of the alternative methods were first implemented as MATLAB R2014a code, specifically taking advantage of MATLAB's Image Processing Toolbox for filtering, accessing, and applying other image processing operations on images. Additionally, with the intention of reducing computation time, the Parallel Computing Toolbox was utilized so as to parallelize operations within the exemplar and proposed methods.

After successfully completing the inpainting methods in MATLAB, which consisted of not only writing the software but also creating the appropriate testing software to ensure each implementation's correctness, each of the inpainting methods were then developed in C++11, utilizing the 2.4.10.0 release of OpenCV for its highly optimized functionality ranging from linear algebraic functions to image processing. The software already developed in MATLAB were ultimately used as a references in the C++ implementations. For the sake of simplicity, all the inpainting methods were implemented to operate over single-precision floating-point intensity values for each pixel and to only operate over a single channel. Finally, only the C++ implementation of the prosed method is integrated in both the simulation and the two demonstration programs.

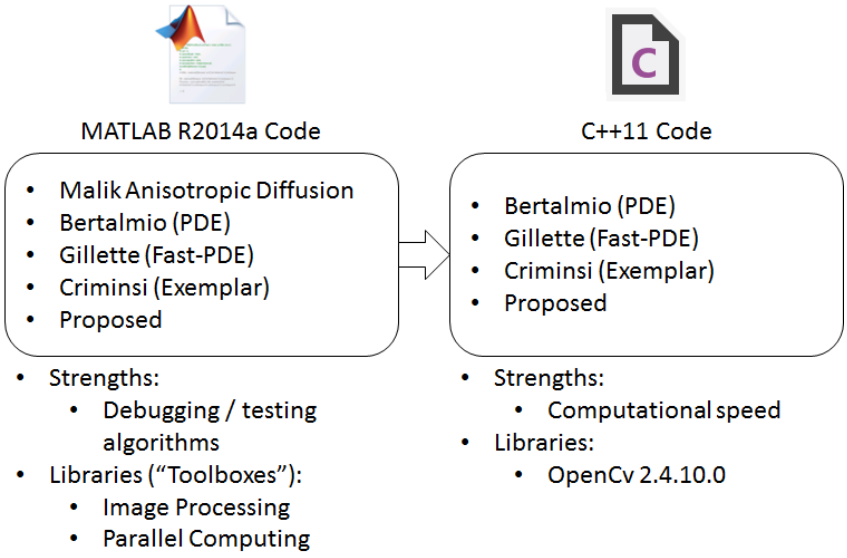

Figure 4.1: Software development evolution

4.1 Simulation

The simulation consists of running each of the *four inpainting methods* over each of the *six selected test images*, while collecting the elapsed time and the inpainted image. The simulation script itself was developed within MATLAB simply for its analytical tools, including plotting inpainted images and measuring elapsed time of each inpainting method. As described, the four inpainting methods are PDE (Bertalmio), Fast-PDE (Gillette), Exemplar (Griminsi), and finally the proposed method.

The six test images can be divided into two groups. The first group is composed of *four simple images*, all of which were created with MS Paint and MATLAB's blurring operations from its Image Processing Toolbox. The four simple images test how well each inpainting method is able to operate over simply structures, such as contours and distinctive shapes. The characteristics of the four simplistic test images range from clear and distinct straight lines to curvy and smooth contours. Apart from the structural information, however, the four simplistic images all have the same resolution and *simple* inpainting regions.

The second group of test images are the *complex images* consisting of actual photographs. As such, the complex images are naturally closer to a real application for which this project is intended. Moreover, the complex images not only contain structural information,

but also textural. Unlike the four simplistic test images, though, the inpainting regions for the complex images are chosen differently. Specifically, the first complex image, which is the picture of Albert Einstein shown in Figure 4.6, is degraded with small, yet *scattered* inpainting regions. On the other hand, the second complex image, which happened to be a picture of a women standing in front of a waterfall in Figure 4.7, has a *large* occlusion that entirely covers the women.

Sections 4.1.1 to 4.1.1 each display a test image with inpainting region and respective information. The noisy areas of each image are the selected inpainting regions, chosen based on the strength and weaknesses of each inpainting method.

