University of Kaiserslautern

Department of Electrical and Computer Engineering

Thesis

Dewarping Documents Using a Stereo System

submitted for the degree of Master of Science

by

Soner Özgün Pelvan

Supervisors:	Prof. Dr.	Thomas Breuel
	Dipl.-Inf.	Adrian Ulges

Kaiserslautern, 2007

Declaration

Acknowledgments

I would like to thank everybody that has helped me to build up a better knowledge on computer engineering. I would like to specially thank my family for their support and their belief in me. Other that that I want to thank *Prof. Norbert Wehn* for letting me work on hardware design and help me learning new technologies. I want to thank the whole *Fraunhofer ITWM Optimization Team* for making me a member of their group. At last I want to thank the whole *IUPR Team* especially to *Adrian Ulges* and *Daniel Keysers* for their support on making this thesis possible.

Motivation

The first step in every research is a motivation to achieve a task that is worth spending time for.

Printed material scanning to create electronic versions of printed material has become more important in the last decade as scanning methods become easier and cheaper. However naive approaches using a flatbed scanner in the case of a book do not always create expected results. This leads to a new research area. Using cameras to capture images of documents and the apply some post processing on these images to obtain a version as if taken from a flat document. Companies like "Google" already begin a research on this topic and have made quite a success with its "Google Book Search". However methods used by Google and other companies are still very complicated and expensive.

In this thesis we approach the same problem by using a stereo camera system. This stereo system is used to create a triangular mesh that approximates the book surface. Then we implement a $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ mapping that preserves the angles of triangles in $3D$ mesh in the $2D$ mesh. All these are done on software and can be used in any computer. The whole system can be a solution for home users that want to digitize their printed media.

Related Work

As we have mentioned in motivation part, In this thesis, we present a dewarping method to flatten document images. We use a $3D$ model to dewarp [1] the document images.

Kakumanu and N. Bourbakis [1] purposed an image-text dewarping methodology based on robust estimation of text-lines. To remove both these distortions and to produce a flattened view of the text, they use the cues present in the image-text, i.e., the text-lines on the surface of the page are straight. Kakumanu and N. Bourbakis [1] used single camera image. In case of book image, single camera image can miss text information near to spine of the book. The state-of-the-art OCR (optical character recognition) systems have a very low performance on such images.

Adrian Ulges, Christoph H. Lampert and Thomas M. Breuel [2] presented a new algorithm to dewarp document images.It requires only a single camera image as input. This algorithm relies on a priori layout information instead of additional hardware. In our approach to dewarping, we used $3D$ model of document image. We work with stereo image pair to make $3D$ model of document image. In this way, we gain high depth information of document surface. We do not miss any information even near to the spine of the book. The accurate $3D$ model produces nice dewarped document images, which are well suited for The state-of-the-art OCR (optical character recognition) systems.

[1]First part of this thesis, presented by Khawar Parvez

Abstract

In this thesis we address the problem of the de-warping document images using a conformal mapping. Instead of finding a direct mapping between the document image and the *flattened* image, we first devise a way to build a $3D$ model of the document and compute a mapping from $3D$ to $2D$.
We then apply this mapping between original document image and its corresponding *flattened* image to de-skew the original document image. Our experiments show that this method is not very robust to noise and works well under conditions like having a uniform point distribution with no or small noise margin. These problems proved to be very hard to tackle, however we have devised some methods to solve some of the problems to gain acceptable results.

Contents

List of Tables

List of Figures

1 Introduction

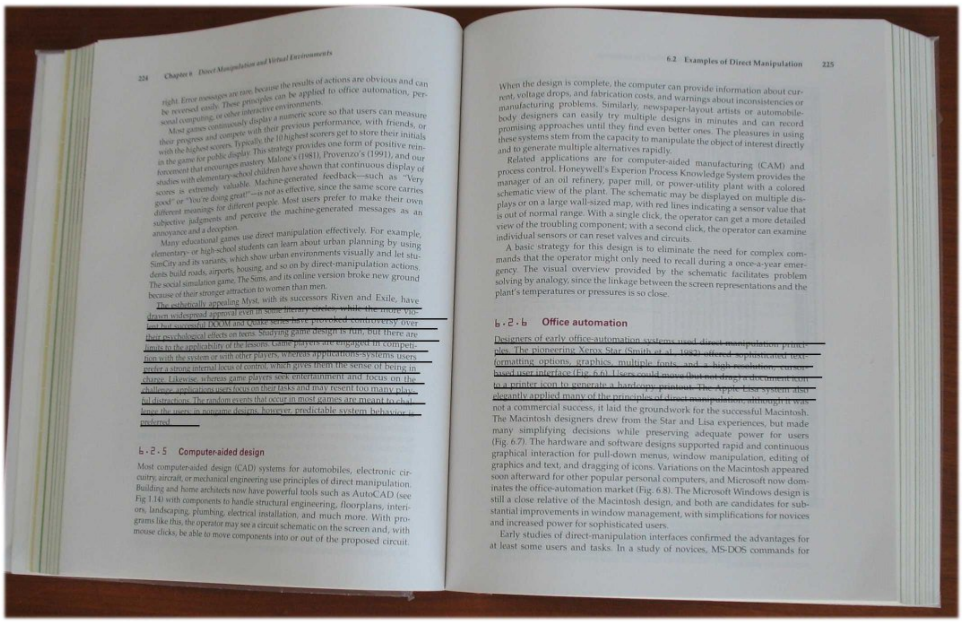
This thesis is the second part of collaborating work with Khawar Parvez on document de-warping using stereo vision. In this part we are going to talk about the procedure for de-warping the image using 3 dimensional data. The first part of the thesis,written by Khawar Parvez, covers the topic of 3 dimensional model reconstruction using stereo images.

Digitizing printed materials have become more and more popular in the last decade due to the increase in the quality of digitizing equipment used and also due to storage media becoming larger and larger. Traditionally flatbed scanners are used to digitize documents however as digital camera technology advanced and became more common in use, they evolved into another method to digitize printed materials. Unlike flatbed scanners cameras do not really guarantee images without skew (Look at Figure 1). This problem however can be solved by doing some post processing on the image. Correcting the distortions in an image is called de-warping.

Image correction can be easily achieved if the document in question is planar(Look at Figure 2). However in the case of curled papers and books, it is more complicated to correct these distortions. However it is possible to produce a $3D$ model using a stereo system. We can then use this $3D$ model to reconstruct an image without distortions. The problem in hand then be described as:

Problem:

Given a pair of images of a document taken by two cameras with different viewpoints, points in $3D$ can be reconstructed. Using these $3D$ points a triangulated mesh can be built to approximate the $3D$ document surface. From this approximation to the surface we can produce a restored image as it would appear in its planar format.(Look at Figure 3)



224 Chapter 6 Direct Manipulation and Virtual Environments

right. Error messages are rare, because the results of actions are obvious and can be reversed easily. These principles can be applied to office automation, personal computing, or other interactive environments.

Most games continuously display a numeric score so that users can measure their progress and compete with their previous performance, with friends, or with the highest scorers. Typically, the 10 highest scorers get to store their initials in the game for public display. This strategy provides one form of positive reinforcement that encourages mastery. Malone's (1981), Provenzo's (1991), and our studies with elementary-school children have shown that continuous display of scores is extremely valuable. Machine-generated feedback—such as "Very good" or "You're doing great!"—is not as effective, since the same score carries different meanings for different people. Most users prefer to make their own subjective judgments and perceive the machine-generated messages as an annoyance and a deception.

Many educational games use direct manipulation effectively. For example, elementary- or high-school students can learn about urban planning by using SimCity and its variants, which show urban environments visually and let students build roads, airports, housing, and so on by direct-manipulation actions. The social simulation game, The Sims, and its online version broke new ground because of their stronger attraction to women than men.

The esthetically appealing Myst, with its successors Riven and Exile, have drawn widespread approval even in some literary circles, while the more violent but successful DOOM and Quake series have provoked controversy over their psychological effects on teens. Studying game design is fun, but there are limits to the applicability of the lessons. Game players are engaged in competition with the system or with other players, whereas applications-systems users prefer a strong internal locus of control, which gives them the sense of being in charge. Likewise, whereas game players seek entertainment and focus on the challenge, applications users focus on their tasks and may resent too many playful distractions. The random events that occur in most games are meant to challenge the users; in nongame designs, however, predictable system behavior is preferred.

6.2.5 Computer-aided design

Most computer-aided design (CAD) systems for automobiles, electronic circuitry, aircraft, or mechanical engineering use principles of direct manipulation. Building and home architects now have powerful tools such as AutoCAD (see Fig 1.14) with components to handle structural engineering, floorplans, interiors, landscaping, plumbing, electrical installation, and much more. With programs like this, the operator may see a circuit schematic on the screen and, with mouse clicks, be able to move components into or out of the proposed circuit.

6.2 Examples of Direct Manipulation 225

When the design is complete, the computer can provide information about current, voltage drops, and fabrication costs, and warnings about inconsistencies or manufacturing problems. Similarly, newspaper-layout artists or automobile-body designers can easily try multiple designs in minutes and can record promising approaches until they find even better ones. The pleasures in using these systems stem from the capacity to manipulate the object of interest directly and to generate multiple alternatives rapidly.

Related applications are for computer-aided manufacturing (CAM) and process control. Honeywell's Experion Process Knowledge System provides the manager of an oil refinery, paper mill, or power-utility plant with a colored schematic view of the plant. The schematic may be displayed on multiple displays or on a large wall-sized map, with red lines indicating a sensor value that is out of normal range. With a single click, the operator can get a more detailed view of the troubling component; with a second click, the operator can examine individual sensors or can reset valves and circuits.

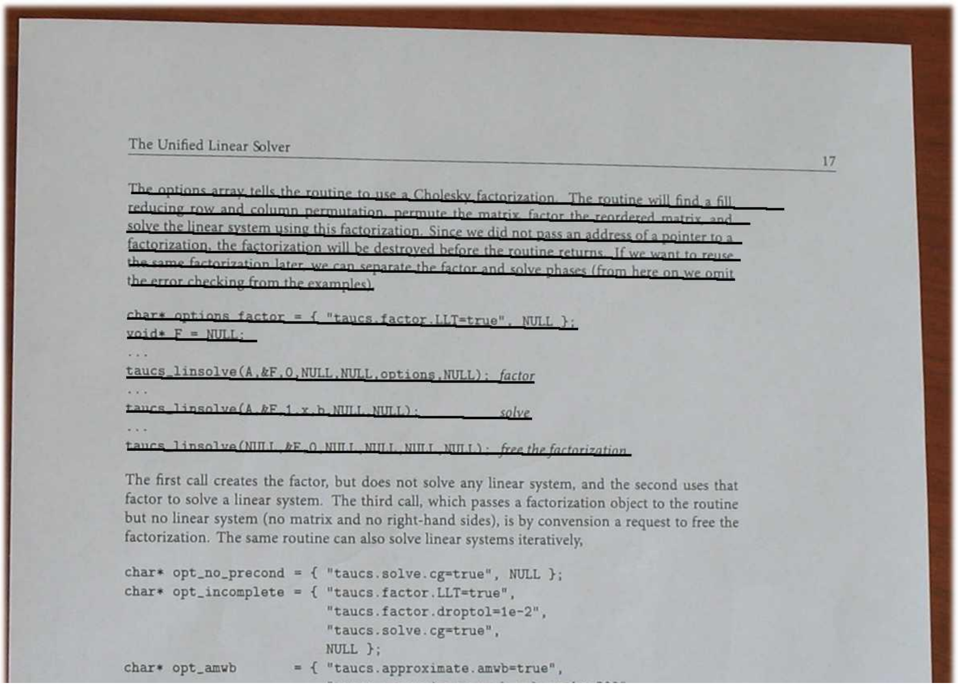
A basic strategy for this design is to eliminate the need for complex commands that the operator might only need to recall during a once-a-year emergency. The visual overview provided by the schematic facilitates problem solving by analogy, since the linkage between the screen representations and the plant's temperatures or pressures is so close.

6.2.6 Office automation

Designers of early office-automation systems used direct manipulation principles. The pioneering Xerox Star (Smith et al., 1982) offered sophisticated text-formatting options, graphics, multiple fonts, and a high-resolution, cursor-based user interface (Fig. 6.6). Users could move (but not drag) a document icon to a printer icon to generate a hardcopy printout. The Apple Lisa system also elegantly applied many of the principles of direct manipulation, although it was not a commercial success, it laid the groundwork for the successful Macintosh. The Macintosh designers drew from the Star and Lisa experiences, but made many simplifying decisions while preserving adequate power for users (Fig. 6.7). The hardware and software designs supported rapid and continuous graphical interaction for pull-down menus, window manipulation, editing of graphics and text, and dragging of icons. Variations on the Macintosh appeared soon afterward for other popular personal computers, and Microsoft now dominates the office-automation market (Fig. 6.8). The Microsoft Windows design is still a close relative of the Macintosh design, and both are candidates for substantial improvements in window management, with simplifications for novices and increased power for sophisticated users.

Early studies of direct-manipulation interfaces confirmed the advantages for at least some users and tasks. In a study of novices, MS-DOS commands for

Figure 1: This figure shows an image of a book taken by one of the cameras in the setup. The lines are drawn to show the curl in the text of the book image.



The Unified Linear Solver 17

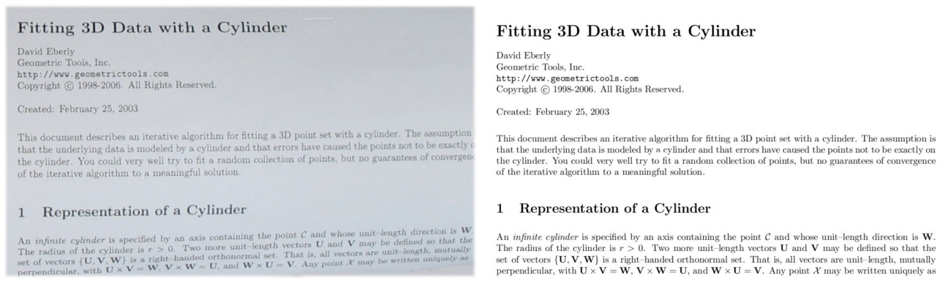
The options array tells the routine to use a Cholesky factorization. The routine will find a fill reducing row and column permutation, permute the matrix, factor the reordered matrix, and solve the linear system using this factorization. Since we did not pass an address of a pointer to a factorization, the factorization will be destroyed before the routine returns. If we want to reuse the same factorization later, we can separate the factor and solve phases (from here on we omit the error checking from the examples).

```
char* options_factor = { "taucs.factor.LLT=true", NULL };
void* F = NULL;
...
taucs_linsolve(A,&F,0,NULL,NULL,options,NULL);  factor
...
taucs_linsolve(A,&F,1,x,b,NULL,NULL);           solve
...
taucs_linsolve(NULL,&F,0,NULL,NULL,NULL,NULL);  free the factorization
```

The first call creates the factor, but does not solve any linear system, and the second uses that factor to solve a linear system. The third call, which passes a factorization object to the routine but no linear system (no matrix and no right-hand sides), is by convension a request to free the factorization. The same routine can also solve linear systems iteratively,

```
char* opt_no_precond = { "taucs.solve.cg=true", NULL };
char* opt_incomplete = { "taucs.factor.LLT=true",
                         "taucs.factor.droptol=1e-2",
                         "taucs.solve.cg=true",
                         NULL };
char* opt_amwb       = { "taucs.approximate.amwb=true",
```

Figure 2: This figure shows an image of a planar document. The lines are again drawn to show that there is no curl in this image



Fitting 3D Data with a Cylinder

David Eberly
Geometric Tools, Inc.
`http://www.geometrictools.com`
Copyright © 1998-2006. All Rights Reserved.

Created: February 25, 2003

This document describes an iterative algorithm for fitting a 3D point set with a cylinder. The assumption is that the underlying data is modeled by a cylinder and that errors have caused the points not to be exactly on the cylinder. You could very well try to fit a random collection of points, but no guarantees of convergence of the iterative algorithm to a meaningful solution.

1 Representation of a Cylinder

An *infinite cylinder* is specified by an axis containing the point $\mathcal{C}$ and whose unit–length direction is $\mathbf{W}$. The radius of the cylinder is $r > 0$. Two more unit–length vectors $\mathbf{U}$ and $\mathbf{V}$ may be defined so that the set of vectors $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ is a right–handed orthonormal set. That is, all vectors are unit–length, mutually perpendicular, with $\mathbf{U} \times \mathbf{V} = \mathbf{W}$, $\mathbf{V} \times \mathbf{W} = \mathbf{U}$, and $\mathbf{W} \times \mathbf{U} = \mathbf{V}$. Any point $\mathcal{X}$ may be written uniquely as

Figure 3: This figure shows the problem addressed. Left image shows the input picture and right picture shows the aimed output.

In solving this problem we use the method devised in [3]. This method as explained in detail in Section 5 uses a conformal mapping which preserves the angels between $3D$ and $2D$ meshes. The difference between our approach and the approach in [3] is the methods we use to build $3D$ models of document surfaces.

After giving a brief introduction to the problem in hand, we want to give first a brief system overview and explain which parts will be explained in detail in this thesis.

System Overview:

As viewed in Figure 4 our system consists of 8 different steps. The steps up to separation of pages include the methods used for constructing the $3D$ model of the document surface. The steps after focuses on the actual dewarping problem.

In this thesis we address the parts matching, smoothing, separation of pages, dewarping and deskewing . We first want to give a brief explanation of each step we address in this thesis and make the reader familiar with the topic.

Matching:

This is the problem of finding correspondences in both images. By correspondences we mean corner points on each image. The problem is given a corner point in the left image we try to find the same point in the right image.

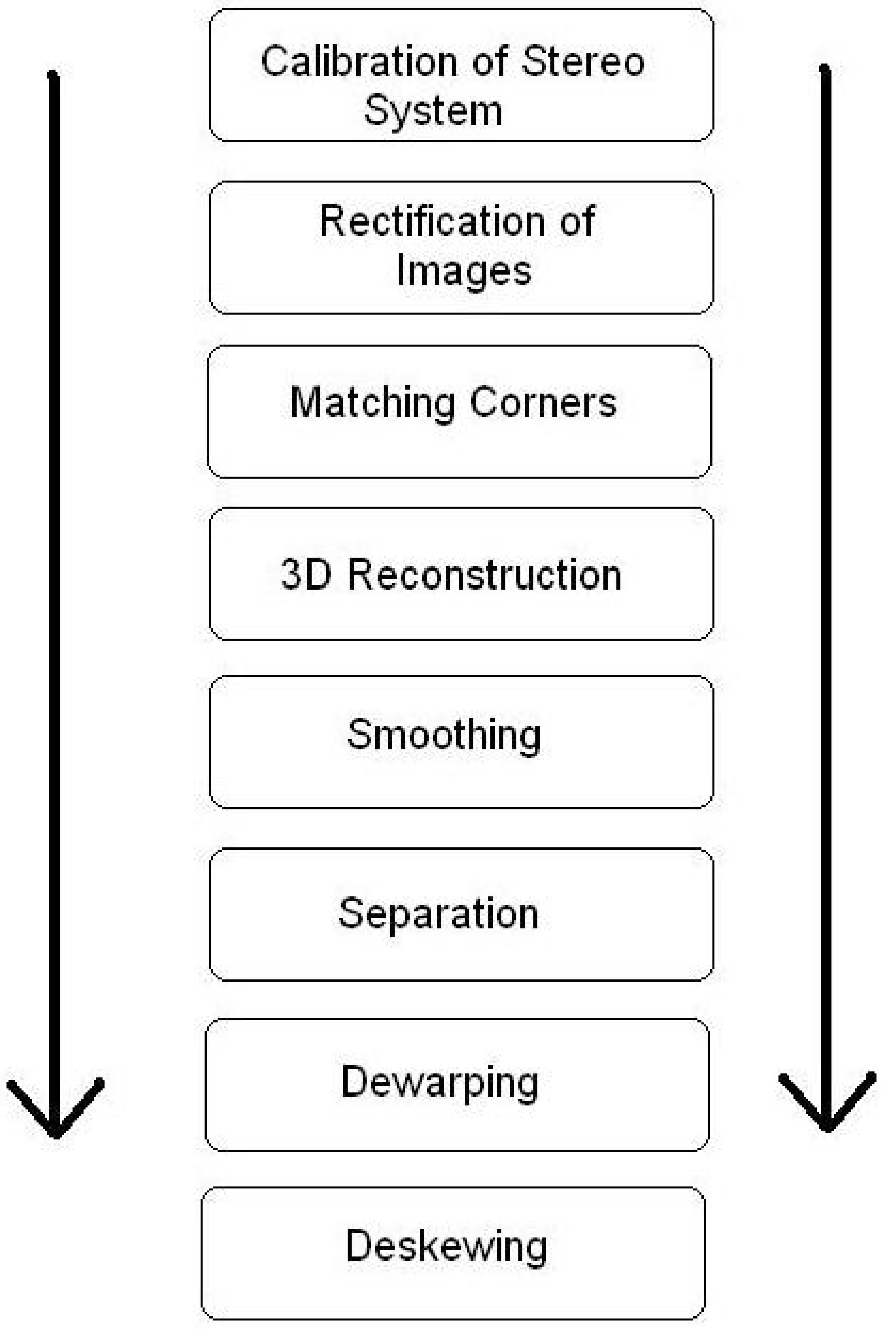


Figure 4: This figure shows the system overview of the whole software.

Smoothing:

After $3D$ reconstruction from given matches as explained in [4] some of the points that actually should lie on surface lie away due to mismatches. So we need a way to move these points to the actual surface. The process of moving so called outliers to the book surface is called smoothing.

Separating Book Pages

This process is finding the middle of a book and separating the book image into two pages.

De-warping

This is the process of finding the transformation that will give us the uncurled version of the same image.

De-skewing

This is the process of actually building the image using the transformation found in de-warping.

