Surface Reconstruction Using Probability Based Photo-Consistency

Master of Science Thesis in Applied Mathematics

Åsmund Kjørstad

Department of Mathematics
University of Bergen

December 5, 2009

Preface

I would like to thank everyone that has helped me with this thesis. Special thanks to my supervisor Xue-Cheng Tai, not only for the help, but for the opportunity to spend a semester in Singapore.

I dedicate this thesis to all my friends at the institute. Without you life at UiB would suck, and you are the main reason why this thesis exists.

Contents

Chapter 1

Introduction

1.1 Image Processing

Images are all around us, from the computer to the TV to the pet scanner, the staggering development in information technology has made digital images an integral part of modern life. What is unknown to most people is that these images often have been processed in order to make them look good. It should not be surprising though. Anyone who has ever taken photos will know that some of them will come out less then perfect. This is where the field of image processing comes in. By using mathematical tools, we can improve the images. We can enhance certain qualities, remove unwanted noise, and a lot of other similar tasks. Image processing is a large and diverse field with a lot of different methods. And with all these different methods, there is one common factor that ties them all together, and that is the fact they use computers to process digital images.

1.1.1 Computer Vision and 3-D Reconstruction

Our focus in this thesis have been on computer vision. In broad terms this is an intuitively easy concept, our aim is simply to create computers that can see. Creating them however, is easier said than done. The human eye is an extremely complex organ, but this is not the main problem, we can simulate the eyes fairly well with cameras. The real problem is interpreting what we see, and that is the work of our brains. The brain is an expert in interpreting what we see, but it can still be fooled by optical illusions as illustrated below in Figure 1.1. So our brains have a lifetime of experience, not to mention 4 billion years of evolutionary training, and sometimes still get it wrong! This shows the daunting task of computer vision, and in order get somewhere, we must restrict ourselves.

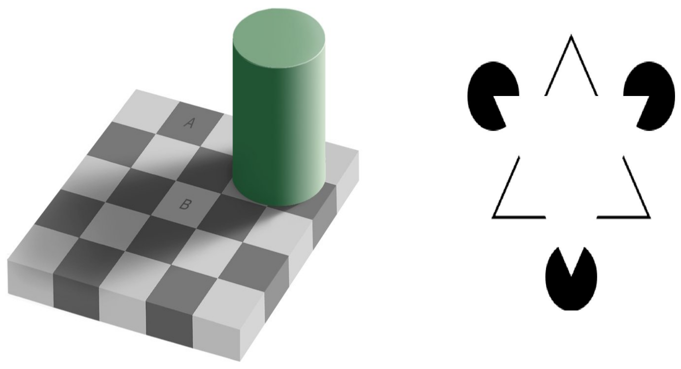

Figure 1.1: *Optical illusions. We see how easily the brain is fooled. Although they look completly different, square A and B are actually the same color. There is also no white triangle in reality, even though it looks like there is.* `Images: Wikimedia Commons`

Strictly speaking, the aim in computer vision is to get a computer to obtain information from images. What kind of information one wants, and what to do with it, is up to the individual programmer. There are a myriad of interesting problems we could look at, but since our time is finite, we must choose to explore only a small spot in the map of computer vision. In this thesis we have chosen to look at the reconstruction of 3-D objects from a set of images, that is, the task of making a 3-D model of an object seen from multiple images.

Why do we seek to do this with images? After all, there exists highly accurate *time-of-flight* 3-D laser scanners, used in everything from gaming to art conservation [18], that gives almost perfect reconstruction, so why do we bother? In fact, reconstruction from images has several advantages over reconstruction with 3-D lasers. First of all, laser scanners are expensive, digital cameras are not. By developing methods for 3-D reconstruction from images, we make reconstruction available not only for big corporations, but for small companies, doctors, and the population in general. Second, to be able to use a laser scanner, the object you wish to reconstruct must be in your possession. One can easily imagine situations where one would wish to reconstruct objects that are far away, in space for example, or objects that only exist in images, like the Buddhas of Bamyan, which were destroyed by the Taliban in 2001. See Figure 1.2.

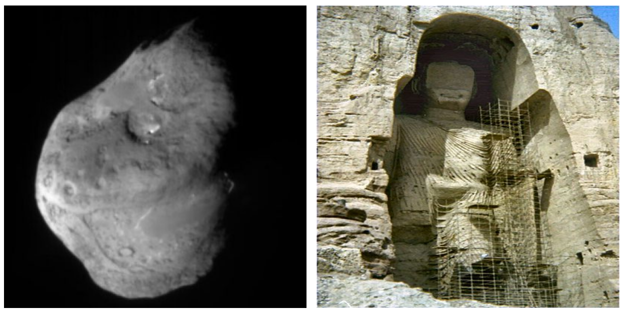

Figure 1.2: *Can not be reconstructed via laser scans.*

1.2 Applications

As all art, the process of doing mathematics is in itself worthwhile, so no further justification should be needed. However, our problem do have several applications in real life. As mentioned above, the gaming industry uses a lot of 3-D models, and a cheap method to obtain these would definitely do them good. Transport companies have also shown some interest in 3-D reconstruction. When they deliver packages they are paid by the weight, but light and large, or oddly shaped packages can be just as much trouble as heavy ones. They would therefore like a way to measure the volume of the packages as well as the weight, and this can be done by creating a 3-D reconstruction of the package. Also, in the past years there has been a trend to make cultural treasures available through the internet. A lot of famous paintings and old books can be find online, and the idea of making virtual museums, i.e. a complete 3-D model of a museum and all the objects inside it, are being discussed. Such a thing is impossible without good reconstruction methods.

Lastly there is the medical industry, where one could use the images from PET-scan, CAT-scan and other machines to reconstruct good 3-D models of internal organs. If the reconstruction is good enough, this would be a great tool for the doctors.

1.3 Thesis Outline

Let us take a brief look at the structure of the thesis. First, in the next chapter, we will introduce and explain the mathematical tools and ideas utilized throughout this thesis. In Chapter 3, will take a quick look at the segmentation problem, a common problem in image processing. Then, in Chapter 4, we will look at concepts from computer vision. Since this is a mathematical thesis, no knowledge of computer vision is expected from the reader. Then, in Chapter 5, we will use ideas and tools explained in Chapter 2, 3 and 4 to develop a reconstruction model, posed as energy minimization. In Chapter 6 we will see the results of our method. Finally, in Chapter 7, we discuss the results, and look at what we could do further to improve upon the method. To summarize:

- Chapter 1 : Introduction
- Chapter 2 : Mathematical Background and Ideas
- Chapter 3 : A Quick Look at The Segmentation Problem
- Chapter 4 : The Reconstruction Problem from Computer Vision
- Chapter 5 : Developing the Energy Minimization Reconstruction Model
- Chapter 6 : Results
- Chapter 7 : Discussions

Once again, so we do not forget. Our goal in this thesis is to construct realistic 3-D models of objects given from different image sets. That is, we will use a set of images of an object, taken at different angles, to reconstruct the object in the images. This will be achieved by using visibility constraints and photo-consistency to formulate a convex energy minimization problem, which we then solve with the piecewise constant level set method of Tai et. al. from [19].

Chapter 2

Mathematical Background

Before we can begin discussing the problem at hand, we first need to define and explain key concepts used throughout the thesis. Although important for understanding, this chapter can easily be skipped by the experienced reader, and simply used when needed.

2.1 Digital Image Representation

It might be self evident, but since this is a thesis regarding digital image processing, we should first explain what a digital image is. A digital gray scale image is a discrete representation of the continuous real world. It is obtained by superimposing a regular, usually rectangular, grid on the continuous world, and then assigning a number to each square on the grid, for example the average brightness in that square. These squares are called *pixels*, and its value is called the *gray level* or brightness. Mathematically, we can look at these images as an intensity function, $I : m \times n \to [t, T]$, where $t = 0, T = 255$ is the most common. Often, these intensity functions are written as $I = u(x)$, where $u(x)$ returns the intensity value of pixel x.

The size of the image is called the resolution, and it increases as the size increases. Digital images are found in all sizes, but to give the idea, typical grids can be $460 \times 320, 720 \times 576, 1920 \times 1440$, and so on.

2.1.1 Colour Images

Most image processing is done on gray scale, also called monochromatic or "black and white", images. This is because colour images are made by combining sets of gray scale images, and methods for gray scale images can therefore often be easily expanded into colour images. The most common color image composition is to combine three gray levels, each representing either red, green or blue. This is called the RGB model.

2.2 Normalized Cross Correlation

Cross correlation is a standard tool in image processing. It is used for matching images, that is, finding whether a given object or region of interest, called the *template*, is contained in another image.

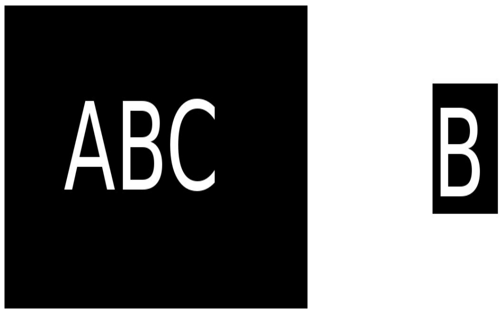

Figure 2.1: *Image and Template. Cross correlation finds the area in the image in which the template fits best.*

Essentially, what we do, is that we slide the template across the entire image, calculating the cross correlation for each position. The higher the value, the closer the images resemble each other. When we have gone through the image, we find the position that gave the maximum score, and this is the part of the original image that is most similar to the template.

We see in Figure 2.1 and 2.2 how this works. Imagine the B template from Figure 2.1 sliding across the middle of the ABC image. Starting from the left, sliding the template to the right. We would expect the largest similarity to be when the B template is in the middle, near one of the letters. Of course, we would also expect to get the greatest value when the B template is over the letter B. As we see on the graph in Figure 2.2, this is exactly what we get.

What else can these images tell us? Most importantly, we see that even though we know the B template to be exactly equal to the B sub image, they are after all taken from the same picture, it is only in a narrow band that we get top score. This tells us that we have to be careful when moving the template. If we move it too many pixels at a time, we risk losing valuable information.

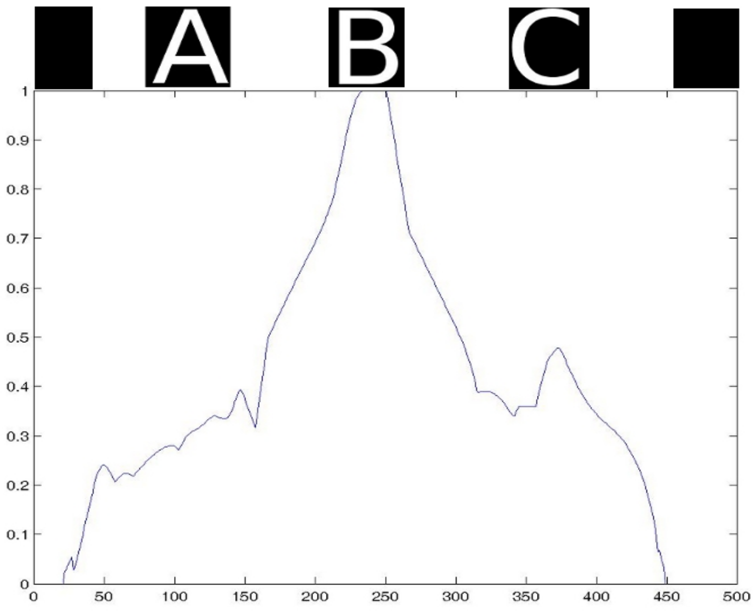

Figure 2.2: *The normalized cross correlation score for different parts of the ABC image using the B template as shown in Figure 2.1. The location of the letters on top are approximate and for visual benefits only.*

2.2.1 Calculating cross correlation

The calculation of cross correlation is motivated by the use of the squared Euclidean distance as a measure of difference. Using the squared Euclidean distance, we can formulate the difference between a part of the image, and the template as

$$d^2(u,v) = \sum_{x,y} [I(x,y) - t(x-u, y-v)]^2,$$

where I is the image and the sum is over (x,y) under the region containing the template t positioned at (u,v). If we expand this, we get

$$d^2(u,v) = \sum_{x,y} [I(x,y)^2 - 2I(x,y)t(x-u,y-v) + t(x-u,y-v)^2].$$

Since the template does not change, we see that the last term is constant. Also, if the term $\sum_{x,y} I(x,y)^2$ is approximately constant, we get that

$$c(u,v) = \sum_{x,y} I(x,y)t(x-u,y-v) \tag{2.1}$$

is a measure of the similarity between the template and sub image. This is the standard formula used for calculating cross correlation, but unfortunately is has some disadvantages. For example, our assumption that the sum $\sum I(x,y)^2$ is approximately constant need not be true, and the equation is not invariant to amplitude changes, which we expect to get under different lightning. To overcome this, [12] introduces the *correlation coefficient* which normalizes the image and its feature vectors. Using this, we get

$$\gamma(u,v) = \frac{\sum_{x,y}[I(x,y) - \overline{I}_{u,v}][t(x-u,y-v) - \overline{t}]}{\{\sum_{x,y}[I(x,y) - \overline{I}_{u,v}]^2 \sum_{x,y}[t(x-u,y-v) - \overline{t}]^2\}^{0.5}}. \tag{2.2}$$

Here $\overline{t}$ is the mean of the template, and $\overline{I_{u,v}}$ is the mean of $I(x,y)$ in the region under the feature. The expression in Equation (2.2) is called the *Normalized Cross Correlation* (NCC).

One of the reasons that Equation (2.1) is used instead of Equation (2.2) is that Equation (2.1) can be computed fast in the Fourier domain, something Equation (2.2) can not. However, as we will see below, [12] provides us with a fast way to compute the NCC.

2.2.2 Fast Normalized Cross-Correlation

Looking at the numerator in Equation (2.2), we can rewrite it as

$$\gamma_{num} = \sum_{x,y} I(x,y)t'(x-u,y-v) - \overline{I}_{u,v} \sum_{x,y} t'(x-u,y-v),$$

where $t'(x,y) = t(x,y) - \overline{t}_{u,v}$. We note that t' has zero mean, and thus the sum is also zero. This leaves us with

$$\gamma_{num} = \sum_{x,y} I(x,y)t'(x-u,y-v)$$

which is a normal convolution, and can be calculated fast using the Fourier transform.

Having found a way to calculate the numerator efficiently, we turn our attention to the denominator. The tricky part here is the term

$$\sum_{x,y}[I(x,y) - \overline{I}_{u,v}]^2.$$

This is the image mean and local energy, and it must be computed at every point (x,y). Since an image usually consists of very many points, this can lead to a very slow algorithm.

However, the term can be efficiently calculated using precomputed tables of the running sum of the image and the image square over the search area. That is, we calculate the cumulative sums for the image in all the points,

$$s(u,v) = I(u,v) + s(u-1,v) + s(u,v-1) - s(u-1,v-1)$$

$$s^2(u,v) = I^2(u,v) + s^2(u-1,v) + s^2(u,v-1) - s^2(u-1,v-1)$$

with $s(u,v) = s^2(u,v) = 0$ when we are outside of the image, that is, when either of $u, v < 0$. For a short explanation of running sums, see Figure 2.3.

Now the energy of the image under the feature can be computed by

$$\begin{aligned} e_f(u,v) &= s^2(u+N-1, v+N-1) - s^2(u-1, v+N-1) \\ &- s^2(u+N-1, v-1) + s^2(u-1, v-1). \end{aligned}$$

Using the precomputed tables, the problematic term $\sum_{x,y}[I(x,y) - \overline{I}_{u,v}]^2$ can now be calculated efficiently, and we have obtained a fast method for calculating the NCC.

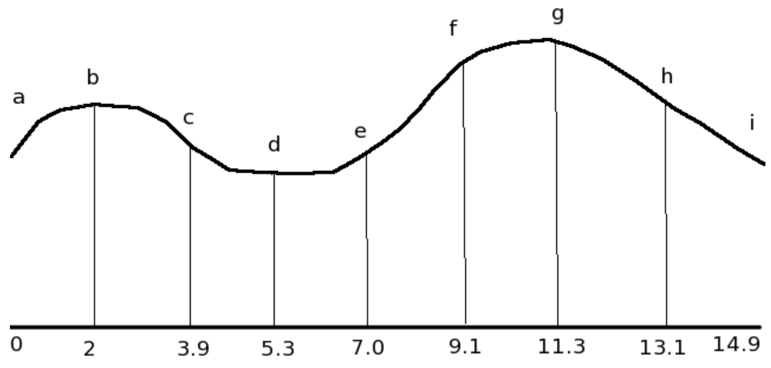

Figure 2.3: *Explaining running sums. Imagine we needed the integral of all the different line segments [a-b], [a-c], ..., [a-i], [b-c], [b-d], ..., [b-i], and so forth. We could of course calculate the integral from one point to another for each segment, but this would not be cost effective. If we instead calculated the running integral sums from [a-i], we can easily compute all the others with little effort. To calculate the integral [c-e], we could just take the cumulative sum [a-e], which is 7, and subtract the cumulative sum [a-c], which is 3.9. Hence, the integral from point c to point e is 7-3.9 = 3.1. This is the method we have used when calculating the NCC.*

2.3 Transformations and Homogenous Coordinates

Homogenous coordinates are commonly used in computer graphics as a way to make calculations with different types of transformations easier. The three basic types of transformations are, *translation*, *rotation* and *scaling*. We can represent these as the following matrix operations

$$P' = T + P$$

$$P' = R \cdot P$$

$$P' = S \cdot P.$$

We see that while translation is addition, the others are multiplication. This is unfortunate, because it means we can not treat all the transformations the same. And in turn, this can decrease the speed and simplicity of our code.

