Interactive Image Filling-In

Teryl Lynne Arnold
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Computer Sciences Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/300

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
INTERACTIVE IMAGE FILLING-IN

by

Teryl Arnold

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
April 2005
BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Teryl Arnold

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

__________________________________________  _______________________________________
Date                                              Bryan S. Morse, Chair

__________________________________________  _______________________________________
Date                                              Parris K. Egbert

__________________________________________  _______________________________________
Date                                              Kevin D. Seppi
As chair of the candidate's graduate committee, I have read the thesis of Teryl Arnold in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Bryan S. Morse
Chair, Graduate Committee

Accepted for the Department

David W. Embley
Graduate Coordinator

Accepted for the College

G.Rex Bryce,
Associate Dean, College of Physical and Mathematical Sciences
ABSTRACT

INTERACTIVE IMAGE FILLING-IN

Teryl Arnold

Department of Computer Science

Master of Science

Removing unwanted scratches or objects from an image in an undetectable manner is a technique that has been researched for its many useful and varied applications, such as removing scratches, defects, super-imposed text, or even entire objects from a scene. Currently there is a wide variety of algorithms that fill in unwanted regions, none of which incorporate user preferences into the structure completion process.

By building a framework to incorporate user preferences into the filling-in process, user input can be utilized to more effectively fill in damaged regions in an image. User input can influence the filling-in process in a variety of ways, including identifying the region to remove, guiding the completion of structure in the damaged region, influencing priority in the searching process for texture completion, and picking the best combination of structure and texture completion in the damaged region.

The framework to achieve the interactive filling-in process contains five main steps. First, the scratch or deformity is detected. Second, the edges outside the deformity
are detected. Third, curves are fit to the detected edges. Fourth, the structure is completed across the damaged region. Finally, texture synthesis constrained by the previously computed curves is used to fill in the intensities in the damaged region. Scratch detection, structure completion, and texture synthesis can be influenced or guided by user input when given.

Defects have successfully been removed from images that contain structure, images that contain texture, and images that contain both structure and texture. A user is able to successfully complete images that contain ambiguous structure in more than one viable way by gesturing the cursor in the direction of desired structure completion.
ACKNOWLEDGMENTS

I would like to acknowledge Dr. Morse for his support throughout the entire research process. I greatly appreciate all of the time and feedback both Dr. Egbert and Dr. Seppi offered. I would also like to thank Heather Ipson for her collaboration of ideas, and my family for their support.
# Contents

1 Introduction .................................................. 1
   1.1 Overview of Thesis ..................................... 5

2 Previous Work ............................................... 7
   2.1 Structure Completion Methods .......................... 7
   2.2 Texture Synthesis Methods ............................... 10
   2.3 Structure and Texture Completion Methods .......... 11

3 Methods ..................................................... 17
   3.1 User Interaction ........................................ 18
       3.1.1 Identifying Regions ............................... 18
       3.1.2 Structure Completion and Over-Constrained Searches .. 19
       3.1.3 Filling-in Finalization ........................... 19
   3.2 Scratch Detection ....................................... 19
   3.3 Edge Detection .......................................... 21
   3.4 Curve Fitting ............................................ 24
   3.5 Structure Completion ................................... 26
       3.5.1 First Search ....................................... 27
## CONTENTS

3.5.2 Second Search ............................................. 31

3.6 Texture Synthesis ........................................... 32
  3.6.1 Numerous Iterations ................................... 33
  3.6.2 Smoothness Prior ....................................... 34
  3.6.3 Constrained Search ..................................... 34
  3.6.4 Over-Constrained Search ............................... 36

4 Results and Limitations ................................. 39
  4.1 Structure Examples .................................... 40
  4.2 Texture Examples ....................................... 45
  4.3 Structure and Texture Examples ........................ 49
  4.4 Ambiguous Examples .................................... 53
  4.5 Limitations ............................................. 57

5 Conclusions and Future Work ......................... 61
  5.1 Future Work ............................................ 62
    5.1.1 User Interaction .................................... 62
    5.1.2 Texture ............................................. 62
    5.1.3 Region Segmentation ............................... 63

Bibliography ................................................ 65
# List of Figures

1.1 Reconstruction example ............................................. 2
1.2 Ambiguous cases .................................................... 2

2.1 Example from “Image Inpainting” ................................. 9
2.2 Example from “Fast Digital Image Inpainting” .................. 10
2.3 Example from “Fragment-Based Image Completion” ............ 12
2.4 Limitation image from “Fragment-Based Image Completion” ... 12
2.5 Example from “Simultaneous Structure and Texture Image Inpainting” 13
2.6 Example from “Object Removal by Exemplar-Based Inpainting” ... 13
2.7 Example from “Image Repairing” ................................... 14

3.1 User Interaction Process ............................................. 20
3.2 Demonstration image ................................................. 21
3.3 Example scratch detection ........................................... 22
3.4 Regions used in edge detection ..................................... 23
3.5 Example edge detection .............................................. 24
3.6 Curve fitting ............................................................ 25
3.7 Example curves fit to a sample image .............................. 26
3.8 Initial structure completion ......................................... 28
LIST OF FIGURES

3.9  User motion example ............................................. 29
3.10 Neighboring colors example ...................................... 30
3.11 Final structure completion ....................................... 32
3.12 Example of iterated texture synthesis ......................... 35
3.13 Search example .................................................. 36
3.14 Initial texture segmentation ..................................... 37
3.15 Over-constrained texture segmentation ....................... 37
3.16 Final product of the algorithm .................................. 38

4.1  Simple structure example ........................................ 41
4.2  Curved structure example ....................................... 42
4.3  Example with no structure detected ............................. 43
4.4  Spurious structure example ..................................... 44
4.5  Fine grained texture example ................................... 46
4.6  Coarse grained texture example ................................. 47
4.7  Smooth texture example ......................................... 48
4.8  Structure and texture example — people ...................... 50
4.9  Structure and texture example — bird .......................... 51
4.10 Similar textures with different colors example ............... 52
4.11  Crossing lines example ......................................... 53
4.12 @ symbol example ................................................ 55
4.13 Real world ambiguous case ...................................... 56
4.14 Exact structure limitation ....................................... 58
4.15 Large texel size limitation ...................................... 59
4.16 Insufficient valid data limitation ............................... 60
Chapter 1

Introduction

Removing unwanted scratches or objects from an image in an undetectable manner is a technique that has been researched for its many useful and varied applications. Its uses include removing scratches, defects, superimposed text, and even entire objects from a scene.