The thesis will address these problems by first giving a detailed problem explanation and then explaining the approach we devised to solve it. Section 2 will cover the matching and will be followed by the Section explaining smoothing. Section 4 addresses the problem of separating pages whereas Section 5 will explain the de-warping method. After that we will describe our de-skewing approach in Section 6. These sections will only consist of theoretical part whereas we have Section 7 on explaining some of the experiments we have devised and will in detail explain all the results we reached.

2 Stereo Matching

In this section we cover the problem of stereo matching. First we are going to give a brief summary to the work done by Khawar Parwez. Section 2.1 will cover the stereo system that we have constructed and the rectification process done to images. After that we will in detail address the matching problem and then explain our approach to it.

2.1 Stereo System and Rectification of Images

A stereo system yields two different images of the same view taken from different angles. These different views enable us to build a $3D$ model for the given document surface.(Look at Figure 5 for the stereo system) To build a $3D$ model from a stereo camera system we need to first find correspondences between two pictures. This process is called matching. Matching means finding the exact location of one point in $3D$ space (a point on book surface for example) in both images. Instead of trying to find matches in input images we decide to preprocess it so that solving the matching problem becomes easier. This preprocessing step is called the rectification of input images. Rectification is the process of projecting both images to a reference frame so that we have an alignement in the $y-axis$ of both images(see Figure 6). This can be explained as:
Let $h(u)$ be the transformation for our rectification ($u=(x,y)$ be a pixel in an image) and also let $f(x_f,y_f)$ be a point in left image and $g(x_g,y_g)$ be the corresponding point in right image.Then:

$$f_h = h(f) = (x_f^h, y_f^h) \text{ and } g_h = h(g) = (x_g^h, y_g^h)$$

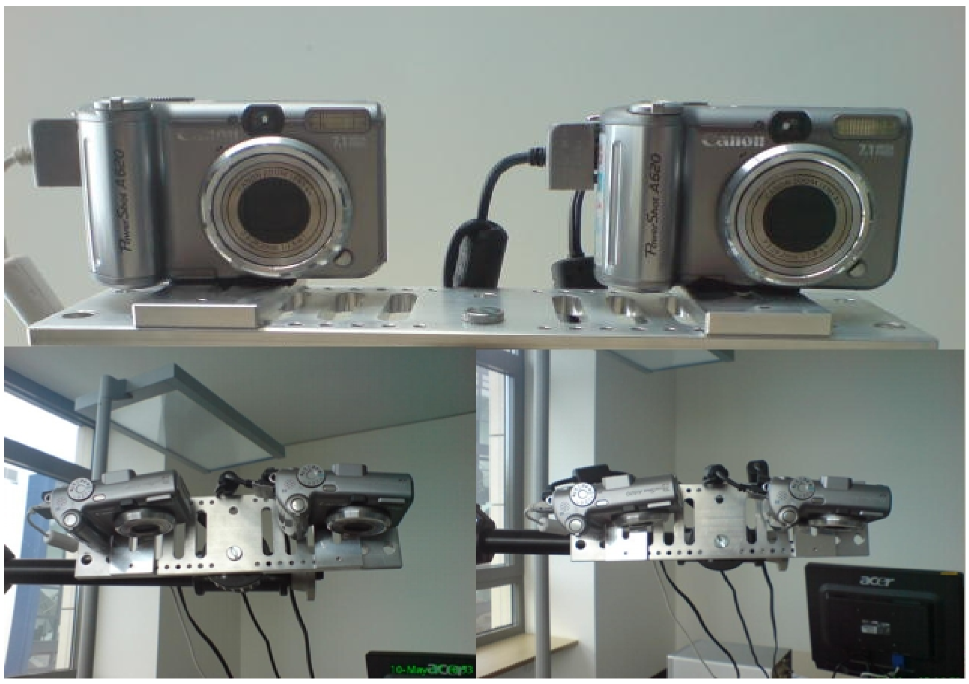


Figure 5: Our stereo system for stereo view. Both cameras covers the same view thus enabling us to get a good $3D$ model

.

The rectification impose on $y - axis$ of both f and g that:

$$y_f^h = y_g^h$$

As can easily seen from this equation this will enables us to make a search for matches only in the $x - direction$. For further details on how to implement this transformation please look at [4].

2.2 Matching Problem

The matching problem is to find point correspondences between two pictures taken with different cameras on different views. This problem can be seen as finding correspondences between the two pictures. In more formal terms we can define the problem as:

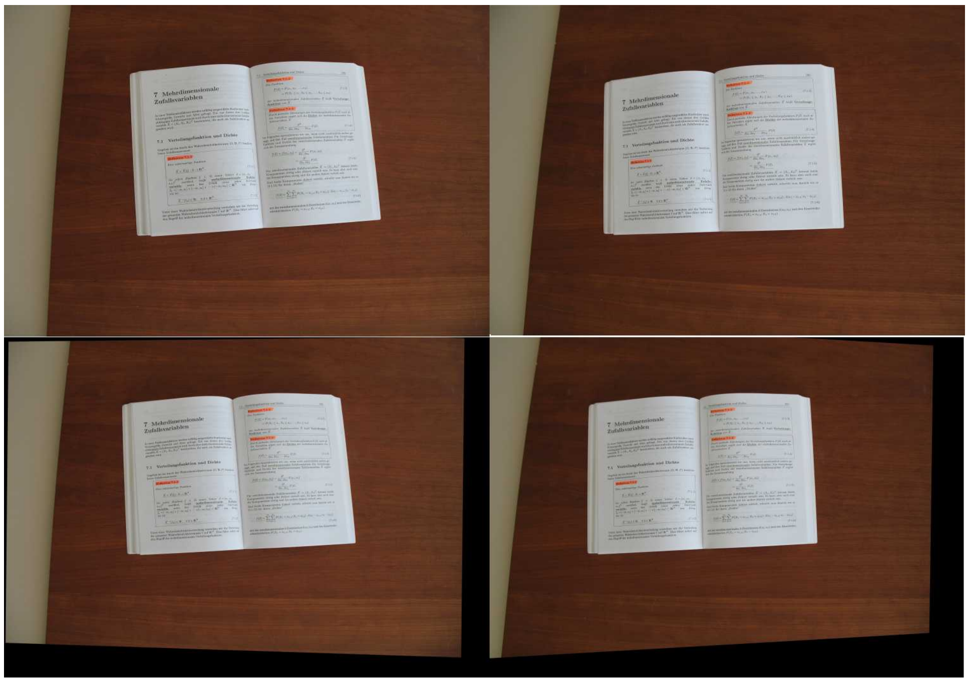


Figure 6: This figure shows an image pair taken by our stereo sytem and their rectified versions. The top images are the original images and the bottom pair is the rectified versions.

Problem:

Let $f = (x_f, y_f)$ be a point in the image taken by our left camera of the system. The matching problem is to find the exact point $g = (x_g, y_g)$ in the image taken by our right camera that corresponds to the same point as f.
Since doing it pixel by pixel will not be conventional we redefine the problem from point matching to block matching. This can be explained as:
Let $f = (x_f, y_f)$ be a point in the image taken by our left camera of the system then we define a block γ_f as a set of all points with $p = (x_p, y_p)$ $|x_p - x_f| < d_x$ and $|y_p - y_f| < d_y$ where d_x and d_y is distance given in pixels.Now we try to find a block on γ_g right image with the same size as γ_f that minimizes a similarity measure. The center of block γ_g ($g = (x_g, y_g)$), minimizing the similarity measure, is the correspondence of $f = (x_f, y_f)$.
Now our problem becomes matching the whole box rather than a point with

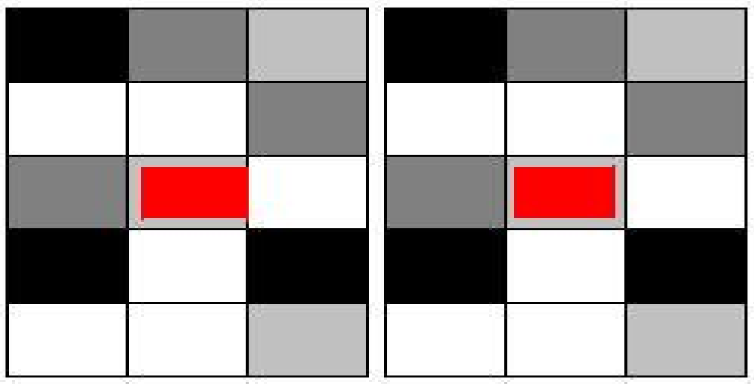


Figure 7: This figure shows an example of matching blocks taken from different images. Red rectangle is the point we try to match. Here $d_x = 1$ and $d_y = 2$

boxes on the right image.Figure 7 shows a matching block pair taken from different images. This method however forces some properties on the points to be used for these blocks. These are:

1. These points should be easily detected on both pictures.

2. These points should have a high difference in both contrast and brightness from neighboring points.

For this purpose as explained in [4] we are using some feature points extracted by **Harris Feature Extraction Method** [5](see Figure 8). This method, as explained in detail by [4], returns us corners on the source image, which satisfy both of the properties above. So, basically we first define some edge points on one of the images and then try to find the corresponding images in the stereo image. Our approach is defined in detail in the next section. Figure 9 shows an example for matching.

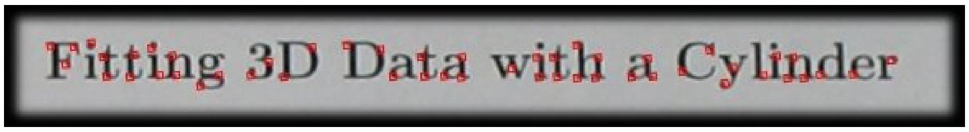


Figure 8: This is an example of feature points returned from Harris Feature Extraction Method. As you can see they are the best approximations to the corners of letters

5 The primary UMFPACK routines
5 The primary UMFPACK routines

Figure 9: This figure shows parts of images taken by the stereo system. The white boxes on the left image shows the Harris feature Points. The white boxes on the right image shows the matches of these feature points

2.3 Our Approach

After finding the corners on the source image, we again divide the matching algorithm into two steps. The first step tries to guess a mean value for the displacement vector in $x - axes$ of two images. By displacement vector what we mean is:

Let $f(x_f, y_f)$ be a feature point in left image and $g(x_g, y_g)$ be its exact match in the right image. Then the displacement vector is $d = \begin{bmatrix} d_x & d_y \end{bmatrix}^T$ with $d_x = x_f - x_g$ and $d_y = y_f - y_g$.

However as explained in Section 2.1 our images are already rectified so usually $d_y = 0$. In this first step what we get is an estimation to the mean of all displacement vectors. This can be viewed as:

$$d_m = \begin{bmatrix} \frac{\sum_{i=1}^{n} d_{x_i}}{n} \\ \frac{\sum_{i=1}^{n} d_{y_i}}{n} \end{bmatrix} \text{ n is the number of all feature points}$$

After estimating a global displacement vector we can actually go to the next step of our algorithm and find the exact matches. In our matching algorithm we need prediction points to begin our search with and we use the global displacement vector to find these prediction points. We can actually use an approach like this since we have a smooth surface and the displacement vectors for each feature point does not really differ very high from this mean value. The next step will be to find the exact matches using the predicted pixels as starting values.

2.3.1 Determining the Displacement Vector

For this purpose we are actually using a software library devised by Prof. Thomas Breul. This software named RAST(Recognition by Adaptive Subdivisions of Transformation Space) will be used to recognize the book on left and right image and then to give us the displacement vector between them. To do this, a first step is to run a canny edge detection on both images so that we can detect the books inside our pictures(see Figure 10). After this step, we only calculate the required displacement vector by feeding these corners to the software and in return we find the required displacement vector.

In brief what this software actually does is to first finding the canny edges on both pictures and then try to match these edges on both images and find a median value for the displacement vector of each match from one picture to another.For more information on how this software and algorithms work, please see [6]

2.3.2 Finding the Matches

For finding the corresponding matches as explained in [4], we are using a block matching connected with a discrete gradient descent method (for more information on the discrete gradient descent method please refer to Section A.3). This method is proposed in [7] and is based on matching blocks in the

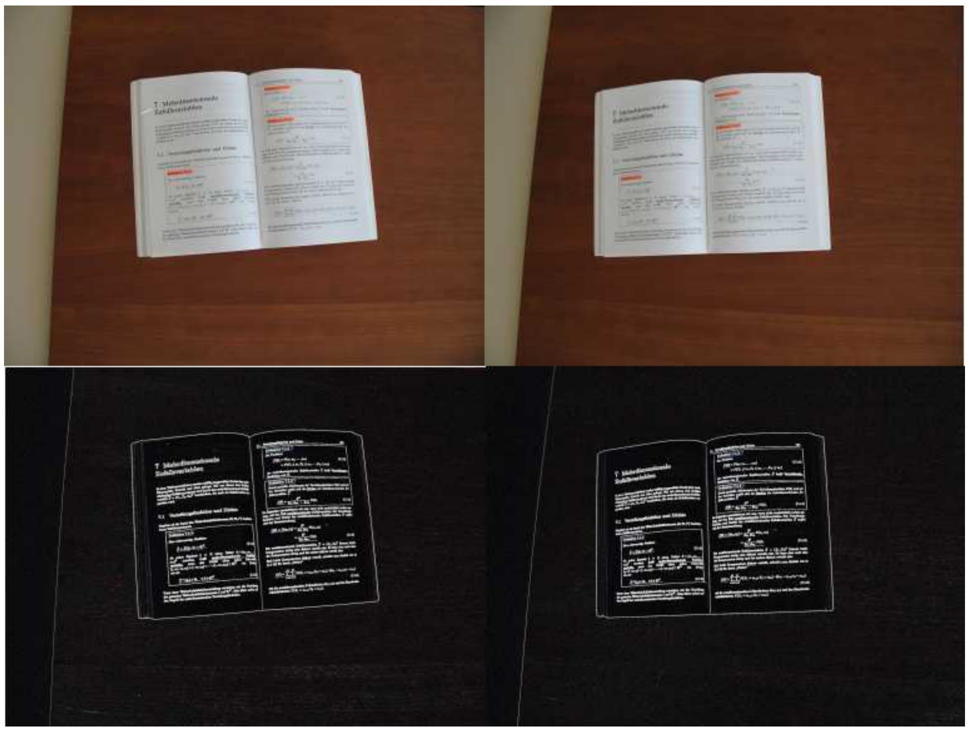


Figure 10: This figure shows an image pair taken by our stereo system and images of edges detected by RAST.

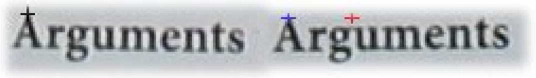
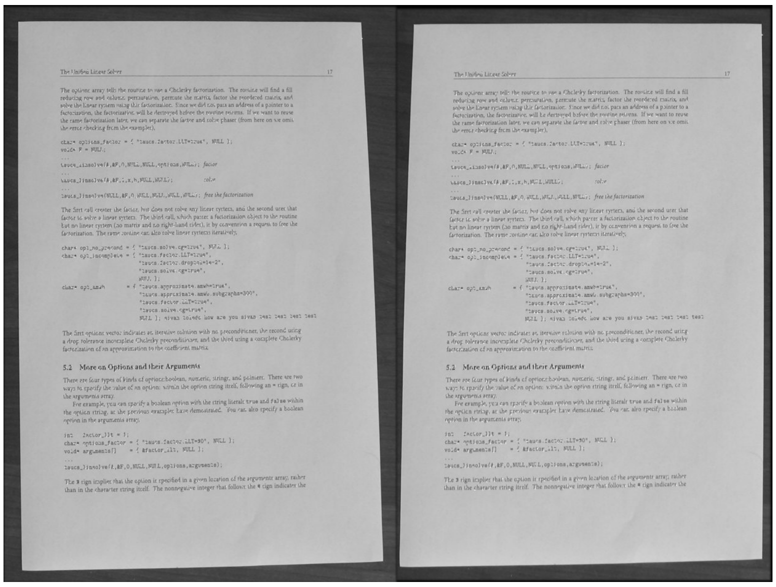


Figure 11: This figure shows an image pair taken by our stereo system. In the left picture black cross shows a feature point to be matched. In the right picture the red cross shows the predictor location whereas the yellow cross shows the location of exact match.

surronding of the expected positions of an interest point. As explained in [4] after an extentive testing we choose to use sum of square differences(SSD) of pixel values as our similarity measure between blocks. To summarize the method basically we represent images as $f_{left}(x, y)$ as a point in left image and $f_{right}(x+d_x, y+d_y)$ as our candidate for the corresponding point in the right image. We define our block as a set of pixels υ so the block SSD becomes:

$$\sum_{(x,y)\ \epsilon\ \upsilon} (f_{left}(x, y) - f_{right}(x + d_x, y + d_y))^2$$

So by calculating the corresponding similarity measure for all points and using gradient descent algorithm to find the local minimum of SSD, we find matches. As explained in [7], this method is prone to contrast and image noise caused by movement. In our case, on the other hand, these effects are not as much of a concern since our images usually have no or very little contrast change and we have no skew caused by motion since our images are being taken without any motion. One of the problems we suffer from is the similarity measure being the same when blocks near our starting point on both directions tend to look alike (See figure 13 below). To solve this problem, we have increased the block size so that these redundancies can be overcome. This really slows the algorithm since more computational work is needed to find an exact match but is actually the best solution to solve this problem. However we have to take into account that larger block size can mean a larger skew, which is a disadvantage of this method.
To produce predictors, as explained in Section 2.3.1, we first estimate a displacement vector $d_m = (d_x, d_y)$ between the images of stereo image pair.



The Unified Linear Solver 17

The options array tells the routine to use a Cholesky factorization. The routine will find a fill reducing row and column permutation, permute the matrix, factor the reordered matrix, and solve the linear system using this factorization. Since we did not pass an address of a pointer to a factorization, the factorization will be destroyed before the routine returns. If we want to reuse the same factorization later, we can separate the factor and solve phases (from here on we omit the error checking from the examples).

```
char* options_factor = { "taucs.factor.LLT=true", NULL };
void* F = NULL;
...
taucs_linsolve(A,&F,0,NULL,NULL,options,NULL); factor
...
taucs_linsolve(A,&F,1,x,b,NULL,NULL);          solve
...
taucs_linsolve(NULL,&F,0,NULL,NULL,NULL,NULL); free the factorization
```

The first call creates the factor, but does not solve any linear system, and the second uses that factor to solve a linear system. The third call, which passes a factorization object to the routine but no linear system (no matrix and no right-hand sides), is by convention a request to free the factorization. The same routine can also solve linear systems iteratively.

```
char* opt_no_precond = { "taucs.solve.cg=true", NULL };
char* opt_incomplete = { "taucs.factor.LLT=true",
                         "taucs.factor.droptol=1e-2",
                         "taucs.solve.cg=true",
                         NULL };
char* opt_amwb       = { "taucs.approximate.amwb=true",
                         "taucs.approximate.amwb.subgraphs=300",
                         "taucs.factor.LLT=true",
                         "taucs.solve.cg=true",
                         NULL }; sivan toledo how are you sivan test test test test
```

The first options vector indicates an iterative solution with no preconditioner, the second using a drop tolerance incomplete Cholesky preconditioner, and the third using a complete Cholesky factorization of an approximation to the coefficient matrix.

5.2 More on Options and their Arguments

There are four types of kinds of options:boolean, numeric, strings, and pointers. There are two ways to specify the value of an option: within the option string itself, following an = sign, or in the arguments array.

For example, you can specify a boolean option with the string literals true and false within the option string, as the previous examples have demostrated. You can also specify a boolean option in the arguments array.

```
int   factor_llt = 1;
char* options_factor = { "taucs.factor.LLT=#0", NULL };
void* arguments[]    = { &factor_llt, NULL };
...
taucs_linsolve(A,&F,0,NULL,NULL,options,arguments);
```

The # sign implies that the option is specified in a given location of the arguments array, rather than in the character string itself. The nonnegative integer that follows the # sign indicates the

Figure 12: This figure shows an image pair taken by our stereo system. The left picture shows the picture where we extract featurs and features are shown by white squares. The right picture shows the matching points in white squares.

After this for each feature point extracted $p_{ext} = (x_{ext}, y_{ext})$ we create an initial prediction point p_{pred} as:

$$p_{pred} = (x_{ext} - d_x, y_{ext} - d_y)$$

To have a view how these predictor points see Figure 11.

Most of the results obtained in planar document cases proved to be quite good.(see Figure 12 fo an example).

However in the case of spine of a book, this method tends to give a lot of mismatches. This is due to the fact that on the spine of the book the images tend to change in shape and skew a lot. Other than that, some of the points available in one image may not be covered in the next one. As can be seen from Figure 14 the difference of shape, contrast and skewness between the two images is too much for our similarity method to comprehend.

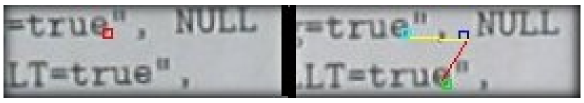


Figure 13: This figure shows the images taken from both cameras in the case that same blocks are very near. In this picture, the left side shows the source image and the right side shows its stereo pair. The blue rectangle shows the initial start point where the matching algorithm begins its search, whereas the green rectangle shows the matched point. As easily observed, instead of going through the yellow line to find the light blue point,which is the actual match, our algorithm goes through the red line to find the green point which is a mismatch.