In order to overcome this, it is usual to introduce *homogenous coordinates*. The trick is to represent each point (x,y,z) as an infinite number of four dimensional vectors, and not as a single vector in three dimensional space as is usually done. So what we do is

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \longrightarrow \begin{pmatrix} Tx \\ Ty \\ Tz \\ T \end{pmatrix}.$$

Using this representation we can do translation as matrix multiplication. The translation matrix will look like

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where (t_x, t_y, t_z) are the translations in the x,y,z directions. This means we now can do all forms of basic transformations as matrix multiplication.

2.4 Functional Analysis

Often, in applied mathematics, we seek solutions that are not just a numbers, but functions, satisfying some given constraints. For example finding the shortest path connecting two point, or how a particle behaves in a gravity field. This is the field of functional analysis and what we are dealing with are equations involving unknown functions.

Such expressions dealing with unknown functions are called *functionals* and are used a lot in image processing. Formally, a functional F is a mapping from a vector space X to a scalar field,

$$F : X \to \mathbb{R}.$$

The functionals we will be working on in this thesis are formed by integrals over unkown functions, and we are trying to minimize the functionals. That is, we have a functional

$$F(u) = \int_a^b f(x, u(x))dx,$$

where $f(x, u(x))$ is an unknown function, and we are trying to find

$$\inf F(u).$$

Usually, the unknown function have certain constraints it must uphold. We therefore search for functions belonging to function spaces that have those constraints. Therefore, to understand functional analysis, we must first thoroughly define and explain some concepts about function spaces.

2.4.1 Functions Spaces

We define a *real vector space* as a triple (X,+,$\cdot$) in which X is a set, and + and $\cdot$ are binary operators which satisfy certain axioms, cf. [6]. These axioms are:

- If x and y belong to X, then so does x + y. (closure axiom)
- x + y = y + x. (commutativity)
- x + (y + z) = (x + y) + z. (associativity)
- X contains a unique element, 0, such that x + 0 = x for all x in X. (identity element)
- With each element x there is associated a unique element, -x, such that x + (-x) = 0. (inverse)
- if x $\in$ X and $\lambda \in \mathbb{R}$, then $\lambda\cdot$ x $\in$ X. (closure axiom)
- $\lambda\cdot$ (x+y) = $\lambda\cdot$ x + $\lambda\cdot$ y ($\lambda \in \mathbb{R}$). (distributivity)

- $(\lambda + \mu) \cdot \mathrm{x} = \lambda \cdot \mathrm{x} + \mu \cdot \mathrm{x}$ $(\lambda \in \mathbb{R})$ (distributivity)
- $\lambda \cdot (\mu \cdot \mathrm{x}) = (\lambda\mu) \cdot \mathrm{x}$. (associativity)
- $1 \cdot \mathrm{x} = \mathrm{x}$. (identity element)

Once we have a vector space, we can define a *norm* on it. A norm is a real valued function, written as $\| \; \|$ which satisfies three axioms:

- $\|x\| > 0$ for each nonzero element in X.
- $\|\lambda x\| = |\lambda| \|x\|$ for each $\lambda \in \mathbb{R}$ and each $x \in$X.
- $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in$X. (Triangle Inequality)

If we have a vector space with a norm on it, we call it a *normed linear space*. This makes it possible to study sequences whose convergence is measured by a norm. Of special importance here is the so called *Cauchy sequences*

Definition 1. *A sequence $[x_n]$ in a normed linear space X is said to be a Cauchy sequence if*

$$\lim_{n \to \infty} \sup_{i \geq n, j \geq n} ||x_i - x_j|| = 0.$$

If every Cauchy sequence in the space X is convergent, then the space X is said to be *complete*. A complete normed linear space is a *Banach space*. These are the spaces we will be working in throughout the thesis.

2.4.2 Total Variation

The total variation norm is a measure of the oscillation of a function and is commonly used in image processing. We will just give the definition and not go into details. For details, the interested reader can look at [3].

The total variation of a real-valued integrable and differentiable function f defined on a bounded domain $\Omega \subset \mathbb{R}^n$ is simply

$$TV(f) = \int_\Omega |\nabla f|. \tag{2.3}$$

Since the total variation measures oscillation, we can interpret it as a measure of smoothness. It is therefore often used to ensure correct smoothness in a solution.

2.4.3 Convex Analysis

If there is one thing mathematicians enjoy, it is extending good methods and theory into more general settings. It should therefore come as no surprise that there has been a lot of work in extending the "standard" convex optimization results from calculus into a more general functional analysis setting. This work has grown into the field of *convex analysis*, which we will now take a quick look at.

First of all, we need to know what it means for a set to be convex. Loosely, we can describe a convex set as a set which has no bumps or corners. More mathematical, we can say that it is a set in which any two given points x and y can be joined by a straight line. This can be written into a formal definition, cf. [16].

Definition 2. *A set C is convex if and only if*

$$x, y \in C \quad \Longrightarrow \quad \alpha x + (1-\alpha)y \in C \qquad \forall \alpha \in [0,1], \;\; x, y \in C.$$

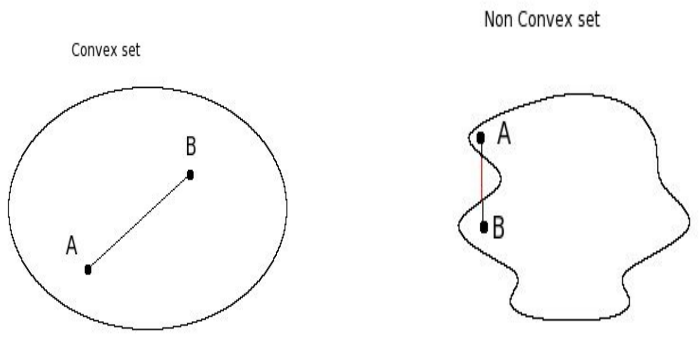

Figure 2.4: *A convex and a non convex set*

Knowing what a convex set is, we now take a quick look at the concept of a *convex hull*. A understanding of the convex hull will be useful in the next chapter, when we introduce the concept of a visual hull.

Definition 3. *The convex hull for a set of points P in a real vector space X is the minimal convex set containing P.*

An intuitive way of visualizing this is the rubber band analogy. If we have a set of points in the plane, we can imagine a rubber band being stretched

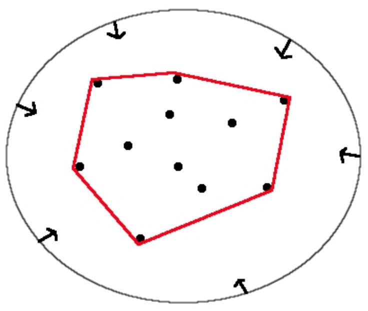

Figure 2.5: *Convex hull. A rubber band is stretched to surround all the points, and then released. The set within the resulting red lines is the convex hull.*

to surround all the points, and then released. The resulting set inside the rubber band would then be the convex hull. This is shown in Figure 2.5.

Having looked at convex sets, we shall now turn our attention to convexity for functionals. We go straight to the definition.

Definition 4. *A functional $F$is convex if and only if*

$$F(\alpha x + (1-\alpha)y) \leq \alpha F(x) + (1-\alpha)F(y) \qquad \forall \alpha \in [0,1],\ x, y \in X.$$

We now know what it means for a functional to be convex. We will use this and known properties from convex optimization in this thesis. In particular, we will use a extremely useful property from convex optimization, namely that all minimums are global. Hence, we can avoid getting stuck in local minimums when optimizing convex functionals, as all local minimums are global.

2.5 The Variational Method

We remember from the last section that we are dealing with a minimization problem over unknown functions, and that we want these functions to be from a Banach space. We will therefore take a closer look at optimization problems in Banach spaces.

2.5.1 The Direct Method in Calculus of Variations

If we have a minimization problem over a Banach space, we first need to be sure that the solution actually exists. This is often done by using the so called *direct method in calculus of variations*. The method will be explained below, but to understand, we need some definitions. Let $(X, +, |\cdot|)$ denote a real Banach space. We denote by X' the topological dual space of X:

$$X' = \{l : X \to R \text{ linear such that } |l|_{X'} = sup_{x \neq 0} \frac{|l(x)|}{|x|_X} < \infty\}.$$

Classically, X can be endowed with two topologies.

Definition 5. *Topologies on X*

- *The strong topology, denoted by* $x_n \underset{X}{\to} x$, *is defined by* $|x_n - x|_X \to 0$ $(n \to +\infty)$.
- *The weak topology, denoted by* $x_n \underset{X}{\rightharpoonup} x$, *is defined by* $l(x_n) \to l(x)$ $(n \to +\infty)$ *for every* $l \in X'$.

Now, back to the minimization problem. Let $F : X \to \mathbb{R}$, and consider the problem

$$\inf_{x \in X} F(x).$$

What can we say about the existence of a solution of this problem? As mentioned above, proving existence is usually done by the following steps, which constitute the direct method in the calculus of variation.

- Construct a minimizing sequence $x_n \in X$, i.e., a sequence satisfying
$$\lim_{n \to +\infty} F(x_n) = \inf_{x \in X} F(x).$$
- If F is coercive, i.e.
$$\lim_{|x| \to +\infty} F(x) = +\infty,$$
one can obtain a uniform bound $|x_n|_X \leq C$. If F is not coercive, i.e., reflexive, then use properties of sequential compactness.[1]
- To prove x_0 is a minimum point of F it suffices to have the inequality
$$\underline{\lim}_{x_{n_j} \rightharpoonup x_0} F(x_{n_j}) \geq F(x_0)$$
which obviously implies that
$$F(x_0) = \min_{x \in X} F(x).$$

[1]See [3] for details.

This last property is called *weak lower semi continuity*, and can be defined more precisely as

Definition 6. *F is called lower semi continuous (l.s.c) for the weak topology if for all sequence $x_n \rightharpoonup x_0$ we have*

$$\varliminf_{x_{n_j} \rightharpoonup x_0} F(x_{n_j}) \geq F(x_0).$$

The same definition can be given with a strong topology.

The problem with l.s.c. for the weak topology is that it generally very hard to prove. However, this is where convexity comes in. We take the following theorem from [3].

Theorem: 1. *Let $F : X \to \mathbb{R}$ be convex. Then F is l.s.c. for the weak topology if and only if F is l.s.c. for the strong topology.*

This is a quite useful theorem, as in most cases l.s.c. for the strong topology is easy to prove. In practice this means that for most cases convexity is a sufficient condition for the existence of a minimizer of the original minimization problem.

2.5.2 Optimization in Calculus

Having seen that convexity is usually enough to ensure existence of some minimizers, we can now turn our attention on how to find them, i.e., the optimality conditions. We will first remind ourselfs how this is done in regular calculus.

We know that a optimization problem is a problem of the form

$$\max f(x)$$

where $f(x)$ is a continously differentiable function. This kind of problem is easily solved by finding the extremum points, that is, the points where $f'(x) = 0$. But sometimes, in addition to the optimization, we have other conditions that the solution must satisfy. For example, a factory owner might wish to optimize the production of goods. But he can not make his workers work more than ten hours a day, nor can he make the machines produce more than their limit. This leads to a *constrained optimization problem.* Mathematically, we can write this as

$$\max f(x) \quad \text{subject to} \quad g(x) = c$$

where $g(x)$ is the constraint, or condition, the solution must satisfy.

How do we solve such a problem? The solution is *Lagrange multipliers*. They work by introducing a new variable λ, called a Lagrange multiplier. This variable λ is then used to define a *Lagrangian function*,

$$\Lambda(x, \lambda) = f(x) - \lambda(g(x) - c).$$

The point of this is that a solution to the original constrained optimization problem corresponds to a stationary point of the augmented Lagrangian function. So just as we had to find the points where $f'(x) = 0$ for the normal optimization problem, we need to find the points where $\Lambda'(x, \lambda) = 0$ for the constrained optimization problem.

It should be noted however, that just like in regular optimization, not all stationary points of the Lagrangian functional is a solution to the original problem. Some test, or other justification, should be used before concluding that a maximum[2] is found.

We now know how to do both regular and constrained optimization in regular calculus. As we will see below, both these methods extend nicely into function spaces as well.

2.5.3 Optimization in Function Spaces

Instead of the usual derivative, we now need a generalized version that works on function spaces. We choose the *Gâteaux derivative*, which is also sometimes called the Gâteaux differential.

Definition 7. *Let X be a Banach space and $F : X \to R$. We call*

$$d[F(u;h)] = \lim_{\lambda \to 0^+} \frac{F(u + \lambda h) - F(u)}{\lambda}$$

the directional derivative of F at u in the direction h if the limit exists. Moreover, if there exists $\tilde{u} \in X'$ such that $d[F(u;h)] = \tilde{u}(h)$, $\forall h \in X$, we say that F is Gâteaux differentiable at u and we write $d[F(u)] = \tilde{u}$.

Now, if F is Gâteaux differentiable and if the problem inf $F(u)$ has a solution u_0, then we have

$$d[F(u_0)] = 0.$$

Conversely, if F is convex, then a solution u_0 of $d[F(u)] = 0$ is a solution of the minimization problem. This type of equation is called an *Euler-Lagrange*

[2]We have been using the maximum in this section, but Lagrangian multipliers work just as well for minimization problems.

equation. This means that in order to minimize a functional, we need to solve that functionals corresponding Euler-Lagrange equation.

But what if we have some constraints? A theorem from [16] tells us the following.

Theorem: Lagrange Multiplier 1. *Let X and Y be Banach spaces, $F : X \to \mathbb{R}$, and $G : X \to Y$. Suppose that:*

- *F and G are Frẽchet*[3] *differentiable on an open set $\mathbb{O} \subseteq X$.*
- *The derivatives $x_0 \mapsto d[F(x_0)]$ and $x_0 \mapsto d[G(x_0)]$ are continous.*
- *$x_o \in \mathbb{O}$ is a regular point*[4] *of the constraints $G(x)$.*

If F has a local extremum under the constraint $G(x_0) = 0$, then there is a Lagrange multiplier $\lambda \in Y$ such that the Lagrangian

$$F(x) + \lambda G(x)$$

is stationary at x_0. That is, we have

$$d[F(x_0)] + \lambda \circ d[G(x_0)] = 0$$

This theorem tells us that given functional F and constraint G, we can use Lagrange multipliers, just as in regular calculus.

We now know we can formulate our minimization problem by using the Euler-Lagrange equation and Lagrange multipliers. But we still do not know how to solve the equations we get. We will therefore now take a look at one method that is commonly used to solve the kind of partial differential equations generated by the Euler-Lagrange condition, namely the gradient descent method.

2.6 Gradient Descent

Gradient descent is an optimization method which uses gradient information to find a minimum. It does this by moving in steps proportional to the *negative* of the gradient. It is well known that the gradient points in the direction of the greatest rate of increase. Therefore, going in the negative

[3]We use the Gâteaux derivative, but this is not a problem. If a function is Fréchet differentiable at a point then it is also Gâteaux differentiable at that point. The converse is not true though.

[4]A regular point is a boundary point that is not a singular point.

direction of the gradient we get the greatest rate of decrease. Due to this, gradient descent is also known as method of steepest descent.

Gradient descent is an *iterative method*. This means that we start out with an initial guess x_0, and then produce a sequence of iterates x_{n+1}. If the algorithm is successful, then x_n converges to the real solution x when $n \to \infty$. A good initial guess x_0 will make the algorithm converge fast, but a bad one might destroy the convergence altogether.

Lets see how this works. If we let $f(x)$ be a function differentiable in the neighborhood of a point x_0. Then $f(x)$ decreases the most if we go in the negative direction of the gradient in point x_0. That is, if

$$x_1 = x_0 - \alpha \nabla f(x_0)$$

then

$$f(x_0) \geq f(x_1)$$

for a suitable α. Continuing this, we get $f(x_0) \geq f(x_1) \geq \cdots \geq f(x_n)$. If this sequence converges, then $f(x_n)$ will be either a local or a global minimum of $f(x)$. We see an example of the gradient descent method in 1-D in Figure 2.6. Notice how easy it gets stuck in a local minimum. This is a big problem, but it can be avoided by a good initial guess x_0, or if we have a convex function or functional, as discussed above.

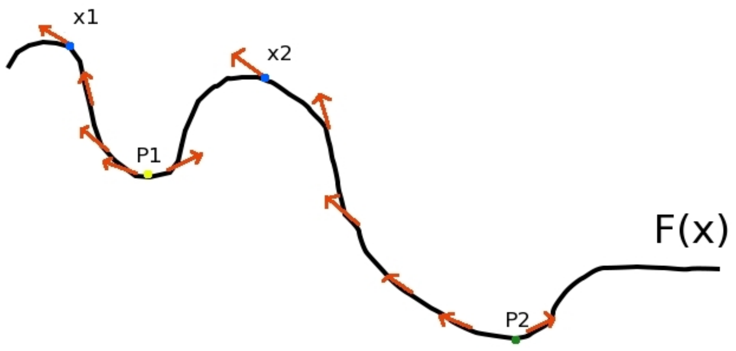

Figure 2.6: *Gradient descent in 1-D. We see how easy the method get stuck in local minimums. The bad initial guess x1 will get us stuck in local minimum P1. A move in any direction from point P1 will lead to an increase of the functional value of f(x), so the algorithm will stop. To avoid getting caught in local minimums, we either need a good initial guess, like x2, or a convex function.*

2.7 Level Set Method

In applied mathematics, when we solve equations, we often want to track how regions move and evolve. We might want to investigate how a pocket of gas dissolves or how a wave breaks against the beach. And in our case, we want to see how our 3-D object evolves in time under our given constraints.