Currently there are many algorithms that propose filling in the unwanted regions in an image. Figure 1.1 shows an example of the removal of scratches in an image using a technique developed by Bertalmio et al. [4].

The reconstruction process in Figure 1.1 sufficiently filled in the scratches in the image by propagating the structure of the image in order to preserve edges. Unlike the image in Figure 1.1(a), some images contain damaged regions that can be reconstructed in more than one viable manner. As a result, there is not just one and only one correct way to reconstruct the image. For example, Figure 1.2 presents two images with structure that could be completed in more than one effective way.

The white square in Figure 1.2(a) could be filled in with either the dark line...
Figure 1.1: Reconstruction example

crossing over the light line or the light line crossing over the dark line. The light square in Figure 1.2(b) could be completed in several different ways. For example a circle could be completed around the ‘a’ or the image could be completed to look like the @ symbol.

All of the algorithms that attempt to fill in images like the ones presented in Figures 1.1 and 1.2 can be categorized into one of three different types. One variety of algorithms attempts to fill in using partial differential equations (PDEs) and/or diffusion methods [2, 3, 4, 21, 22, 6]. In general, these inpainting techniques attempt to

Figure 1.2: Ambiguous cases
fill in a damaged region by propagating the missing structure and intensities into the
target region via diffusion. Some of these algorithms are derived from PDEs similar to
those modeling physical heat flow or fluid dynamics. These methods usually work well
for removing small, thin scratches or areas in an image, but they tend to introduce
some blurring for large areas or textured regions.

A second variety of algorithms attempts to fill holes in an image using texture
synthesis methods and/or correspondence maps [9, 12, 25, 1, 7, 24, 13]. This variety
performs nicely for areas that do not require any structure to be filled in and typically
do not preserve edges. Often times these algorithms can be very computationally
expensive as well.

A final variety of algorithms attempts to fill in both structure and texture [8, 17,
5, 26]. Most of these methods produce convincing images with large objects removed
from the original. However, the large majority of these algorithms are computa-
tionally expensive and do not offer any feedback to the user until the algorithm is
completely finished.

These algorithms, along with the other two previously mentioned types, have
no notion of what the user wants the region to look like. Users may not like the
results these algorithms produce in cases where there exist more than one valid and
convincing way to fill in the damaged region, such as in Figure 1.2(a). Some previous
methods are able to complete one line or the other, but none of the current algorithms
allow the user to influence which case he/she prefers, other than through broad global
preferences. The user never actually directly influences the structure completion
process.

By creating an algorithm that leverages user motion to fill in damaged regions in
an image, cases such as the one displayed in Figure 1.2(a) can be completed more accurately according to a user’s preferences. This ability requires methods to run at interactive speeds. User motion can easily fit into a framework similar to the one that Jia and Tang created in their image synthesis method [17]. Jia and Tang segmented the image to identify the underlying structure, then automatically completed the structure into the missing parts. A texture synthesis algorithm then completes the process by filling in the pixel intensities. Like Jia and Tang, the interactive algorithm proposed here first segments the image to detect the underlying structure, but only in the region surrounding the defect and under the cursor. Unlike Jia and Tang, user motion can then influence the completion of the structure into the damaged region. A texture synthesis algorithm is executed, taking into consideration the location of the previously detected structure.

User motion can be used to more appropriately fill in damaged regions in an image according to a user’s perception or preference. User input influences the filling-in process in four main ways:

1. The user influences which region to remove and what surrounding data to consider in the filling-in process.

2. The direction the user strokes the cursor guides the completion of the underlying structure in the damaged region. This ability provides a credible solution to damaged regions where the structure could be arranged in more than one legitimate manner.

3. The user motion determines a priority of which edges to consider for places where valid pixel intensities cannot be found in relation to the detected edges.
4. The user decides when the filled-in region is satisfactory by viewing continuous new results produced by the algorithm and directing the algorithm to stop. This ability creates opportunities for the user to pick the combination of structure and texture that best fits the surrounding data, instead of just one mathematically driven solution to filling in the damaged region.

1.1 Overview of Thesis

The remainder of the thesis is organized as follow. Chapter 2 presents background research, including algorithms that complete just structure, algorithms that complete just texture, and algorithms that attempt to complete both. Chapter 3 covers the details of a five-step filling-in algorithm and the effect each step has on a simple example. Chapter 4 presents a variety of results using a basic implementation of the filling-in algorithm. Chapter 5 suggests possible ways to extend these ideas in future work.
Chapter 2

Previous Work

This chapter reviews details of related work. Section 2.1 reviews those techniques that fill in damaged regions by propagating structure only. Section 2.2 discusses texture synthesis methods. Section 2.3 discusses the details of those algorithms that attempt to complete both structure and texture.

2.1 Structure Completion Methods

The completion of structure is a necessary part in edge preservation in the filling-in process. In general, structure completion methods attempt to fill in a damaged region by propagating the missing structure and intensities into the target region using partial differential equations and/or complicated diffusion methods. Bertalmio et al. [4] attempt to propagate structure in their technique called Image Inpainting by trying to replicate the basic techniques used by professional restorators. After a user selects various defective regions, the algorithm automatically completes the regions by using the surrounding information. The filling-in process is achieved by smoothly propagating information from the surrounding areas in the direction along each isophote. Isophotes are simply lines of equal gray values. This technique fills
 CHAPTER 2. PREVIOUS WORK

in the damaged regions by using a series of PDE’s to extend the correct isophotes. Isophote directions are computed at each pixel by calculating a discretized gradient vector and by rotating the resulting vector by 90 degrees. A 2-D Laplacian is then used to locally estimate the variation in intensity smoothness and that variation is propagated in the isophote direction. After each step of the inpainting process, anisotropic diffusion is used to smooth the inpainted region while simultaneously attempting to preserve boundaries [4]. Figure 2.1 demonstrates one example of the results of this technique by removing the superimposed text in the image.

This algorithm works well for the removal of small, thin scratches or areas in an image, but it tends to introduce some blurring for large areas or textured regions. This algorithm typically requires several minutes on personal computers for inpainting relatively small areas. Such a time is less than ideal for interactive applications.

This limitation motivated Oliveira et al. [21] to design a simpler and faster algorithm capable of producing similar results in just a few seconds. This algorithm is based on an isotropic diffusion model extended with the restriction of user-provided diffusion barriers. A diffusion barrier is a two-pixel wide line segment that stops the smoothing process from mixing information from both sides of the line [21]. Figure 2.2 displays the results of the algorithm successfully removing a scratch from an image. The process took about half a second [21].