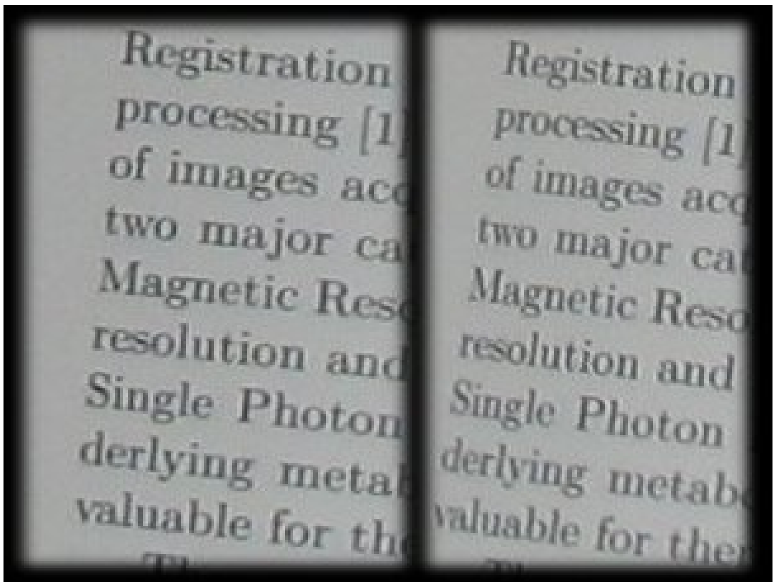


Figure 14: This figure shows the images taken from both cameras for the spine of a book. The drastical change in shape, curl and contrast in both pictures can easily be observed

This causes our algorithm to find a different local minimum instead of the matching point and hence the mismatches. To solve this problem we actually need a new method that takes into account these changes between the two images. For this purpose, we used the method derived in [8]. This is a method used for feature tracking in video processing. In this method instead of supposing simple changes between two images, changes are modeled using a more complex model such as an affine map. This affine map is used to fit the image views to each other so that we can actually find a better match.(For more on the method please refer to [8]).However there are some differences between [8] and the problem we have. Since we have a stereo system we have very little noise associated with movement. Other than that, contrast measure is again not such a high priority. The part that is most useful for us in [8] is its ability to change the size and orientation of source blocks. This property allows us to change the source blocks thus enabling to find an exact match. For our results on experimenting with this new method, please refer to Section 7.

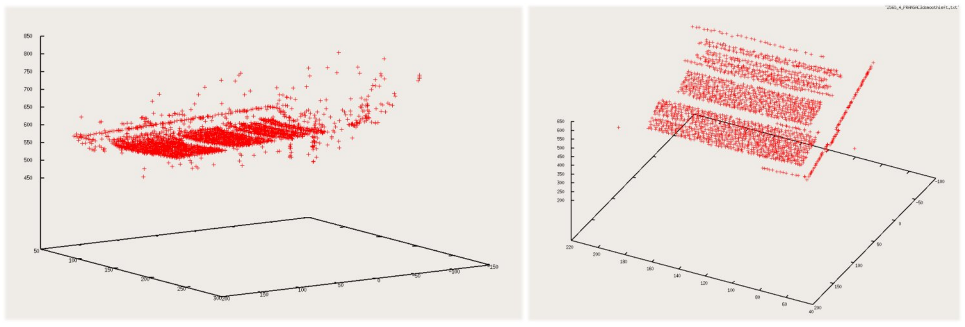


Figure 15: This picture shows an example of a smoothing. The left image shows the model without smoothing whereas the right image shows the smoothed version

3 Smoothing

In this section we will explain the smoothing problem and methods used to solve it. After $3D$ reconstruction we obtain a $3D$ model of our document surface. However when we look at Figure 17, we can easily observe that some of the points are not on the document surface. These points are called outliers and are a direct result of mismatches. As explained in Section 2 we have devised different methods to solve mismatches problems but we still need to do a post processing on our $3D$ model to have a perfect surface model. To achieve this we have to first detect and then discard or move these outlier points to document surface. So now we can define our problem as:

Problem: Given a set of $3D$ points named Ω (which is all the points in our $3D$ model), we try to find the best fitting surface to this set of points. (see Figure 15)

In the case of a planar document actually what we can do is just to find the best matching plane to Ω. However a book surface is not a known geometric shape and even if we find a model to fit to one book surface(say Ω), we can not be sure that it will be valid for all different book surfaces. You can have a visual of this problem by looking at Figure 16. As can be seen from Figure

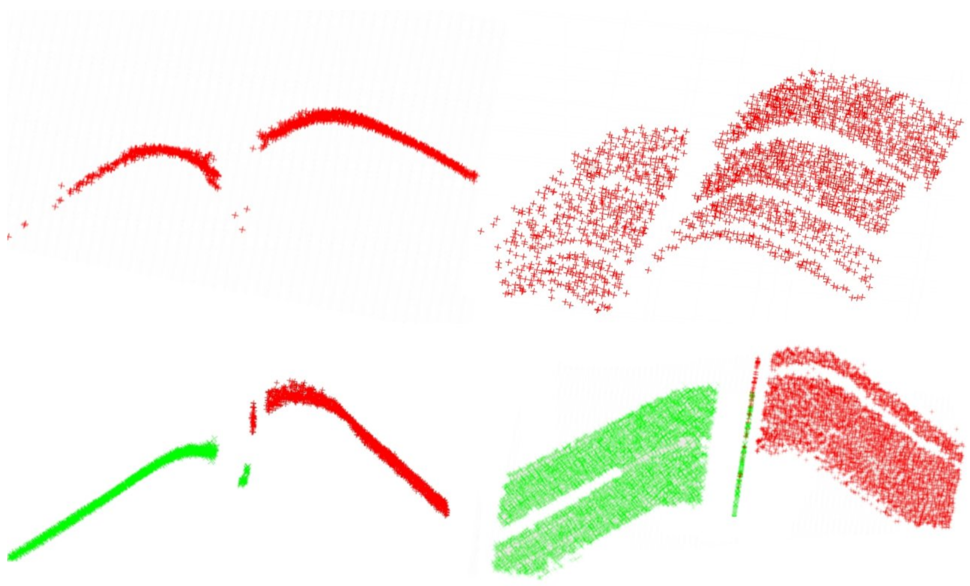


Figure 16: This figure shows different book surface models. We can see apparently that the two book surfaces are totally different. This roves that it is not possible to find a function fitting to all book surfaces.

16 finding a global surface to fit to different book images is a very hard task. Instead of finding a global surface to all points we decide to use a divide and conquer method to our problem. This can be defined as:

Let $w = (x_w, y_w, z_w)$ be a point in our $3D$ model. Then we define a set containing points that are in a local neighborhood of w as β. The set β consist of all point in our $3D$ model given that a point $p = (x_p, y_p, z_p)$ holds the following equations.

$$|x_w - x_p| < \sigma_x \tag{1}$$

$$|y_w - y_p| < \sigma_y \tag{2}$$

Here σ_x and σ_y are maximum distances from the source point. After finding all the elements of β we still need to find a fitting surface. However we now have a smaller neighborhood which lets us to make an assumption.

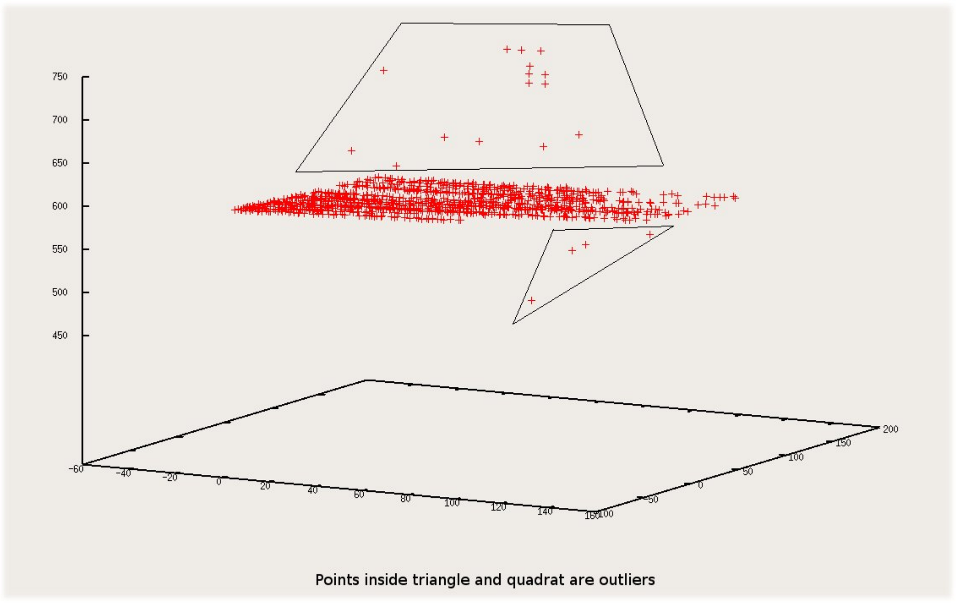


Figure 17: This is a picture of an un-smoothed $3D$ model. As you can see outliers are highlighted.

- We can pick an adequate value for both σ_x and σ_y so that we have a surface that an be approximated to linear functions

By using this assumption we will use a linear regression method to find a fit to set of $3D$ points β. Having this in mind we began our implementation of algorithm first by fitting to a plane. In this method we try to find the best fitting plane to β and instead of moving all points in β to the plane we only change the location of w to this plane. We do the same operation to all points in Ω. After trying this method on different images we found out that fitting to a plane is not suitable for a book surface especially on the spine. Thus we change our approach to fitting to an applicable surface instead. In both approaches we use **RANSAC** "RANdom SAmple Consensus" algorithm. We have used RANSAC since as explained in [4] we have a much better performance using it.

3.1 Linear Least Squares Method

It is an optimization method to find the best fitting function to a cloud of data points. This method is used to minimize the square of the distance between the line and the point on the cloud set. The reason for using squares of distances is to treat the residuals as a continuous differentiable quantity, which is not the case when we use absolute differences instead. We have done two different types of least squares fitting. The first one is to fit to a plane and the second one is to fit the paraboloid.

3.1.1 Fitting to a plane

In this method we try to find a fitting plane to a given set of three dimensional points $P_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ $i = 1, ..., n$ to a plane with an equation $N.(P - P_0) = 0$. In this equation $N = (N_x, N_y, N_z)$ is the normal to the fitting plane and P_0 is a point on the plane.The optimum fitting plane should pass through P_0 which is the mean of all points that we want to fit on the plane thus we first find the mean of all points as:

$$(x_m, y_m, z_m) = \frac{\sum_{i=1}^{n}(x_i, y_i, z_i)}{n}$$

So to find the best fitting plane to these points what we need is to solve the next equation:

$$\begin{bmatrix} (x_1 - x_m) & (y_1 - y_m) & (z_1 - z_m) \\ (x_2 - x_m) & (y_2 - y_m) & (z_2 - z_m) \\ . & . & . \\ . & . & . \\ . & . & . \\ (x_n - x_m) & (y_n - y_m) & (z_n - z_m) \end{bmatrix} \begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix} = 0$$

More on linear least squares fitting can be found on [9]. This system is a rectangular system and needs to be converted into a symmetric system which can be easily done by multiplying both sides of the equation with the transpose of our error matrix A^T. After this multiplication, we have a new matrix $B = A^T A$, which is symmetric and our new equation system becomes $BN = 0$. Another specification we can impose on our system is that $\|N\| = \sqrt{N_x^2 + N_y^2 + N_z^2} = 1$. This results in solution being equal to the eigenvector of the smallest eigenvalue. The proof of the theorem can be found in [10]

3.1.2 **Fitting to a paraboloid**

Near the spine of a book, our estimation of fitting to a plane seems apparently not so effective. As can be seen easily from Figure 18 a better choice for a surface is a paraboloid. For this reason, we also try to do a smoothing based on fitting our surface on a paraboloid.

To solve this problem, first we need the equation of a paraboloid which

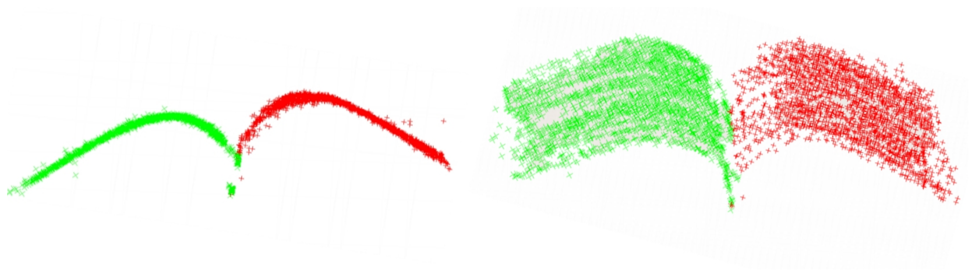


Figure 18: This picture shows a $3D$ model for the spine of the book.

looks like $f(x,y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$. In our case, we take $f(x,y) = z$. Our residual function looks like:

$$R^2 = \sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))^2$$

After computing the partial derivatives on all unknowns A, B, C, D, E, F we have a system that looks like:

$$\frac{\partial R^2}{A} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))x_i^2 = 0$$
$$\frac{\partial R^2}{B} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))y_i^2 = 0$$
$$\frac{\partial R^2}{C} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))x_iy_i = 0$$
$$\frac{\partial R^2}{D} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))x_i = 0$$
$$\frac{\partial R^2}{E} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F))y_i = 0$$
$$\frac{\partial R^2}{F} = -2\sum_{i=1}^{n}(z_i - (Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F)) = 0$$

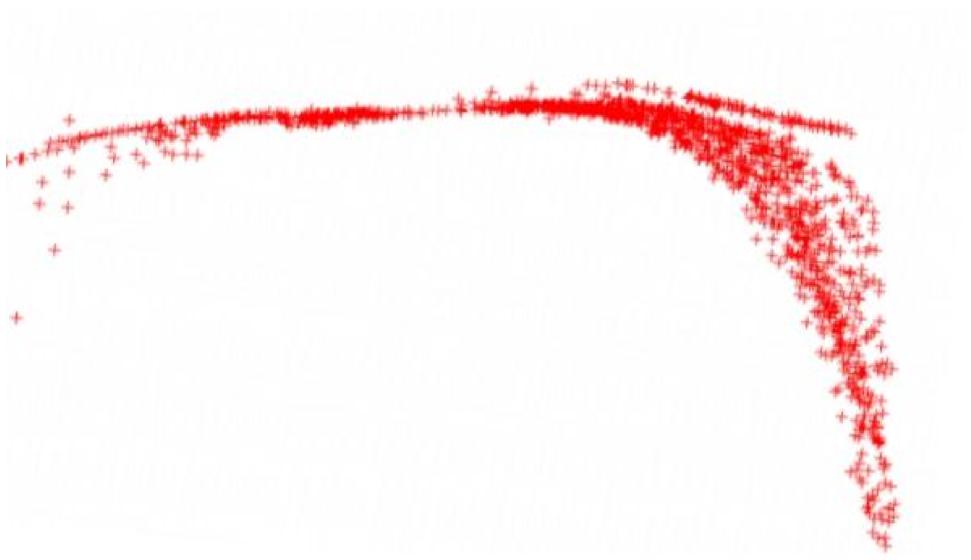


Figure 19: This picture shows an over estimated curl in the spine of the book.

Writing these equations in a matrix format and using the fact that this is a Vandermonde matrix (For more information on this conversion please refer to [2]) we can actually build our solution system as:

$$\begin{bmatrix} x_0^2 & y_0^2 & x_0y_0 & x_0 & y_0 & 1 \\ x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 & 1 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_n^2 & y_n^2 & x_ny_n & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ . \\ . \\ . \\ z_n \end{bmatrix}$$

Solving this equation using numerical methods will give us the best fitting paraboloid to our points. However as can be seen from Figure (19) near the spine of a book a paraboloid can have an overestimated curl on both

[2]http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html

directions. This also means that a paraboloid has a Gaussian curvature that is not equal to 0. This is not acceptable since book surface as an applicable surface has the property that its Gaussian curvature becomes zero so we need to change our model to have a better fit to the book surface. We already know that terms D, E and F have very little effect on the curl and is only showing direction of the plane so we can ignore them. However the terms A, B and C affect the curl. To get rid of the least effecting direction for the curl we build the matrix:

$$\begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

Unlike matrix solution in this case we do not take the eigenvector associated with the minimum eigenvalue(λ_{min}) but the eigenvector associated with the maximum eigenvalue(λ_max). This in return will give us a solution minimizing the effect of low curl direction and maximizing the high curl direction. To find the new values associated with A, B and C:

$$z_{new} = \lambda_{max}^2 (x_{max} \begin{bmatrix} x \\ y \end{bmatrix})^2 + Dx + Ey + F$$

where $x_max = \begin{bmatrix} a_1 & a_2 \end{bmatrix}^T$ is the eigenvector associated with λ_{max}. From this we have:

$$\begin{aligned} A &= a_1^2 \lambda_{max}^2 \\ B &= a_2^2 \lambda_{max}^2 \\ C &= a_1 a_2 \lambda_{max}^2 \end{aligned} \tag{3}$$

This method explained above is used for our paraboloid model to look more like an applicable surface thus having a better fit to our book surface.

3.2 RANSAC

After this brief introduction to the fitting methods we use, we want to emphasize on our implementation of a RANSAC algorithm for locally smoothing $3D$ points. Before giving details of our implementation, we want to give a brief introduction to the general RANSAC algorithm.

3.2.1 General Algorithm

This is an algorithm to fit data into models robustly in the presence of many data outliers.This algorithm was first proposed by Fischler and Bolles in [11].
The algorithm tries to estimate a set of parameters (say a_i for $i = 1, ..., n$)for a function and then fit the data according to these given parameters. There are two rules that data has to hold for RANSAC to be applicable. These are:

- The parameters can be estimated from n data items
- There must be at least $n + 1$ data items in total

For the RANSAC algorithm we define a set of points that holds both conditions above as η. η has m number of elements. The general algorithm for RANSAC is very easy. The algorithm looks like:

1. Select k points randomly where $k > n$ and $k < m$
2. Estimate parameters $a_1, ..., a_n$ for your fitting function

3. Find the total number of points(say ϱ_i where i is the number of iteration) whose distance from the given parameterized surface is smaller than a given threshold distance τ by the user.

4. Check if the number of points (ϱ_i) is larger than the largest number (ϱ_{best}) so far. If it is larger change the current ϱ_{best} to the new solution and store the parameters for the best solution so far.

5. Do the same five steps above for l iterations(l given by user)

6. Return the best solution(parametrization corresponding to ϱ_{best}) found by the algorithm as the best fitting solution.

3.2.2 Our implementation

We have two different parametrization for our RANSAC algorithm. These as explained in 3.1 are plane fitting and paraboloid fitting. Since we are using a local neighborhood of points for our algorithm we first want to define some of the variables we use in our RANSAC. First we run the RANSAC on each point $w = (x_w, y_w, z_w)$ and for each point w we define a local neighborhood as a set β with points from the global set that holds the requirements explained in Section 3

For plane fitting we need three parametrization variables which means in each iteration of our RANSAC we pick three points from our set β randomly. From this three points we find the corresponding plane by using a least squares fitting.Our least squares fitting method returns us a normal to the plane N. After that for all points in β we find the distance between the point and the plane fitted to these picked three points. We calculate it by using the formula:

$D_{x_i} = N(x_0 - x_i)$where x_i is a point in β, x_0 a point on plane(any of three

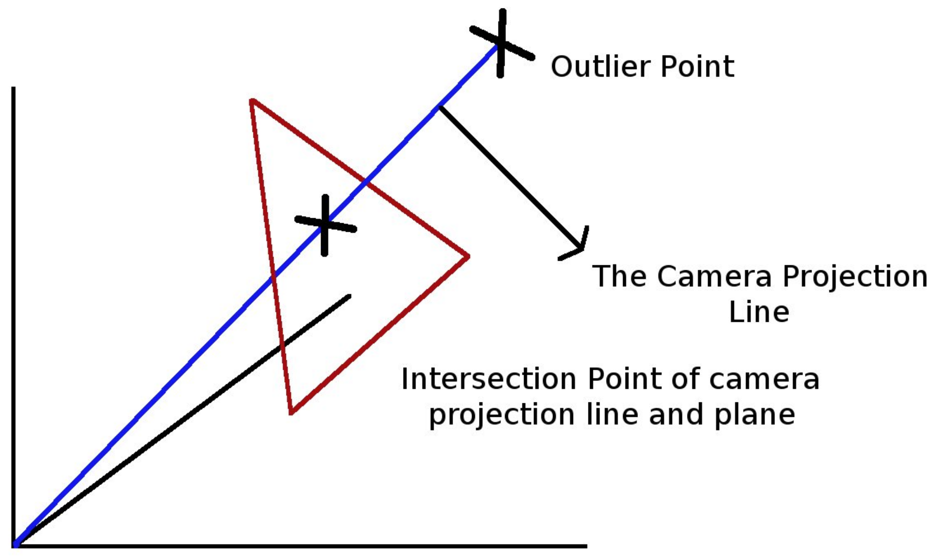

Outlier Point
The Camera Projection Line
Intersection Point of camera projection line and plane

Figure 20: This picture shows the process of moving an outlier through the camera projection line to the intersection with the fitting plane.

picked) and N is a normal to plane.

After finding the total number of points(say ϱ_j)for this iteration that holds $D_{x_i} < \tau$. We check this number with the largest value for number of points (say ϱ_{best}) and if it is bigger we store the parametrization values and change ϱ_{best} otherwise we discard both. After doing the same operation for l iterations our algorithm returns the parametrization variables for ϱ_{best}. After this we move the point w to the point where the projection line and the plane returned by RANSAC intersects.(Look at Figure 20)

In the second method to parametrize a paraboloid we need six different variables so the first change in RANSAC is the number of points picked randomly. We pick six points and from them again using our linear regression model we find the best fitting paraboloid. After that to calculate the distance we use the following equation:

$D_{p_i} = Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i + F - z_i$ where $p_i = (x_i, y_i, z_i)$ is a

point in β and A, B, C, D, E and F are the parameters for the paraboloid. After finding the total number of points(say ϱ_j)for this iteration that holds $D_{p_i} < \tau$. We check this number with the largest value for number of points (say ϱ_{best}) and if it is bigger we store the parametrization values and change ϱ_{best} otherwise we discard both. After doing the same operation for l iterations our algorithm returns the parametrization variables for ϱ_{best}. To move points to the fitting surface we only change the value of z_w to
$z_w = A_{best}x_w^2 + B_{best}y_w^2 + C_{best}x_w y_w + D_{best}x_w + E_{best}y_w + F_{best}$ where $w = (x_w, y_w, z_w)$ and $A_{best}, B_{best}, C_{best}, D_{best}, E_{best}$ and F_{best} are the parameters for the paraboloid returned by RANSAC.
This method is used because a line and a paraboloid intersects on more than one point and we do not know which point will be the solution of our problem.