This can be done explicitly, by choosing markers on the boundary and "tying a rope" between the markers. So when the markers move, the boundary of the region follows. We see this in Figure 2.7.

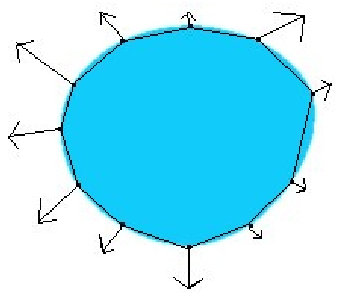

Figure 2.7: *Explicit tracking. When the markers move, the boundary follows. But what happens when a region splits in two, or two regions merge?.*

In simple cases, when the region just expands or shrinks, this is a perfectly reasonable method. However, when we have more complex cases, things start to get difficult. What happens if the region splits in two, or two separate regions merge into each other after some time, or two markers cross each other? In these cases you must add or remove markers, and the explicit method mentioned above quickly becomes quite complicated and computationally hard.

The solution to these problems? We enter the next dimension! Instead of a explicit formulation, we make an implicit one by regarding our object in $\mathbb{R}^n$ as a surface in $\mathbb{R}^{n+1}$. By representing our object as a surface in the next dimension, we can easily keep track of merging and splitting of regions. We see how this works in Figure 2.8.

Let us formulate this a bit more mathematically. We are representing our region Γ as the *zero level set* of a auxiliary function φ,

$$\Gamma = \{(x, y)|\varphi(x, y) = 0\}.$$

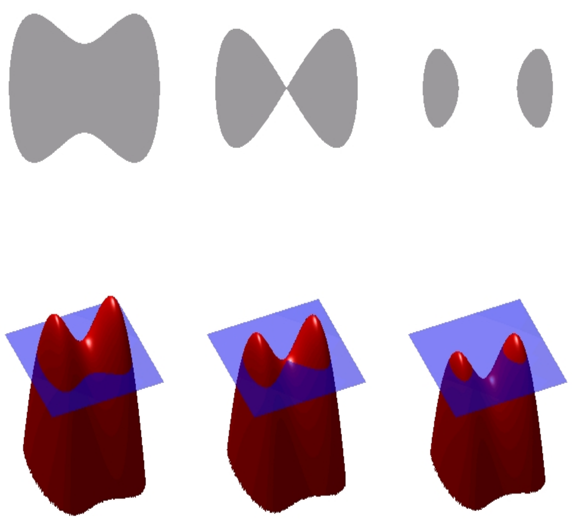

Figure 2.8: *Implicit tracking. We see how easy the level set method handles splitting by using the next dimension. Instead of keeping track of all the boundary markers, we simply translate the level set upwards. The blue plane is the zero level set Γ. The function ϕ gives the distance from a given point to the plane Γ.* ***Image: Wikimedia commons****.*

This auxiliary function φ can be defined in several ways, but we will only look at the two most used. First, the "standard" way to describe φ, which is by the *signed distance function.*

$$\varphi(x,y) = \begin{cases} d(\Gamma,x) & \text{if } \xi \text{ is inside } \Gamma \\ 0 & \text{if } \xi \text{ is at } \Gamma \\ -d(\Gamma,x) & \text{if } \xi \text{ is outside } \Gamma \end{cases}$$

where $d(\Gamma,x)$ is the euclidean distance between Γ and x. We see that φ gives a positive value inside the region Γ, is exactly 0 at the boundary, and is negative outside of Γ. With this φ, we now have a one to one correspondence between the region Γ and the function φ, and thus if we know the solution to one, we know the solution to the other. We know from [20] that the motion of φ is described by

$$\frac{\partial\varphi}{\partial t} + \mathbf{v}\cdot\nabla\varphi = 0$$

where $\mathbf{v}$ is the velocity of the boundary. So by solving this, we can follow the progress of Γ in time.

2.7.1 Piecewise Constant Level Set Method

The second way we will define our auxiliary function φ is as a piecewise constant function. This way of defining φ is due to [19], and is called the *piecewise constant level set method.*

The advantage of the piecewise constant level set method is that one can identify an arbitrary number of sub regions using just one level set function. Assume we want to partition our domain Ω into several sub domains Ω_i, $i = 1, 2, \dots, n$, where n is known. We can then assign a value to the piecewise constant level set function φ for each sub domain, such that

$$\varphi = i \quad \text{in } \Omega_i, \quad i = 1, 2, \dots, n$$

With such a level set function, the characteristic functions of the sub domains are given as

$$\psi_i = \frac{1}{\alpha_i} \prod_{j=1,\, j\neq i}^{n} (\phi - j), \quad \alpha_i = \prod_{k=1,\, k\neq i}^{n} (i - k).$$

If we now define a piecewise constant function as

$$u = \sum_{i=1}^{n} c_i \psi_i$$

we can make sure that this function is uniquely represented, i.e. that different values of φ gives different values of $u(\varphi)$, by defining

$$K(\varphi) = (\varphi - 1)(\varphi - 2) \dots (\varphi - n) = \prod_{i=1}^{n} (\varphi - i)$$

and forcing

$$K(\varphi) = 0.$$

If this is satisfied, then for all functions $\varphi : \Omega \to \mathbb{R}$, we have a unique $i \in \{1, 2, \dots, n\} \quad \forall x \in \Omega$ such that $\varphi(x) = i$. This means that a point $x \in \Omega$ can only be part of one region, which again means that the function is well defined.

The simplicity of the basis functions we have defined makes it possible for us to measure both the length of the curve surrounding Ω_i and the area of Ω_i simply by

$$|\partial\Omega_i| = \int_\Omega |\nabla\psi_i \, \mathrm{dx} \quad \text{and} \quad |\Omega_i| = \int_\Omega \psi_i \, \mathrm{dx},$$

where $|\partial\Omega_i|$ is the total variation (TV) norm.

The piecewise constant level set method is the one we have chosen to use in this thesis. One of the reasons for this approach is that the piecewise constant level set method is the easiest to extend into several regions. Although we do not use more then 2 regions in this thesis, it is a natural point in which one could extend the work, and by using the piecewise constant level set method here, we are making this extension easier for the future. Also, it is known from the work of Knudsen in [13] that this method works for 3-D reconstruction.

Chapter 3

Image Segmentation and the Chan-Vese Method

As we will see below in Chapter 4, a crucial part of the reconstruction method we are using depends on your ability to segment images. We will therefore use this chapter to explain what is meant by segmentation and how it is done. It would have been no problem writing an entire thesis about this topic alone, but since segmentation is just the initial part of our reconstruction method, a concise introduction will have to do.

3.1 Image Segmentation Basics

We can describe image segmentation as the process of partitioning an image into non-overlapping regions. This is done in order to find special regions of interest. The standard way is to partition the image into two regions, background and object. See Figure 3.1. However, it is not necessary to restrict ourselves to only background and object, if necessary, we can segment an image into as many regions as we like. Mathematically, we can write the decomposition as

$$I = \bigcup_{i=1}^{n} I_i$$

where I is the entire image, I_i the segmented sub images and n the number of segmented regions.

Image segmentation is one of the oldest problems in image processing, and there is a multitude of different methods. To better illustrate the idea of segmentation, we will now give an example of a segmentation method. We will explain one of the most basic methods. This is the method of *thresholding*. In this method we simply label all pixels above a certain value, called

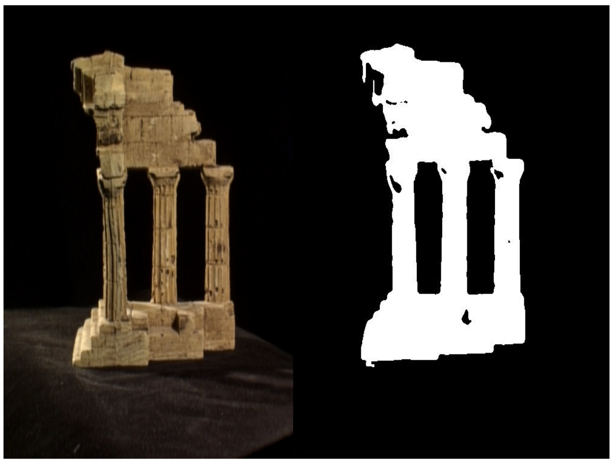

Figure 3.1: *Original and segmented image. Only background (black) and object (white) is part of the segmented image*

the threshold level, as the object. Those pixels below the threshold level are labeled as the background. That is, we create the thresholding function $f : I \to [0,1]$ corresponding with threshold level t_0 for every pixel x by

$$f(x) = \begin{cases} 1 & \text{if } x \geq t_0 \\ 0 & \text{if } x < t_0 \end{cases}$$

Although very primitive, this method can work well on certain images, especially if the images are smoothed[1] first.

3.2 Mumford-Shah Model

For more difficult images, we need more advanced methods. The Mumford-Shah model is a classic in image processing and constitutes the basis for our method of choice. We will therefore quickly explain this model.

The original Mumford-Shah model is formulated as a optimization problem. We want to find the optimal approximation of the perfectly segmented image. Let Ω be a bounded open set, Γ the set of edges, $u_0(x)$ represent the initial image and u be the true segmented image that we are trying to find. We can then without loss of generality assume that $0 \leq u_0(x) \leq 1$ a.e. $x \in \Omega$. The idea is then to search for a pair (u, Γ) which minimizes

$$F(u,\Gamma) = \int_\Omega (u - u_0)^2 \, dx \, dy + \alpha \int_{\Omega-\Gamma} |\nabla u|^2 \, dx \, dy + \beta \int_\Gamma d\sigma$$

[1]Smoothing is a sort of blending of the pixels in a region, making the region more homogenous. For example using the average of all neighborhood pixels.

where α and β are non-negative constants and $\int_\Gamma d\sigma$ is the length of the boundary Γ. This is the classical Mumford-Shah model. The three terms have the following properties.

- Term 1 : Minimizing the difference between the perfectly segmented image, u, and our approximation u_0
- Term 2 : Ensuring smooth regions outside of the edges.
- Term 3 : Minimizing the length of the boundary, ensuring a tight fit.

Having explained the Mumford-Shah model, we can move on to the method of choice for this thesis, the *Chan-Vese method.*

3.3 Chan-Vese Method

Chan and Vese, in their paper [11] used the Mumford-Shah model above to develop a new *region based* segmentation method. That is, a method that is able to segment regions without clear edges. See Figure 3.2.

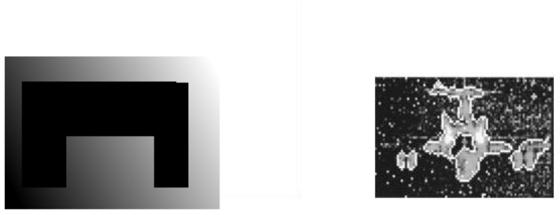

Figure 3.2: *The block to the left has clearly defined edges and can easily be found by a edge based segmentation method such as the Mumford-Shah model. The noisy image of the plane to the right however, has no clear edges, and need a region based segmentation method.*

The Chan-Vese method can be taught of as the level set implementation of the piecewise constant version of the Mumford-Shah model. Knowing this, we take a look at the piecewise constant version of the Mumford-Shah model.

When we assume a piecewise constant image, the second term of the Mumford-Shah disappears. Thus, we can now write the functional as

$$F(u,\Gamma) = \sum_i \int_\Omega (u-u_0)^2 \, dx \, dy + \beta \int_\Gamma d\sigma.$$

The minimization of this is simply

$$c_i = \text{mean}(u_0) \text{ in } \Omega_i.$$

Using this, Chan and Vese came up with a new functional to be minimized,

$$F(c_1, c_2, \Gamma) = \lambda_1 \int_{Inside(\Gamma)} (u_0(x,y) - c_1)^2 dx\, dy$$

$$+\lambda_2 \int_{Outside(\Gamma)} (u_0(x,y) - c_2)^2 dx\, dy + \beta \int_{\Gamma} d\sigma$$

where λ_1 and λ_2 are fixed non-negative constants, Γ the contour, and u_0 is the initial image as always. The first two terms can be interpreted as two forces. The first term is there in order to shrink the contour and the second term is there in order to expand the contour. These two forces get balanced when they reach the boundary of the region of interest. The last term is a regularizing term regarding the length of the contour.

To represent this in the level set form, we use the level set function from Section 2.7 in the last chapter, $\phi(x),\ x \in \Omega$.

$$\begin{array}{ll} \phi(x) > 0 & \text{inside the object} \\ \phi(x) = 0 & \text{on the boundary of the object} \\ \phi(x) < 0 & \text{outside the object} \end{array} .$$

and the Heaviside function $H(x)$ $x \in \Omega$,

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Thus our level set representation becomes

$$F(c_1, c_2, \Gamma) = \lambda_1 \int_{\Omega} (u_0 - c_1)^2 H(\phi) dx\, dy$$

$$+\lambda_2 \int_{\Omega} (u_0 - c_2)^2 (1 - H(\phi)) dx\, dy + \beta \int_{\Omega} |\nabla H(\phi)| dx\, dy$$

where c_1 and c_2 are the average brightness over the inside and outside

$$c_1 = \frac{\int_{\Omega} u_0 H(\phi) dx\, dy}{\int_{\Omega} H(\phi) dx\, dy}, \quad c_2 = \frac{\int_{\Omega} u_0 (1 - H(\phi)) dx\, dy}{\int_{\Omega} (1 - H(\phi)) dx\, dy}.$$

This functional is then minimized by finding the Euler-Lagrange equation and then solving it by gradient descent as explained in the last chapter. Details of how this is done will not be given here, as we will do much the same for another functional later on in the thesis. The details can be found in [3] and [11] if the reader is interested.

We have used the Chan-Vese method to segment the images we have in this thesis. Its region based approach makes it a good choice for both normal and noisy images[2]. This makes our reconstruction method more robust against noise.

[2]Noisy images are images corrupted by some function. This removes information from the image, and makes them harder to analyze

Chapter 4

Multi-view Surface Reconstruction

The reconstruction of 3-D objects from images is one of the oldest in computer science. However, until recently it was not possible to create good reconstruction due to the lack of computational power. This has all changed the last years, and now there are a multiple of different methods. In this thesis, we are looking at the task of recreating an object seen from many different angles. The cameras are situated all around the object, making all of the object in question seen. This allows for good reconstruction of the object, but also makes the problem quite heavy computationally.

4.1 The Multi-view Reconstruction Problem

First, to clarify, we give a concise statement of the problem.

> *"The multi-view reconstruction problem is the problem of reconstructing a 3-D object as seen in a set of images."*

That is, using only the information from a set of images, we want to partition a volume V into *object* and *background*. This partitioning can be written as $V = f(x)$, where $f(x) \in \{0, 1\}\; \forall\, x \in \mathbb{R}^3$. A voxel of value 1 is said to be part of the object, and a voxel of value 0 is part of the background. We see in Figure 4.1 an illustration on how the set of images we are using is obtained.

Unfortunately, this is, as the cliché says, "easier said than done". What we are dealing with here is an *inverse problem*. Given the 3-D object, we could easily create the images in our set, but to go the other way around is a different matter. Inverse problems are notoriously difficult, and in fact, often insolvable. To understand what this means, we look at the concept of *well posedness*.

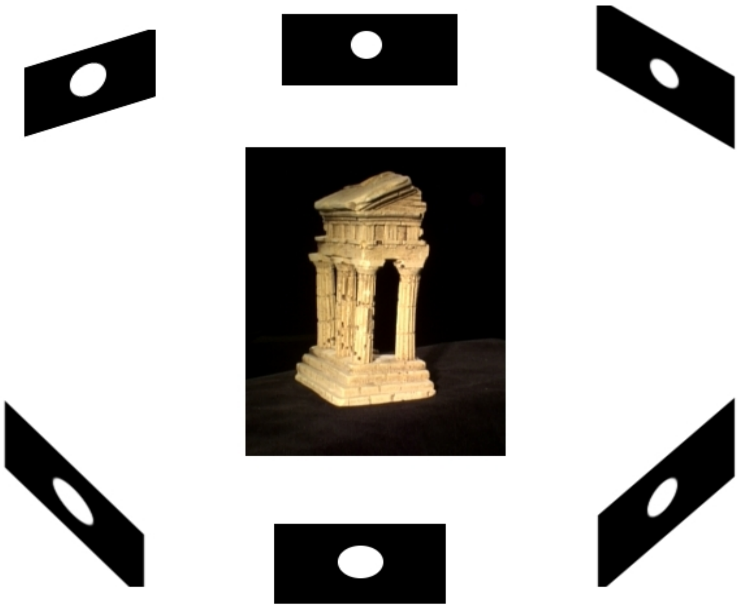

Figure 4.1: *The 3-D object we wish to reconstruct are surrounded by a set of cameras. Each camera will take one photo of the object, and this, in addition to information on the location of the cameras, are used to reconstruct the 3-D model*

4.1.1 Well Posedness

The French mathematician Hadamard believed that all mathematical models describing the physical world, should fulfill certain properties. These properties are:

1. A solution exists for the problem.
2. The solution is unique.
3. The solution depends continuously on the data given in the problem

If the conditions are fulfilled, we say that the problem is *well-posed*, otherwise, it is *ill-posed*.