This algorithm is computationally faster but still struggles in completing damaged texture regions. This technique requires user input, but only aiding in the location and preservation of edges. The user never actually influences the direction of structure completion directly.
Since 1699, when French explorers landed at the great bend of the Mississippi River and celebrated the first Mardi Gras in North America, New Orleans has brewed a fascinating melange of cultures. It was French, then Spanish, then French again, then sold to the United States. Through all these years, and even into the 1900s, others arrived from everywhere: Acadians (Cajuns), Africans, indige-

![Figure 2.1: Example from “Image Inpainting”](image1.png)

**(a) Before**

(b) After
2.2 Texture Synthesis Methods

Texture synthesis methods are useful in the completion process if the image is to be filled in in an undetectable manner. Of particular interest is an algorithm designed by Efros and Leung called “Texture Synthesis by Non-parametric Sampling” [13]. This technique grows a new image outward from an initial seed, one pixel at a time. A Markov random field model is assumed, and the conditional distribution of a pixel given all its neighbors synthesized so far is estimated by querying the sample image and finding all similar neighborhoods. Although the method aims at preserving as much local structure as possible, it cannot guarantee that local structure is preserved, especially when the search space is not exceptionally large. The actual method is quite simple, though, and works well for a variety of textures. Also, this texture synthesis is useful if you need one that works on a pixel-by-pixel basis versus a patch-based method.

Efros and Freeman later proposed a patch-based texture synthesis method to try
to decrease the time complexity involved with image completion [12]. Their method, called "Image Quilting", stitches patches made of several pixels together. This technique should be faster because most pixels have their values already determined by what has been filled in so far. Cohen et al. [7] continued work in patch based texture synthesis by using Wang Tiles to create textures. Wang Tiles are squares in which each edge is assigned a color. A valid tiling requires all shared edges between tiles to contain matching colors. A stochastic tiling algorithm is used to nonperiodically tile the plane with Wang Tiles [7]. Wei and Levoy try to decrease time complexity by using a tree-structured vector quantization to accelerate the synthesis process. Their method is derived from Markov Random Field texture models and generates textures through a deterministic searching process [24]. All of these methods are useful and successful in replicating texture. However, none of these methods are able to successfully preserve structure as well.

2.3 Structure and Texture Completion Methods

Several algorithms have recognized the importance of completing both structure and texture in the filling-in process. Dior et al. [11] introduced a method that removes large objects from an image by filling in using a composition of fragments under combinations of spatial transformations. To attempt to solve some ambiguous cases they modified their algorithm to allow a user to specify a point of interest or a direction to bias their search in finding the composition of fragments. However even with the added information provided by the user, this algorithm still struggles to complete images that contain ambiguous structure. Figure 2.3 demonstrates one example of the successful completion of an image, and Figure 2.4 displays a limitation of the algorithm. Their algorithm also is not the most ideal for interactive sessions
CHAPTER 2. PREVIOUS WORK

Figure 2.3: Example from “Fragment-Based Image Completion”

Figure 2.4: Limitation image from “Fragment-Based Image Completion”

because of the time and space complexity.

Bertalmio et al. [5] continued their work in Digital Image Inpainting by developing an algorithm that could fill in with texture as well as structure. The algorithm first segments the image to find those parts that should be inpainted and those parts that should be completed with texture. The image is then reconstructed by adding in the two separate parts together. Figure 2.5 displays the results of their algorithm. Decomposing and reconstructing an image is a time consuming process, especially when dealing with color images and would therefore not be ideal for interactive applications. Furthermore, by filling in structure and texture independently, the filled-in structure does not constrain or match the texture.

Criminisi et al. [8] employ an exemplar-based texture synthesis technique modu-
2.3. STRUCTURE AND TEXTURE COMPLETION METHODS

Figure 2.5: Example from “Simultaneous Structure and Texture Image Inpainting”

Figure 2.6: Example from “Object Removal by Exemplar-Based Inpainting”

lated by a unified scheme for determining the fill order of the target region. Pixels maintain a confidence value, which together with image isophotes, influence their fill priority [8]. Figure 2.6 demonstrates some results of this algorithm.

Zang et al. [26] remove large objects from an image by propagating reasonable texture information into the image in three main steps. First, a spatial-range model is determined to establish the searching order of the target patch. Second, a source patch is selected by measuring the adjusted appearance of the source patch with the
target patch and enforcing the searching area in the neighborhood around the previous source patch. Third, a graphcut patch updating algorithm is designed to ensure the non-blurring updating [26]. This algorithm, like most of the before mentioned algorithms, is computationally expensive because of the searching that is required to complete large textured areas. The algorithm also does not take into account user preferences when reconstructing missing texture.

Jia and Tang [17] presented a new algorithm that could automatically repair damaged images that included large holes where missing details can be complex and inhomogeneous. To complete an image, the image is first segmented and curves are fit to the edges detected in the surrounding valid data. Curves are then connected in the damaged or defective area. ND tensor voting is used to infer missing curves and pixel values [17]. Figure 2.7 demonstrates results of this algorithm.

Although this algorithm also does not account for user preferences, user motion could easily influence the process of combining curves together in the damaged region. This set-up could easily provide the right amount of user influence in the filling-in process so that the user could easily fill in without unknowingly adding too much human error into the completion process.
2.3. *STRUCTURE AND TEXTURE COMPLETION METHODS*

Many of the previously proposed filling-in algorithms have been quite successful in removing large objects and/or defects from an image. However, the majority if not all of these algorithms struggle to complete cases containing ambiguous structure, like those shown in Figure 1.2. By building a framework that incorporates user preferences in the filling-in process, a user can successfully remove objects or defects from an image, even for cases that include ambiguous structure.
Chapter 3

Methods

The process of removing objects and scratches from an image in an undetectable manner is a compilation of smaller image processing tasks. Performing each task at the right time in a sufficient manner builds a framework that allows the user to influence the filling-in process.

The interactive filling-in algorithm contains the following five basic steps:

1. Detect the scratch or deformity.

2. Detect the edges outside of the deformity that lie under the cursor.

3. Fit curves to the detected edges.

4. Complete the structure missing from the damaged region.

5. Fill in the damaged region by applying a texture synthesis algorithm constrained by the previously completed curves.

Each of these tasks, except structure completion, can easily be implemented by building on well known and previously established image processing algorithms. Scratch
detection, structure completion, and texture synthesis can be influenced or guided by user input when given. To understand how each step works and how the steps interact, it is first important to understand how the user interacts with the user interface.