4.1.1 Test Images

bar image

Figure 4.2: Test Image

Table 4.1: *bar* information

Parameter	Value
Resolution (pixels)	264x136
Mask Area (pixels)	341
Mask Type	Simple

ball image

Table 4.2: *ball* information

Parameter	Value
Resolution (pixels)	264x136
Mask Area (pixels)	341
Mask Type	Simple

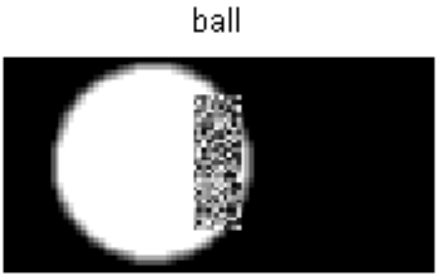

Figure 4.3: Test Image

***slide* image**

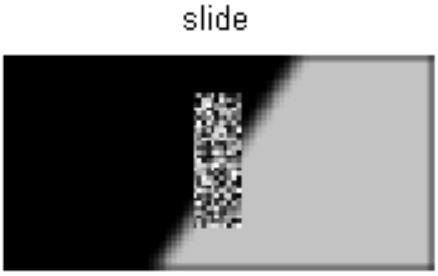

Figure 4.4: Test Image

Table 4.3: *slide* information

Parameter	Value
Resolution (pixels)	264x136
Mask Area (pixels)	341
Mask Type	Simple

***balls* image**

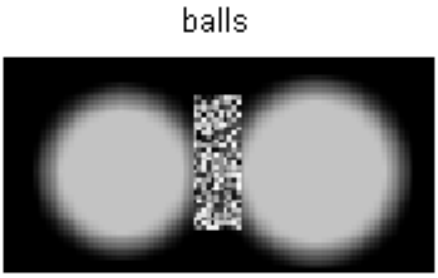

Figure 4.5: Test Image

Table 4.4: *balls* information

Parameter	Value
Resolution (pixels)	264x136
Mask Area (pixels)	341
Mask Type	Simple

***einstein* image**

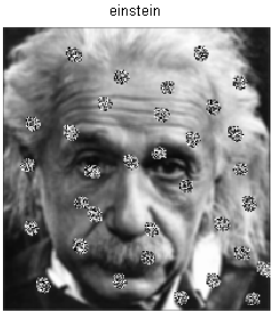

Figure 4.6: Test Image

Table 4.5: *einstein* information

Parameter	Value
Resolution (pixels)	364x298
Mask Area (pixels)	3388
Mask Type	Scattered

***waterfall* image**

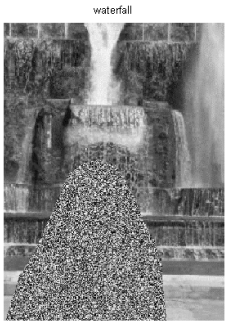

Figure 4.7: Test Image

Table 4.6: *waterfall* information

Parameter	Value
Resolution (pixels)	411x418
Mask Area (pixels)	21662
Mask Type	Large

4.1.2 Results

***bar* image**

Table 4.7: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,20000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.00001

Gillette (Cahn) Parameter	Value
Total Stages	2
Total Iters	10000,15000
ϵs	5,.5

Exem Parameter	Value
Patch Size (Pixels)	20

Proposed Parameter	Value
Patch Size (Pixels)	15
Distance (Pixels)	150
Skip Width (Pixels)	2
Skip Height (Pixels)	1
(Cahn) Total Stages	0
(Cahn) Total Iters	N/A
(Cahn) ϵs	N/A