Let us look at these criterion for the general reconstruction problem. Since we know that the images we are given comes from a real object, a solution must exist. So criterion 1 is fulfilled. But even though a solution exists, it does not necessarily imply we can find it, and also, this is for the physical problem only. It is not given from this that our mathematical model of the problem has a solution.

Is the solution unique? The answer is unfortunately no. An image does not capture all the information about the object, and even if it did, we are only given a limited amount of images to work on. This means that he reconstruction problem is inherently ill-posed. No matter how we try to model it mathematically, we can not hope to attain more than a good approximation of the true object.

Since we are dealing with the physical world, condition 3 is fulfilled. A small perturbation in viewpoint will never radically change an image. Note that once again this is for the general reconstruction problem. It is not given that our mathematical model fulfill this criterion.

Even though we know our problem is ill-posed, we would like to try to solve it. We therefore add certain constraints to the problem. That is, we try to make the problem better by restricting ourselves to certain cases, as we will see below. But even when we do this, we can not be sure our problem is well-posed. In the end it is up to our own judgment to deem the results as realistic and trustworthy or not.

4.1.2 Camera Models

When dealing with inverse problems, it is always helpful to understand the original problem, i.e., how an image captures the 3-D object in the first place. We therefore quickly explain some principles of how cameras work.

The easiest of the camera models, is the so called *pinhole camera.* This is quite simply a light-proof container, with a small hole in it. The principle of the pinhole camera is quite similar to the way our own eyes work. Light rays from the object all pass through the single hole and these project an inverted image of the object onto the other side of the container. We see how this works in Figure 4.2.

Now, if the hole in the camera is too big, the light rays will not pass through the same point, causing disturbance in the inversion. The solution seems obvious, reduce the size of the hole. However, this will also create problems. Due to the wave properties of light, we will experience *diffraction.* Diffraction is a physical phenomena concerning waves, which was first described, posthumously, in 1665 by Francesco Maria Grimaldi [10]. It tells us that light passing through a small passage will “break up” and form specific patterns, which in our case will cause blurring of the image.

This means that both a too small and a too large hole in the camera will cause blurring, so what do we do? The solution is ancient, humans have used glasses to correct for blurry vision for thousand of years, and there is no reason why we can not use glasses in a camera. Glasses are of course just lenses that focus the light, and by putting a lens in the camera we can achieve a much sharper image. This is known as *lens-based imaging.*

In lens-based imaging, the convex lens focuses the light beams of certain

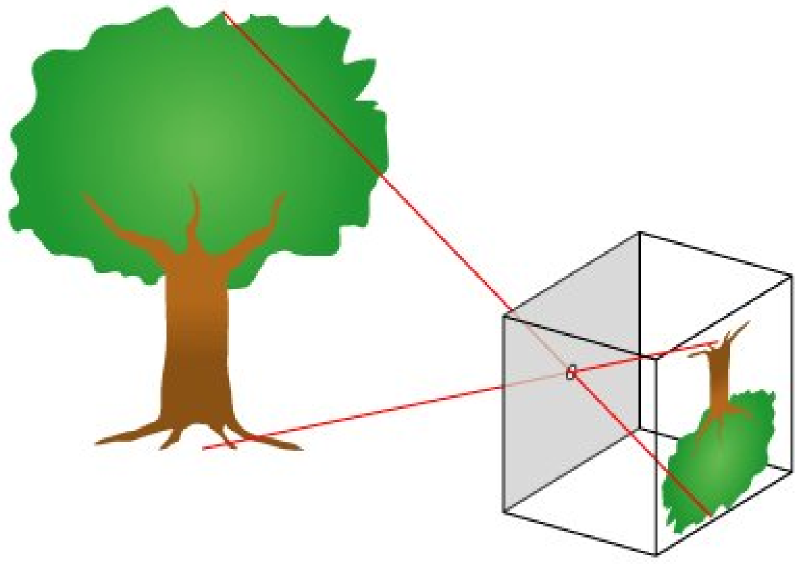

Figure 4.2: *Pinhole camera model. Light projected from the object through the hole creates an inverted image at the back of the box, but only if they all pass through the same point.* `Image: wikimedia commons`

wave lengths through a single point called the *optical center*. The advantage of this is that the image becomes much sharper, but in exchange only certain ranges are in focus. But if our entire scene is in the given range, we get sharp and good images.

Lens-based imaging is not perfect, and it carries with it some problems, but all of these are manageable, and we will not go into the details here.

4.1.3 Light and Reflectance Models

Having looked at how cameras create images, it is natural to look at the process which makes it all possible, light itself. In order to reconstruct a scene, we need a basic model for the light and reflectance in the images which describe the scene. As always when we want to model a natural phenomenon, we seek inspiration from the grandest of all models, nature itself.

In nature, photons are emitted from a light source, and then travels in a straight[1] line until the photons hit a surface. When the photons hit, any combination of four effects can happen. These are `absorption, reflection, refraction and fluorescence.`

[1]We ignore relativistic effects here.

Absorption means that the energy from the photons is transferred into other kinds of energy, and thus the light "disappear", it is absorbed by the object.

Reflection is that the photons are "bounced back" from the object, as we know from common mirrors.

Refraction is what we observe when light hits water or other surfaces. The angle of the light is changed, giving us a straight line that is bent where the photons hit the surface.

Fluorescence is an form of cold body radiation where a photon hits an object which then absorbs it and the object then releases a photon of lesser energy, often leading to nice color schemes.

All of these effects are something we would have liked to model, however, it would be time consuming and would be almost impossible to do without extensive knowledge about the object in question. We therefore do as is common in the image processing community and assume a *Lambertian surface* for the rest of this thesis.

A Lambertian surface, or Lambertian reflectance as it is sometimes called, is a surface where the brightness is independent of the angle of the surface. In other words, a given spot on a Lambertian surface looks the same from all angles that can see it. Although we can experience this in the real world when we look at rough surfaces, like unfinished wood, most surfaces are not Lambertian, and if they are far from Lambertian, like transparent surfaces, we can not expect good results.

4.2 Projections, Rotations and Translations

A difficult part of the multi-view surface reconstruction is how to transform the 2-D information in the image into 3-D information about the object. One of the reasons for this is that we are dealing with 3 different coordinate systems, the camera coordinates, the image coordinates and the real 3-D world coordinates. We have to somehow combine all of these together. To do this we need to know the location of the camera, the cameras intrinsic calibration and also the orientation of the camera. We will soon explain in detail how this is done.

The cameras intrinsic calibration parameters can be found by doing so called camera calibration, using a few fixed points in each image. However, this is prone to error, time consuming, and not what this thesis is about. And even if we did find the the camera calibration parameters, we would also need to find the relative location and orientation of the cameras involved. This would be exceptionally hard, so instead, we are going to use an image set where all these parameters are given. This image set is supplied by the Middelbury Vision Group [21].

Still, given the camera parameters we need to combine the three coordinate systems together. Following [8], we do this by creating a so called 3×4 *Projection matrix*. By strict definition this is not a projection matrix as it is not square, but we can expand it into 4×4 by adding a row consisting of zeros and a 1 in the lower right corner. However, this is not necessary for computations, so we will be using the 3×4 matrix in the rest of this thesis.

4.2.1 Camera Coordinates

First, let us look at how the image is captured by the camera. If we assume a pinhole model of the camera, the 3-D object is projected into a *viewing plane*. We remember from linear algebra that this can be represented with a *perspective projection*. That is, a projection that maps each point (x, y, z) onto an image point $(x^*, y^*, 0)$ so that the two points and the center of projection, i.e. the pinhole camera, are on a line.

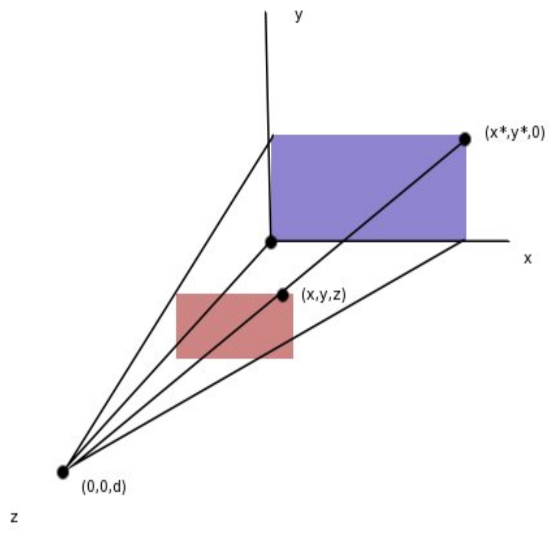

Figure 4.3: *Perspective projection.*

We let X_c, Y_c, Z_c be the points projected into the image points and use homogeneous coordinates as described in Chapter 2. Then this perspective projection can be represented, up to a scale factor, as a matrix.

$$\begin{pmatrix} y \\ x \\ f \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix}.$$

4.2.2 Image Coordinates

Next, lets look at the internal camera coordinates. These are different for different cameras, and describe how motion is handled by the camera. To express this, let us define a coordinate system using the top left corner as origo. We then define u_0 and v_0 to be the center pixel. We can now express all the pixels in the image as

$$k_u x = u - u_o$$

$$k_v y = v_0 - v,$$

where k is the pixels. See Figure 4.4. In matrix form we can write this as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ f \end{pmatrix} = \mathbf{C} \begin{pmatrix} x \\ y \\ f \end{pmatrix}$$

where the 3×3 upper triangular matrix $\mathbf{C}$ is called the *camera calibration matrix*. This is the matrix that gives the transformation between image points and a ray in 3-D space. The scaling is given by the parameters fk_u and fk_v and the point (u_o, v_0) is the *principal point*, i.e. the point where the optical axis intersects the image plane.

4.2.3 Real World Coordinates

Now we look at the 3-D world coordinates. As above, let X_c be the coordinates from the camera point of view. We also introduce X_w as the coordinates in the real world. We can then write the Euclidean transformation between the camera and the real world as

$$X_c = \mathbf{R}X_w + \mathbf{t},$$

where $\mathbf{R}$ is a rotation matrix and $\mathbf{t}$ is a translation matrix.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}.$$

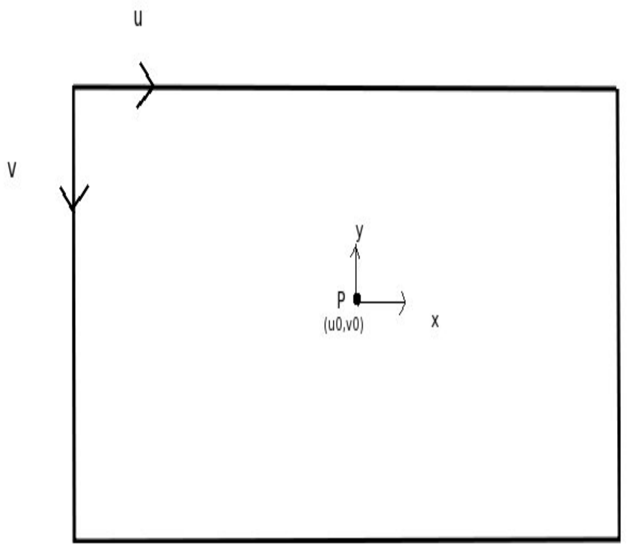

Figure 4.4: *Image coordinates.*

Combining this with the camera calibration matrix **C** and the expression for the perspective projection as described above, we finally get the 3×4 projection matrix we can use.

$$\mathbf{x} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{C} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \mathbf{C} \left(\mathbf{R} | \quad \mathbf{t} \right) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}.$$

So we get that the projection matrix is $\mathbf{P} = \mathbf{C}(\mathbf{R}|\mathbf{t})$. Hence, we can find the projection for an image point into the real world, by taking

$$\mathbf{x} = \mathbf{P} \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}.$$

Now that we know the projections from image to real world, we can start working on what to do with them.

4.3 Visual Hull

In real life, when a human being looks at an object, the first thing one usually notices is the shape of the object. The shape, or the silhouette, contains a lot of information, and of special importance for us is that it gives an accurate description of the location of the object. This trivial fact, that an object must lie within its own silhouette, is quite useful for us.

If we assume a perfectly segmented binary image, and extend a ray from each pixel into 3-d space, then the physical 3-D object must lie inside this volume given by the silhouette ray. Using this, we can combine the silhouette information from all the images, and obtain a volume which encloses the true object. This silhouette constructed volume is know as the *visual hull.* A more formal definition given by Laurentini in [17] states that the visual hull is the maximal volume consistent with an objects silhouettes as seen from some set of viewpoints.

Mathematically we can write this as the intersection of all the silhouette rays, that is $S = I_1 \cap \cdots \cap I_i$, where $S \in \{0, 1\}$ is the visual hull and I_i is the projection of the silhouettes into 3-D.

Although the visual hull is guaranteed to enclose the true object, it will usually not be the same as the true object, since the visual hull does not capture depth concavities. This is because a camera can only see in straight lines and contain no depth information. Hence, all depth concavities in the object will be "filled". Although not entirely correct, the visual hull can be taught of as a sort of collection of convex hulls in 3-D.

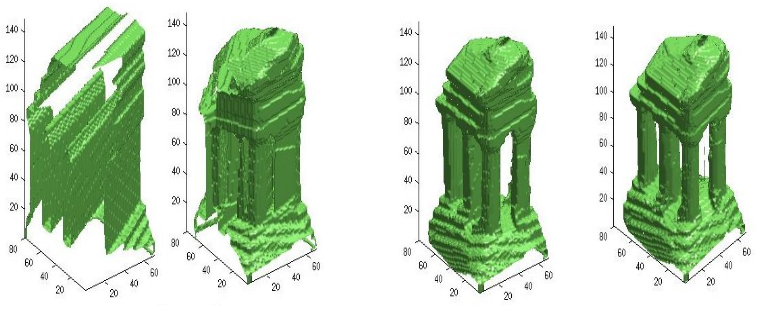

Figure 4.5: *Visual hulls. From left to right, 1,5,10 and 16 viewpoints.*

An implementation of the Chan-Vese model was used to segment the images in this thesis. [2]

[2]In retrospect, this was unnecessary advanced, as some smoothing and a simple thresh-

4.4 Photo-Consistency

Once we have obtained the visual hull, we can start removing parts that should not be there. Remember, the visual hull contains the entire object, so we only need to remove, not add, voxels. We do this by looking at the images and checking how consistent they are. We know that if we assume a Lambertian surface, the value of a voxel that is part of the true object should be approximately the same in all cameras that can see it. See Figure 4.6.

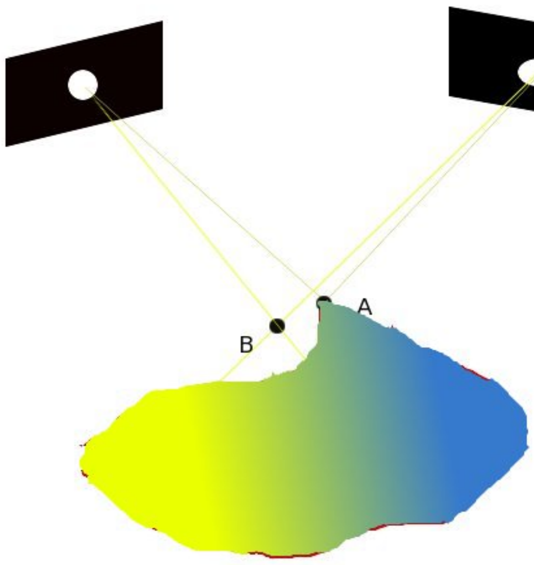

Figure 4.6: *Point A is a true surface point, so both cameras will observe approximately the same value in that point. Point B is a false surface point, the cameras will have different values there.*

A voxel, or a group of voxels, is assigned a photo-consistency value by projecting the voxels into all cameras that see it, and then comparing how similar the values are in the different images. In other words, we are comparing the projection of the voxels into different images. This is done for all voxels in the visual hull.

Now there are several ways we can compare the values we get in the in the different images. There are of course the direct comparison, we can use the average of neighborhood pixels, and a host of other simple statistical tools. In this thesis however, we will be using the Normalized Cross Correlation (NCC), as explained in chapter 2. The NCC method have shown very good

olding did just as well on the images used in this thesis results in [14] and [24] and should give good results for us as well. Details about how we actually calculate the photo-consistency values will be shown in the next chapter.

Chapter 5

Surface Reconstruction using Photo-Consistency

The basic idea of photo-consistency was given in the last chapter. We will now explain in more detail how we can use this idea to compute actual values for the voxels, and how this can be used in surface reconstruction. We will also look at some of the problems we encounter when calculating photo-consistency, and what tools we use to overcome this.

5.1 Photo-Consistency

We remember from the last chapter that in order to calculate photo-consistency values, we need an approximate shape, and that this shape is given by the visual hull. Now, the visual hull is an ideal place to start, as it not only is a good approximation of the shape, but it is also known to contain the entire true object. Strictly speaking, this is only true if the images are perfectly segmented, but if the images are reasonable well segmented it has little effect on the end result. This means that if a voxel is outside of the visual hull, we know that it is a part of the background. Thus, we can set it permanently to be part of the background. This reduces the risk of artifacts[1] in our reconstruction.

Once we have obtained the visual hull, we can start going through each voxel or block of voxels on the surface. We project each voxel into every camera that can see it. Using the values in the different cameras, we can calculate a score. If we let I_i be the image captured by camera i, and $\pi_i(x)$ the projection of surface points x into camera i we can write the value of

[1]Artifacts are erroneously reconstructed objects that can appear due to noise or other corruption.

points x projected into camera i as $I_i(\pi_i(x))$. Using this terminology we can write a very general pseudo algorithm for surface reconstruction using photo-consistency. See Algorithm 1.