4 Separating a Book Image

In this section we will address the topic of separating a book image into its pages. This method is needed since in our book images it is easier to de-warp them when we separate the image into its pages. In the first section we will give the explanation of our problem. In the next section I will give details about our approach and how it works.

4.1 Problem

We are trying to find the spine of a book in an image. The spine of the book is the connection between two consecutive pages in a book(look Figure 21). This particular part of the book is very important since it is the place where the highest amount of curl is present. Other that this we have the problem that it is not possible to create a model of this particular place since it is totally white and no features can be detected on images however this place particularly contains important depth knowledge that must be taken care of. Instead of dealing artificially created feature points we separate the book image into two pages and also add the feature for the user to choose the page he wants to de-warp only.

4.2 Method

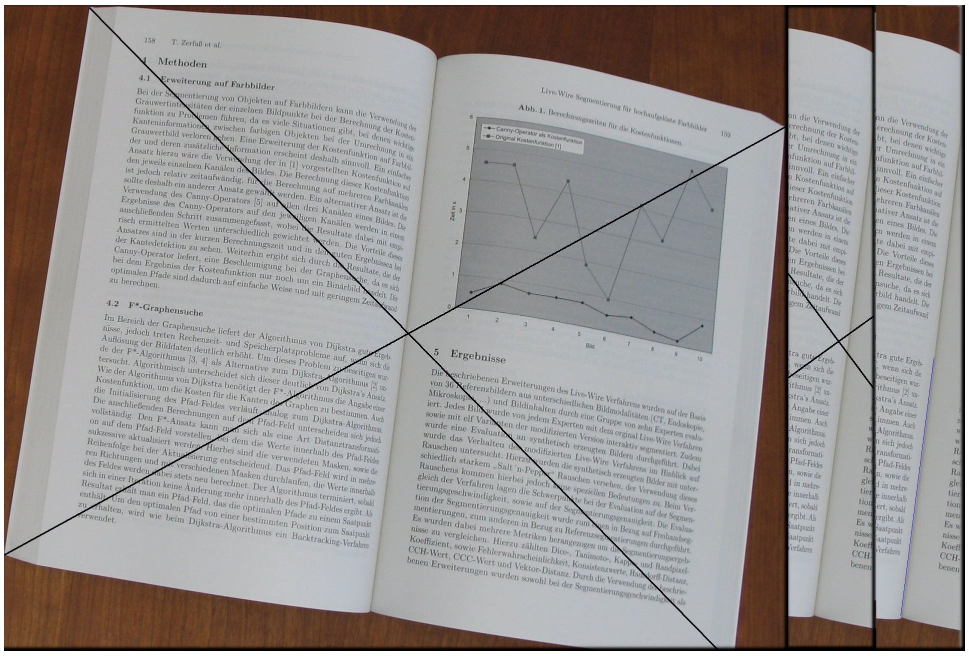
For finding the location of the spine of the book inside the image we first have to extract the book from the image. The method we use to achieve this purpose is explained in [4]. This method returns us the four corner points of the book in the image.There are however some assumptions that has to be made before we explain our method. These are:

- The shape of the book in image should be approximated as a rectangle

- Perspective effects on the image should be minimal.

These assumptions usually hold in the case that we have taken the photo from a distance not too near to the book or the book lies on a surface that is perpendicular to the camera. In our images both of these rules hold. After this we should easily declare that the spine of the book should lie somewhere in the middle of our extracted image. To find the middle point we find the intersection point ($P_{intersect} = (x_i, y_i)$) of diagonals from the corner points of the extracted image.(See Figure 21)After this step we define a sub image neighboring $P_{intersect}$. This neighborhood can be explained as:
All pixels ($p_j = (x_j, y_j)$) on extracted image with $|x_j - x_i| < \alpha$ pixels
After some experimentation we found out that 50 is a good estimate for α. After this we again use RAST software devised by Prof. Thomas Breuel and explained in [6] to find the exact location of the spine. This time the software works in a two step process. First we find canny edges on the extracted image(Look at Figure 10). From these canny edges then software builds up lines. After finishing execution this software returns us line segments and it is now our duty to pick which one is the spine of the book. For this purpose after experimentation with different images we found out that the longest segment actually turns out to be the spine of the book. This line segment is then modeled by a point ($p_{line} = (x_l, y_l)$)and the slope of the line m. After finding these values we just check for each pixel in the image if it lies on the right or left of this line. This can be done easily by using the below equations: A point $P_1 = (x_{P_1}, y_{P_1})$ lies on the left side of the line if $mx_{P_1} + (y_p - mx_p) < y_{P_1}$ and in the right otherwise.
By applying this equation on every pixel of the image we have now separated the book in the image into its pages.(Look at Figure 22)

This method however is not actually very robust. There are some cases in


158 T. Zerfaß et al.

4 Methoden

4.1 Erweiterung auf Farbbilder

Bei der Segmentierung von Objekten auf Farbbildern kann die Verwendung der
Grauwertintensitäten der einzelnen Bildpunkte bei der Berechnung der Kosten-
funktion zu Problemen führen, da es viele Situationen gibt, bei denen wichtige
Kanteninformationen zwischen farbigen Objekten bei der Umrechnung in ein
Grauwertbild verloren gehen. Eine Erweiterung der Kostenfunktion auf Farbbil-
der und deren zusätzliche Information erscheint deshalb sinnvoll. Ein einfacher
Ansatz hierzu wäre die Verwendung der in [1] vorgestellten Kostenfunktion auf
den jeweils einzelnen Kanälen des Bildes. Die Berechnung dieser Kostenfunktion
ist jedoch relativ zeitaufwändig, für die Berechnung auf mehreren Farbkanälen
sollte deshalb ein anderer Ansatz gewählt werden. Ein alternativer Ansatz ist die
Verwendung des Canny-Operators [5] auf allen drei Kanälen eines Bildes. Die
Ergebnisse des Canny-Operators auf den jeweiligen Kanälen werden in einem
anschließenden Schritt zusammengefasst, wobei die Resultate dabei mit empi-
risch ermittelten Werten unterschiedlich gewichtet werden. Die Vorteile dieses
Ansatzes sind in der kurzen Berechnungszeit und in den guten Ergebnissen bei
der Kantedetektion zu sehen. Weiterhin ergibt sich durch die Resultate, die der
Canny-Operator liefert, eine Beschleunigung bei der Graphensuche, da es sich
bei dem Ergebniss der Kostenfunktion nur noch um ein Binärbild handelt. Die
optimalen Pfade sind dadurch auf einfache Weise und mit geringem Zeitaufwand
zu berechnen.

4.2 F*-Graphensuche

Im Bereich der Graphensuche liefert der Algorithmus von Dijkstra gute Ergeb-
nisse, jedoch treten Rechenzeit- und Speicherplatzprobleme auf, wenn sich die
Auflösung der Bilddaten deutlich erhöht. Um dieses Problem zu beseitigen wur-
de der F*-Algorithmus [3, 4] als Alternative zum Dijkstra-Algorithmus [2] un-
tersucht. Algorithmisch unterscheidet sich dieser deutlich von Dijkstra's Ansatz.
Wie der Algorithmus von Dijkstra benötigt der F*-Algorithmus die Angabe einer
Kostenfunktion, um die Kosten für die Kanten des Graphen zu bestimmen. Auch
die Initialisierung des Pfad-Feldes verläuft analog zum Dijkstra-Algorithmus.
Die anschließenden Berechnungen auf dem Pfad-Feld unterscheiden sich jedoch
vollständig. Den F*-Ansatz kann man sich als eine Art Distanztransformati-
on auf dem Pfad-Feld vorstellen, bei dem die Werte innerhalb des Pfad-Feldes
sukzessive aktualisiert werden. Hierbei sind die verwendeten Masken, sowie die
Reihenfolge bei der Aktualisierung entscheidend. Das Pfad-Feld wird in mehre-
ren Richtungen und mit verschiedenen Masken durchlaufen, die Werte innerhalb
des Feldes werden dabei stets neu berechnet. Der Algorithmus terminiert, sobald
sich in einer Iteration keine Änderung mehr innerhalb des Pfad-Feldes ergibt. Als
Resultat erhält man ein Pfad-Feld, das die optimalen Pfade zu einem Saatpunkt
enthält. Um den optimalen Pfad von einer bestimmten Position zum Saatpunkt
zu erhalten, wird wie beim Dijkstra-Algorithmus ein Backtracking-Verfahren
verwendet.

Live-Wire Segmentierung für hochaufgelöste Farbbilder 159

Abb. 1. Berechnungszeiten für die Kostenfunktionen.

Canny-Operator als Kostenfunktion
Original Kostenfunktion [1]
Zeit in s
Bild

5 Ergebnisse

Die beschriebenen Erweiterungen des Live-Wire Verfahrens wurden auf der Basis
von 36 Referenzbildern aus unterschiedlichen Bildmodalitäten (CT, Endoskopie,
Mikroskopie, ...) und Bildinhalten durch eine Gruppe von zehn Experten evalu-
iert. Jedes Bild wurde von jedem Experten mit dem orginal Live-Wire Verfahren
sowie mit elf Varianten der modifizierten Version interaktiv segmentiert. Zudem
wurde eine Evaluation an synthetisch erzeugten Bildern durchgeführt. Dabei
wurde das Verhalten des modifizierten Live-Wire Verfahrens im Hinblick auf
Rauschen untersucht. Hierzu wurden die synthetisch erzeugten Bilder mit unter-
schiedlich starkem „Salt´n-Pepper" Rauschen versehen, der Verwendung dieses
Rauschens kommen hierbei jedoch keine speziellen Bedeutungen zu. Beim Ver-
gleich der Verfahren lagen die Schwerpunkte bei der Evaluation auf der Segmen-
tierungsgeschwindigkeit, sowie auf der Segmentierungsgenauigkeit. Die Evalua-
tion der Segmentierungsgenauigkeit wurde zum einen in Bezug auf Freihandseg-
mentierungen, zum anderen in Bezug zu Referenzsegmentierungen durchgeführt.
Es wurden dabei mehrere Metriken herangezogen um die Segmentierungsergeb-
nisse zu vergleichen. Hierzu zählten Dice-, Tanimoto-, Kappa- und Randpixel-
Koeffizient, sowie Fehlerwahrscheinlichkeit, Konsistenzwerte, Hausdorff-Distanz,
CCH-Wert, CCC-Wert und Vektor-Distanz. Durch die Verwendung der beschrie-
benen Erweiterungen wurden sowohl bei der Segmentierungsgeschwindigkeit als

Figure 21: In this figure we have all three steps of separation of a book image together. In the first step we find a point near to the spine by intersecting the diagonals from the corner of the book. In the second image we cut out a neighborhood around the intersection point.The third picture shows the longest vertical line returned from the RAST algorithm

which separation gives unexpected results. These cases usually include spines where the middle point is far more than 50 pixels away from the spine. In these cases we have usually a thick book in which you try to separate pages near to the cover or near to the last page of the book. You can see some examples for these kind of book images and their separation results in Figure 23.

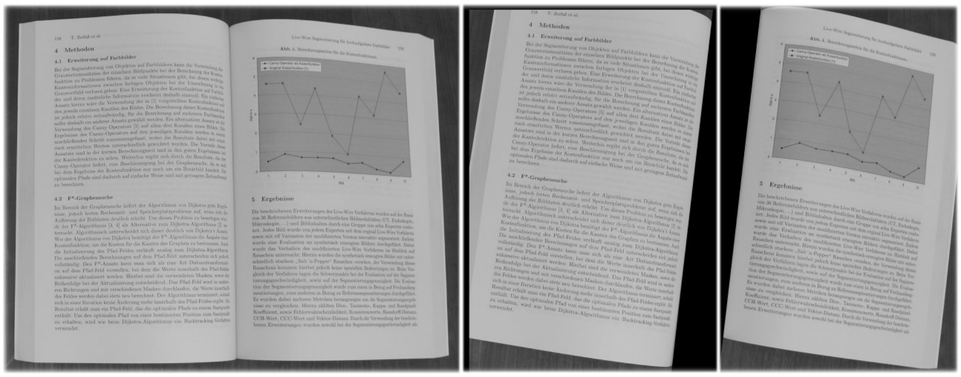


Figure 22: This picture shows the result of the page separation of the book image in Figure 21.

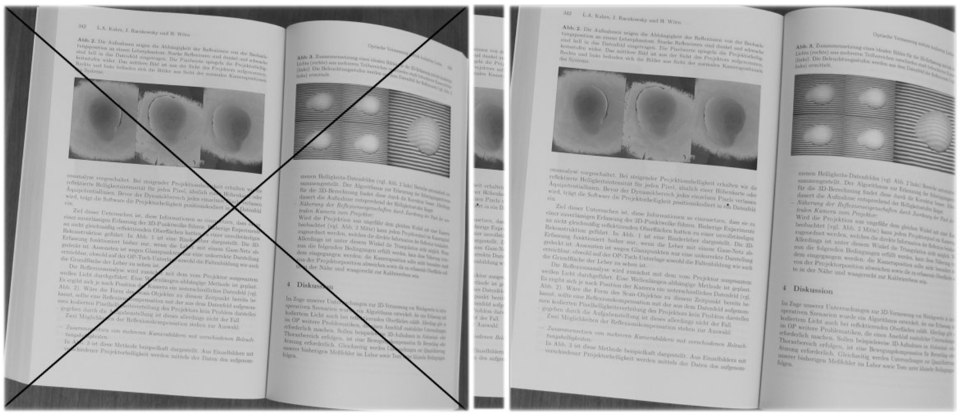


Figure 23: This picture shows a case in which our separation algorithm fails. The reason is the intersection point of the diagonals is not very near to the spine thus the image separated from the original image does not include spine in it.

5 De-warp

Digital Image De-warping is the technique that deals with the geometric transformations on an image. Interests on this topic dates back to 1960s. Since that time it has found its use in different areas of image processing from medical imaging to computer vision. This technique becomes popular as our computing power becomes bigger and thus making it possible to apply this to images in a considerable time. This also leads to its use in document capturing area thus letting us to create planar documents using a basic digital camera instead of a flat bed scanner. In this section we are going to discuss on a de-warping method that will be used to recover book images using a stereo system. This method is actually proposed in [3]. We have only implemented the same method to solve a different problem than in [3]. In [3] a structured light device is used to build $3D$ model whereas we have used a stere imaging system. The approach is used to solve crumbled pages whereas we try to use the same method on book images.

We first give a brief definition of the de-warping problem and then every detail will be cleared in the next subsections.

Problem:

Given a set of $3D$ points Ω_{3D} modeling the surface of a book and a triangular mesh for these points say Θ, we try to find a mapping f that maps these $3D$ points set Ω_{3D} to a $2D$ points set Ω_{2D} preserving all the angles of the triangles in the triangular mesh Θ.

In mathematical terms we can describe the problem as:

$\Omega_{3D} = p_1, ..., p_n$ where $p = (x, y, z)$

$\Theta = \triangle_1, ..., \triangle_m$ where $\triangle = (p_a, p_b, p_c)$ are the corners of each triangle and $(p_a, p_b, p_c) \in \Omega_{3D}$

$\Omega_{2D} = T_1, ..., T_n$ where $T = (x', y')$

The mapping f is $f(p) = T$ where f is a mapping from $\mathbb{R}^3 \to \mathbb{R}^2$ preserving all inner angles of the triangles $\triangle_1, ...\triangle_m \in \Theta$.

Before explaining de-warping method in detail we first want to emphasize some of the theoretical background needed for the method. The topics include conformal mapping and Delaunay triangulation. These are only brief explanations however references are given for the avid reader to learn more about the topic.

5.1 Conformal Mapping

A conformal mapping is a transformation $w = f(z)$ that preserves angles.

Definition:

Given a set of points ρ and let z be a point in ρ, if our mapping f preserves the angles between all curves passing through z then f is conformal on z. If this property holds $\forall z \in \rho$ our mapping is conformal in ρ

There are two important properties of a conformal mapping $w = f(z)$ where z is a complex number($z = x + iy$).

- It is an analytical function which has a nonzero derivative for all the points z in ρ. This meaning $\frac{\partial f}{\partial z} \neq 0$
- Conformal mappings have to satisfy both Laplacian equation and Cauchy Riemann equations.

  Laplace Equation

  $$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

  Cauchy-Riemann Equations

  $$\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y} = 0$$

This property actually helps us since the idea of our de-warping method is to construct an angle preserving mapping. For detailed information on conformal mapping please refer to [3]

5.2 Delaunay Triangulation

As described in the problem we need a triangular mesh for $3D$ point set Ω_{3D}. To achieve this task we use the well known Delaunay triangulation method. This method dates back to the paper written by Boris Delaunay in 1934.[12]

The approach to the triangulation is that we have no points X on a plane that lies in the circumcircle of any of the triangles in our triangles set Θ. Thus leading to a maximization of the minimum angle of all the angles of triangles in our solution set. One of the main results of this feature is that we have no sliver triangle in our solution set. A sliver triangle has the property that the difference between the angles of the triangle are very large. (look figure 24) For a Delaunay triangulation to be unique it has to hold the following properties for an n-dimensional space.

- No n+1 points should lie on the same **hyperplane**.
- No n+2 points should lie on the same **hypersphere**.

In the case of n being 2, no three points shall lie on the same **line** and no four points shall lie on the same **circumcircle**. For a proof of these two properties please refer to [4]

Since implementation of this triangulation is done using the qhull library we will not describe the algorithm for the triangulation but how we use the tool instead. For making the triangulation we use the points extracted by Harris

[3]http://mathworld.wolfram.com/ConformalMapping.html
[4]http://en.wikipedia.org/wiki/Delaunay_triangulation

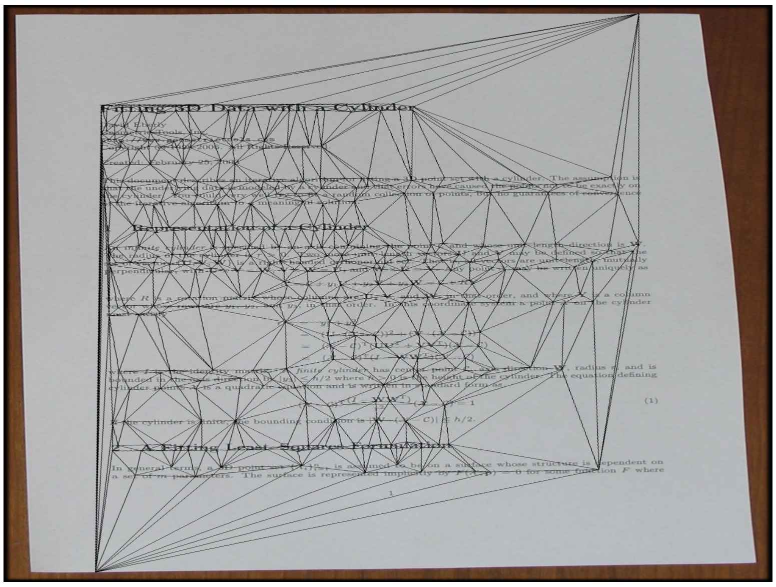


Figure 24: Example of a triangulation done on the image

Feature extraction method [5]. Using these points we can build triangles and these triangles will be used to cover the $3D$ model. For more information on qhull and its implementations please refer to [5] or [13].

5.3 De-warping Method

After making a brief introduction of theoretical background we now want to emphasize on method to de-warp images. As already explained the method we use for de-warping is proposed in [3]. In this method the first step is to create a three dimensional model of the given document(explained in detail in Section 2 and 3). After this step we have to find a suitable triangulation for the given three-dimensional model points. For this purpose we are using a Delaunay triangulation method explained in 5.2. The de-warping method is based on a conformal mapping (explained in Section 5.1) between the generated $3D$ points to $2D$ points as given in the problem explanation. One of

[5]http://www.qhull.org

the most important property of this mapping is that it does not preserve the lengths and areas of the triangles in the triangular mesh so after the solution there should be a resizing of given parameters. This resizing however does not guarantee preserving the length and area. It is only an approximation to the size of the document by finding its size on original image and multiplying this value with the resulting $2D$ points.

The reason that a conformal mapping should preserve angles in case of a book surface is that it is an applicable surface and that applicable surfaces have Gaussian Curvature equal to zero. This means this surface can be mapped to a plane without loss of any data. By taking this property into account we hoped to get uncurled versions of our source images.

5.4 Method

Given our de-warping problem as in Section 5 to find conformal mapping first consider we have represented the set of $3D$ points (Ω_{3D}) by a vector $\kappa(u, v)$, parametrized by u and v, with components or x, y, z. This is a $\mathbb{R}^3 \longrightarrow \mathbb{R}^2$ mapping and conformal on $u - v$ plane iff:

$$\nabla^2 \kappa = \frac{\partial^2 \kappa}{\partial u^2} + \frac{\partial^2 r}{\partial v^2} = 0$$

where ∇^2 is the Laplacian operator on κ.