3.1 User Interaction

User interaction is an integral part of the filling-in process. User input aids the filling-in process in four main ways:

1. User input aids in identifying which region to remove and what valid data to consider to complete the damaged region.

2. User input influences the completion of the underlying structure in the defective region.

3. User input determines a priority of which edges to consider for places where valid pixel intensities cannot be found in relation to the detected edges.

4. User input determines when the filling-in process is done.

Section 3.1.1 describes the specific user interaction that achieves the identification and separation of the defective region from the valid data. Section 3.1.2 explains the user interaction that provides a direction used for structure completion and to search for edges to use in over-constrained searches. Section 3.1.3 depicts the user interaction that completes the filling-in process. Figure 3.1 illustrates the steps of how a user interacts with the user interface to aid in the filling-in process.

3.1.1 Identifying Regions

The user is able to help in identifying the region to remove by right-clicking the mouse in the damaged region. The user then drags the cursor with the right mouse
3.2. **SCRATCH DETECTION**

button down to identify other regions to consider. Every region the user drags the
cursor over will be considered and determined as either a part of the deformity or a
part of the valid data. Figures 3.1(a) and (b) show the cursor before and after the
user identifies the damaged region and the surrounding valid data by dragging the
cursor vertically along the black line.

3.1.2 **Structure Completion and Over-Constrained Searches**

The user can aid in the structure completion process and the over-constrained
searches by clicking the left mouse button and dragging the mouse in the direction
of desired structure completion. Figures 3.1(c) and (d) demonstrate the before and
after images of a user providing a direction for structure completion. The cursor is
dragged horizontally to indicate that the edges between stripes should be connected.

3.1.3 **Filling-in Finalization**

The user is able to decide when the filling-in process is complete by right-clicking
the mouse to terminate the process. Figure 3.1(e) shows the final image produced
from the filling-in algorithm after the user clicked the right mouse button.

3.2 **Scratch Detection**

The first step in the filling-in process is to identify the damaged region that needs
to be filled in. The majority of existing algorithms identify the damaged region by
having the user create a mask for the image by categorizing each pixel as either inside
or outside the damaged region. For this algorithm, that method is too tedious and
time consuming. A faster approach that requires less input from the user is much
more appropriate for an interactive algorithm. Instead of requiring the user to mark
every pixel included in the scratch, a user can supply two things: a seed pixel found
somewhere in the middle of a scratch and an approximate intensity deviation for how
Figure 3.1: User Interaction Process. First, the user right-clicks the mouse in the defective region and drags the cursor across the regions to consider in identifying valid data (a, b). Second, if necessary, the user left-clicks the mouse and drags the cursor in a direction of desired structure completion (c, d). In this case, the user is indicating the completion of the flag’s stripes through the horizontal motion of the cursor. Finally, the user right-clicks the cursor to indicate the completion of the filling-in process (e). The green arrows represent the direction of the cursor motion.
much the pixel intensities vary within the defect. The user easily supplies a seed pixel by right-clicking the mouse when the center of the cursor is located in the damaged region. An approximate intensity deviation is provided by the user by adjusting a slider for intensity sensitivity. At this point, the algorithm automatically detects an approximation for the damaged region by simply growing the region outward from the seed pixel. Any neighboring pixels that differ more than the given threshold are not considered a part of the scratch.

Consider the image in Figure 3.2(a). A user places the middle of the cursor somewhere in the region of the white square and clicks the right mouse button. The image in Figure 3.2(b) shows the magnified region of both the scratch and the valid data found under the cursor in the image in Figure 3.2(a). The red region in Figure 3.3 demonstrates the region that the algorithm detects as the scratch.

3.3 Edge Detection

The next step in the filling-in process is to detect the underlying structure in the region surrounding the damaged portion. Areas that are included under the cursor that are not detected as a part of the defect are considered to be valid data.
Figure 3.3: Example scratch detection. The red region shows the region that the algorithm detects as the defective region.

To successfully detect the structure, each pixel in the valid data is considered to determine whether or not it is an edge point. To achieve this classification, anisotropic diffusion is first used to smooth out noise while preserving strong edges. Then, the average intensity for each color channel is computed in four separate three by three regions surrounding the pixel (Figure 3.4). This region size is used to prevent edges from becoming too thick while still considering the surrounding texture. If a larger region size is used, a sharp edge is included in more regions. As a result, the edge affects the average intensity of more pixels, and edges are represented by thicker edges. If a smaller region is used, fine texture does not affect the segmentation process. For example, Sobel or Prewitt kernels do not account for texture because they consider only neighboring pixels.

An average combined color intensity is also calculated for each of the four regions. This average measure of color is computed with the following equations [20]:

\[ I_1 = (R + G + B)/3 \]  
(3.1)  
\[ I_2 = R - B \]  
(3.2)  
\[ I_3 = (2G - R - B)/2 \]  
(3.3)
Using these values, the following equation computes a color value:

$$\text{color} = I_1 + 0.5 \times I_2 + 0.25 \times I_3$$

(3.4)

Once the average intensities are computed across the four neighboring regions, the differences between the regions are computed. Absolute differences are calculated horizontally, vertically, and diagonally. Region one is compared to region three and region two is compared to region four when considering the difference horizontally. To calculate the vertical differences, region one is compared to region two and region three is compared to region four. Region one is compared to region four and region three is compared to region two to determine diagonal differences. The maximum difference computed for each pixel is used to decide a gradient threshold. Maximum differences above that threshold are considered edge pixels. Figure 3.5 demonstrates the edges detected in the example image shown in Figure 3.2. After the edge pixels are detected in the valid data, groups of pixels are divided to represent separate edges.
CHAPTER 3. METHODS

Figure 3.5: Example edge detection

This process is achieved by grouping together neighboring pixels.

3.4 Curve Fitting

After edge detection, each edge is represented as a group of pixels. Figure 3.6(a) shows points that could represent an edge. To accurately complete structure, the location of pixels relative to edges must be known during the texture synthesis process. There is a variety of methods to achieve this, but for this algorithm a computationally fast method is needed. Low-degree rotated polynomials can be represented by a simple equation so that it is easy to determine on which side of a curve a point or pixel lies. This representation is computationally fast enough to work at interactive speeds. This capability provides a means for determining the orientation of damaged pixels relative to the underlying structure detected. By using low-degree rotated polynomials, rounded curves as well as straight lines can be used to represent the underlying structure of the region.