Algorithm 1 Surface Reconstruction with Photo-Consistency

```
Create Visual Hull
while Reconstruction not complete do
  for All voxels p on the surface of object do
    Project p into all cameras in which it is visible, i.e find $I_i(\pi_i(p))$ for
    all cameras in which point p is visible.
    Calculate photo-consistency value for voxel p.
  end for
  Reconstruct object using updated photo-consistency.
end while
```

Although crude, this algorithm tells us what tools we need in order to accomplish our goal. The projection matrix and the visual hull have both been explained earlier, so that leaves us with only two more areas to explore, visibility of voxels, and the concrete calculation of photo-consistency when given the image values.

5.1.1 Visibility

Visibility is intuitively simple, but how to express it mathematically and implement it is not. We can create a visibility function associated with a given camera i, such that. $\chi_i(\xi) : \Omega \to [0, 1]$ where

$$\chi_i(\xi) = \begin{cases} 1 & \text{if } \xi \text{ is visible from camera i} \\ 0 & \text{if } \xi \text{ is not visible from camera i} \end{cases} .$$

But in order to know if a given voxel is visible from a camera, we not only need the position of that voxel and the camera, we need to know if there are any other voxels blocking the view. That means that for each voxel, we have to check the position of a large number of other voxels! This quickly turns out to be computationaly hard.

We therefore choose a simple and intuitive method. We partition our 3-D domain into different parts, and say that a voxel is visible in a camera if it is in the same region as the camera, or a neighboring region. This is shown in Figure 5.1. For example, a camera in region 1, can see voxels in region 1,2 and 8.

This visibility model might not be the most sophisticated, but simplicity is not to be frowned upon, and as we will see, the model achieves good results.

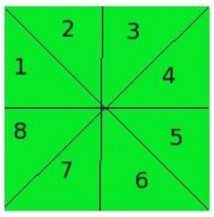

Figure 5.1: *Dividing our space into regions. A voxel is set as visible if it is in the same or neighboring region as the camera.*

5.1.2 Calculation using NCC

We now have all the tools we need to calculate the photo-consistency, so lets take a closer look at how this is done. As mentioned, there are several ways to do this, but we will focus on the calculation of photo-consistency using the normalized cross correlation described in Chapter 2. This method is known to give very good results.

As before, let I_i be image i, and $\pi_i(x)$ the projection of voxels x into image i. Then $I_i(\pi_i(x))$ is the projection of voxels x into image i. In other words, $I_i(\pi_i(x))$ is the subimage of image i in which we can see the part of the object where the voxels x are situated.

Given a set of voxels, we now project these into all cameras in which they are visible, and obtain several subimages, one from each image who can see the voxels.

We can now use the NCC to compare these images with each other, and find how similar they are. We do this by the method proposed by Kolev et.al. in [14].

Let $V \subset \mathbb{R}^3$ be the domain containing our scene and $S \subset V$ the surface of our reconstruction. For each $x \in S$ we can then express the photo-consistency on the surface by

$$C(x) = \frac{1}{N} \sum_j \sum_i NCC(I_i(\pi_i(x)), I_j(\pi_j(x))) \tag{5.1}$$

where N is the number of relevant camera pairs, and $i \neq j$. By relevant pairs, we mean that both cameras are in a region that can see the voxels.

This gives us a photo-consistency value for all the voxels on the surface. Our initial surface is of course the visual hull, and we could integrate the photo-consistency values calculated by the surface of the visual hull to get a new surface. Using the new surface we could once again calculate the

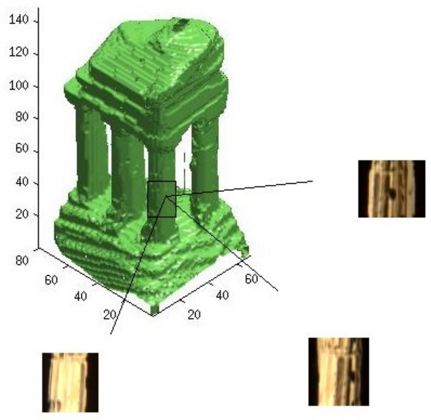

Figure 5.2: *Voxels from the column projected into 3 different images in which the voxels are visible. Number of voxels are exaggerated for visual convenience.*

photo-consistancy and so forth. This method would achieve decent results. However, we would like to go further, and use a probability based photo-consistency along visual rays, as done by Kolev et.al. in [14] and [15].

5.2 Probability Based Photo-Consistency

The main problem with just using the method described above is that it uses only the information on the surface. It could easily get stuck on a surface that looks good, not knowing that a even better surface lies beyond. Therefore, it is possible that the method might not capture concavities well, and could favor a surface reconstruction that would be close to the visual hull.

This is solved by propagating classic on-surface photo-consistency into values inside and outside of the volume. This gives us much more information about the volume, and makes us better equipped to find the best surface. See Figure 5.3.

Our ability to do this comes from the following basic property.

Property 1. *Let S be an arbitrary surface, which is consistent with the silhouettes of a set of given images $I_1, \ldots, I_n$. Then each visual ray passing through a point x in the interior of S intersects the real observed surface $\tilde{S}$ at least once.*

If there is a visual ray going through a point x, which does not intersect the real surface $\tilde{S}$, then that point x will not project within the silhouette

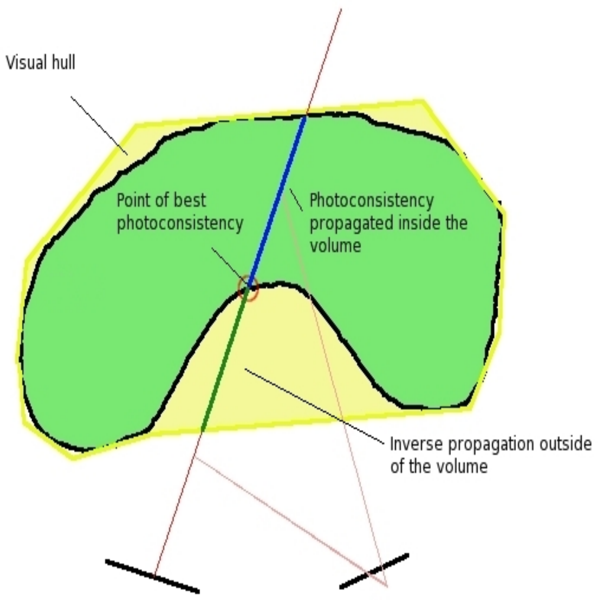
Visual hull
Point of best photoconsistency
Photoconsistency propagated inside the volume
Inverse propagation outside of the volume

Figure 5.3: *Propagation of photo-consistency. Voxels inside and outside of the volume are given probability values calculated using the point of best photo-consistency.*

of the image, and can therefore not lie inside a silhouette consistent shape, which the visual hull is.

This leads to the final idea. We can calculate photo-consistency along visual rays, and take the position of the maximum value as the point of intersection between the visual ray and the real surface $\tilde{S}$. Using this, we can convert classical photo-consistency which describes the likelihood of lying *on* the surface into likelihood of being *inside* or *outside* of the volume.

Let $r_j(x,t)$ be the visual ray of camera j, where t is the position, starting at the initial camera position, and let t_{cur} be the current position. Also, as usual, let $\pi_i(x)$ be the projection of x into camera i. We then measure photo-consistency along the ray j, according to another camera i as

$$C_i^j(x,t) = NCC(\pi_i(r_j(x,t)), \pi_j(r_j(x,t))). \tag{5.2}$$

Adding over several cameras, we get the final score for the visual ray of camera j

$$C^j(x,t) = \sum_{i=1}^{n} C_i^j(x,t). \tag{5.3}$$

There still is a problem in how to turn these values into probability values, namely that the NCC gives values between -1 and 1. In order to give the voxels a probability score, we need to turn these numbers into numbers in the range 0 to 1. This can be done by the following function from [23]

$$f(x) = 1 - \exp(-\frac{\tan(\frac{\pi}{4}(x-1))^2}{\sigma^2}), \tag{5.4}$$

where $\sigma \in [0, 1]$. In this thesis σ was set to 0.25, as this have been used to good effect by others

We can now determine the maximum photo-consistency along the visual ray, and the position where the maximum is found by using

$$C_{max}(x) = \max_t C^j(x, t), \quad t_{max}(x) = \arg C_{max}(x). \tag{5.5}$$

We now define interior and exterior photo-consistency along a ray j as

$$\rho^j_{obj}(x) = H(t_{max} - t_{cur}) \cdot (1 - f(C^j_{max})) + (1 - H(t_{max} - t_{cur}) \cdot f(C^j_{max})$$

$$\rho^j_{bck}(x) = H(t_{max} - t_{cur}) \cdot f(C^j_{max}) + (1 - H(t_{max} - t_{cur}) \cdot (1 - f(C^j_{max})).$$

Where H is the heavyside function

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$

and $f(x)$ as in Equation (5.4). Notice that $\rho_{obj}(x) + \rho_{bck}(x) = 1 \; \forall x \in V$, as we would expect from a probability based method.

5.2.1 Choosing the Size of the Visual Ray

Having defined photo-consistency along visual rays, a natural question arises. What size should the visual ray be? This is a classical case of having to weigh two options against each other. If we choose a small visual ray, we should be able to reconstruct more details, but it is harder computationally. Also, if the visual ray is too small, there will not be enough information in the images to compare them properly. On the other hand, if we choose a larger visual ray, we will experience faster computation, but less detailed reconstruction.

It is clear that we need the visual ray to be big enough to capture enough information from the images, otherwise the NCC will give us nothing useful, and the reconstruction will fail. Also, we need some details, so the visual ray can not be too large. It is therefore natural to choose something in between, the “golden mean”, as the philosophers would say.

We would of course like to be more precise than this, but what a suitable “golden mean” is will depend on the object being reconstructed. In practice we will mainly have to find the size of the visual rays through experiments.

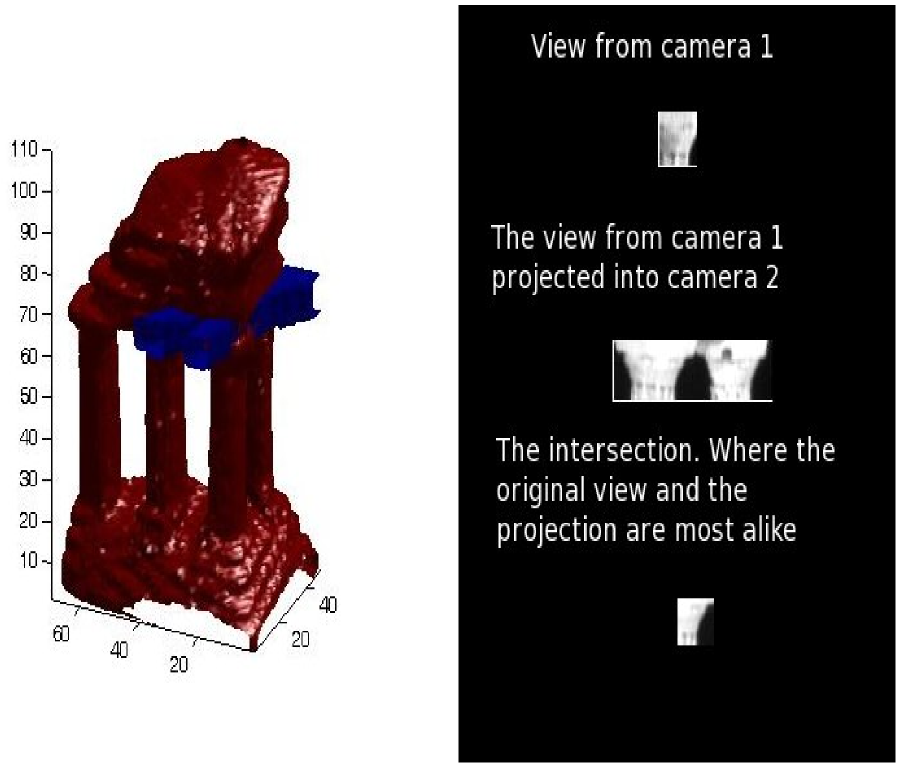
110
100
90
80
70
60
50
40
30
20
10
60
40
20
40
20
View from camera 1
The view from camera 1 projected into camera 2
The intersection. Where the original view and the projection are most alike

Figure 5.4: *Probability based photo-consistency. The highest probability is set at the intersection where the two beams meet. We see here how this intersection point is found. The view from one camera beam is projected into a second camera and NCC is used to find the location where they are most similar.*

5.2.2 Obtaining the Energy Functional

Having obtained expressions for background ($\rho_{bck} \in [0,1]$), foreground ($\rho_{obj} \in [0,1]$) and classical on-surface photo-consistency ($\rho \in [0,1]$), we can now construct an energy functional suitable for surface reconstruction. If we as usual let $V \in \mathbb{R}^3$ be a volume containing our scene and let S be a surface estimation, we can use S to divide the volume V into background and object, $V = R^S_{obj} \cup R^S_{bck}$. We can also assign each point in the volume a probability for being on the surface by using classical photo-consistency and turning it into probability by using Equation (5.4). Using this, we get the final optimization problem,

$$E(u) = \int_{R^S_{obj}} \rho_{obj}(x)\, dx + \int_{R^S_{bck}} \rho_{bck}(x)\, dx + \nu \int_S \rho(x)\, dx. \tag{5.6}$$

The surface we are trying to find, $\hat{S}$, is then

$$\hat{S} = \arg\min_{S \subset V} E(S).$$

Here $\rho_{obj}(x)$ and $\rho_{bck}(x)$ impose correct division into object and background and the last term, $\rho(x)$, seeks a minimum surface with respect to photo-consistency, and can therefore be seen as a smoothing constraint. The parameter $\nu \in [0, 1]$ is there to regulate this smoothing property.

5.3 Global Minimization

Having described our problem as an energy functional needing to be minimized, we must take a look at the properties of such functionals. A common problem in variational image processing techniques is that the energy functional needing to be minimized has local minimums, and can get stuck there. This is a big problem, because these local minimums often have the wrong level of detail and scale, and in fact, the local minimum might not look like the real solution at all!

Adding to our problems, the solutions of the variational problems we encounter in image processing are often based on gradient descent methods, and these are quite sensitive to local minimums. Therefore, we either need a very good initial guess, or we need a way to avoid local minimums.

From basic calculus, it is well known that all the minimums of a *convex* function are global minimums. Thus the idea arises, if we can somehow force the energy functional to become convex, without losing its essential properties, we can minimize without fear of local minimums.

5.3.1 Global Minimization of Binary Images

We can look at surface reconstruction as a binary problem. We want to partition our domain V into either background or object. We are given a shape, the visual hull, that consists of the "real" shape with some added distortions, and we want to recapture the original shape. In 2-D this is called *binary image denoising* and have been studied quite a lot, and even though we are in 3-D, some of the tools developed for the 2-D case might be applicable.

As mentioned, in binary image denoising the noise manifests itself as pertubations in the geometry of the shape of the object, and the goal is to find the true object. This is just as in our reconstruction case, and it is therefore not far fetched to seek inspiration from the binary image denoising field. We will not go into details about binary image processing here, it will suffice to mention one article from Chan et al [5], which ideas we will utilize.

Given a optimization problem, it is, surprisingly, harder to minimize in the binary case than it would in the normal case. This is because the space of binary functions is non-convex, and thus the optimization problem itself becomes non-convex for the binary case. However, in [5] it was shown that by introducing a convex penalizer, we can for many cases turn the problem into a convex one.

This is due to the fact that when we minimize the total variation norm of the space of all functions, $u : \Omega \to \mathbb{R}$, the values of $u(x)$ converge to $\pm\infty$ almost everywhere. Hence, we can get a convex problem on the convex space of functions $\Omega \to [0, 1]$ by enforcing $0 \leq u(x) \leq 1$, using the convex penalizer in [5]

$$\theta(u) := max \left\{ 0, 2|u - \frac{1}{2}| - 1 \right\}. \tag{5.7}$$

Now, if we minimize over the space of the real valued functions, and then threshold the result, we will get a global minimizer for the original non-convex problem.

5.3.2 Ensuring Global Minimization in Our Model

As discussed above, [5] gave us a framework for constructing convex minimization problems out of non-convex ones. In this section, we will follow [15] and prove that the same technique can be applied to our case, and that we can formulate the reconstructing problem as a convex optimization problem.

Remember that our problem is the minimization of Equation (5.6). If we wish to formulate this as a convex optimization, we need to represent the surface S implicitly by the characteristic function $u : V \to \{0, 1\}$. In other words, $u = 1_{R^S_{bck}}$ and $1 - u = R^S_{obj}$. Using this, we can write the original functional, Equation (5.6), as

$$E(u) = \int_V (\rho_{bck}(x) - \rho_{obj}(x))u(x)dx + \nu \int_V \rho(x)|\nabla u(x)|dx, \tag{5.8}$$

such that $u(x) \in \{0, 1\}$. Now, as noted above, the space of binary functions is non-convex, and thus our minimization problem is also non-convex. If we now forget about the binary condition, and instead let u take all values, $u(x) : V \to \mathbb{R}$, we will, as above, get that $u(x)$ converges to $\pm\infty$ almost everywhere. To get around this, we keep $0 \leq u(x) \leq 1$ by adding the convex penalizer $\theta(u)$ from Equation (5.7). This gives us

$$E(u) = \int_V (\rho_{bck}(x) - \rho_{obj}(x))u(x)dx + \int_V \nu\rho(x)|\nabla u(x)| + \alpha\theta(u)dx, \tag{5.9}$$

where we choose α big enough, such that we are ensured $u(x) \in [0, 1]$. This gives us a convex minimizer. Although note that it is not strictly convex, i.e. the minimas are not unique.