Since our input is a set of $3D$ points what we actually seek is an inverse mapping from $(x, y, z) \longrightarrow (u, v)$. Now consider another function say $g(x, y)$ on $2D$. This function returns a $2D$ point (u, v). In $2D$ case we know that a conformal mapping has to satisfy the Cauchy-Riemann equation (look at Section 5.1) so:

$$\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y} = 0$$

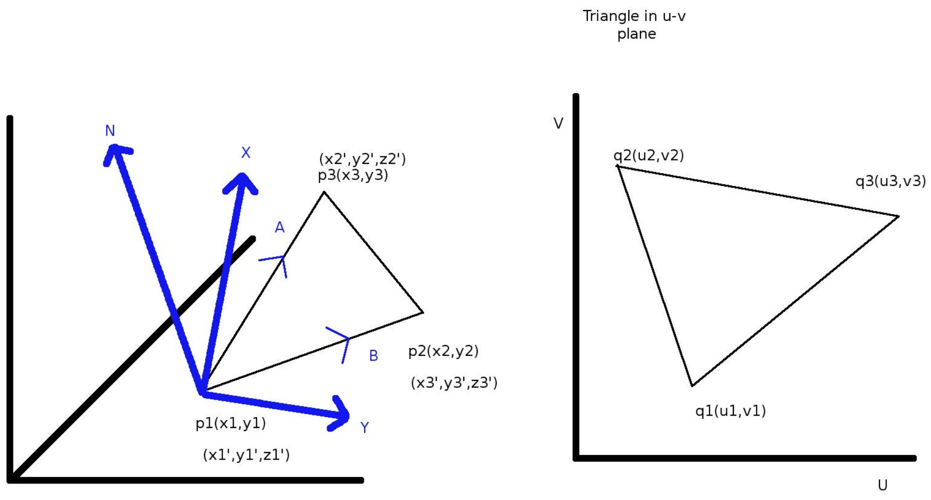


Figure 25: This figure shows the parametrization done from $mathbbR^3 \rightarrow \mathbb{R}^2$. All the vectors N, X, Y, A and B can be visualized. The corresponding triangle in $u - v$ plane is also showed.

from this:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \text{ and } \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

This means we can now reduce the $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ mapping to a $\mathbb{R}^2 \rightarrow \mathbb{R}^2$. The only problem left is to find a way to parametrize the $3D$ triangles in $u - v$ plane.

Parametrization:

As can be seen in Figure 25 A and B are two sides of a triangle($\triangle_a$) in Θ. We need to represent a local plane ς for $\triangle_a$ and for that reason we need to find a normal(N) to plane ς and a point (P_0) on ς so that we can build the equation for ς:

$$N.(P - P_0) = 0$$

To find a normal to ς we already have two vectors that lie on ς,A and B. By applying a cross production on them and dividing the resulting vector with its length we find a nor mal to ς, say N.

$$N = \frac{AxB}{|AxB|}$$

After finding the normal to ς we need to change the parametrize the $3D$ points to their values in $u - v$ plane. To achieve this we take one of the corners in $\triangle_a$ as origin and then find the unit vectors in both u and v dimensions. We name these unit vectors as X in u direction and Y in v direction. These are calculated as:

$$X = \frac{NxB}{|NxB|}$$

$$Y = \frac{XxB}{|XxB|}$$

After finding unit vectors X and Y on ς it is now very easy to parametrize the $3D$ points to $2D$. Each point in $2D$ system is represented as $p = x', y'$. All these points build up a new mesh $M = p_{i\in[1n],\triangle_{j\in[1m]}}$ where p_i are parametrized $2D$ points and $\triangle_j$ represents the new triangles resulting after parametrization. For this new parametrization we define two new sets.These are Ω_{2Dpar} with n elements and Θ_{2DPar} with m elements. The parametrization done on each $T_j \in \Theta_{2DPar}$ with $\triangle_j = C_{j1}, C_{j2}, C_{j3}$ where C_{ji} is a vertex in $\triangle_j$ can be defined as:

$$C_1 = (x'_1, y'_1) = (0, 0)$$

$$C_2 = (x'_2, y'_2) = (B.X, B.Y)$$

$$C_3 = (x'_3, y'_3) = (A.X, A.Y)$$

Now we can define a mapping between Ω_{2Dpar} and Ω_{2D}(as explained in Section 5). An element of Ω_{2D} is q and it is parametrized as $q = (u, v)$. So our mapping will be from $p(x', y') \rightarrow q = (u, v)$. For notation simplicity we take $p = (x, y)$ from now on. As explained in [3] a triangle to triangle mapping has a unique affine transformation between original and destination triangle. An affine mapping $g(p) \rightarrow q$ with $p = (x, y) \in \Omega_{2Dpar}$ and $q = (u, v) \in$ with each triangle $\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3) \in \Theta_{2DPar}$ described by its vertices:

$$\begin{aligned} f(x,y) = & \frac{AREA(\triangle(x,y)(x_2,y_2)(x_3,y_3))(u_1,v_1)}{AREA(\triangle(x_1,y_1)(x_2,y_2)(x_3,y_3))} \\ + & \frac{AREA(\triangle(x,y)(x_3,y_3)(x_1,y_1))(u_2,v_2)}{AREA(\triangle(x_1,y_1)(x_2,y_2)(x_3,y_3))} \\ + & \frac{AREA(\triangle(x,y)(x_1,y_1)(x_2,y_2))(u_3,v_3)}{AREA(\triangle(x_1,y_1)(x_2,y_2)(x_3,y_3))} \end{aligned}$$

Before applying the Cauchy-Riemann equations on our affine mapping we want to give the formulation of the **AREA** function. Given $\triangle_1 = ((x_1, y_1), (x_2, y_2), (x_3, y_3))$ we can formulate the area of this triangle as:

$$AREA(\triangle((x_1,y_1),(x_2,y_2),(x_3,y_3))) = \frac{1}{2!}\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

which can be written as:

$$AREA(\triangle((x_1,y_1),(x_2,y_2),(x_3,y_3))) = \frac{1}{2!}(-x_2y_1+x_3y_1+x_1y_2-x_3y_2-x_1y_3+x_2y_3)$$

Now we compute the partial derivatives of our mapping function:

$$\frac{\partial f}{\partial x} = \frac{(u_1,v_1)(y_2-y_3)+(u_2,v_2)(y_3-y_1)+(u_3,v_3)(y_1-y_2)}{2AREA(\triangle(x_1,y_1)(x_2,y_2)(x_3,y_3))}$$

and

$$\frac{\partial f}{\partial y} = \frac{(u_1, v_1)(x_2 - x_3) + (u_2, v_2)(x_3 - x_1) + (u_3, v_3)(x_1 - x_2)}{2AREA(\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3))}$$

Now we define the Cauchy-Riemann equations for the mapping obtaining a linear Equation system. For u we have:

$$\frac{\partial u}{\partial x} = \frac{(y_2 - y_3) + (y_3 - y_1) + (y_1 - y_2)}{2AREA(\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3))}$$

$$\frac{\partial u}{\partial y} = \frac{(x_2 - x_3) + (x_3 - x_1) + (x_1 - x_2)}{2AREA(\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3))}$$

and for v we have:

$$\frac{\partial v}{\partial x} = \frac{(y_2 - y_3) + (y_3 - y_1) + (y_1 - y_2)}{2AREA(\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3))}$$

$$\frac{\partial v}{\partial y} = \frac{(x_2 - x_3) + (x_3 - x_1) + (x_1 - x_2)}{2AREA(\triangle(x_1, y_1)(x_2, y_2)(x_3, y_3))}$$

Before constructing the equation system we define for triangle $\triangle((x_1, y_1), (x_2, y_2), (x_3, y_3))$ $\Delta c_1 = (y_2 - y_3), \Delta c_2 = (y_3 - y_1), \Delta c_3 = (y_1 - y_2), \Delta b_1 = (x_2 - x_3), \Delta b_2 = (x_3 - x_1), \Delta b_3 = (x_1 - x_2)$

Now we can write our equation system for one triangle as:

$$\frac{1}{2A_T} \begin{bmatrix} \Delta c_1 & \Delta c_2 & \Delta c_3 & -\Delta b_1 & -\Delta b_2 & -\Delta b_3 \\ \Delta b_1 & \Delta b_2 & \Delta b_3 & \Delta c_1 & \Delta c_2 & \Delta c_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The above equation is valid for one triangle but we need to solve it for all triangles in Θ_{2DPar}. This in return should give us the values of all $q = (u, v)$ in Ω_{2D}. This can be written as an equation system like $AQ = 0$ where $Q = \begin{bmatrix} U_i & V_i \end{bmatrix}$.Here U_i stands for all u_i in Ω_{2D} and V_i for all v_i in Ω_{2D}. Here again Δc_i and Δb_i defined as above.

$$X_{ij} = \begin{cases} \frac{\Delta b_i}{2A_{T_j}} & \text{if it is a corner point;} \\ 0 & \text{otherwise.} \end{cases}$$

$$Y_{ij} = \begin{cases} \frac{\Delta c_i}{2A_{T_j}} & \text{if it is a corner point;} \\ 0 & \text{otherwise.} \end{cases}$$

Now writing the A matrix for all triangles in Θ_{2DPar} what we have is:

$$\begin{bmatrix} Y_{ij} & -X_{i(N+j)} \\ X_{(M+i)j} & Y_{(M+i)(N+j)} \end{bmatrix} \begin{bmatrix} U_i \\ V_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The above equation system is an underdetermined one meaning there are more equations that the number of unknowns. This causes us to have infinitely many solutions with different orientation and size for the Ω_{2D}. We have to pick a solution between these. To find a unique solution, we have to set some of the points in Q to predefined constants. This makes sure we pick a unique orientation for our solution set Ω_{2D}. In our case fixing the values of two constraints will give us a unique solution. We ,rather for calculation purposes, pick to set the values of the first two entries in Ω_{2D} to (0,0) and

(1,1). This constraints cause our equation system to change and become $A_{new}Q_{new} = Result$ where:

$$Result = -\begin{bmatrix} X_l & X_m & X_{N+l} & X_{N+m} \end{bmatrix} \begin{bmatrix} U_l \\ U_m \\ V_{N+l} \\ V_{N+m} \end{bmatrix}$$

and A_{new} with the first two columns and columns at n and $n+1$ removed from A. We also remove the elements in first row and on n^{th} row from Q to get Q_{new}. We are now left to solve the system $A_{new}Q_{new} = Result$. One of the most important features of A_{new} is that it only has 6 nonzero entries per row. This means A_{new} is a sparse matrix. A sparse matrix is a matrix that mainly consists of zeros. This property is very useful in computation of matrices since we can take into account that most of the entries are zero and have no or very little effect on our solution. Other than this there are actually a lot of implementations for linear sparse matrix solvers and we can use one of these for our system.

After a thorough research we decide to use so called **CHOL**esky **MOD**ification toolbox [6](all following papers define the algorithms of CHOLMOD [14][15] [16][17][18]) to solve the equation system. This toolbox uses the so called Cholesky factorization to factor out the matrix. This factorization however factors out a symmetric matrix(which is usually not the case for us) into its lower left matrix and its transpose. To achieve this factorization we must build a symmetric matrix and we do this by multiplying our matrix by its transpose. . We cannot find a Cholesky Factorization for A_{new} since it is not symmetric. However we already know from our basic linear algebra courses

[6]www.cise.ufl.edu/research/sparse/cholmod/

that we can actually obtain a symmetric matrix from a rectangular matrix by multiplying it with its transpose. $(A_{new}^T * A_{new})$ We also know that in a linear system when we multiply left side with a matrix we should multiply the right side also with the same matrix thus causing our system to change from:

$$A_{new}Q = Result$$

to:

$$A_{new}^T A_{new} Q = A_{new}^T Result$$

After doing these modifications our system is now ready to be solved with the toolbox. The Q_{new} contains the values of all $q = u, v \in \Omega 2D$ meaning the de-warped $2D$ points. After finding the values for all elements of Ω_{2D} we only need to build the image which will be explained in the next section.

6 De-skew

This is the process of using the input image to build an uncurled version of the same image. For this purpose we need a mapping between the source points and our de-warped points. The source points are the points extracted by Harris Feature Extraction and for the set Ω_{Harris}, whereas destination points are the de-warpep points and are contained in the set Ω_{2D} (look at Section 5.3). This $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ mapping is actually very easy to find since we already know a mapping between $3D$ points (Ω_{3D}) and de-warped $2D$ points(Ω_{2D})(look at Section 5.3). We also know the relationship between the $3D(\Omega_{3D})$ and source points on original image (Ω_{Harris}) so actually we only need to connect these two correspondences. We have a $3D$ point, say $(x_1, y_1, z1)$, and its corresponding point on the source image (k, l). As explained above we can actually build a mapping from $(k, l) \rightarrow (u, v)$ where $(u, v) \in \Omega_{2D}$. For this purpose we use a triangle to triangle mapping. We can actually do such a trick because we can easily build the corresponding triangular mesh for both Ω_{2D} and Ω_{Harris}. We define the triangular mesh sets as Θ_{2D} for Ω_{2D} and Θ_{Harris} for Ω_{Harris}.
Let $\triangle_{2D}$ be an element of Θ_{2D} with corners,
$(u_1, v_1)(u_2, v_2)(u_3, v_3)$. The corresponding triangle in Θ_{Harris} is $\triangle_{Harris}$ with corners
$(k_1, l_1)(k_2, l_2)(k_3, l_3)$. So our mapping function between these two images look like:

$$\begin{aligned} f(k,l) = & \frac{AREA(\triangle(u,v)(u_2,v_2)(u_3,v_3))(k_1,l_1)}{AREA(\triangle_{2D})} \\ + & \frac{AREA(\triangle(u,v)(u_3,v_3)(u_1,v_1))(k_2,l_2)}{AREA(\triangle_{2D})} \\ + & \frac{AREA(\triangle(u,v)(u_1,v_1)(u_2,v_2))(k_3,l_3)}{AREA(\triangle_{2D})} \end{aligned}$$

where (k, l) is the corresponding pixel of (u, v) in the source image. We solve this equation for all the image points that lie in $\triangle_{2D}$ thus finding their position on the source image. From the source image position by the use of bilinear interpolation (see A.1) we found the corresponding RGB value for (u, v). To find which points lie inside the triangle $\triangle_{2D}$ we use the method explained in Section A.2.

7 Experimental Results

This section addresses experiments that were done on our system. Since there are different parts in our system, we have tested each of them separately. Each experiment is geared towards understanding how to improve the performance of our system. We have divided the testing into three different parts. The first part will show some results of our experiments on our stereo matching approach. The second part will be a comparison between the two different smoothing approaches. The third and last section will show the results of our de-warping method. In each section, we first define our experiments and our data sets. After this, we will explain and discuss the results.

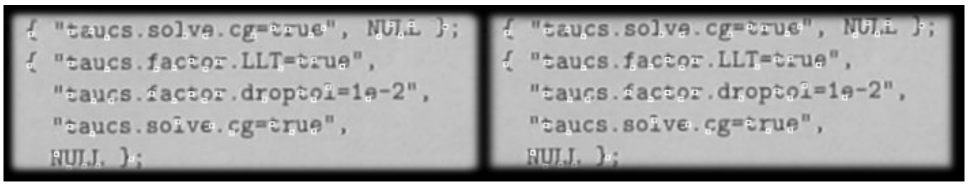
7.1 Experiments on Stereo Matching

As explained in Section 2, matching is the process of finding correspondences between two images of the same object. We have already given a brief idea about our approach to solve matching problem in section 2. The experiments are divided into two parts. First we experimented on planar documents, and then continue our experiments on book images.

7.1.1 Experiments on Planar Documents

These experiments are designed to prove that our approach is working well under the basic case in which we try to match two images that has very little skew or disorientation between them(see Figure 28). Since there is no or very little difference in terms of the contrast, size and shape between two images, we are expecting almost perfect matching. The only problem that can occur in this case is for blocks that are very similar to be of equal distance from our starting point (Look at figure 13). First we represent some of the results from small parts of the images. As can be seen from these images(Figure 26), all matches are perfect and as expected, our algorithm works very well. However, if we look at another part of the same image where we have similar blocks that actually lie very near to each other, our algorithm can not differ between these two(see Figure 13). For solving this problem, we try to impose larger block sizes and as can be seen from Figure 13 we have succeeded in solving this problem. However this very simple solution causes us to lose a lot of time since this means we have a higher computation time for our similarity measures. This solution can cause mismatches in images with a higher skew since larger blocks can have different contents on the correspondences.

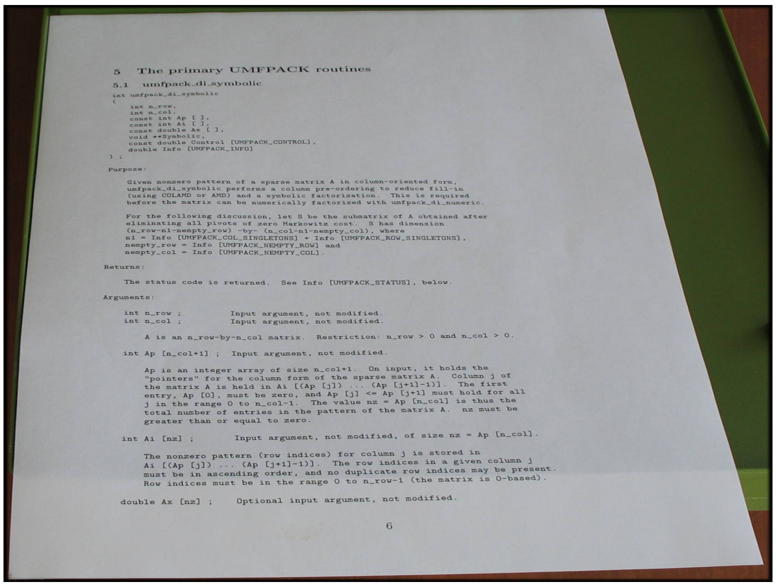
Other than this, as can be seen in Figure 17 number of outliers in the whole image is very low and can be easily smoothed out by using one of our smoothing algorithms. This test proves that with a good initial guess, we can actually have an almost perfect $3D$ model of a plane paper using this method. However, I have to mention that since our initial starting point is just a guess based on the median displacement vector (look at Section 2.3.1), we have to take care of the variance of the displacement vectors. This means, in the case of a large variance for our displacement vectors, we have to take into account that our initial guess could be somewhere away from the exact point and in that case we have to add more prediction points so that our algorithm can find the exact match . However this can also be problematic since we can again have the same problem with similar blocks. This can mean a higher rate of mismatches.(see Figure 27).



```
{ "taucs.solve.cg=true", NULL };
{ "taucs.factor.LLT=true",
  "taucs.factor.droptol=1e-2",
  "taucs.solve.cg=true",
  NULL };
```

```
{ "taucs.solve.cg=true", NULL };
{ "taucs.factor.LLT=true",
  "taucs.factor.droptol=1e-2",
  "taucs.solve.cg=true",
  NULL };
```

Figure 26: Parts of a planar document with perfect matching between two images



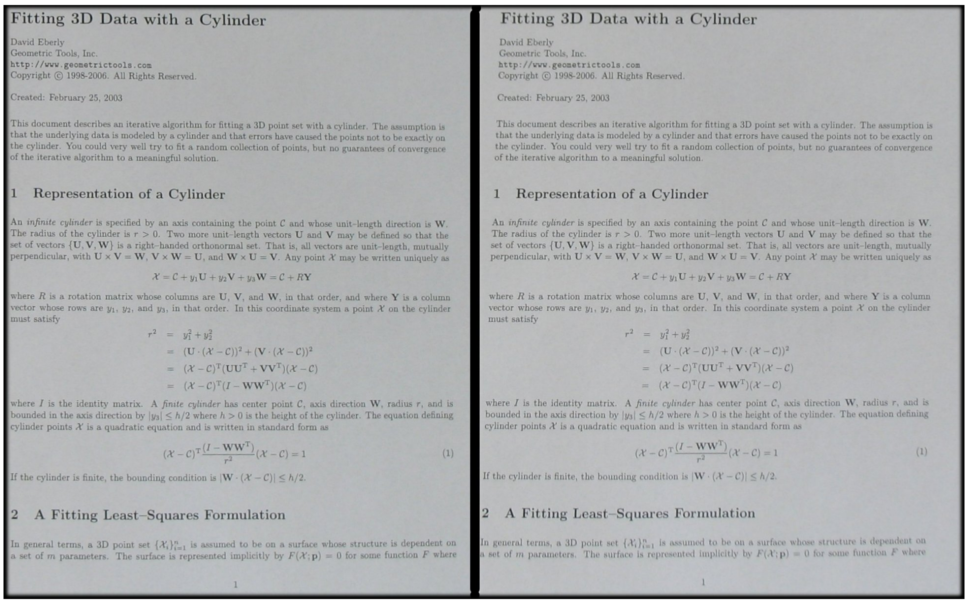
5 The primary UMFPACK routines

5.1 umfpack_di_symbolic

```
int umfpack_di_symbolic
(
    int n_row,
    int n_col,
    const int Ap [ ],
    const int Ai [ ],
    const double Ax [ ],
    void **Symbolic,
    const double Control [UMFPACK_CONTROL],
    double Info [UMFPACK_INFO]
) ;

Purpose:

    Given nonzero pattern of a sparse matrix A in column-oriented form,
    umfpack_di_symbolic performs a column pre-ordering to reduce fill-in
    (using COLAMD or AMD) and a symbolic factorization.  This is required
    before the matrix can be numerically factorized with umfpack_di_numeric.

    For the following discussion, let S be the submatrix of A obtained after
    eliminating all pivots of zero Markowitz cost.  S has dimension
    (n_row-n1-nempty_row) -by- (n_col-n1-nempty_col), where
    n1 = Info [UMFPACK_COL_SINGLETONS] + Info [UMFPACK_ROW_SINGLETONS],
    nempty_row = Info [UMFPACK_NEMPTY_ROW] and
    nempty_col = Info [UMFPACK_NEMPTY_COL].

Returns:

    The status code is returned.  See Info [UMFPACK_STATUS], below.

Arguments:

    int n_row ;          Input argument, not modified.
    int n_col ;          Input argument, not modified.

        A is an n_row-by-n_col matrix.  Restriction: n_row > 0 and n_col > 0.

    int Ap [n_col+1] ;   Input argument, not modified.

        Ap is an integer array of size n_col+1.  On input, it holds the
        "pointers" for the column form of the sparse matrix A.  Column j of
        the matrix A is held in Ai [(Ap [j]) ... (Ap [j+1]-1)].  The first
        entry, Ap [0], must be zero, and Ap [j] <= Ap [j+1] must hold for all
        j in the range 0 to n_col-1.  The value nz = Ap [n_col] is thus the
        total number of entries in the pattern of the matrix A.  nz must be
        greater than or equal to zero.

    int Ai [nz] ;        Input argument, not modified, of size nz = Ap [n_col].

        The nonzero pattern (row indices) for column j is stored in
        Ai [(Ap [j]) ... (Ap [j+1]-1)].  The row indices in a given column j
        must be in ascending order, and no duplicate row indices may be present.
        Row indices must be in the range 0 to n_row-1 (the matrix is 0-based).

    double Ax [nz] ;     Optional input argument, not modified.
```

6

Figure 27: Images of a planar document that has a high depth change and skew



Fitting 3D Data with a Cylinder

David Eberly
Geometric Tools, Inc.
http://www.geometrictools.com
Copyright © 1998-2006. All Rights Reserved.