Edges, like that shown in Figure 3.6(a), can be represented by a rotated polynomial by first computing the line of orientation of the points. The line of orientation can be computed by calculating a direction of elongation using the equation

$$\theta = \frac{1}{2} \tan^{-1} \frac{2\mu_{11}}{\mu_{20} - \mu_{02}}$$

where $\mu_{ij}$ is the $ij$th statistical central moment. Each point is then translated so that the mean of all the points is at the origin. Figure 3.6(b) shows the
Figure 3.6: Curve fitting. Each red dot represents the center of a pixel where an edge point was detected. The blue line displays the line of orientation for the points. The green line demonstrates the curve computed to approximate the edge points.
**Figure 3.7:** Example curves fit to a sample image. The green lines demonstrate the eight curves computed to approximate the underlying structure.

points found in Figure 3.6(a) after they are translated. After translating the points, they are then rotated so that the line of orientation aligns with the $x$-axis as shown in Figure 3.6(c). A least squares fitting of the points is used to approximate the points by a rotated polynomial, which produces a curve similar to the green curve in Figure 3.6(d). All curves extend to the boundary of the search area. This means that they could extrapolate beyond the fitted edge points. Figure 3.7 shows an example of the result of fitting curves to edge points.

### 3.5 Structure Completion

Some of the detected structure could be obstructed or interrupted by the damaged region. As a result, some of the previously computed curves should be combined to form the correct structure across the damaged region. For example, consider the eight curves in Figure 3.7, which are terminated by the square in the middle. For this case, the square in the middle is the region to remove, so those eight edges are the only ones to be considered in the structure completion process. Without the square there, it is apparent that the top two edges should connect respectively to the bottom two while the two edges on the left should connect respectively to the two edges on the right.
3.5. *STRUCTURE COMPLETION*

A combination of user input and the mathematics of the fitted curves can successfully combine the computed eight edges into four so that the underlying structure extends across the damaged region successfully. Figure 3.8 demonstrates what the structure looks like after all of the curves have been combined successfully.

More specifically, there are two main steps for connecting the underlying structure. The first step is a search that is executed only if user input is given. If user input is not given, the algorithm continues straight to the second step. The second step executes a similar search to attempt to match up any curves that have not been matched up yet. By using a variety of different mathematical criteria, the best match for a function should be chosen as described in the following subsections.

**3.5.1 First Search**

This first search uses the following three metrics to determine whether curves should be combined:

1. mean squared error (MSE),

2. angular difference between user motion and the direction between endpoints (diff$_\theta$), and

3. difference between colors adjacent to the edges (diff$_c$).

These values are discussed in Sections 3.5.1.1–3.5.1.3, respectively. The first search completes structure by minimizing the total summation of the equation

$$\text{total} = w_1 \cdot \text{diff}_\theta + w_2 \cdot \text{diff}_c$$  \hspace{1cm} (3.5)

where MSE < $T_1$ and diff$_\theta$ < $T_2$. The values $w_1$ and $w_2$ are constant weights. In our experience, the values work well when they are within the ranges 0.95 < $w_1$ < 1.05 and
Figure 3.8: Initial structure completion. The completed structure is based on a vertical user gesture

0.00015 < \( w_2 \) < 0.00025. The values \( T_1 \) and \( T_2 \) are constant thresholds that specify maximum values allowed for a specific criterion. Typically, the ranges \( 10 < T_1 < 20 \) and \( 0.5 < T_2 < 0.7 \) eliminate those curves that would erroneously be connected because they do not represent the data close enough or the direction between the endpoints does not closely match the user direction. Figure 3.8 shows the result after the first search is implemented on the image shown in Figure 3.2 when the user strokes the cursor in a vertical direction.

3.5.1.1 Mean Squared Error

Mean squared error is used in the first search to eliminate any curves that do not represent the edge data closely enough. A curve is created to test the outcome of combining two curves. If this trial representation does not represent the data well enough, it is discarded to prevent growing erroneous data in the subsequent texture synthesis step of the algorithm.

3.5.1.2 Angular Difference

To understand how to compute the angular difference between user motion and the direction between endpoints, it is first important to consider the user input. This first search executes only if the user gestures the cursor in the direction he or she
3.5. *STRUCTURE COMPLETION*

![Image of user motion example](image)

**Figure 3.9:** User motion example. The user connects the edges by stroking the cursor in the direction of desired structure completion. The red area represents the defective region, and the black lines represent the edges. The blue points represent the endpoints of the curves.

would most prefer to complete the structure in. Consider the example in Figure 3.9.

If a user wanted to connect curve A with curve C, he/she would stroke the cursor in a vertical direction. Conversely, if the user wanted to connect curve B with curve C, he/she would stroke the cursor in a direction that is at a 45 degree angle to the x-axis.

The second criterion used in the search draws from this user motion. In considering if two curves should be connected, the direction between the two endpoints of the curves is calculated and compared to the direction provided by the user’s cursor gesture. The angular difference between the user motion and the direction between endpoints is calculated with the equation

\[
\text{diff}_\theta = |\theta_u - \theta_c|
\]  

(3.6)

keeping in mind the modular nature of angles. In this equation, \(\theta_u\) represents the direction of user motion. It is computed by tracking the motion of the cursor and computing the direction of elongation of the detected points using the equations.
Figure 3.10: Neighboring colors example. The red area represents the defective region while the black lines represent the edges.

described in Section 3.4. The value $\theta_e$ represents the direction between the endpoints terminated by the defective region of the curves. This value is calculated by the equation

$$
\theta_e = \tan^{-1} \frac{dy}{dx} 
$$

(3.7)

where $dx$ and $dy$ are the $x$ and $y$ components of the normalized vector between the two endpoints.

3.5.1.3 Difference Between Neighboring Colors

The difference between neighboring colors is the third value considered when connecting the various curves. The closer the neighboring color values are between two curves, the more probable that the two curves should be combined. Figure 3.10 conveys this principle more fully. By examining the neighboring colors, it is obvious which edges should be combined. For example, edge A has white above it and blue below it. It should connect to the only other edge with white above it and blue below it, edge B.

To compute the difference between neighboring colors, two pixel values for each curve are found, one to represent each side of the curve. These pixels are located by
calculating a small distance from the mean of the curve in both directions orthogonal to the direction of elongation of the curve. A sample curve is created that represents the structure if the two curves are combined. The four pixels’ topological relationship to the sample curve is determined. The two pixels on the same side of the curve are compared, and the two pixels on the other side of the curve are compared. The difference between the two pixels is found by adding the absolute value of the differences in each color plane between the two pixels. The value \( \text{diff}_c \) is calculated by adding these two differences.