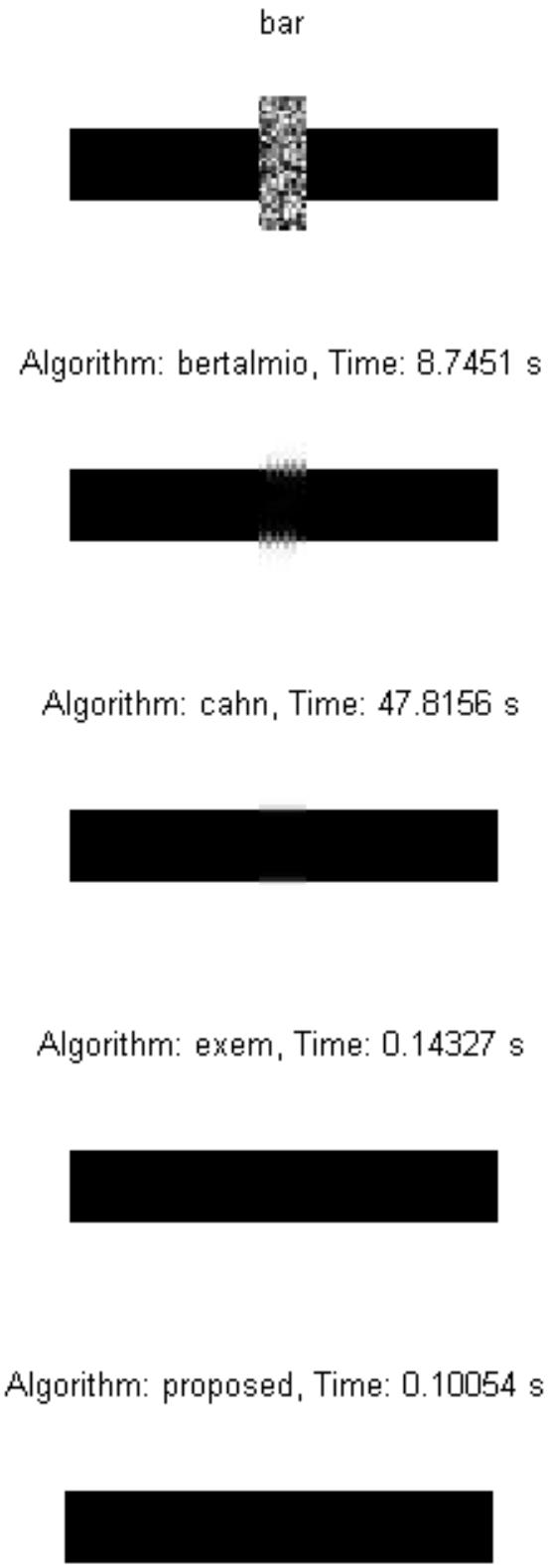

Figure 4.8: Result Image

***ball* image**

Table 4.8: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,20000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.00001

Gillette (Cahn) Parameter	Value
Total Stages	2
Total Iters	10000,15000
ϵs	5,.5

Exem Parameter	Value
Patch Size (Pixels)	20

Proposed Parameter	Value
Patch Size (Pixels)	15
Distance (Pixels)	150
Skip Width (Pixels)	1
Skip Height (Pixels)	1
(Cahn) Total Stages	2
(Cahn) Total Iters	10000,10000
(Cahn) ϵs	5,0.5

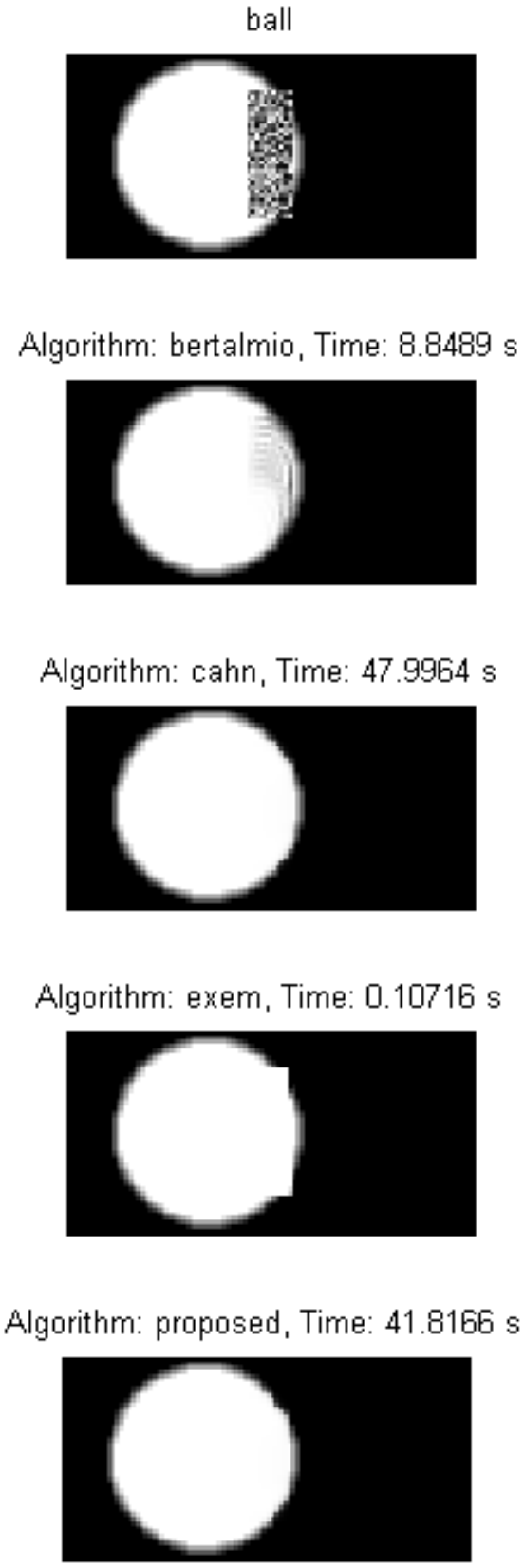

Figure 4.9: Result Image

***slide* image**

Table 4.9: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,10000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.001