Created: February 25, 2003

This document describes an iterative algorithm for fitting a 3D point set with a cylinder. The assumption is that the underlying data is modeled by a cylinder and that errors have caused the points not to be exactly on the cylinder. You could very well try to fit a random collection of points, but no guarantees of convergence of the iterative algorithm to a meaningful solution.

1 Representation of a Cylinder

An *infinite cylinder* is specified by an axis containing the point $\mathcal{C}$ and whose unit-length direction is $\mathbf{W}$. The radius of the cylinder is $r > 0$. Two more unit-length vectors $\mathbf{U}$ and $\mathbf{V}$ may be defined so that the set of vectors $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ is a right-handed orthonormal set. That is, all vectors are unit-length, mutually perpendicular, with $\mathbf{U} \times \mathbf{V} = \mathbf{W}$, $\mathbf{V} \times \mathbf{W} = \mathbf{U}$, and $\mathbf{W} \times \mathbf{U} = \mathbf{V}$. Any point $\mathcal{X}$ may be written uniquely as

$$\mathcal{X} = \mathcal{C} + y_1\mathbf{U} + y_2\mathbf{V} + y_3\mathbf{W} = \mathcal{C} + R\mathbf{Y}$$

where R is a rotation matrix whose columns are $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, in that order, and where $\mathbf{Y}$ is a column vector whose rows are y_1, y_2, and y_3, in that order. In this coordinate system a point $\mathcal{X}$ on the cylinder must satisfy

$$\begin{aligned} r^2 &= y_1^2 + y_2^2 \\ &= (\mathbf{U}\cdot(\mathcal{X}-\mathcal{C}))^2 + (\mathbf{V}\cdot(\mathcal{X}-\mathcal{C}))^2 \\ &= (\mathcal{X}-\mathcal{C})^{\mathrm{T}}(\mathbf{U}\mathbf{U}^{\mathrm{T}} + \mathbf{V}\mathbf{V}^{\mathrm{T}})(\mathcal{X}-\mathcal{C}) \\ &= (\mathcal{X}-\mathcal{C})^{\mathrm{T}}(I - \mathbf{W}\mathbf{W}^{\mathrm{T}})(\mathcal{X}-\mathcal{C}) \end{aligned}$$

where I is the identity matrix. A *finite cylinder* has center point $\mathcal{C}$, axis direction $\mathbf{W}$, radius r, and is bounded in the axis direction by $|y_3| \leq h/2$ where $h > 0$ is the height of the cylinder. The equation defining cylinder points $\mathcal{X}$ is a quadratic equation and is written in standard form as

$$(\mathcal{X}-\mathcal{C})^{\mathrm{T}}\frac{(I - \mathbf{W}\mathbf{W}^{\mathrm{T}})}{r^2}(\mathcal{X}-\mathcal{C}) = 1 \qquad (1)$$

If the cylinder is finite, the bounding condition is $|\mathbf{W}\cdot(\mathcal{X}-\mathcal{C})| \leq h/2$.

2 A Fitting Least–Squares Formulation

In general terms, a 3D point set $\{\mathcal{X}_i\}_{i=1}^{n}$ is assumed to be on a surface whose structure is dependent on a set of m parameters. The surface is represented implicitly by $F(\mathcal{X};\mathbf{p}) = 0$ for some function F where

1

Figure 28: Images of a planar document. There is a very low difference between two images

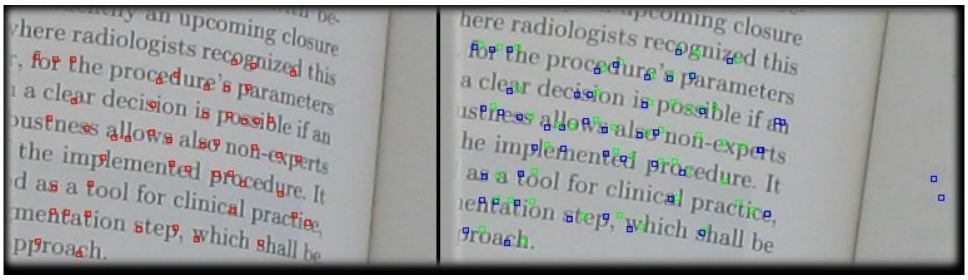

here radiologists recognized this
, for the procedure's parameters
a clear decision is possible if an
usiness allows also non-experts
the implemented procedure. It
d as a tool for clinical practice,
mentation step, which shall be
pproach.

Figure 29: This figure shows a part of book image near the spine of the book.Green points show the exact points of the feature points whereas the blue points show the matches.

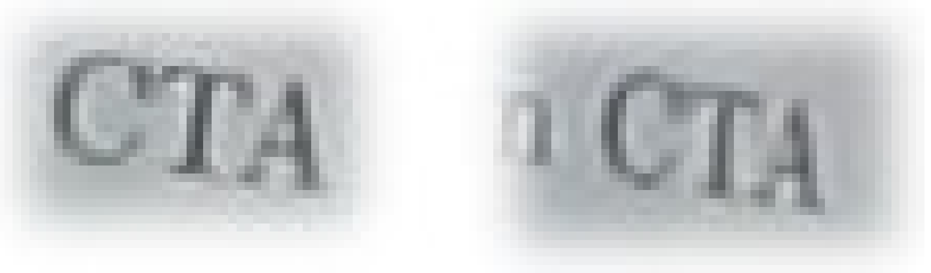


Figure 30: This figure shows blocks taken from the image near to the spine of the book. We can observe the difference in size and orientation.

7.1.2 Experiments on Spine of a Book

This section covers the experiments made on curled book images. These experiments are handled to improve the quality of the $3D$ model of our book surface. We are interested especially in the spine of the book since it contains the most curled data and also is very hard to model with naive similarity measure methods.Problems special to this area are as explained in section 2:

1. There are not enough features to model this area very well.
2. The difference between two stereo images in this particular area of the book image is very high in contrast,size and shape.

After making these observations we first increased the number of extracted points on the image to have more feature points near the spine of the book. This however did not improve our results as much as expected since new points found by Harris extraction method also tend to be in the middle of the book image rather than near the spine. We then focus our interest on finding matches for the points we have near the spine. For this purpose we first cut out small parts from the book image and try to reach to an optimum value for block size and number of predictors for our matching algorithm.After experimenting heavily on these values we have in the end found out these experiments are futile as can be seen from Figure 29 the matches tend to vanish as feature points approach to spine. The problem is that with a fixed block size the contents on one of the pictures differs too much that our algorithm can not handle it(see Figure 30). After that we consider using a new algorithm based on making an affine transformation to the source block so that it can find a match on the other picture 2. For this purpose we again take small parts of the book images near the spine of the book (look figure 31).In these pictures, the white crosses correspond to the points to be matched and black points are the furthest prediction points that new algorithm returns perfect matches. We have tested points with different skew levels. Actually the skew level can be seen as their distance to the spine of the book. The nearer the point to the spine the higher the skew level is for that point. Table 1 shows the displacement vectors between the starting points and the perfect match. The results on Table 1 show us that

Table 1: Results of distance vectors for predictors

Image Name	dx	dy
Figure9-a	6	12
Figure9-b	5	10
Figure9-c	6	10
Figure9-d	5	10

this method is not robust and needs very good predictors to work properly. Since our predictors tend to be more than ten pixels away from the original point we do not add this algorithm. Instead we decided to solve the problem in the smoothing module of our software.

7.2 Experiments on Smoothing

This section explains and discuss on experiments to test different smoothing methods introduced in Section 3. The first subsection challenges the experiments on planar document images whereas the second subsection covers the experiment on book images.

7.2.1 Experiments on Planar Documents

We begin our experiments again on planar documents because it is easier to observe the results in this case. We expect our algorithm to move all points in Ω_P(for description see Section 3.This one models the surface of a planar document) to a plane . For this purpose, we took an image of a simple

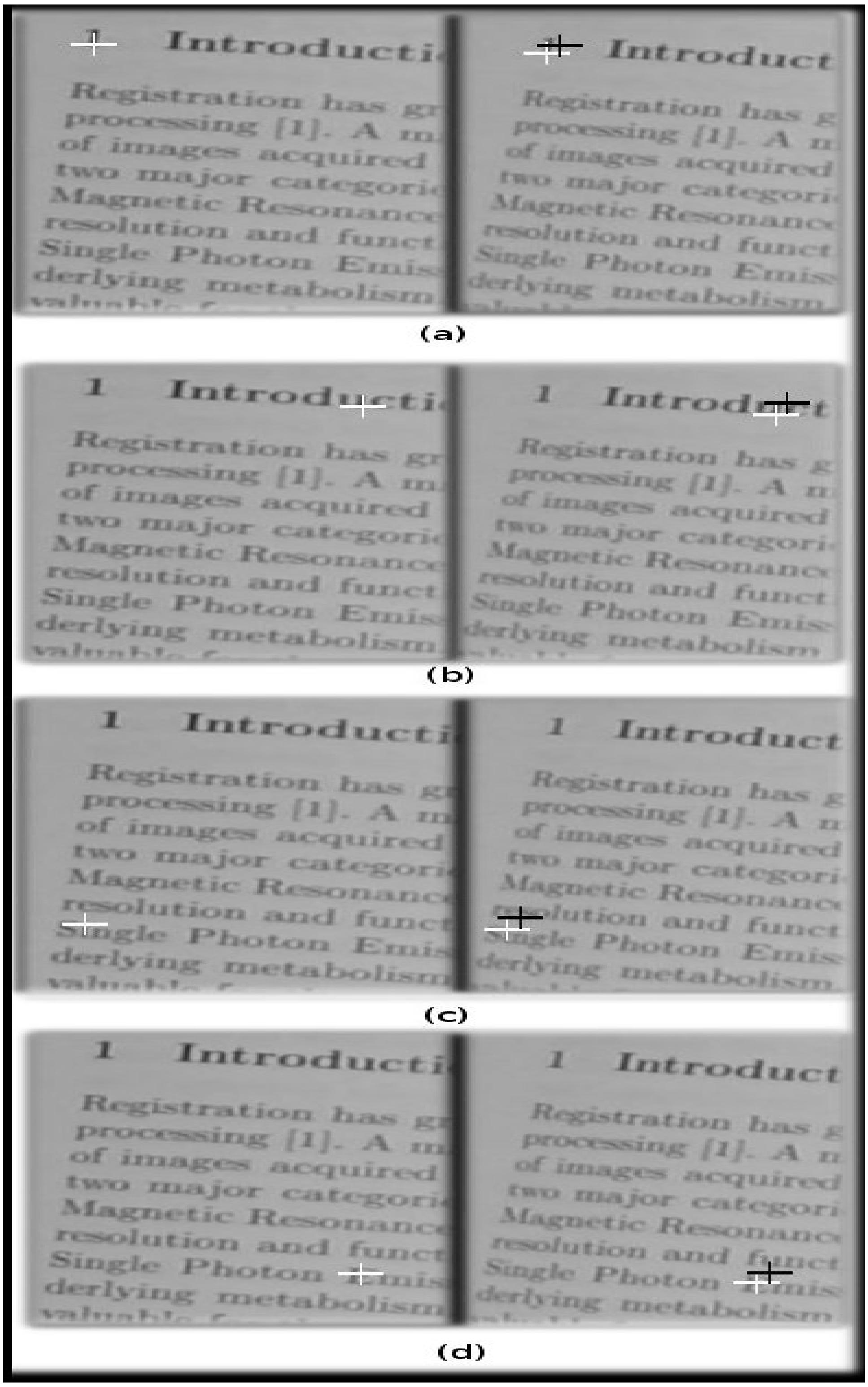


Figure 31: This picture shows the results of our second matching algorithm. As can be seen easily from the pictures, the point to be matched and the initial guess points should be near to each other

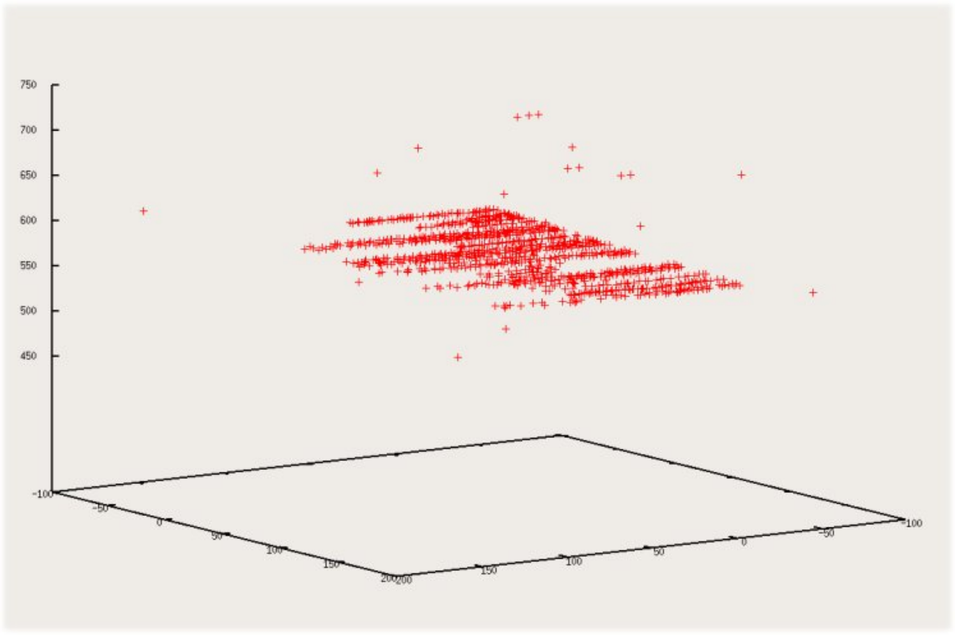


Figure 32: This picture shows an unsmooth model of a planar document

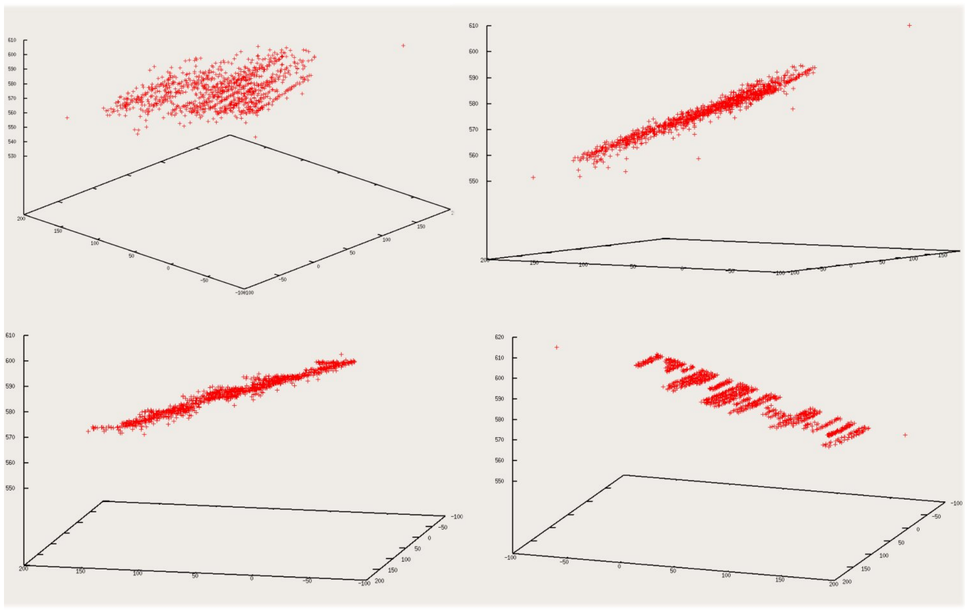


Figure 33: This picture shows results of smoothing(fitting to a plane) for different variable values. In all pictures above we fix $l = 200$ and $\tau = 1mm$. For the left top image $sigma_x = 10$ and $sigma_y = 20$,for the right top $sigma_x = 20$ and $sigma_y = 40$, for left bottom $sigma_x = 40$ and $sigma_y = 80$and last right bottom $sigma_x = 100$ and $sigma_y = 150$

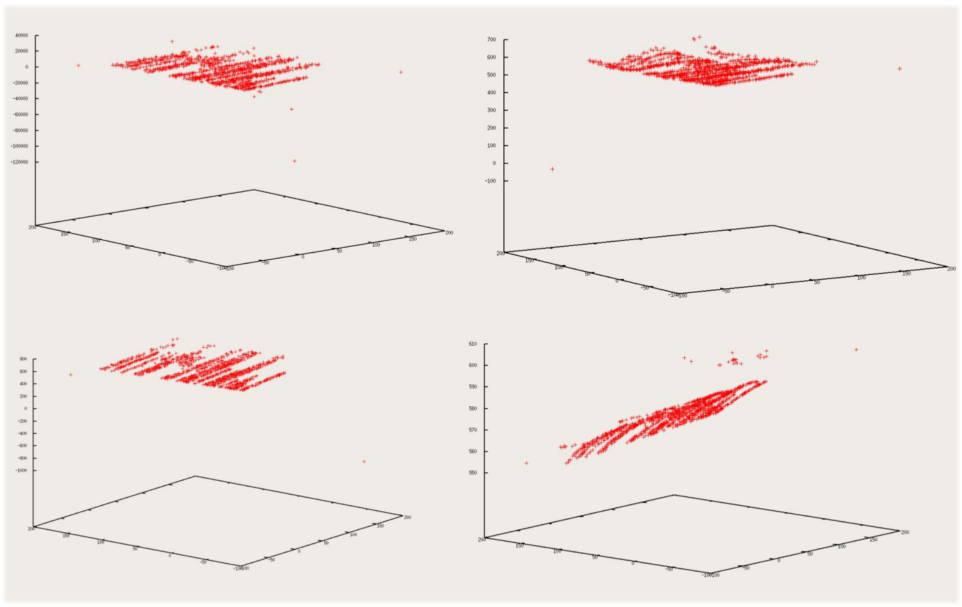


Figure 34: This picture shows results of smoothing (fitting to a parabola) for different variable values. In all pictures above we fix $l = 200$ and $\tau = 1mm$. For the left top image $sigma_x = 10$ and $sigma_y = 20$,for the right top $sigma_x = 20$ and $sigma_y = 40$, for left bottom $sigma_x = 40$ and $sigma_y = 80$and last right bottom $sigma_x = 100$ and $sigma_y = 150$

$A4$ document(look at Figure 28 to see stereo image pair of $A4$ document). After running the matching algorithm in the image pair in Figure 28, we have our input Ω_P. As seen in Figure 32 we have a nearly perfect matching with only a small number of mismatches. The problem now is to tailor out the values of neighborhood sizes (σ_x and σ_y), threshold (τ) and number of iterations (l)(all variables explained in Section 3). We first try to find the values of variables for case of plane fitting. Figure 33 show us the effect of each variable. As expected the most important variable tends to be the sizes of neighborhood (σ_x and σ_y). This directly affect the size of local points set (β) thus in the case of a planar image we see that we need larger sizes to have a perfect smoothing. We can actually use the number of outliers that still exist as a merit of measurement on how good the smoothing is. As we can see from Figure 33 a larger neighborhood($\sigma_x = 100mm$ and $\sigma_y = 150mm$)

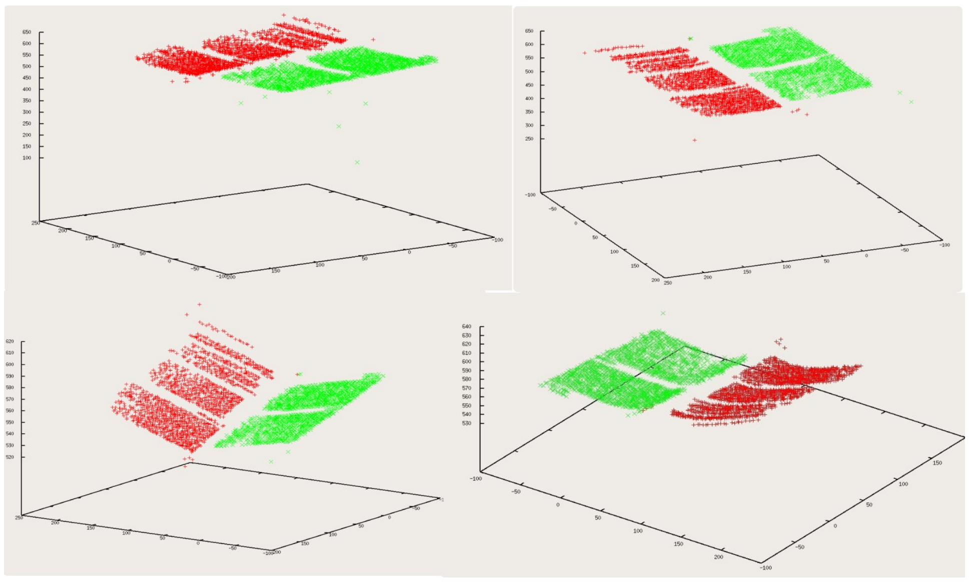


Figure 35: This picture shows results of smoothing(fitting to a plane) for different variable values(for a book image). In all pictures above we fix $l = 200$ and $\tau = 1mm$. For the left top image $sigma_x = 20$ and $sigma_y = 50$,for the right top $sigma_x = 30$ and $sigma_y = 30$, for left bottom $sigma_x = 100$ and $sigma_y = 150$and last right bottom $sigma_x = 35$ and $sigma_y = 50$

size with the number of iterations fixed to 200 and threshold value set to $1mm$ fits the points perfectly. Increasing the neighborhood sizes (σ_x and σ_y) further do not improve results whereas causes a longer computation time. After finishing tests on plane fitting we do the same tests with paraboloid fitting. As we can see from Figure 34 paraboloid fitting results in undesired curls on the model. This even toying with variables can not be solved. So after some experimentation we conclude that for planar images using a plane fitting algorithm with a neighborhood size of $\sigma_x = 100mm$ and $\sigma_y = 200mm$, threshold value set to $1mm$ for 200 times iteration.