3.5.2 Second Search

The second search uses the following four metrics to determine whether curves should be combined:

1. mean squared error (MSE),

2. difference between adjacent colors to the edges (\( \text{diff}_c \)),

3. Euclidean distance between the endpoints of the curves (\( \text{dist} \)), and

4. difference between curve orientations (\( \text{diff}_d \)).

These values are discussed in Sections 3.5.1.1, 3.5.1.3, 3.5.2.1, and 3.5.2.2, respectively. The second search completes structure by minimizing the total summation of the equation

\[
\text{total} = w_3 * \text{MSE} + w_4 * \text{diff}_d + w_5 * \text{diff}_c + w_6 * \text{dist}
\]  

(3.8)

where \( \text{MSE} < T_1 \). The values \( w_3, w_4, w_5, \) and \( w_6 \) are constant weights that can be tuned to weight the relative importance of the mathematical values. In our experience, the ranges \( 0.95 < w_3 < 1.05; 0.6 < w_4 < 0.7; 0.001 < w_5 < 0.002; \) and \( 0.1 < w_6 < 0.2 \)
typically work well for these weights. The value $T_1$ is the same rejection threshold discussed in Section 3.5.1.

3.5.2.1 Comparison of Curve Orientation

The orientation of each curve is represented by the direction of elongation of each curve. The direction of elongation is calculated using the equations described in Section 3.4. The value $\text{diff}_d$ is computed by taking the absolute difference of the directions of elongation of the curves, keeping in mind the modular nature of angles.

3.5.2.2 Distance Between Endpoints

The mathematical value $\text{dist}$ is the Euclidean distance between the endpoints of curves. The closer two endpoints are, the more likely that the two curves should be connected.

3.6 Texture Synthesis

Once the curves have been connected, the R,G,B intensities are then ready to be computed for each pixel in the damaged region. To achieve this task, a texture synthesis method is used that can be constrained by the previously computed curves. An example of a texture synthesis method that can accomplish this is the method derived by Efros and Leung called “Texture Synthesis by Non-parametric Sampling” [13]. In their method, texture is propagated by estimating a conditional distribution of a pixel given all its neighbors. This conditional distribution is computed by querying
the sample image and finding all similar neighborhoods. In our method a similar conditional distribution is calculated using the neighboring data, but our method differs from Efros and Leung’s method in three important ways:

1. Our method iterates more than once when determining which pixel to fill in and uses previous pixels filled in from the previous iteration in calculating the conditional distribution.

2. Besides just referencing the data in the search, a smoothness prior is also used.

3. Instead of searching through the entire image to build the conditional distribution, only the regions that are considered valid data are used to influence the distribution.

3.6.1 Numerous Iterations

Unlike Efros and Leung’s method of texture synthesis, this algorithm runs continuously across the data. Therefore, after the first iteration of filling in the pixel intensities, the algorithm runs again. By executing the search more than once, the user is able to decide what results he/she prefers the most. Iterating more than once also allows for a more appropriate pixel intensity to be chosen the second or third time through the search than what was chosen on the first iteration. The first pass through only considers those neighbors that are valid data. If all of a pixel’s neighbors are included in the damaged region, then a pixel is chosen based on the frequency of the pixel intensities found in the eligible valid data. For all successive iterations, all neighboring pixel intensities computed during the previous iteration are used to build a pixel’s conditional distribution. Each iteration builds on the results from the previous iteration. As a result, the conditional distribution does not have to be based
solely on a comparison of neighboring data. Another advantage to completing the search more than once is that a smoothness prior can be calculated and factored into the conditional distribution using results provided from the previous iteration.

3.6.2 Smoothness Prior

On the first iteration through the texture synthesis, the conditional distribution for each pixel is built using only the neighboring data. The second time through, the neighboring data and a smoothness prior is used in the search. The smoothness prior is calculated by taking an average intensity across the neighborhood of a pixel. This average is computed from the values determined by the previous pass of the texture synthesis algorithm. As the texture synthesis algorithm iterates, more weight is put on the prior and less on the data until the weights are even. When user input is provided, the weights reset so that for the first pass through, all of the conditional distributions depend on the data again. A smoothness prior helps to eliminate noise and builds structure by allowing pixels to influence their neighbors’ intensities into becoming similar to their own intensities. Figure 3.12 shows the results of the first few iterations of the texture synthesis algorithm.

3.6.3 Constrained Search

When building a conditional distribution for a particular pixel, it is important to consider only pixels that have the same topological relationship to the underlying structure as the pixel to be filled in. By enforcing this restriction, the structure is guaranteed to be preserved as long as the curves approximate the actual structure closely enough. For example, consider a pixel in the region with the question mark in Figure 3.13. A pixel in this region is to the left of both of the vertical lines and occurs above the two horizontal lines. As a result only those pixels that occur to the
Figure 3.12: Example of iterated texture synthesis. These images show the results of the first few iterations of the texture synthesis algorithm.
Figure 3.13: Search example. The region labeled by the question mark should only consider the pixels in the region labeled A. These pixels contain the same topological relationship to the edges as the pixels in the region labeled by the question mark.

left of the vertical lines and above the horizontal lines should be considered in the conditional probability. Those pixels are the pixels in the gray region labeled A.

Once the image is searched and all topologically similar regions are found for each pixel, each pixel is filled in as is displayed in Figure 3.14. Notice that not all of the regions in the image are correctly filled in yet. Over-constrained parts of the search prevent some of these regions from correctly being filled in.

3.6.4 Over-Constrained Search

In some cases, a defective pixel search might not find any valid pixels that have the same topological relationship to the detected edges. An example of one such pixel would be the center pixel of the square in Figure 3.14 if the structure is completed as it is in Figure 3.11. This pixel is in the middle of all four lines, but none of the valid data is. To solve this predicament, some edges are ignored when performing a search to find valid pixel intensities for this pixel. To determine which edges to
3.6. **TEXTURE SYNTHESIS**

![Image](image_url)

**Figure 3.14:** Initial texture segmentation. Not all pixel values have been filled in. The red line in the image outlines the defective region.