The point of all this is that a minimizer of the convex problem in Equation (5.9) is also a minimizer of the original non-convex problem in Equation (5.8)! This is given from the following theorem in [15], based on the work in [5].

Theorem: 1. *If* $u : V \rightarrow \mathbb{R}$ *is any minimizer of the functional in Equation (5.9), then for any threshold* $\mu \in (0, 1)$*, the binary function* $\mathbf{1}_{\Sigma_\mu u*}(x) : V \rightarrow \{0, 1\}$ *with* $\Sigma_\mu(x) := \{x : u(x) > \mu\}$ *is also a minimizer of Equation (5.9)*

Proof. See [15] $\square$

Now, the theorem does not mention the non-convex problem in Equation (5.8), but any thresholded minimizer of the convex problem in Equation (5.9) must be binary, and since Equation (5.9) fulfills the minimization criterion of Equation (5.8), we have achieved a convex binary optimization. This method can be summarized in two steps:

1. Find a minimizer of the convex problem in Equation (5.9).
2. Threshold the result from 1 with $\mu \in \{0, 1\}$.

So, now the problem becomes how to minimize the convex problem in Equation (5.9). We do this via the piecewise constant level set and gradient descent methods.

5.4 Minimization by Piecewise Constant Level Sets and Gradient Descent

We now wish to use the piecewise constant level set method to solve Equation (5.9). From the discussion about this method in Chapter 2, we get the following problem

$$E(u) = \int_V (\rho_{bck}(x) - \rho_{obj}(x))u(x)dx + \int_V \nu\rho(x)|\nabla u(x)| + \alpha\theta(u)dx$$

subject to

$$K(u) = 0,$$

where $K(u) = u(u - 1)$. This is a constrained optimization problem and to solve it we use the method of Lagrangian multipliers as described in Chapter

2. Expanding our functional with a Lagrangian mulitiplier, we get that a minimizer of $E(u)$ corresponds to a saddle point of the following functional

$$H(u, \lambda) = E(u) + \int_V \lambda K(u)\, dx \tag{5.10}$$

We know that at a saddle point, the following must be satisfied;

$$\frac{\partial H}{\partial u} = 0\ , \quad \frac{\partial H}{\partial \lambda} = 0. \tag{5.11}$$

What we want to do is to minimize H with respect to u, and maximize H with respect to λ. To do this, we use a gradient descent method. The derivatives of the functional H will give gradient information, and using these gradients, we can decrease or increase the functional value. Moving in the negative of the gradient direction will decrease the functional value, and so we will do this for u. Similary, moving in the positive gradient direction will increase the functional value, and this will be done for λ.

Normaly, this would be done with a small step size and would only give local information. We could not know if we had converged at the right solution or if we were caught in local minimum or maximum. However, since we have turned our functional into a convex one, we know that all local minimums and maximums are global, and thus we have no fear of getting caught in one. This allows us to be less careful with the inital values u_0 and λ_0.

To do the minimization, we introduce an artificial time parameter, t, and define the following

$$\frac{\partial u}{\partial t} = \frac{\partial H}{\partial u}\ , \quad \frac{\partial \lambda}{\partial t} = \frac{\partial H}{\partial \lambda} \tag{5.12}$$

with $u = u_0$, $\lambda = \lambda_0$ at time $t = 0$. We now want to minimize $\frac{\partial u}{\partial t}$ and maximize $\frac{\partial \lambda}{\partial t}$.

We remember from Chapter 2 that in order to solve the original constrained problem we have to find the extremum points of the augmented Lagrangian functional. To do this we need to find the *weak derivative.* To illustrate how this is done, we will show the differentiation for the most difficult part of our functional, $\int_V \rho(x)|\nabla u(x)|dx$. If we define that term to be f, what we need to do is find a g such that

$$\int_V f \frac{\partial h}{\partial x_i} dx = -\int_V g_i h\, dx\,,$$

for all $h \in C_0^\infty(V)$. Such a g will be the weak derivative of f.

$$
\begin{aligned}
d[\int_V \rho|\nabla u|dx](u;h) &= \lim_{\tau\to 0}\int_V \frac{\rho|\nabla(u+\tau h)| - \rho|\nabla u|}{\tau}dx \\
&= \int_V \lim_{\tau\to 0}\frac{\rho|\nabla u+\nabla\tau h| - \rho|\nabla u|}{\tau}dx \\
&= \int_V \lim_{\tau\to 0}\frac{\rho|\nabla u+\nabla\tau h| - \rho|\nabla u|}{\tau}dx \cdot \frac{\rho|\nabla u+\nabla\tau h| + \rho|\nabla u|}{\rho|\nabla u+\nabla\tau h| + \rho|\nabla u|} \\
&= \int_V \lim_{\tau\to 0}\frac{\rho^2|\nabla u+\nabla\tau h|^2 - \rho^2|\nabla u|^2}{\tau\rho[|\nabla u+\nabla\tau h| + |\nabla u|]} \\
&= \int_V \lim_{\tau\to 0}\frac{\rho^2[(\nabla u)^2 + 2\nabla u\nabla\tau h + (\nabla\tau h)^2 - (\nabla u)^2]}{\tau\rho[|\nabla u+\nabla\tau h| + |\nabla u|]} \\
&= \int_V \lim_{\tau\to 0}\frac{\rho^2[2\tau\nabla u\nabla h + \tau^2(\nabla h)^2]}{\rho[2\tau|\nabla u| + \tau^2\nabla h|]} \\
&= \int_V \rho\frac{\nabla u\nabla h}{|\nabla u|}.
\end{aligned}
$$

We know we can write $\nabla h = (\frac{dh}{dx_1}, ..., \frac{dh}{dx_i})$. We also define a vector

$$\psi = \rho\frac{\nabla u}{|\nabla u|} = [\psi^{(1)}, ..., \psi^{(n)}]$$

and use standard integration by parts. We then get that for each direction i,

$$\int_V \frac{dh}{dx_i}\psi^{(i)}dx = -\int_V h\frac{d\psi^i}{dx_i}\, dx + \int_S h\psi^{(i)}\mathbf{n}(i)\, dx,$$

where $\mathbf{n}(i)$ is the outward unit surface normal and S is the surface of the volume V. Since $h \in C_0^\infty(V)$, the last term disappers. If we do summation over all i, we get that

$$\int_V \rho\frac{\nabla u\nabla h}{|\nabla u|} = \int_V \operatorname{div}(\rho\,\frac{\nabla u}{|\nabla u|})\, h\, dx.$$

Which means we have found the weak derivative for that term. Doing this for all the terms in our functional, we end up with

$$\int_V h[(\rho_{bck} - \rho_{obj}) - \nu \operatorname{div}(\rho\frac{\nabla u}{|\nabla u|}) + \alpha\theta_\epsilon'(u)]\, dx = 0.$$

For this to be zero, we only need whats within the integral sign to become zero. Since h is in every term, we can remove it from the equation. What we end up with is the Euler-Lagrange equations for our problem.

$$(\rho_{bck} - \rho_{obj}) - \nu \operatorname{div}(\rho\frac{\nabla u}{|\nabla u|}) + \alpha\theta_\epsilon'(u) = 0 \tag{5.13}$$

$$K(u) = 0 \tag{5.14}$$

The first equation is not well defined at points where $\nabla u = 0$, due to the term $1/|\nabla u|$. It is common to overcome this problem by a slight perturbation of the $|\nabla u|$ term. We say

$$\frac{1}{|\nabla u|} \approx \frac{1}{\sqrt{|\nabla u|^2 + \beta}}$$

where β is a small positive parameter. It is shown in [1] that the solution of the perturbed problem converge to the solution of the original problem when $\beta \rightarrow 0$. Using this, we get the following equations to be solved

$$\frac{\partial u}{\partial t} = (\rho_{obj} - \rho_{bck}) + \nu \operatorname{div}(\rho \frac{\nabla u}{\sqrt{|\nabla u|^2 + \beta}}) - \alpha \theta'_{\epsilon}(u) \tag{5.15}$$

$$\frac{\partial \lambda}{\partial t} = K(u) \tag{5.16}$$

These coupled equations are solved by a modified version of the algorithm from the original piecewise constant level set method [19].

Algorithm 2 Piecewise Constant Level Set Method using Gradient Descent

```
Set $u_0$ to visual hull
Initialize $\lambda_0$
Calculate $\rho_{obj}$, $\rho_{bck}$ and $\rho$
$\epsilon = \text{tol}$                                  //Set the tolerance
while $|u_i - u_{i-1}| > \epsilon$ do
    Update $\rho$ given new surface S                   //Updating photo-consistency
    $u_i = u_{i-1} + \Delta t \frac{\partial H(i_{i-1}, \lambda_{i-1})}{\partial u}$          //Updating level set using G.D.
    $\lambda_i = \lambda_{i-1} + rK(u)$                   //Updating Lagrangian multiplier
    $i = i + 1$
end while
```

This algorithm will solve our minimization problem and will give us our final reconstruction. Note that the probability based photo-consistency terms, ρ_{obj} and ρ_{bck} are only calculated once. This is because these terms are calculated using only the information in the images and the corresponding projections, and not the surface itself, so these do not change, while the standard photo-consistency term, ρ, is dependent on the surface, and therefore has to be updated in each iteration.

Chapter 6

Results

Having described and explained all the tools and methods we are using, we can now take a look at how this turns out in practice.

6.1 Image set

Making our own image sets would be very complicated, so we have tried our method on a image set from the Middelbury Vision Group [21]. This set comes with given camera set up, so there is no need to do camera calibration, we can just follow the description from Chapter 3 on how to make a projection matrix and use it.

Our image set is the sparse temple grid, which consists of 16 images of the temple of the Dioskouroi in Sicily. See Figure 6.1.

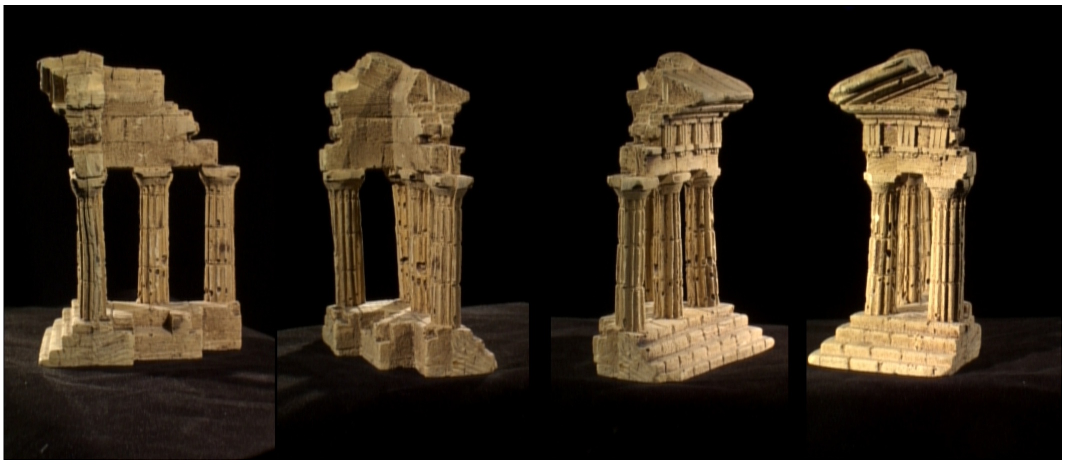

Figure 6.1: *Four of the original sixteen temple images.*

6.2 Visual hull

The first step in our method is to construct the visual hull. In order to do this, we must first define a rectangular grid for us to work in. The size of this grid is known as the *resolution* of the model. A large grid gives more detail and thus better resolution, but a large grid is also harder computationally. The opposite is of course true for a small grid. We are therefore once again faced with the problem of quality vs computation. What to choose depends on the application of the result. In practical application it is often necessary to sacrifice some quality for speed, but since the purpose of this thesis is mere presentation, we can choose quality. Still, we are working in 3-D, and a small increase in the resolution leads to a large increase of data points. Thus we quickly find ourselves at the edge of what our computer memory can manage. The unfortunate consequence of this is that we can not achieve the resolutions we would have liked, and hence quality may suffer. For the remainder of this thesis, unless specified otherwise, we use use the resolution $90 \times 120 \times 70$ for the model.

Having defined our grid, we are ready to start the reconstruction. As we remember from Chapter 3, in order to construct the visual hull, we first need to segment the images in the sets properly.

6.2.1 Segmented Images and Projection into 3-D

As explained, we have chosen to use the Chan-Vese method to segment our images. This method works fairly well on our images. Once the segmentation had been achieved, we used the projection matrices to project the silhouette into 3-D. The result can be seen in Figure 6.2.

6.2.2 Visual hulls

Using the segmented images, we can now construct the visual hull for our image set. Having noticed that the segmentation is not perfect, we loosen on the constraints a bit. Instead of demanding that all images agree on the location of the object, which is the standard visual hull algorithm, we say that only 14 of the 16 images must agree, thus allowing for some error in the segmentation.

We see the computed visual hulls in Figure 6.3.

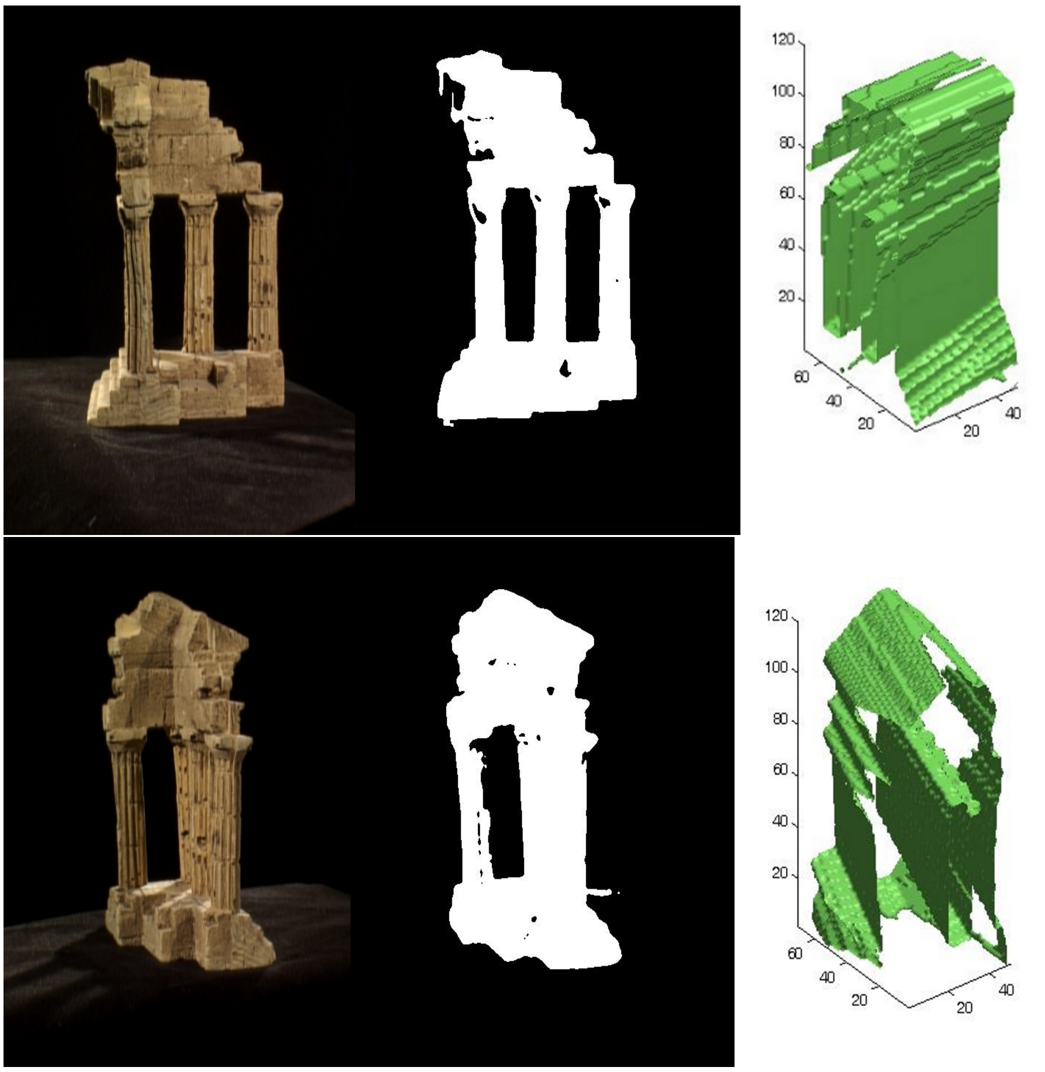

Figure 6.2: *Samples of the original images, the segmented images and the projection of the silhouettes into 3-D.*

6.3 Final Version

Having obtained the visual hull, we can now use the probability based photo-consistency, and our gradient descent method, to complete the reconstruction.