7.2.2 Experiments on Book Images

The next issue in smoothing is to smooth a $3D$ point set Ω_B which models the surface of a book. As explained already in Section 7.1, we will

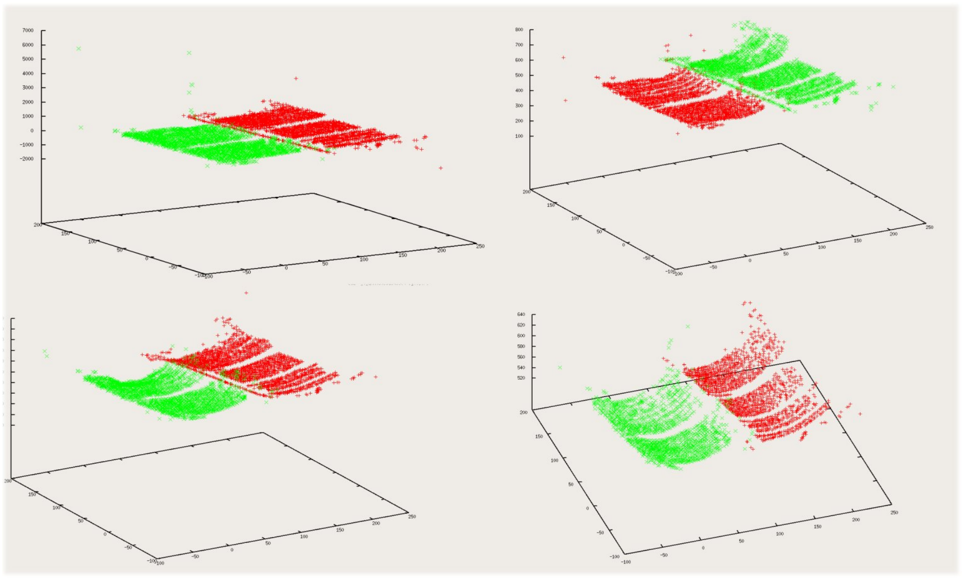


Figure 36: This picture shows results of smoothing (fitting to a paraboloid) for different variable values(for a book image). In all pictures above we fix $l = 200$ and $\tau = 1mm$. For the left top image $sigma_x = 20$ and $sigma_y = 40$,for the right top $sigma_x = 30$ and $sigma_y = 65$, for left bottom $sigma_x = 80$ and $sigma_y = 100$and last right bottom $sigma_x = 100$ and $sigma_y = 150$

try to solve the problem of outliers by using an appropriate smoothing model. For this purpose we first begin our experiments with plane fitting. For the first experiments we decide to use the same set of variables ($\sigma_x = 100mm$ and $\sigma_y = 200mm$, $\tau = 1mm$ and $l = 200$) for Ω_B. As seen in Figure 35 the result is a surface without any curl near the spine of the book. This, of course, proves that we again have to play around with the variables to get a nearly perfect model. We again observe that the actual problem can be reduced to picking the right size of neighborhood. We begin testing different sizes of neighborhoods however this time taking smaller values for σ_x since we observe that by taking smaller neighborhoods on $x - axis$ we preserve the curl near the spine. After some intense experimentation with different values we found out that there really is not an optimum value that gives results with no or very few outliers. However as seen in Figure 35 the results tend to be better

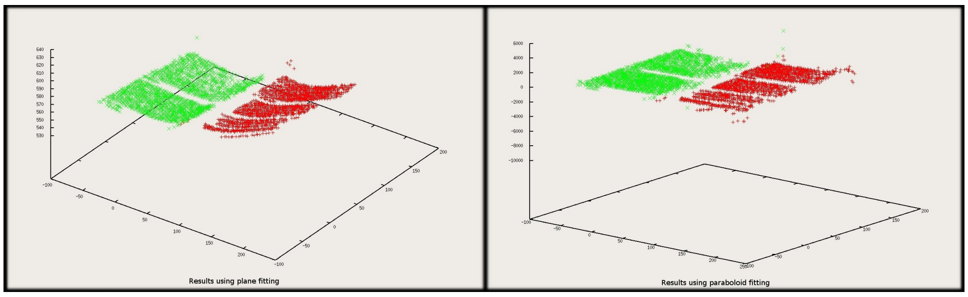


Figure 37: This figure shows the results of both fitting algorithms together in a neighborhood of $35mm - 50mm$ for book image

as $25 \leq \sigma_x \leq 35$ and $55 \leq \sigma_y \leq 65$. The values of l and τ stays the same as in the planar case. After having these results as explained in Section 3 we decide to fit the point set Ω_B to an applicable surface [19]. By this method we know that we keep the curl however we fear that the not so high curled parts of the book surface will also have additional curl added to it. Keeping this in mind we begin our experiments again by changing the values of the variables (σ_x, σ_y, τ and l). As we illustrate our results for different values of variables in Figure 36 we have observed that large sizes of neighborhood (around 125 for σ_x and around 200 for σ_y) gives out better results. These experiments are also done keeping the values of $\tau = 1mm$ and $l = 200$. We conclude from these tests that we need a larger neighborhood for a better fit in the case of fitting to a paraboloid. As we compare the results of paraboloid fitting with the plane fitting we see that paraboloid fitting keeps the curl better, however since it needs a larger neighborhood to achieve that it is much slower that its counterpart. Since we believe that having a better model is more important that the total time spent on calculations. Even though we have tried different smoothing methods our $3D$ model of the surface is not 100% perfect and has some error on it.

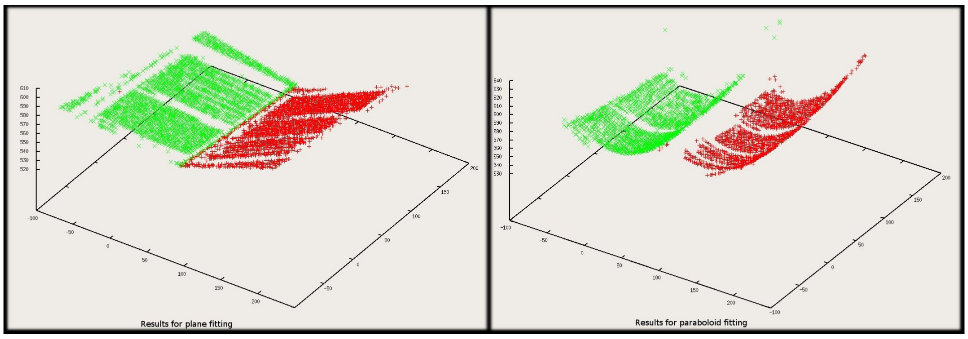


Figure 38: This figure shows the results of both fitting algorithms together in a neighborhood of $105mm - 250mm$ for book image

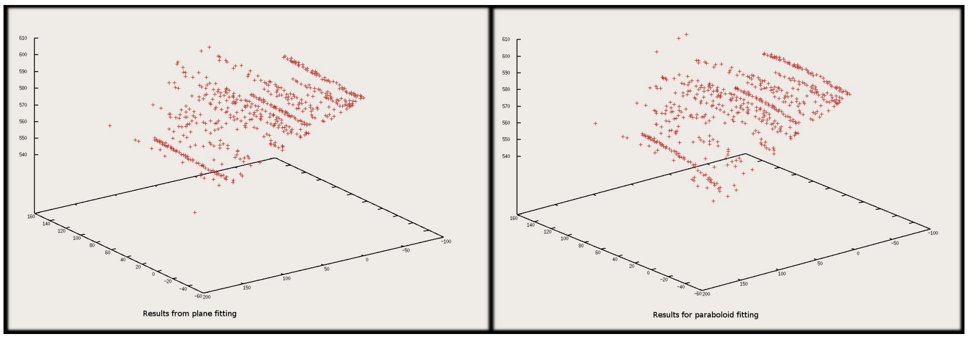


Figure 39: This figure shows the results of both fitting algorithms together in a neighborhood of $105mm - 250mm$ for planar document

7.3 Experiments on De-warping

The last experiments are devised for the De-warping algorithm in use. For this purpose we first begin testing with a $3D$ point set $\Omega_{cylinder}$ which models a cylinder cut in half from its base (see Figure 40). The cylinder has dimensions $r = 10mm$ and $h = 20mm$. These tests are done to prove that the method creates a conformal mapping since a cylinder is a developable surface, meaning it is a surface with zero Gaussian curvature, i.e., it is a surface which can be flattened onto a plane without any distortion, giving us a planar object in the end. After that we continue our test with planar document images. The last experiments are done on the book images.

7.3.1 Experiments with Cylinder

The first tests as explained above are done on a developable surface, a cylinder in this case. The first experiment is made to test if the algorithm works at all. This test, as the output can be seen in Figure 42, turns out that our algorithm works as expected in the case of a developable surface. The orientation is caused because of the fixed variables. We can not control which variables we set and in this case they cause an orientation like this. Then we have altered the cylindrical model and change the values of some points on the cylinder. This means we move some of the points to outside the cylinder surface. This can be viewed to compensate for the error in our $3D$ model reconstruction. In mathematical terms:

Let $p = (x, y, z)$ be a point in set $\Omega_{cylinder}$,then to change the place of p and take it outside the cylinder surface by $p_{new} = (x_{new}, y_{new}, z_{new})$ where $x_{new} = x$, $y_{new} = y$ and $z_{new} = z - 0.01$

In Figure 43 only 5% of the points have been removed from the surface. We can actually observe from Figure 43 that there is a difference between the two results. However we can easily say that this is actually acceptable. The Figure 44 shows that moving the same points only a little further (instead of subtracting 0.01 we subtract 0.02) causes our method to suffer deeply and return an unacceptable result. After observing this result we make one last test with our cylinder data. This test consist of adding an outlier to the surface. This is again done by moving one point away from cylinder surface. The result of this test can be seen in Figure 45. As expected having even one outlier can cause a drastic change in the result of the de-warp method. After finishing these experiments we see that it is futile to have $3D$ models of surfaces with very little or no error in them. This can be explained as the

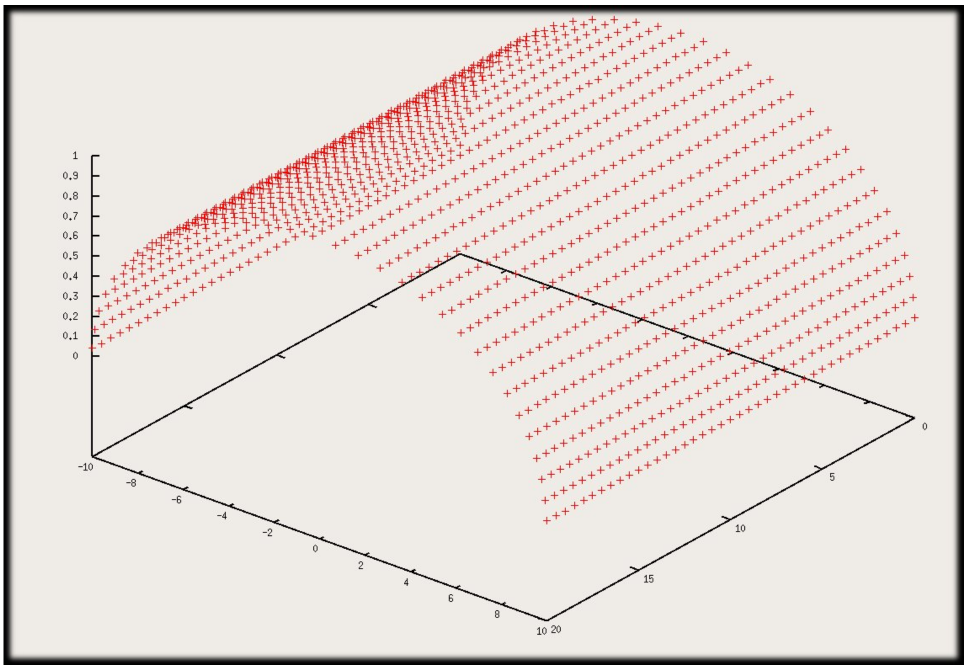


Figure 40: Three dimensional cylinder model.

method is not very robust to models with error on it. To have an idea on cylinder model with points moved from the surface please see Figure 41

7.3.2 Experiments on Flat Paper Images

After finishing our tests on cylinders, we wanted to see how this algorithm works on real sets. For this purpose, we began our testing with basic flat paper images. These images have no curl(actually there is a little curl on the image however this is not visible to human eye)(see Figure 28). Another reason we want to further our tests with these documents is that we are sure that after smoothing the $3D$ point set Ω_P is a perfect model for the planar document surface. The set of points corresponding to Ω_P from the original image can be defined as $\Omega_{P_{Harris}}$. As can be seen from figure (47), the de-warping algorithm works actually quite well for this case. However we can see that in some of the lines there are skewed characters. These skewed

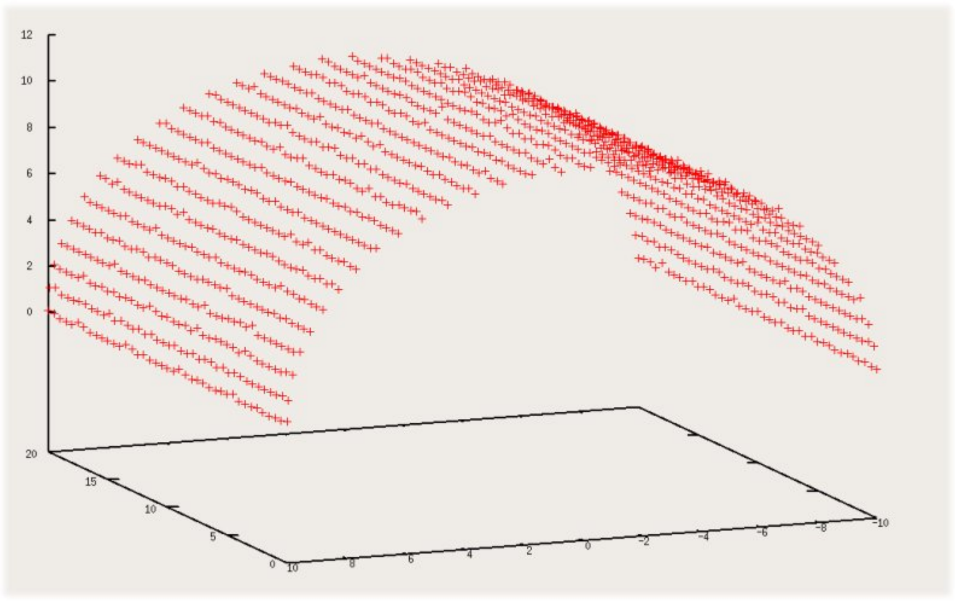


Figure 41: Three dimensional cylinder model with noise added to it.

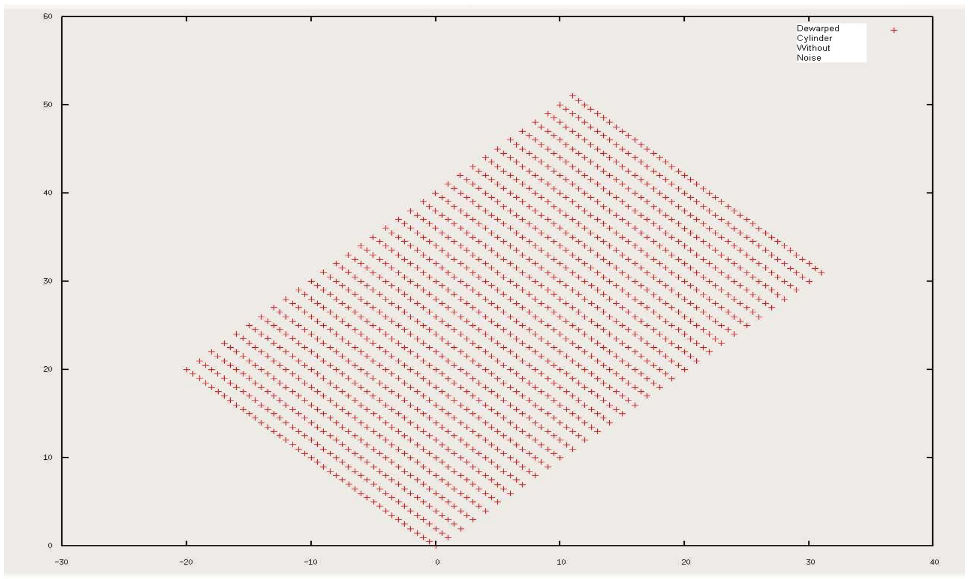


Figure 42: De-warped cylinder model. This is the perfect de-warp model since no noise is added to it

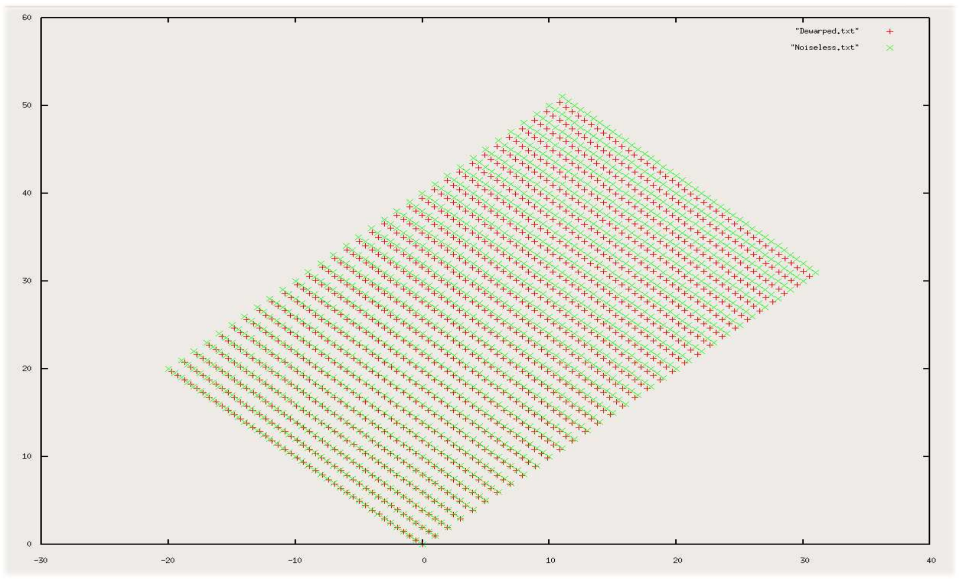


Figure 43: De-warped cylinder model. This model is done with 5% of the points on surface moved. They move in z-axis with a displacement of 0.01 mm

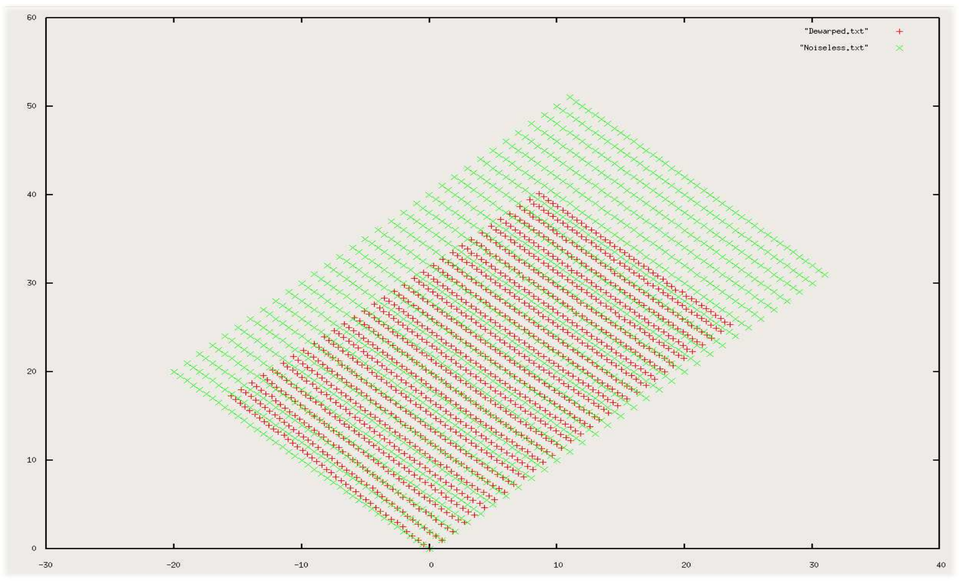


Figure 44: De-warped cylinder model. This model is done with 10% of the points on surface moved. They move in z-axis with a displacement of 0.01 mm

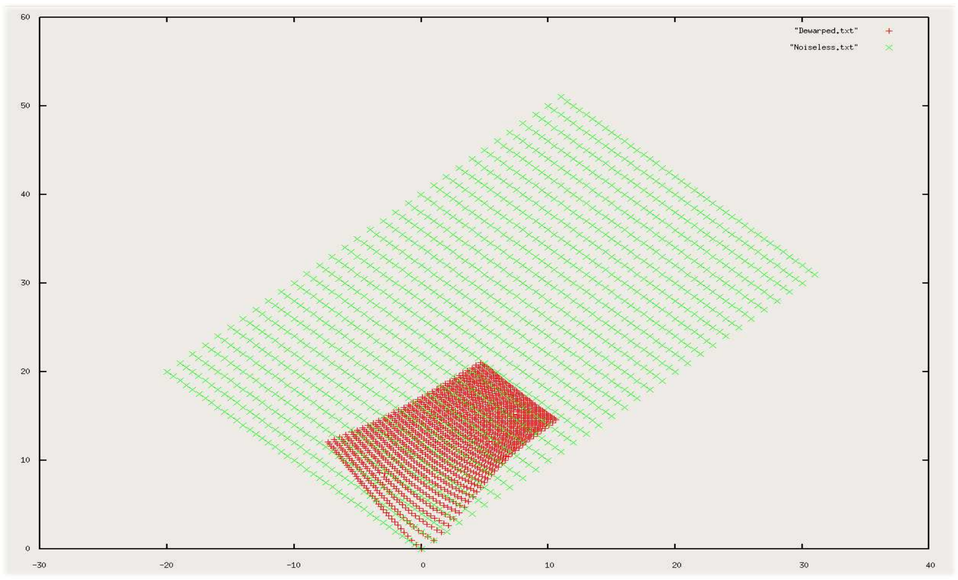


Figure 45: De-warped cylinder model.

characters are caused due to the error when we move an outlier on ray of light. This problem can be easily solved by finding the real corresponding point $u \in \Omega_{P_{Harris}}$ (defined as $u = \begin{bmatrix} x_u & y_u & 1 \end{bmatrix}^T$) to the de-warped image from the source image. We do this by multiplying the known $3D$ point $w \in \Omega_P$(defined as $w = \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix}^T$) with the camera projection matrix P_C(For the definition of camera matrices and please look at [4]). This can be seen mathematically as :

$$u = P_C * x$$

By applying this equation on all points of Ω_P, we find their corresponding points set $\Omega_{P_{Harris}}$. This method solves these skews on the de-warped image. The next experiment is done on an image with a high perspective effect. This means the picture is taken so that we have implemented a higher skew in the image than the usual case(see Figure 48). If we look closely on the Figure 48 we see that already our $3D$ model points have a skew associated

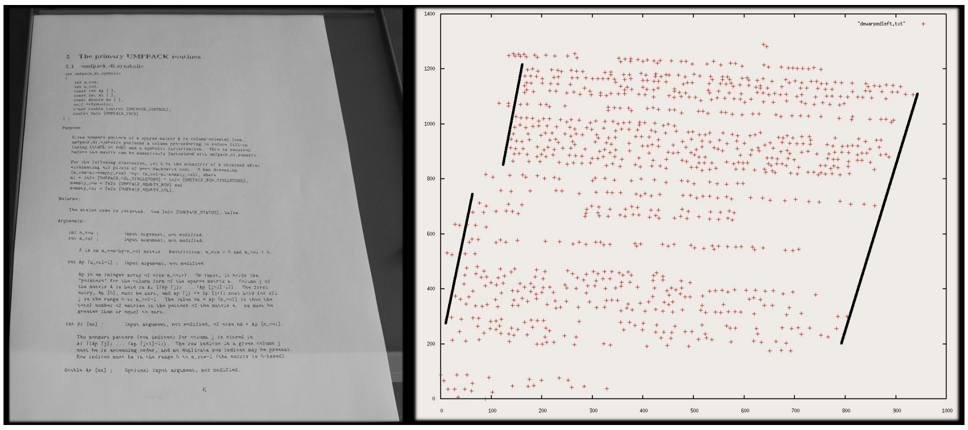


Figure 46: De-warped points shown in a $2D$ plot. The lines on both sides should actually be parallel but due to some skewing affect it is not.

with them. This skew is already preserved in the de-warp point set also. The explanation for this can be that our camera matrices have some error associated with them and in the case of higher error rates, this causes us to have some kind of skew associated with it.