![Image](image_url)

**Figure 3.15:** Over-constrained texture segmentation. The pixel intensities are computed for any regions that are over-constrained in the search in the texture synthesis algorithm. The red line in the image outlines the defective region.
consider and which to ignore, user motion is considered. If user motion is provided, the two edges that are closest to the pixel in the direction perpendicular to the user motion are found. These two edges are then used in comparison to the valid data. All other edges are ignored. On the other hand, if the user motion is not provided, the algorithm selects two edges with the closest orientations to be considered in the search. As a result, all defective pixels will be filled in with valid pixel intensities. Figure 3.15 displays the results when a user strokes with the cursor vertically. The final image appears in Figure 3.16 without any superimposed structural curves or lines. Additional examples are presented in Chapter 4.
Chapter 4

Results and Limitations

The filling-in algorithm successfully removes defects and objects from a variety of images. Section 4.1 demonstrates examples where structure is successfully propagated into the defective regions being removed. Section 4.2 shows examples of the removal of defects from images only containing texture. Section 4.3 deals with the removal of defects from images that contain structure and texture. Examples of images which ambiguous structures have been resolved are given in Section 4.4. Limitations of the filling-in algorithm are discussed in Section 4.5. Throughout these sections numerous images are given displaying the results of the interactive filling-in algorithm. Degree five polynomials are used to represent the underlying structure because they are flexible in their representation, and they do not require too much time to compute. Unless otherwise noted, the intensity standard deviation, as discussed in Section 3.2, is 0 because the majority of defects removed are regions of constant black intensity. Any special interaction the user gave to create the appropriate output is noted. All edges are displayed in blue in the structure images.
4.1 Structure Examples

The interactive filling-in algorithm successfully detects and completes a variety of structure, including both curves and straight lines. Figure 4.1 shows a simple example of structure completion. In this example, the authentic scratch on the man’s shoulder is removed. A standard intensity deviation of 60 is used to detect the scratch. Notice the blue line in Figure 4.1(b) divides the edge between the man’s shoulder and the background. Figure 4.1(c) shows that this edge is preserved to correctly fill in the defective scratch.

The interactive filling-in algorithm can also preserve more complex structure. Figure 4.2 demonstrates an example where the algorithm completes curves. The black lines and squares are successfully removed from the image, specifically the lines that occluded the lower left corner of the red triangle. Figure 4.2(b) displays the curves that are completed in the defective region in the lower right corner of the red triangle. Notice some of these curves extrapolate beyond the valid edge point data, but they do not prevent a valid completion of the damaged portion.

In Figure 4.3, the white feathers in the middle of the duck are removed. The standard intensity deviation used is 20. Notice that no structure is detected in the structure image in Figure 4.3(b). The white portion is still successfully filled in with the appropriate intensities.

Figure 4.4 presents an example of the completion algorithm successfully filling in in spite of the existence of spurious edges. The algorithm effectively fills in the black square found in the butterfly’s orange wing, even though a spurious edge is produced.
4.1. STRUCTURE EXAMPLES

Figure 4.1: Simple structure example. The small scratch on the upper shoulder is removed. The intensity standard deviation used for scratch detection is 60. The algorithm correctly preserves the edge between the man’s shoulder and the background.
Figure 4.2: Curved structure example. In this example the black lines and squares are removed. The curve in the lower left hand corner of the triangle is correctly completed.
4.1. STRUCTURE EXAMPLES

Figure 4.3: Example with no structure detected. The white feathers in the middle of the duck are removed. The intensity standard deviation used for scratch detection is 20. No structure was actually detected, but the white feathers are still successfully filled in.
Figure 4.4: Spurious structure example. Four black squares in the image are successfully removed. Notice the spurious edge produced in the middle of the butterfly’s wing. This edge does not prevent the algorithm from successfully removing the black squares and filling in with convincing texture.
4.2 Texture Examples

The interactive filling-in algorithm completes texture as well as structure. Figure 4.5 demonstrates an example where the algorithm fills in texture over the black line and black square in the image. The algorithm is able to successfully complete fine grained texture.

The interactive filling-in algorithm can also complete coarse grained textures. Figure 4.6 demonstrates this ability by filling in the white squares and line in the berries image. Notice that both the red and black berries are successfully filled in in the image.

Smooth textures can also be completed by the interactive filling-in algorithm. Figure 4.7 shows the results of filling in the black oval and line in the image. Even though this texture is more blurred than the previous examples in Figures 4.5 and 4.6, the defective regions are still effectively removed.
Figure 4.5: Fine grained texture example. This example demonstrates the results of the filling-in algorithm on a fine grained texture.
4.2. TEXTURE EXAMPLES

Figure 4.6: Coarse grained texture example. This example demonstrates the results of the filling-in algorithm on a coarse grained texture. Image from Efros and Freeman “Image Quilting for Texture Synthesis and Transfer”.
Figure 4.7: Smooth texture example. This example demonstrates the results of the filling-in algorithm on a more blurred texture.
4.3 Structure and Texture Examples

The interactive filling-in algorithm successfully fills in damaged regions in images that contain both structure and texture. Figures 4.8, 4.9, and 4.10 demonstrate the algorithm's ability to complete both structure and texture into defective regions.

In the image in Figure 4.8(a), the two black squares and the black oval need to be removed. Notice that the algorithm correctly fills in the texture of the faces and the texture of the red shirt. The image in (b) displays the detection and completion of the edge between the red shirt and the girl’s neck.

Figure 4.9 is another example that shows that the algorithm can complete damaged regions of an image that require both texture and structure. Three black squares are removed from this image.

Figure 4.10 shows that the algorithm can correctly complete edges between grass and sky. Also, the algorithm can correctly fill in damaged regions that separate similar textures with different colors. The removal of the black line that occludes both the green and yellow grass demonstrates this ability to fill in similar textures with different colors.
Figure 4.8: Structure and texture example — people. The two black squares and the black oval are removed. One black square is located in the girl’s forehead while the other black square is located between the other girl’s chin and the background. The black oval occludes the edge between the red shirt and the girl’s neck. The algorithm successfully fills in both structure and texture.
Figure 4.9: Structure and texture example — bird. Three black squares are removed from the image. Structure and texture are filled in.
Figure 4.10: Similar textures with different colors example. In this example, the black lines and black square are removed. The algorithm successfully fills in the edges between grass and sky. Also, the edge between the green grass and the yellow grass is completed effectively. This example demonstrates that the algorithm can fill in edges that separate similar textures with different colors.
4.4. AMBIGUOUS EXAMPLES

Figure 4.11: Crossing lines example. These images present the case of removing the white square that occludes the crossing of the dark and light line (a, b). The white square is filled in based on the user’s motion (c, d).

4.4 Ambiguous Examples

Often times there are instances where structure could be completed in more than one legitimate manner. Figures 4.11 and 4.12 present two simple geometric cases of ambiguous structure. Figure 4.13 provides a real world image where ambiguous structure occurs.