Gillette (Cahn) Parameter	Value
Total Stages	2
Total Iters	10000,10000
ϵs	5,3

Exem Parameter	Value
Patch Size (Pixels)	20

Proposed Parameter	Value
Patch Size (Pixels)	15
Distance (Pixels)	150
Skip Width (Pixels)	1
Skip Height (Pixels)	1
(Cahn) Total Stages	1
(Cahn) Total Iters	100
(Cahn) ϵs	1

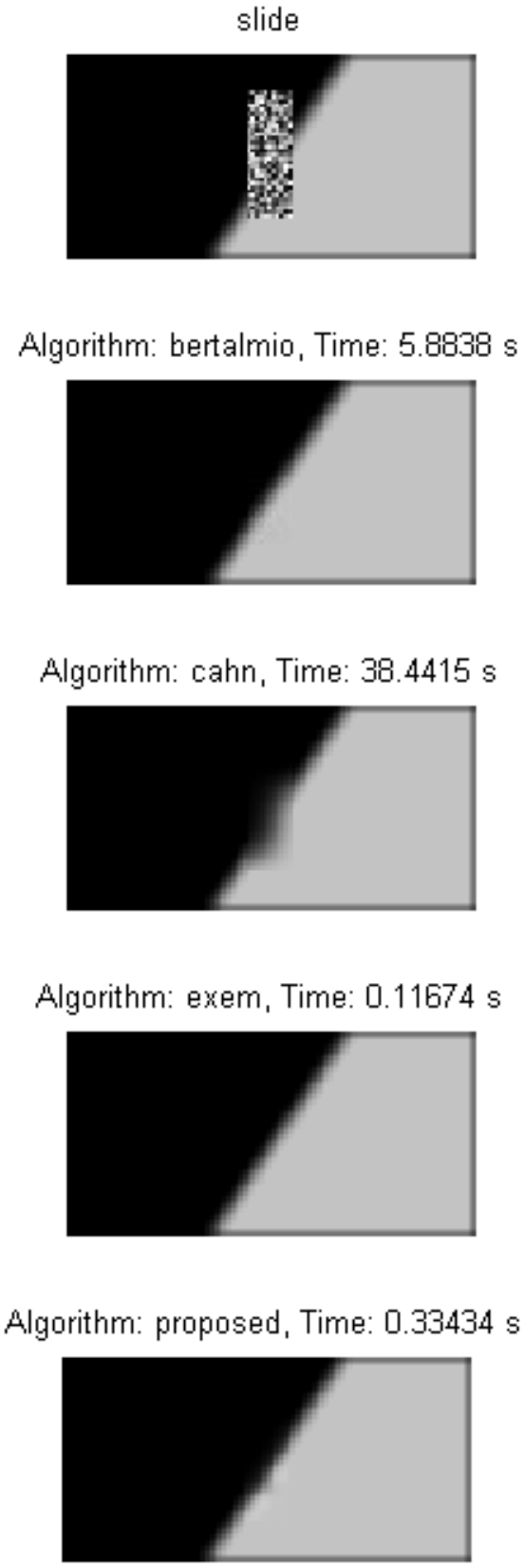

Figure 4.10: Result Image

***balls* image**

Table 4.10: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,20000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.00001

Gillette (Cahn) Parameter	Value
Total Stages	2
Total Iters	10000,10000
ϵs	3,2

Exem Parameter	Value
Patch Size (Pixels)	7

Proposed Parameter	Value
Patch Size (Pixels)	7
Distance (Pixels)	max
Skip Width (Pixels)	1
Skip Height (Pixels)	1
(Cahn) Total Stages	0
(Cahn) Total Iters	N/A
(Cahn) ϵs	N/A