7.3.3 Experiments on Book Images

We do not really have a 100% reliable model for our book surfaces as explained in Section 7.2 . In our experiments in on book images, we used three different page styles to see how robust an algorithm we have. We wanted to see if it works with different cases like pages with figures and images on them. We also examine the case in which there is only text on both pages. After observing not so promising results with planar document and cylinder data, we continued our tests with book images and not surprisingly our results did not seem to improve. As can be viewed in Figure 49 we have not established a decent de-warping for any of the cases we tried. This leas us to search for the reason between results we have and results in [3]. First we want to test if the mapping we use for de-warping really preserves the angles(as it should).

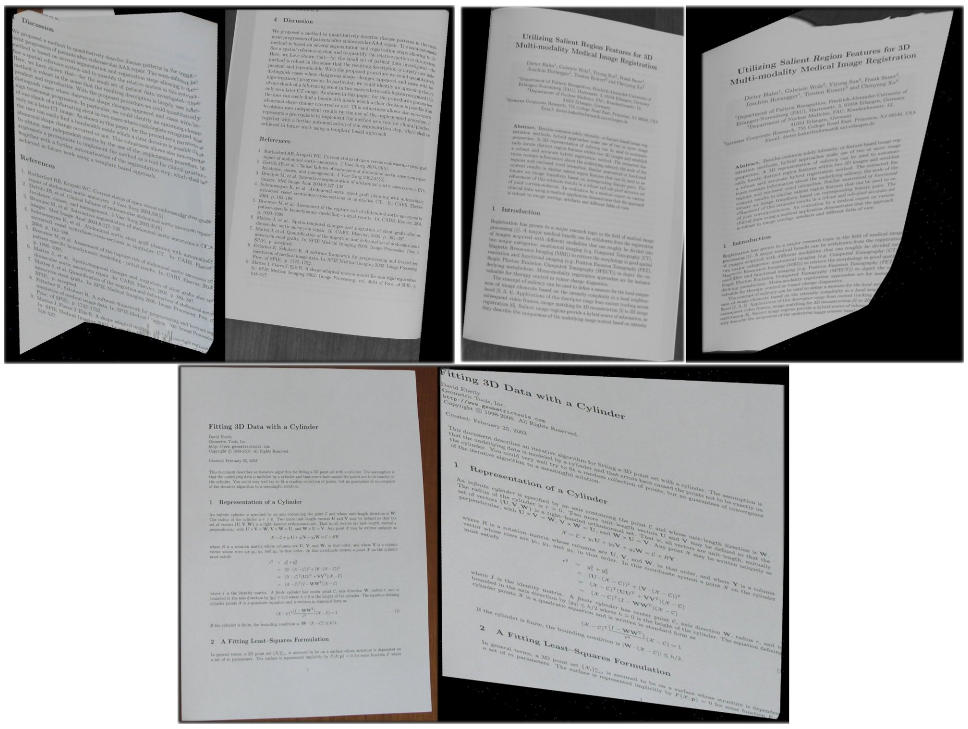


Figure 47: De-warping method implemented on different images

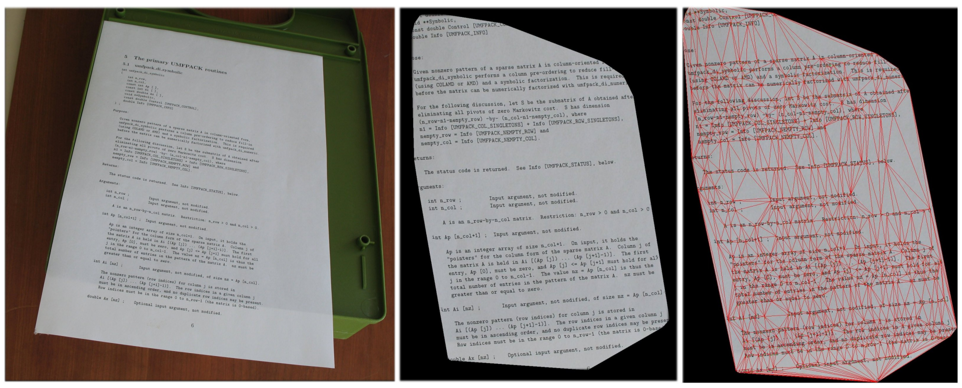


Figure 48: De-warping method implemented on a high skewed document

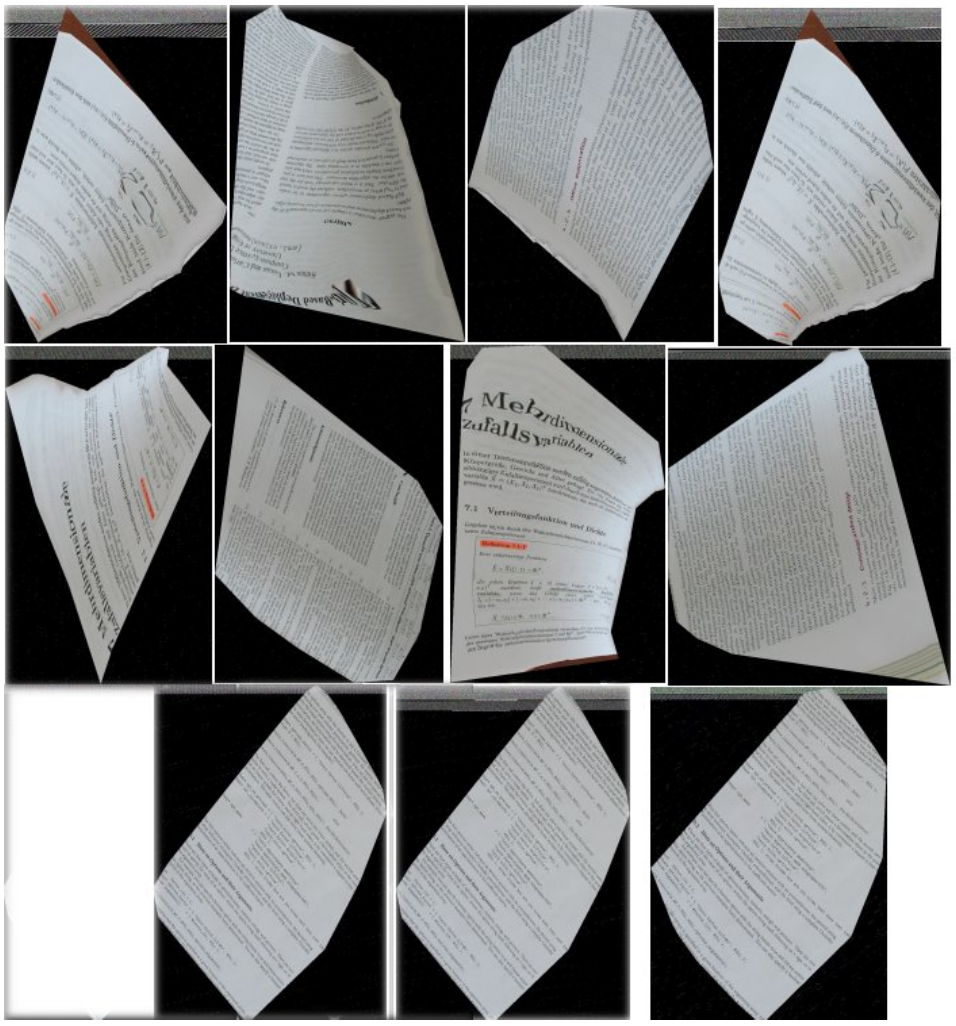


Figure 49: This figure shows different dewarping results we obtain through our experiments.

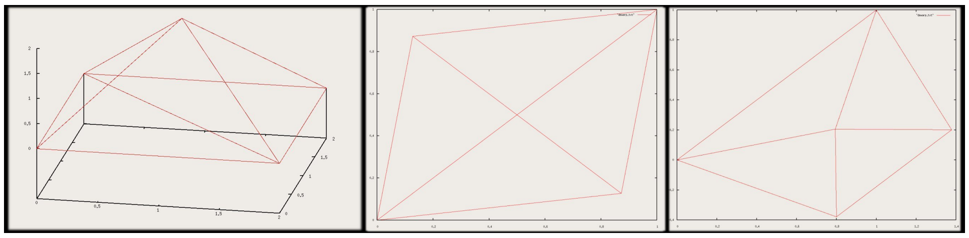


Figure 50: Result of de-warping by setting different points

For that reason, we first used a planar document image first since we can actually find the plane that our document lies and move all points to that plane and have a perfect $3D$ model of the surface. We call the $3D$ point set that models for the planar document case as Ω_{Planar}. The results we obtain can be seen on Figure 46 and proves that the method creates some skew even under this case. When we check the values between the $3D$ triangles and their $2D$ counterparts we see that the maximum difference is in the orders of $1 * e^{-7}$ radians. This also shows that in this case our mapping preserves the angles of triangles since error in this order can be caused by numerical computations. However, when we try the same experiment on a book image, the difference between angles is in the orders of 0.01. This meant there could be up to five degrees of angle change which actually was the reason for all this curled de-warped book images(see Figure 49). Other reasons why our conformal mapping did not work as expected are actually explained in [20]. The reason is:

Let $P_d = (u, v)$ be a de-warped point and $P_{3D} = (x, y, z)$ is the corresponding three dimensional point. Let $S_\triangle$ denote the set of all triangles that has P_d as a corner and $S_\triangle^{3D}$ denote the same for P_{3D}. It is always that the sum of the angles in $S_\triangle$ sum up to 2Π, however this is not true for $S_\triangle^{3D}$.

This means our mapping can never be a hundred percent conformal mapping except for the planar image case. This problem however can be minimized by uniformly distributing the deformation [20]. This also has the implication that the $3D$ model of the surface should be perfect or nearly perfect. Since we can not build a perfect $3D$ model there is no use in trying to have a uniform distribution of points thus triangles. This actually was not a problem on the [3] since their $3D$ model hs a very low error they can actually use this property of uniform distribution and minimize the error caused by this problem. This is not the only problem with the de-warping approach however. As explained in section 5.3, the linear system we had is an oversized system so have infinitely many solutions. We solved that problem by setting some of the parameters however this setting can also give different shaped solutions due to the deformation we had. As can be seen from figure (50), fixing different points(see Section 5.3) to find a unique solution results in totally different shapes. This can be solved trying to fix different points however practically this is not applicable [20]. These experiment prove that to find a conformal mapping that solves the curls in a book image needs a very precise $3D$ model of the surface which can not be implemented using stereo vision methods based on cameras.

8 Conclusions

We have implemented a system that uses a stereo setup to create a $3D$ model of a given document and then by applying a conformal mapping on this model, to get an uncurled version of the document. In our work, we have implemented several different approaches to build a more precise $3D$ model. In matching we have implemented two different matching algorithms. The first method based on block matching with sum of square differences as a similarity measure proved to be very successful in the images with a small difference. However on our tests in the spine of book images, where the contents of the blocks differ more, this method proved to give less accurate results. Therefore we tried a different method in which we could implement some changes on the source block so that we could find better matches. This method gave better results, given that our starting point was very near to the aimed point, which was very hard for us to achieve. We then managed to correct our results by applying a smoothing to our $3D$ model. We also attempted different fitting algorithms. These include fitting to a plane and paraboloid. Fitting to a plane worked better for plane images, whereas fitting to a paraboloid resulted in better book images. These were all done to have a reliable $3D$ model for the de-warping method.

On the second part we work on de-warping method. This method, as we proved on our tests, seemed to work well under the given condition that we had a nearly perfect $3D$ model. However, as explained in section 7, we can not provide a noiseless $3D$ model using a stereo system. Other explanations included in [20] also proved that we needed a better parametrization for our mapping.

This thesis proves that a conformal mapping is not enough to have a very good method for de-skewing documents since we need to take the size parameters also into account. I want to emphasize that with a different parametrization that also keeps sizes of triangles as intact as possible, maybe better results are possible. However we have to implicate that these are just hypothesis and in the end turn out to be wrong as well.

A APPENDIX

A.1 Bilinear Interpolation

This is a method to interpolate functions with two variables. We use this method to find the corresponding pixel values in images we created from our source images. The method can be summarized as finding the value of a discrete function in a point in which we know the values of four points around it and we want to give weights to the points in reverse proportion to their distance from our point of interest. Say we have a point $P = (u, v)$ and we want to know the value of an unknown function $f(x, y)$ at P. We also know values of $f(x, y)$ at four different points which are $f(x_1, y_1) = \Omega_1$, $f(x_2, y_1) = \Omega_2$, $f(x_1, y_2) = \Omega_3$, $f(x_2, y_2) = \Omega_4$. Then we can find the value of f at P by computing:

$$\begin{aligned} f(u,v) \approx &\frac{\Omega_1}{(x_2 - x_1)(y_2 - y_1)}(x_2 - u)(y_2 - v) \\ &+\frac{\Omega_2}{(x_2 - x_1)(y_2 - y_1)}(u - x_1)(y_2 - v) \\ &+\frac{\Omega_3}{(x_2 - x_1)(y_2 - y_1)}(x_2 - u)(v - y_1) \\ &+\frac{\Omega_3}{(x_2 - x_1)(y_2 - y_1)}(u - x_1)(v - y_1) \end{aligned}$$

The value we compute from this method is just an approximation to the actual value.For more information on interpolation methods please refer to [7]

A.2 Barycentric Technic for Determining Triangle Points

This technique is used to determine whether a point lies in a triangle or not. We know the location of three corners of our triangle(points A, B, C).

[7]http://en.wikipedia.org/wiki/Bilinear_interpolation

From these three points, we can actually build our plane equation. To build this equation, we first choose one of the corners as our origin(say A).We take two vectors that passes through A.The best candidates are $(B-A)$ and $(C-A)$. Now we can describe our plane equation.

$$P = A + u * (C - A) + v * (B - A)$$

We have three rules that a point must hold if it is inside the triangle.These rules are:

1. If u or $v < 1$, the point is outside our triangle.
2. If u or $v > 1$, point is also out of our triangle.
3. If $u + v > 1$, point is out of our triangle .

Keeping these in mind the process of finding u and v from three points is an easy task.
First subtract a from both sides:

$$(P - A) = u * (C - A) + v * (B - A)$$

Assume $v_0 = (C - A)$, $v_1 = (B - A)$ and $v_2 = (P - A)$. So our equation becomes:

$$v_2 = u * v_0 + v * v_1$$

We need two equations to solve our system since we have two unknowns. To get two different equations, we impose a scalar product of our equation with both v_0 and v_1. We get:

$$v_2.v_0 = u * (v_0.v_0) + v * (v_1.v_0) \quad (4)$$

$$v_2.v_1 = u * (v_0.v_1) + v * (v_1.v_1) \quad (5)$$

All we are left to do now is to solve this equation and we can do it by writing it in matrix format and solve this equation system:

$$\begin{bmatrix} (v_0.v_0) & (v_1.v_0) \\ (v_0.v_1) & (v_1.v_1) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} (v_2.v_0) \\ (v_2.v_1) \end{bmatrix}$$

This system can be solved using numerical methods. When we have the u and v values we only check if they meet the three conditions for a point to be inside our triangle. For more information on this technique please refer to [8]

A.3 Gradient Descent

This is an optimization algorithm to find a local minimum for a given function in a neighborhood. This algorithm takes steps proportional to the negative of the gradient of function to find the local minimum.
For a real-valued function $F(x)$ gradient descent base its fundemantals on the observation that $F(x)$ in a neighborhood of x is continous (thus differentiable) decreases fastest in the direction of negative gradient of $F(x)$ $(-\nabla F(x))$. From this observation it follows that:

$$y = x - \gamma \nabla F(x)$$

for $\gamma > 0$ and being a small number enough, we have $F(x) \geq F(y)$. Now if we say we begin from a point x_0 and go through the series of points from $x_i i = 1, ..., n$ with equation $x_{n+1} = x_n - \gamma_n \nabla F(x_n)$ with $n \geq 0$ holding you have:

$$F(x_0) \geq F(x_1) \geq F(x_2) .... \geq F(x_n)$$

[8]http://www.blackpawn.com/texts/pointinpoly/default.html

so this series will converge to a local minimum. For more information on gradient descent algorithms please refer to [21]

References

[1] P. Kakumanu, N. Bourbakis, J. Black, and S. Panchanathan, “Document Image Dewarping Based on Line Estimation for Visually Impaired ,” *18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’06)*, pp. 625–631, 2006.

[2] A. Ulges, C. H. Lampert, and T. M. Breuel, “Document image dewarping using robust estimation of curled text lines,” in *International Conference on Document Analysis and Recognition (ICDAR)*, (Seoul, South Korea), pp. 1001–1005, aug 2005.

[3] M. S. Brown and C. J. Pisula, “Conformal Deskewing of Non-Planar Documents,” *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 998–1004, 2005.

[4] K. Parvez, “Stereo Reconstruction of Images,” 2007.

[5] C. G. Harris and M. Stephens, “A combined corner and edge detector,” *4th Alvey Vision Conf.*, pp. 147–151, 1988.

[6] T. Breuel, “Geometric Aspects of Visual Object Recognition,” 1992.

[7] H. N. Timo Zinfler, Christoph Graefll, “High-Speed Feature Point Tracking,” *Vision, Modeling and Visualization*, pp. 49–56, 2005.

[8] T. K. Carlo Tomasi, “Detection and Tracking of Point Features,” *Technical Report CMU-CS-91-132*, 1991.

[9] D. Eberly, “Least Squares Fitting of Data,” *Geometric Tools Documentation*, 2001.

[10] Z. Zhang, “Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting,” *Image and Vision Computing Journal*, vol. 3, 1996.

[11] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Commun.,” *ACM*, pp. 381–395.

[12] B. Delaunay, “Sur la sphère vide,” *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, vol. 7, pp. 793–800, 1934.

[13] D. P. D. C. B. Barber and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Trans. Math. Software*, vol. 22, pp. 469–483, 1996.

[14] T. A. DAVIS and W. W. HAGER, “Dynamic supernodes in sparse Cholesky update/downdate and triangular solves,” *Technical report TR-2006-004, CISE Dept, Univ. of Florida, Gainesville, FL*, 2006.

[15] W. W. H. Y. Chen, T. A. Davis and S. Rajamanickam, "Algorithm 8xx: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," 2006.

[16] T. A. Davis and W. W. Hager, "Row modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, pp. 621–639, 2005.

[17] T. A. Davis and W. W. Hager, "Multiple-rank modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, pp. 997–1013, 2001.

[18] T. A. Davis and W. W. Hager, "Modifying a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 20, pp. 606–627, 1999.

[19] M. Pilu, "UNDOING PAGE CURL DISTORTION USING APPLICABLE SURFACES," *IEEE International Conference on Image Processing, Thessalonica, Greece, September 2001.*, 2001.

[20] K. H. Michael S. Floater, "Surface Parametrization:a Tutorial and Survey," 2003.

[21] M. Avriel, *Nonlinear Programming: Analysis and Methods.* Dover Publishing, 2003.