The temple in the image set has many features that makes for a good visual hull. The columns in particular are easy to recreate for the visual hull, and this gives us a very good initial reconstruction. In fact, the only big flaw of the visual hull is the failure to capture the concavity on the backside of the

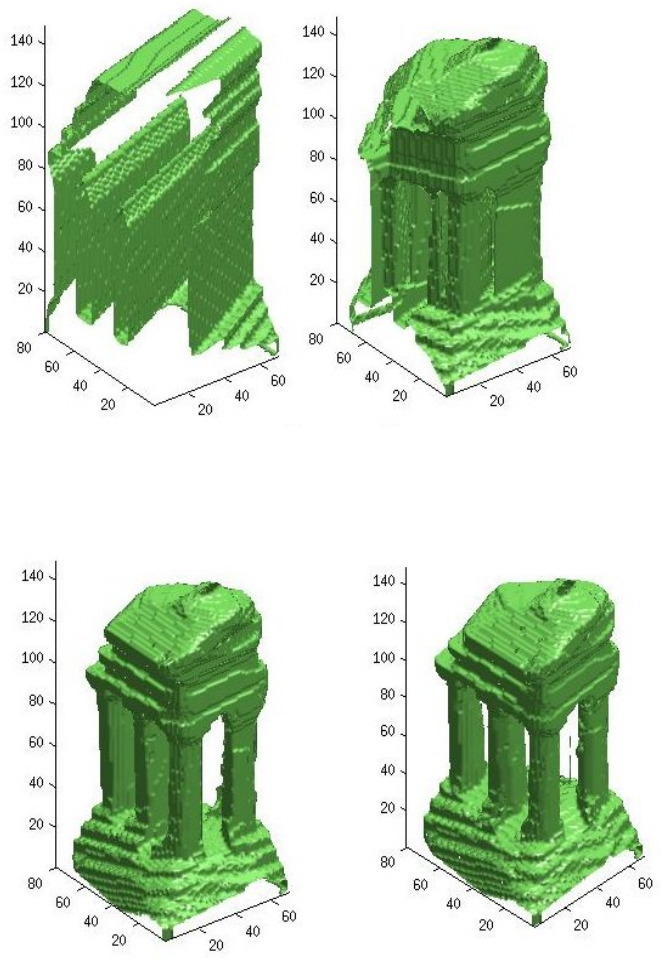

Figure 6.3: *Visual hulls. From left to right, 1,5,10 and 16 viewpoints. Resolution:* $140 \times 80 \times 70$

temple. This concavity will have to be identified by the photo-consistency terms.

We set the smoothing parameter ν to be 10^{-5}. We then calculate the photo-consistency terms ρ_{obj} and ρ_{bck} using a template of 20×20 pixels, moving the template 4 pixels at a time. Having done this, we use Algorithm 2 from Chapter 5 to solve the system. We see the results of the temple reconstruction in Figure 6.4.

These results are quite good. We have captured all the major characteristics of the temple. Especially we have captured the concavity on the back of

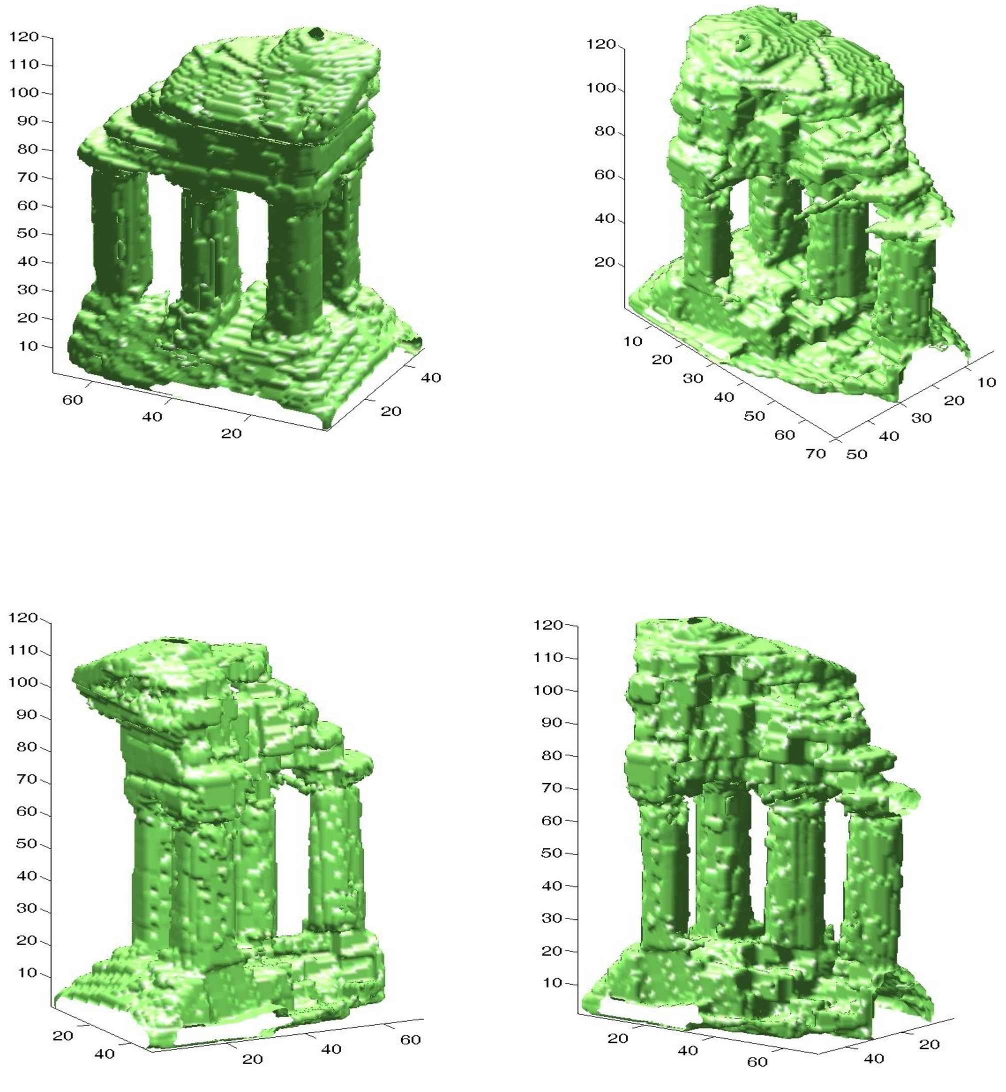

Figure 6.4: *The final reconstruction. Notice the capture of the concavity. Resolution:* $90 \times 120 \times 70$.

the temple very well. We notice there is a lack of small details. The pattern below the roof has not been reconstructed and neither has the stairs on the side of the temple. The pattern can be explained by our low resolution. The stairs on the side however, are harder to explain. We have reconstructed the stairs on the front of the temple, so why are we not able to reconstruct the stairs on the side? It would be easy to point to our simplistic visibility function, but it seems this is not to be blamed. To test the visibility function, we ran reconstructions using only the images that can see those particular stairs and we got the same results as before. One possible reason is that the stairs on the side is much longer than those in front. This makes the edge less noticeable, and could make the reconstruction of those stairs more difficult than the others.

It is unfortunate that we are unable to provide reconstructions of higher resolution. We are certain that we would have captured the pattern and other lesser details at a higher resolution. It is also very likely that a higher resolution would have resolved the stairs issue.

6.4 Quality of Reconstruction

Having reconstructed the object, we would like to get an idea of how good our reconstruction is. This is not the easiest task, as it is difficult to determine the quality of a reconstruction. A horrible reconstruction might be easy to identify, but telling the difference between a good and a very good reconstruction is far worse.

Knowing this, the Middelbury vision group [21] obtained the "ground truth" of the object in our image set using laser scanners. They then created a web page for comparing different methods. A researcher may submit their results to this web page and see how it compares to other methods and the ground truth. Unfortunately, the submitting process is not straightforward, so we will not be submitting the results from this thesis. However, we can use the results from known models to do a comparison with our own. This will give us a fairly good idea of how good our reconstruction is. Keep in mind that the reconstructions we are comparing with are of higher resolution than we can achieve on our computer, so we should not expect the same level of detail.

In Figure 6.5 we compare our temple reconstruction with some of the results from the web page and the ground truth. The results are from Furukawa [9], Delaunoy [7], Auclair [4] and Starck [22]. The percentages are taken from the Middelbury web page.

Although our model is certainly not the best, it can easily stand the comparison with the other models. This must be said to be fairly good. We are after all comparing it with the best results out there. And we must remember that our reconstruction has much lower resolution than the others. We don't know the exact resolution of the models we are comparing with here, but [14] uses a resolution of $256 \times 384 \times 192$ and it is likely that the others use

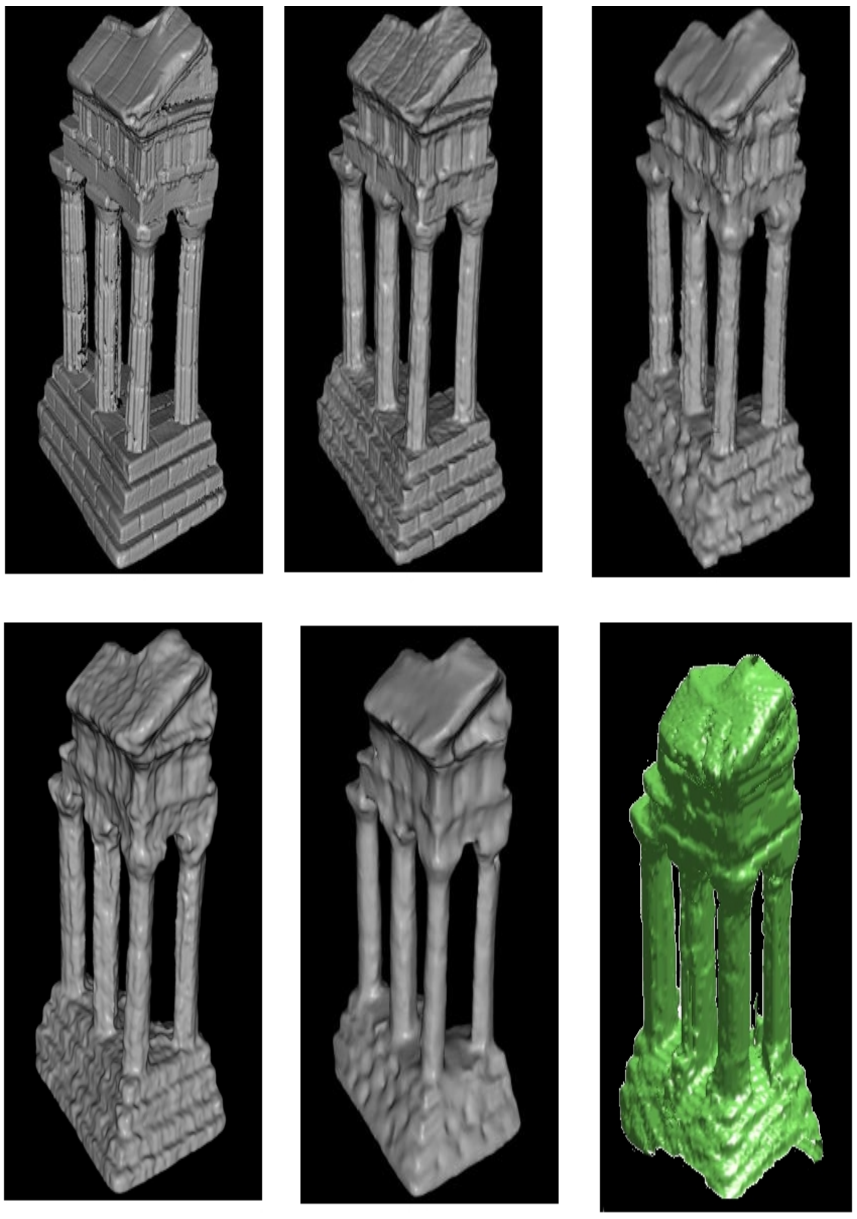

Figure 6.5: *Top left to right; Ground truth 100 %, Furukawa 99,3% , Delaunoy 95,9%. Lower left to right; Auclair 92,5% , Starck 87,7%, Our model. The percentage reflects how many of the voxels are within 1.25 mm of the ground truth mesh.*

a similar resolution. This is huge compared to our resolution of $90 \times 120 \times 70$. In fact, it's over 18 million more points! This explains the difference of the level of detail in the models. To quantify a reconstruction percentage for our model is hard, but it seems to be at least above the 80% mark.

6.5 Effect of Resolution

We have mentioned a couple of times that our inability to use a high resolution in the reconstruction makes it difficult to handle small details. To see how much effect resolution has on details, we reconstructed the temple using different resolutions.

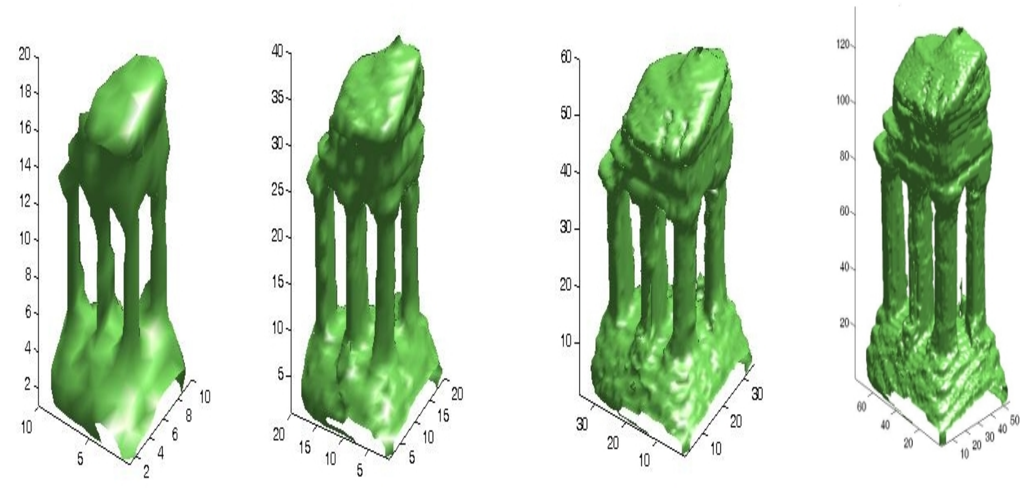

Figure 6.6: *The reconstruction using different resolutions. Resolution, from left to right:* $10 \times 20 \times 10$*,* $20 \times 40 \times 20$*,* $40 \times 60 \times 40$*,* $90 \times 120 \times 70$*.*

As we see in Figure 6.6, the resolution has a big effect on the reconstruction. It is clear that the lower the resolution, the less detail we have.

6.6 Robustness of Algorithm

The ultimate goal of surface reconstruction is to let any person take a few photos of any object, and then reconstruct that object in 3-D. If this is to be achieved we must create algorithms that are robust, that is, not sensitive to errors in the information we use. We can not expect perfect images and perfect camera calibration outside of laboratory conditions, and we must therefore test how our algorithm fares under less then ideal conditions.

6.6.1 Error in Images

One of the most common problems with images is the inclusion of noise. Noise is some kind of disturbance, corrupting the value of the pixels in the image. This results in loss of information and a lesser visual appeal. See Figure 6.7.

There is all kinds of different noise, but in the image processing community it is usually assumed that the noise is *additive noise*. That is, noise that has been added to the original image. This is written

$$I_{observed} = I_{original} + n$$

where n is the noise. Noise removal is a large field in image processing, with many different method. It is likely that by using additional noise removal tools in our algorithm, we could be able to deal with almost all kinds of noise without losing much detail in the final reconstruction. However, for the sake of testing robustness, we will not be using any noise removal methods on the images, except for the standard smoothing of images before segmentation.

We add Gaussian noise to the images. Gaussian noise is just random noise, Gaussian distributed, and independent in each pixel. We find the average of the variance of the original noiseless images, and then use half of that variance in our Gaussian noise. This gives us a 2:1 signal to noise ratio. This is considered a reasonably large amount of noise.

As we can see in Figure 6.7 the segmentation of the images is relatively good. Some of the reason is our segmentation method, but we have to admit that the images used in this thesis are fairly easy to segment, even with the added noise. We could of course make the segmentation harder by adding a difficult background, but we do not wish to spend too much time on the segmentation problem. What is more interesting is how the photo-consistency term copes with noise.

We see the reconstruction from the noisy images in Figure 6.8. The reconstruction is quite good considering the amount of noise in the images. There are some irregularities, but not many.

One of the reasons we get such a good reconstruction even with noisy images is that the noise is random. Since we are using several images to find the photo-consistency value at each point, it is probable that the information lost due to noise in one image, will be there in another. If we had structured

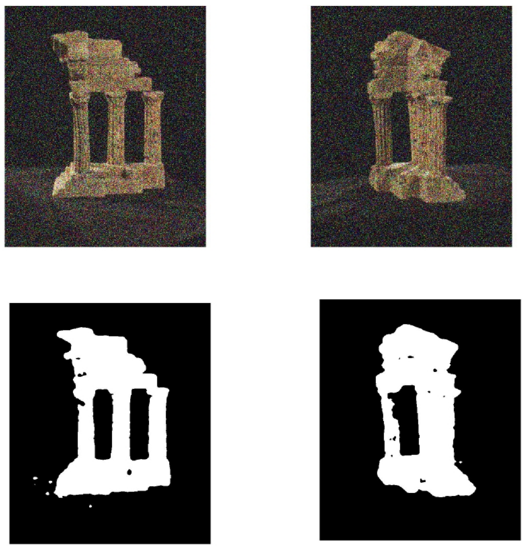

Figure 6.7: *Images with added Gaussian noise and the resulting segmentation. Mean of the noise was set to zero, and the variance to half of the average variance of the noiseless images.*

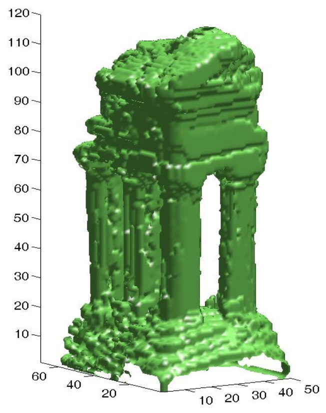

Figure 6.8: *The reconstruction of the temple using noisy images. We notice some distortions on the stairs, the columns and the roof. Otherwise the reconstruction looks fine.*

noise that corrupted the same ares in each image, our reconstruction would have suffered much more. This test shows that our reconstruction, due to its use of multiple images in each point, is quite robust against random noise.