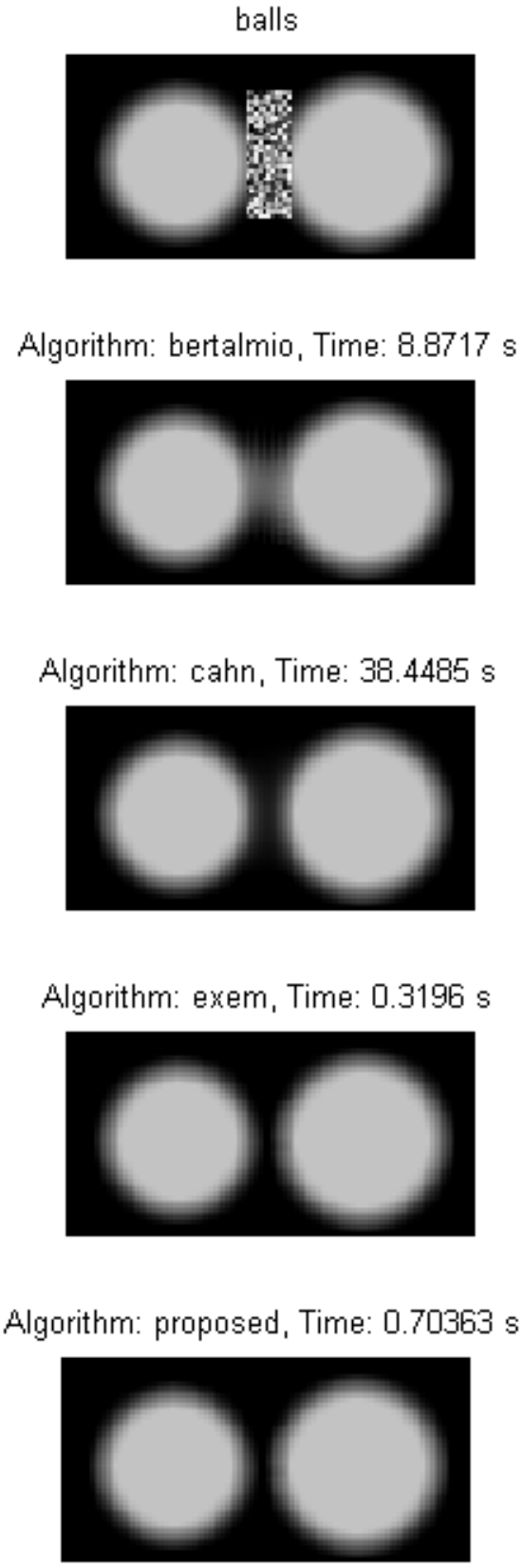

Figure 4.11: Result Image

***einstein* image**

Table 4.11: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,20000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.00001

Gillette (Cahn) Parameter	Value
Total Stages	2
Total Iters	10000,100
ϵs	2,1

Exem Parameter	Value
Patch Size (Pixels)	9

Proposed Parameter	Value
Patch Size (Pixels)	7
Distance (Pixels)	28
Skip Width (Pixels)	1
Skip Height (Pixels)	1
(Cahn) Total Stages	1
(Cahn) Total Iters	100
(Cahn) ϵs	2