Figure 4.11 demonstrates the results for an ambiguous case. In this case, the white square is the object to remove. If a user wants the light line to occlude the dark line, then the user strokes the cursor horizontally. However, if the user wants
the dark line to occlude the light line, the user strokes the cursor vertically. The different results are not based on different structure completions. In this example, the structure is completed in the same manner, regardless of whether the cursor is stroked horizontally or vertically. The results differ because the two vertical lines are used in the over-constrained search to complete the dark line, whereas the two horizontal lines are used in the over-constrained search to complete the light line.

Figure 4.12 demonstrates another ambiguous structure case. In this example the structure is completed two different ways depending on the user input. A circle, as seen in Figure 4.12(e), is completed when the user strokes vertically. Figure 4.12(d) shows the way the structure is completed in order to create a circle around the ‘a’. If the user instead strokes the cursor from the end of the ‘a’ to the upper black line, then an @ symbol is created. The structure formed from this motion is pictured in Figure 4.12(b). This structure produces the @ symbol pictured in Figure 4.12(c).

Figure 4.13 presents a real world image that contains ambiguous structure. In this example, the red square is removed. Structure and texture is completed based on the user motion in identifying which valid regions to search and which structure to complete. To complete the girl’s arm over the tie, the user moves the cursor along the arm when identifying which valid data to use to fill in. The user also stroke horizontally to complete the structure across the arm. Conversely, if the user wants to complete the tie over the arm, the user moves the cursor along the tie to identify the blue tie structure and texture as the valid data to search to fill in. In addition, the user strokes vertically to complete the tie structure.
Figure 4.12: @ symbol example. The gray square is removed. The structure can be completed to produce an @ symbol like the one pictured in (c) or the structure can be completed to produce a circle around the ‘a’ like the image pictured in (e). The user strokes vertically to get a circle around an ‘a’, and the user strokes from the end of the ‘a’ to the top black line to get the @ symbol.
Figure 4.13: Real world ambiguous case. This example shows a real world image that contains ambiguous structure. The image shown in (c) shows the arm completed over the tie, and the image shown in (e) shows the tie completed over the arm. Structure and texture are completed based on the user motion in identifying which valid regions to search and which structure to complete.
4.5 Limitations

This section presents some examples of limitations of the interactive filling-in algorithm. Structure can be difficult to detect and complete, especially when varying factors like complicated lighting or thin edges exist in an image. Occasionally spurious structure is detected, but this inefficacy typically does not prevent the complete process from successfully filling in the defect. Sometimes in the filling-in process, curved structure is produced when straight edges are needed instead, or vice versa. Figure 4.14 demonstrates this limitation in some of the color squares that were filled in with rounded corners.

Large texel sizes in texture are also hard for the algorithm to fill in. To achieve this, large search regions are needed, which require lots of computation time. Sometimes large textured regions of valid data are not even available in the image. Figure 4.15 presents a texture with a texel that is too large to complete in the damaged region.

Figure 4.16 shows an example where there does not exist sufficient valid data to fill in the damaged portion. In this example, the red square occluding the girl’s nose needs to be removed. Unfortunately, there is not enough structure in the valid data surrounding the nose to indicate how the damaged area should be filled in.
Figure 4.14: Exact structure limitation. In this example, the black squares are removed. Curved structure is completed where straight lines are needed to complete the color patches on the balloon.
Figure 4.15: Large texel size limitation. The texel of this texture is too large for the algorithm to be able to successfully complete the correct texture.
Figure 4.16: Insufficient valid data limitation. In this example, the damaged region to remove is the red square that occludes the girl’s nose. There is not enough structure in the surrounding valid data to correctly fill in the nose.
Chapter 5

Conclusions and Future Work

An algorithm framework has been created and discussed that allows a user to influence the filling-in process. Defects have successfully been removed from images that contain structure, images that contain texture, and images that contain both structure and texture. This framework allows the user the opportunity to affect the way in which an object or defect is removed, especially in cases where the missing structure could be filled in in more than one viable manner. The user influences the filling-in process in four main ways:

1. The user influences which region to remove and what surrounding data to consider in the filling-in process.

2. The direction the user strokes the cursor guides the completion of the underlying structure in the damaged region.

3. The user motion determines a priority of which edges to consider for places where valid pixel intensities cannot be found in relation to the detected edges.
4. The user decides when the filled-in region is satisfactory by viewing continuous new results produced by the algorithm and directing the algorithm to stop.

5.1 Future Work

This interactive filling-in algorithm could be extended in a variety of ways to add to or change its functionality.

5.1.1 User Interaction

Sometimes there is not enough structure or texture in the surrounding valid data to leverage off of in order to correctly fill in a damaged portion in an image. Figure 4.16 shows an example where the valid data surrounding the red square does not have enough structure information to complete the girl’s nose. The interactive filling-in algorithm could be extended to allow a user to draw in the missing structure since the data does not provide enough information to automate it. The intensities could be filled in by performing texture synthesis with the structure that was drawn in. If drawing the structure is too time consuming, a user could also provide seed points that indicate an approximate location of the missing structure, without having to draw every missing edge. As the structure provided by the user increases in complexity and curvature, other curves besides polynomials could be considered in representing the edge data.

5.1.2 Texture

Filling in texture can be a very time intensive process, especially for textures that have a large texel size. Figure 4.15 shows an example image that contains a texture with a large texel size. As increases in hardware speed continue, the interactive filling-in algorithm will be able to fill in with texture in shorter amounts of time. Larger search areas could be identified to allow texture with larger texel sizes to be
5.1. FUTURE WORK

filled in. Also, as hardware speed increases, a patch-based texture synthesis could be used to fill in texture for the removal of larger objects. Patch-based texture synthesis methods usually require some set-up time before replicating texture. As this set-up time decreases, using a patch-based method to synthesize texture could also help to fill in with textures that have a large texel size.

5.1.3 Region Segmentation

Many segmentation methods are also very time intensive. To ensure that the structure is more accurately detected, a region segmentation method can be used to separate the valid data into similar regions. Region segmentation should help eliminate spurious edges and would be less likely to let less-distinct edges go unnoticed. Dividing the regions, though, needs to be achieved at interactive speeds. More work needs to be done in increasing the speed of segmentation that accounts for texture and color regions.
Bibliography


surfaces in range images. XVI Brazilian Symposium on Computer Graphics and 

tization. International Conference on Computer Graphics and Interactive Tech-


[26] Y. Zhang, J. Xiao, and M. Shah. Region completion in a single image. Euro-