6.6.2 Error in the Projection Matrix

Having tested out algorithms robustness against noise, we will now see how robust it is regarding errors in the projection matrix. We can not expect the algorithm to handle this very well. The projection matrix tells us which pixels corresponds to which voxels, and too much corruption in this data will lead to a total chaos. However, it is interesting to see how much corruption, or noise if you will, we can handle.

It might be tempting to just add a random matrix to the projection matrix, but this is not the kind of error we would expect in the real world. It is more likely that the error would be in the position of the camera, that is, in the rotation and translation matrices. We will therefore now corrupt the rotation and translation matrices, and use the projection matrix that results from these corruptions.

In contrast to when we had image noise, we do not need to run the entire reconstruction algorithm, it is enough to create the visual hull. This is because the corruption is in the projection itself, and the visual hull is the intersection of the projections. Therefore the visual hull gives us a lot of information about the projections. A good visual hull tells us that the projections are useable for reconstruction, while a bad visual hull of course tells us that the projections are not useable. To calculate the size of the corruption in the matrices we use the relative error,

$$\eta = \frac{|u - u_{approx}|}{|u|}.$$

The results of this test can be seen in Figure 6.9.

We see that the method copes well when there is just errors in the rotation matrix. Errors in the translation matrix are more difficult though, it can not handle more than a small relative error of 2.0%. And it is even worse when we combine the two errors. A relative error of around 3% on both matrices completely ruins the reconstruction. This tells us that we need accurate data when calculating the projections, or else the reconstruction will fail.

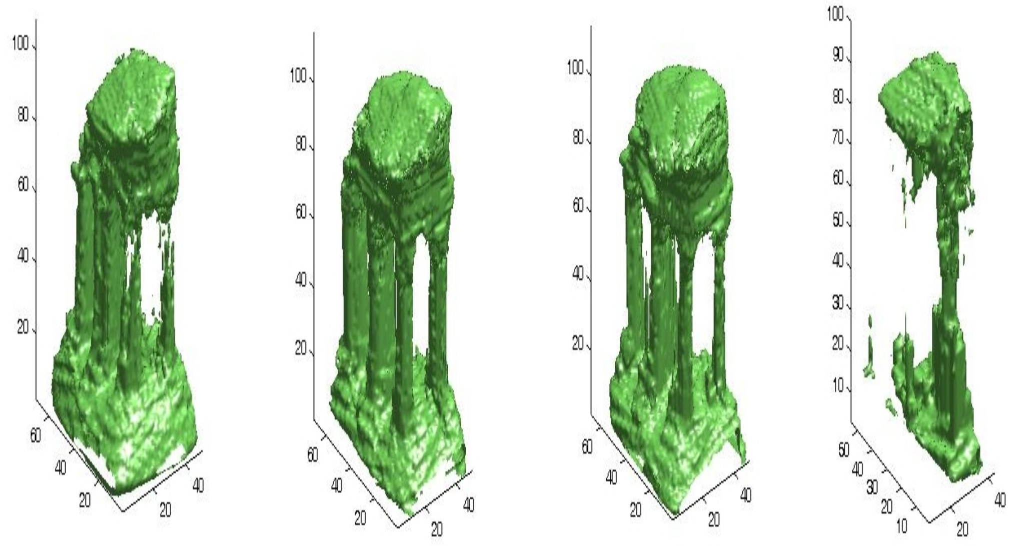

Figure 6.9: *Four visual hulls constructed using corrupted rotation and translation matrices. Relative error, from left to right; Rotation:* 10.0% *Translation:* 0% , *Rotation:* 0% *Translation:* 2.0%, *Rotation:* 1.04% *Translation:* 0.81%, *Rotation:* 3.09% *Translation:* 2.5%.

Chapter 7

Summary and Discussions

7.1 Summary

We have in this thesis looked at the 3-D reconstruction problem. We have shown that we are able to do 3-D reconstruction by using an initial visual hull, probability based photo-consistency along visual rays and the piecewise constant level set method. We used the method first introduced by Chan et.al in [5] to construct a convex functional, making the minimization process easier. We also tested the robustness of our algorithm. We found that we had good robustness against noise in the images, but less so regarding error in the position of the cameras. In order to do the reconstruction we had to assume that our object had a Lambertian surface, as this assumption is the basis for calculating photo-consistency. We know this is not the case in the real world, but it is usually not very wrong. In the end, we achieved very good results for our temple reconstruction. There was some small details missing, but this is most likely due to low resolution. It is clear from this that the probability based photo-consistency method works very well, and we see no reason why it should not be used in the future.

We have not mentioned anything about the running time of our code. This is because the time varies quite a lot, depending on different parameters. But to give an idea of the timescale we are working on, we can tell that the final reconstruction in Chapter 6 took over 5 hours to complete on our slow computer. This is too slow to be usable for industrial purposes, but the running time is by no means optimized, and there are methods to significantly reduce the running time.

7.2 Future work: Improving the method

When writing a thesis, there are always something you would have liked to do, but never had time to. This section is devoted to exploring what could have been done, if more time was available, to improve our method.

7.2.1 Visibility Function

The visibility function used in this thesis was very basic. We simply divided the grid into different areas, and assigned visibility based on this. This worked fine on the temple reconstruction in this thesis, but it might fail on more advanced objects. To improve upon our visibility function would have been a top priority if there was more time.

Using an initial surface like the visual hull, it has been proposed to use the euclidean distance of a point to its nearest surface point to calculate visibility. This is a direction we would have liked to explore.

There is also the option of choosing not to use a visibility function. In [24] Vogiatzis et.al. claim to not need a visibility function, due to a use of a robust photo-consistency metric.

7.2.2 Speed

Speed is always an issue when it comes to computational mathematics. We want our methods to work fast and well, but these two criterions are often in conflict. As mentioned before, we have in this thesis sacrificed some speed to attain better quality. But even when choosing good quality there are some ways to speed up the algorithm.

Graph cuts is a specialized optimization method used to compute the maximum flow in a network. It is a much faster method than gradient descent and have been used with success by many in the image processing community. Although the visibility term would have posed a challenge, we believe it would have been possible to use the graph cut method on the model in this thesis. The reason we choose not implement the graph cut method was that the main time consumption in our algorithm was not the minimization of the functional, but the calculation of the photo-consistency terms. Therefore, it was not that much to gain by using graph cuts.

Another idea to improve speed is to replace the gradient descent with an iterative linear solver. Our functional is almost linear, with $\frac{1}{|\nabla u|}$ being the only source of non-linearity. If we set an initial u_0 we get a system of linear equations which we can solve. We can then compute a new u_1, and solve the resulting new system of linear equations. This can be done until we we have $u_i - u_{i-1} <$ tolerance. This might sound like it would take a lot of time, but systems of linear equations can be solved very fast, especially by methods utilizing the Krylov subspace[1], like GMRES. The interested reader may consult any book about numerical linear algebra for details.

Other ideas to improve speed would be implementation of a adaptive step length in our gradient descent, or the use of quasi-Newton method instead of gradient descent.

To reduce the calculation time of the photo-consistency terms we would have to employ a *multi-resolution scheme*. These as so-called pyramid methods, that first apply the algorithm on a coarse scale, and then gradually move into the smaller scale containing the details. If we had the time and knowledge to implement this properly, it would reduce running time considerably.

7.2.3 Testing on Other Image Sets

We have only seen the reconstruction of the object from the temple image set. It would have been interesting to test our algorithm on other image sets, with other properties. In particular, an image set containing multiple objects would have been interesting. It is our belief that due to the visibility function, and the piecewise constant level set methods ability to deal with multiphase, that the reconstruction of multiple objects would pose no problem for our method.

Another interesting image set would be of a semi-transparent item. We remember that we assumed a Lambertian surface, but it would be interesting to see how dependent we were of this assumption. If we are able to segment the item properly, we believe that there is a good chance to achieve a decent reconstruction, even with a semi-transparent item.

7.3 Future work: Obtaining the “Final Goal”

In the last section we discussed how we could improve on the method used in this thesis. And although it is important to find ways to improve old methods, we feel that this is not where the main focus should be in the future. Our reconstruction result was good, and we know from the work of others that it is possible to create very good reconstructions in a reasonable fast time. Therefore, it is time to move away from the theoretical lab reconstruction, where almost all information is given. We should instead focus on achieving the *final goal* of multi view reconstruction. That is:

[1]A Krylov subspace is a linear subspace spanned by a vector b and a $n \times n$ matrix A $\mathbf{K}_r = \text{span}\,\{b, Ab, A^2b, ..., A^{r-1}b\}$

> *"To be able to reconstruct any object taken by any camera under any conditions"*

Surely a daunting task, but it should be achievable, especially with the help of new technology. We have in this thesis learned what we need in order to reconstruct an object. What we need is:

- Pictures of the object
- Camera calibration matrix
- Rotation matrix
- Translation matrix

The first one is trivial and goes without mentioning. The second one, the camera calibration matrix, is a bit more tricky. However, it is possible to find the camera calibration matrix for any camera by using standard camera calibration techniques. And once found, we do not need to calibrate that camera again. The camera calibration matrix is inherent in the camera, and does not change if the object changes. Therefore, acquiring the camera calibration matrix is not a problem for anyone interested in making 3-D reconstructions.

That leaves us with the last two, the rotation and translation matrices. These are not easy to find, and this is where the research focus must be. Finding these automatically and without error is unlikely without any additional technology. We will therefore first take a look at what can be done without any additional technology, and then a look at what might be done with the help of some technology. Please keep in mind that these are just untested ideas, which may or may not work.

7.3.1 Finding the Projection Matrix Semi-Manually

The underlying idea here is that we humans know what the reconstructed object should look like. When we see the pictures of the object, we construct a 3-D model of it in our minds. Surely this additional information can be used.

The idea we propose is this. Define a suitable grid. Then project the first image directly into 3-D space without any translation or rotation. This first projection will then serve as our reference point for the translation and rotation of the other projections. We then take the second image and using a 3-D visualization program we manually rotate and translate the 3-D space until we find the angle from which the second image was taken. The translation and rotation away from the original projection is now known by the 3-D visualization program. And since all such 3-D visualization programs use matrices to do the translation and rotation, we just need to extract these matrices from the program and we have all the information we need to do the projection for image number two. This can be done for all the images, and we will then have all the information we need to do the reconstruction. This can be summarized in a few simple steps:

- (1) Define suitable 3-D grid
- (2) Project image number one into 3-D space, with translation and rotation matrix as identity
- (3) Using image number one as reference, rotate and translate 3-D space manually until the view corresponds with the view from image number two. See Figure 7.1.
- (4) Extract the rotation and translation matrix from the 3-D visualization program
- (5) Project image number two into space using the extracted matrices and the first projection as reference.
- (6) Repeat for all images

This method is of course sensitive to human error, but with some patience and skill from the user, it should be able to achieve decent results.

7.3.2 Finding the Projection Matrix by Additional Technology

This section could of course be filled with dreams of marvelous new technology that would make everything possible. However, this would not be very productive. We will therefore only look at methods that would use common technology, with only small adjustments or improvements.

The problem of finding the translation and rotation matrix is just a problem of finding the position of the cameras, relevant to each other and the object. And finding the position is exactly what a GPS receiver does. Modern GPS receivers are highly accurate, finding the receivers position with 3 to 5 meters accuracy, and they are bound to get more accurate with time. A camera equipped with a GPS receiver[2] will be able to register the position when

[2]Modern mobile phones already have both camera and GPS

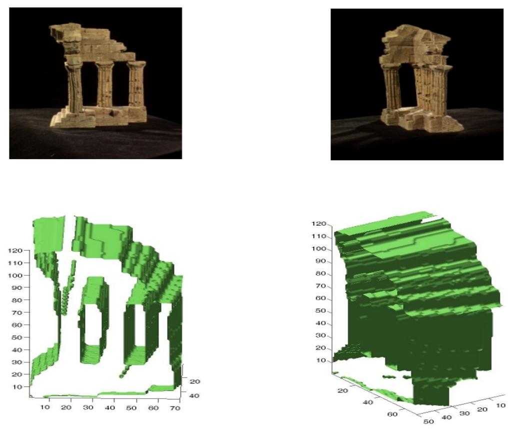

Figure 7.1: *First project image number one into 3-D. Then rotate and translate that projection until you find the view from image number two. The rotation and translation matrices can then be extracted from the 3-D visualization program and used for projection of image number two.*

an image is taken. Using the positions one could calculate both the translation and rotation matrices. With the current GPS accuracy, this would not be suited for small objects, but larger objects, like the statue of liberty or a building, could possibly be reconstructed. And as the GPS accuracy increases, the size of a reconstructable object decreases.

Another technology that has become prevalent in the last years is *Bluetooth technology*. This technology is included in almost all new gadgets. It is a way to exchange information wirelessly over short distances by using short radio waves. As bluetooth enabled devices has become more a part of our world, it has attracted a lot of research interest. One of the areas of interest is *bluetooth triangulation*. This is a way of finding the position of other bluetooth devices nearby. In a report by Almaula and Cheng [2] they claim to achieve an accuracy of 5-10 feet, which is about one and a half to three meters. With this kind of accuracy, the reconstruction of smaller objects might be possible. And as with the GPS, we should expect accuracy to increase as research continues.

Bibliography

[1] R. Acar and C. R. Vogel. Analysis of bounded variation penalty methods for ill posed problems. *Inverse Problems, 10, pp 1217-1229*, 1994.

[2] Varun Almaula and David Cheng. Bluetooth Triangulator. Technical report, Department of Computer Science and Engineering University of California, San Diego, 2007.

[3] Gilles Aubert and Pierre Kornprobst. *Mathematical Problems in Image Processing.* Springer Verlag, 2002. ISBN 0-387-96326-4.

[4] A. Auclair, L. Cohen, and N. Vincent. Using point correspondences without projective deformation for multi-view stereo reconstruction. *ICIP 2008*, 2008.

[5] Tony F. Chan, Selim Esedolu, and Mila Nikolova. Algorithms for Finding Global Minimizers of Image Segmentation and Denoising Models. *SIAM J. Appl. Math. Vol 66*, 2006.

[6] Ward Cheney. *Analysis for applied mathematics.* Springer Verlag, 2001. ISBN 0-387-95279-9.

[7] A. Delaunoy, E. Prados, P. Gargallo, J.-P. Pons, and P. Sturm. Minimizing the multi-view stereo reprojection error for triangular surface meshes. *BMVC 2008*, 2008.

[8] Bob Fisher. Geometric Framework for Vision, 1997.

[9] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *Submitted to PAMI 2008*, 2008.

[10] Francesco Maria Grimaldi. Physico-mathesis de lumine, coloribus, et iride, aliisque adnexis libri duo. 1665.

[11] Andrew I. Hanna. Minimizing functionals. A level set approach. 2007.

[12] J.P.Lewis. Fast Normalized Cross-Correlation. *draft*, 2000.

[13] Orjan Knudsen. Scene Reconstruction Using Level Sets and Graph Cuts. Master's thesis, University of Bergen, 2008.

[14] Kalin Kolev, Maria Klodt, Thomas Brox, and Daniel Cremers. Propagated Photoconsistency and Convexity in Variational Multiview 3D Reconstruction.

[15] Kalin Kolev, Maria Klodt, Thomas Brox, and Daniel Cremers. Continuous Global Optimization in Multiview 3D Reconstruction. 2009.

[16] Andrew J. Kurdila and Micheal Zabarankin. *Convex Functional Analysis*. Birkhaser Verlag, 2005. ISBN 3-7643-2198-9.

[17] Aldo Laurentini. The Visual Hull Concept for Silhouette-based Image Understanding. *IEEE Transactions on pattern analysis and machine intelligence, vol 16. No 2*, 1991.

[18] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, K Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. Proc. SIGGRAPH, 2000.

[19] Johan Lie, Marius Lysaker, and Xue-Cheng Tai. A Piecewise Constant Level Set Framework. *International Journal of Numerical Analysis and Modeling*, 2005.

[20] Stanley Osher and Ronald P. Fedkiw. Level Set Methods: An Overview and Some Recent Results. *Journal of Computational Physics 169*, 2001.

[21] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. *CVPR 2006, vol. 1, pages 519-526*, 2006. http://vision.middlebury.edu/mview/.

[22] J. Starck and A. Hilton. Surface capture for performance-based animation. *CG and A 27(3) page 21-31*, 2007.

[23] G. Vogiatzis, P. Torr, and R. Cippola. Multi-view stereo via volumetric graph-cuts. *Proc. International Conference on Computer Vision and Pattern Recognition, 391-399*, 2005.

[24] George Vogiatzis, Carlos Hernndez Esteban, Philip H. S. Torr, and Roberto Cipolla. Multi-view Stereo via Volumetric Graph-cuts and Occlusion Robust Photo-Consistency. 2007.