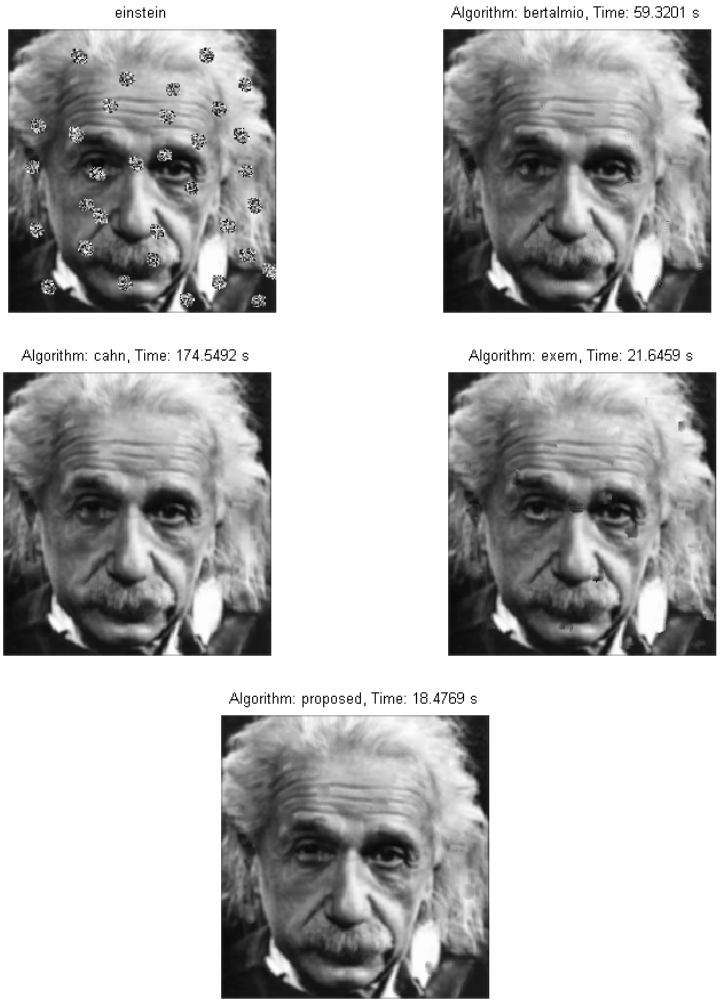

Figure 4.12: Result Image

***waterfall* image**

Table 4.12: Method Parameters

Bertalmio Parameter	Value
Total Stages	2
Total Iters	10000,20000
Total Inpaint Iters	2,2
Total Anidiffuse Iters	2,2
δts	0.001,0.00001

Gillette (Cahn) Parameter	Value
Total Stages	1
Total Iters	100
ϵs	3

Exem Parameter	Value
Patch Size (Pixels)	70

Proposed Parameter	Value
Patch Size (Pixels)	70
Distance (Pixels)	max
Skip Width (Pixels)	2
Skip Height (Pixels)	1
(Cahn) Total Stages	1
(Cahn) Total Iters	1
(Cahn) ϵs	2

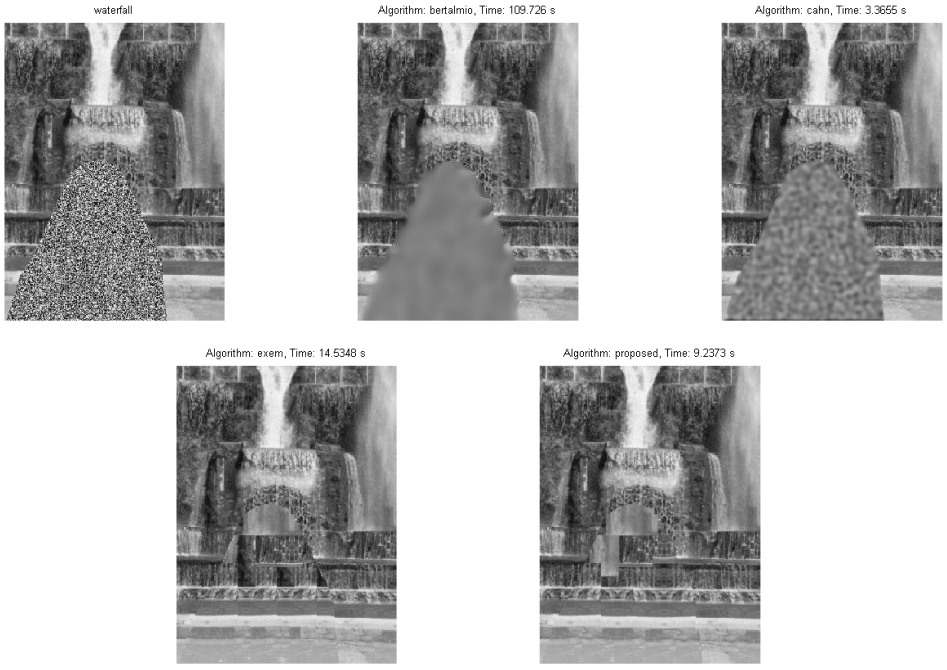

Figure 4.13: Result Image

4.2 Demonstration Programs

Considering the application of most interest in this project is to perform inpainting methods on mobile devices in the hands of regular users, such as those who regularly use Instagram, the inpainting method should provide some form of feedback to the user so

that they may at least know the operation is running. For this reason, another version of the proposed method was implemented in C++—that is to say, a version slightly different from that explained in Section 4.1.

4.2.1 PC Application

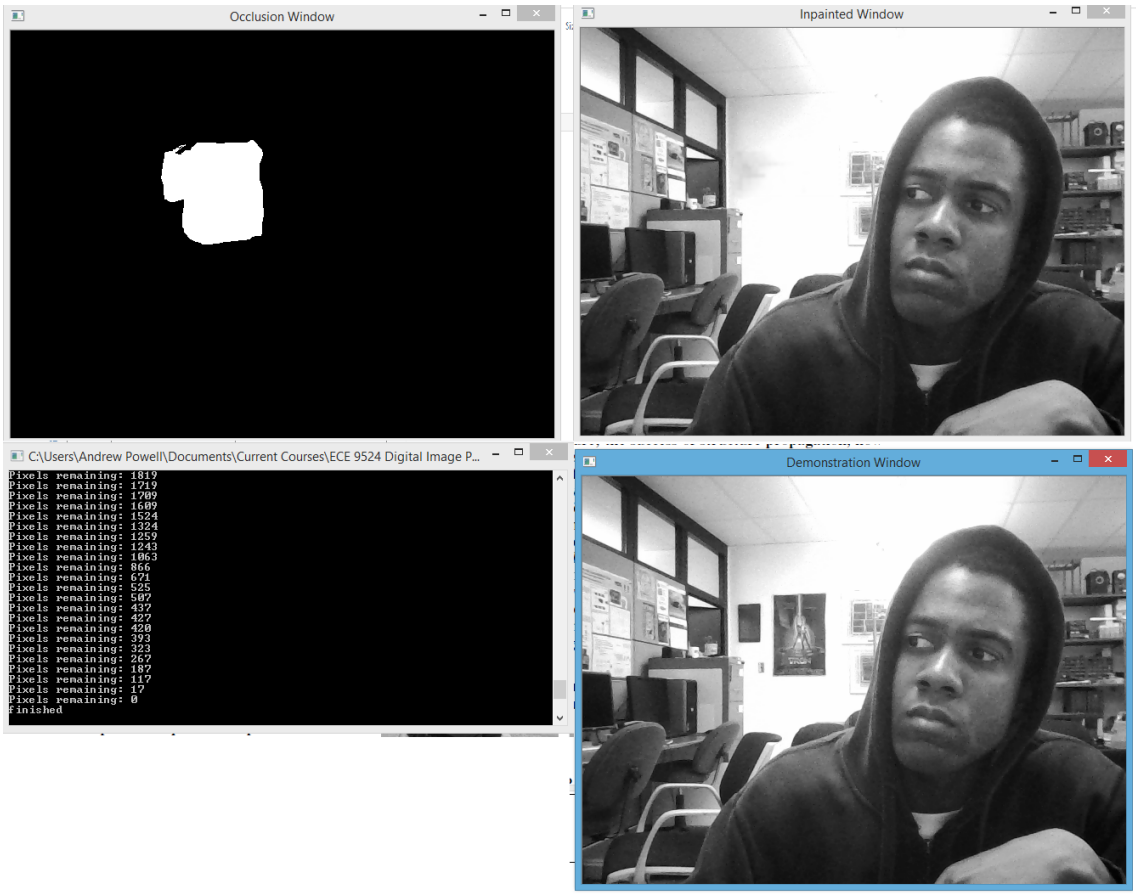

Figure 4.14: PC demonstration program GUI

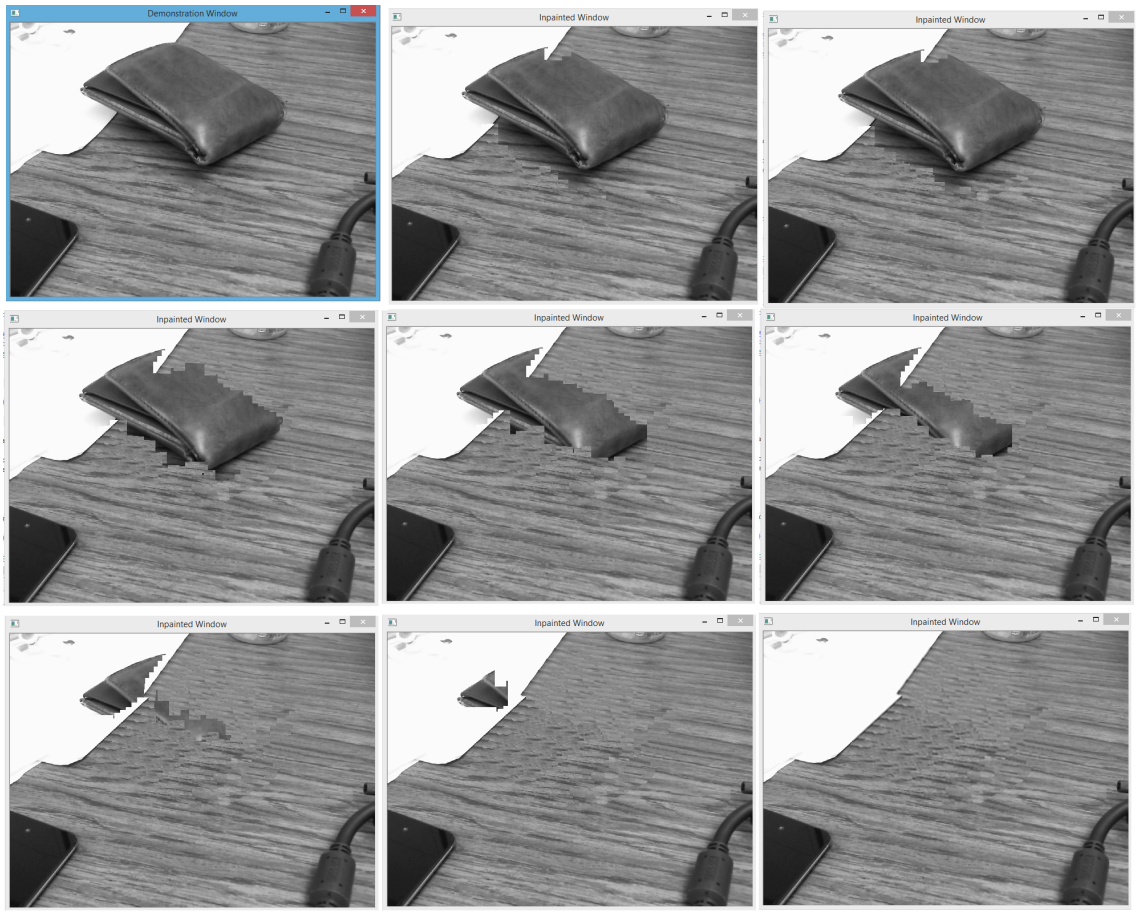

Figure 4.15: PC demonstration program result

4.2.2 Android App

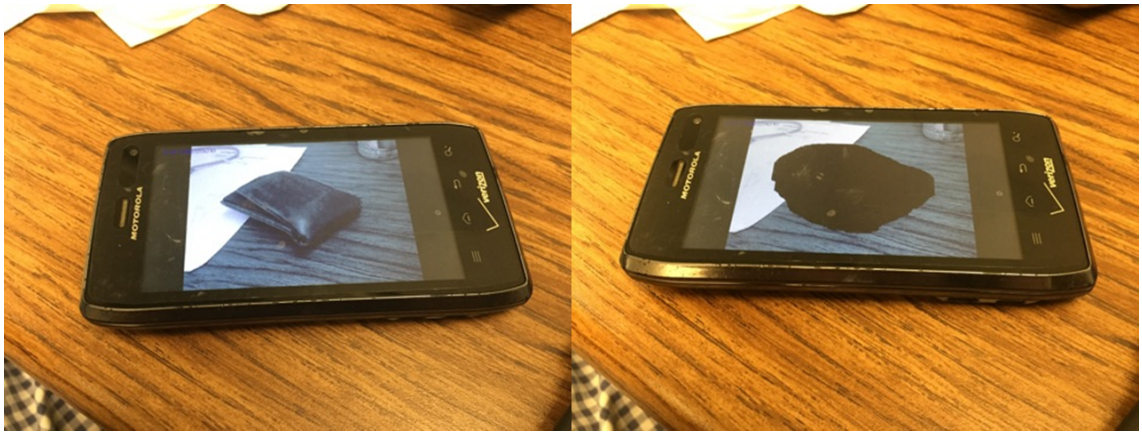

Figure 4.16: Mobile demonstration program interface

Figure 4.17: Mobile demonstration program result

Conclusion

5.1 Challenges Future Work

-different methods are has their ideal and poor applications -good results often depend on a subjective analysis of quality -determining parameters -length of time necessary to execute most operations is still way too long for real applications

-parallization -running operations over a server and then sending the results back

5.2 Closing Remarks

References

[1] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co., 2000.

[2] Antonio Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 13(9):1200–1212, Sept 2004.

[3] Alan Gillette. *Image inpainting using a modified Cahn-Hilliard equation.* PhD thesis, University of California, Los Angeles.

[4] Zhen Xie, Fan Zhang, and Conggui Zhang. An adaptive matching algorithm for image inpainting. In *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pages 1293–1296, Sept 2011.