Scene Reconstruction Using Level Sets and Graph Cuts

Master of Science Thesis in Applied Mathematics

Ørjan Knudsen

Department of Mathematics
University of Bergen

May 30, 2008

Preface

Apart from numerous moments of hair-pulling frustration, this thesis has been a labour of love. I enjoy the wide gamut of applied mathematics, from the hands-on modelling of real-world phenomena to the icy pinnacles bordering the sunless lands of pure mathematics. But I could not make this journey alone.

First, I would like to thank my supervisor Xue-Cheng Tai, for proposing an interesting subject for my thesis, for efficient help with technical details and for his faith in my ability to work autonomously. Also, proper recognition to Egil Bae for our numerous discussions of the minutiae of surface representation, edge weights and Cauchy-Crofton formulas and for his pointers to useful papers.

Cheers to the people who helped proof-read my thesis, for pointing and laughing at my most embarrassing errors so I would never make them again, and for their persistence and attention to detail. Chiefly, Kristine, Eirik and Øystein.

A raised hat to all that make life worth living and put a smile on my face: friends, lovers, colleagues, dance partners, family, vintners and cheesemakers, strangers who smile at me on the street and all the rest. You know who you are.

Finally, to the warmest and driest spring in Bergen for decades: You have tried to draw me away from my studies, but I have persevered. Try harder next time!

Ørjan Knudsen

Bergen, May 2008

Contents

Chapter 1

Introduction

Image processing is the branch of data analysis that concerns itself with visual data. Either images are processed prior to inspection by a human observer, or the act of observation itself is emulated. In either case, a thorough understanding of the mechanics of vision is required. For instance, the knowledge that a high contrast and dynamic range makes it easier to pick out details in an image might motivate contrast-boosting preprocessing. When trying to implement machine vision, knowledge of the process that formed the input images is critical.

Vision in nature The human vision system is the result of millions of years of evolution. The earliest vision system was simply a patch of photosensitive cells that could distinguish light from dark. This area then became more and more recessed, leading to crude directional vision as light would hit different parts of the pit from different angles. Also, the front opening of the pit shrank in size, yielding a crude pinhole camera. The aperture was subsequently covered by a clear layer that protected the cells within. This layer then grew into a full-blown lens, allowing clearer definition of shapes. The photoreceptors diversified into rods and cones, respectively providing low-light and colour vision. To match this sophisticated sensory equipment, a significant part of the brain is dedicated to low-level and high-level vision. Low-level tasks include finding edges and shapes and tracking movement. On a higher level, the vision center maintains a persistent model of the area around us, updating it continually from eye input and experience. It is part of this higher-level vision that the field of **computer vision** seeks to emulate.

Computer vision The human vision center is highly specialized and draws upon a lifetime of experience. Computer vision programs are often run on general-purpose computers and draw only on the priors that their programmers give them. These limitations, along with the inherent complexity of the task, make the problem of vision one of the most challenging in image processing. The field is not yet mature, and algorithms are typically tailored to their specific application. Few general-purpose computer vision algorithms exist.

Applications Computer vision can be applied to a diverse set of problems: Organ or pathology recognition in medical imaging, whether arising from X-rays, ultrasound, computer tomography (CT), nuclear magnetic resonance imaging (MRI), microscopy, angiography, positron emission tomography (PET) or other sources; industrial automation, where machines and robots can be guided by machine vision and defective products can be detected; military applications such as missile guidance, detection and early warning systems and unmanned aerial vehicles (UAV); surveillance, where machine vision can point out regions of interest to a human operator; augmented reality, where real-time video is "augmented" with computer-generated imagery appropriate to the view angle; 3-D graphics and cinematic effects, archaeological reconstruction and autonomous cars are some examples.

This thesis concerns itself with the problem of **scene reconstruction**, which is the task of recovering 3-D models of objects seen in images. We define this problem and our approach to it in Chapter 3. Starting with an existing algorithm introduced by Kolev et.al. in [9], we modify it to use two very different frameworks: the **piecewise constant level set method** and the method of **graph cuts**. The former, introduced by Tai et.al. in [11], has several important advantages over the more common signed distance level set method; we will take a closer look at these in Chapter 4. The graph cut method is an emerging and exiting tool in the minimization of certain functionals in image processing. It has the advantage of speed and guaranteed minimization, though it is less general than other methods and great care has to be taken in converting problems to graph form. We explore this method in Chapter 6. We test the performance of our algorithms in Chapter 7. Finally, we summarize the thesis and outline some avenues for future work in Chapter 8. Before we turn our attention to these advanced topics, however, we introduce the mathematical tools we will be using in Chapter 2.

Chapter 2

Mathematical Tools

In the course of this thesis we will be employing various mathematical tools. Some may be trivial, others harder to comprehend. Which tools fall into which of these categories varies with the background of the reader. We have chosen to include a liberal amount of background matter; the reader is free to skip any part that is instantly familiar. We have tried to keep this section concise yet clear.

2.1 Image representation

It is not surprising that the field of image processing deals extensively with images. Ideally, a **gray-scale image** is a function $I : D \subset \mathbb{R}^2 \to \mathbb{R}$ giving a real value to the observed intensity $I(x, y)$ at each point (x, y) in the image domain D. The meaning of the term **intensity** depends on the nature of the image; for instance, a black and white photography measures the incident light at a point. We shall deal with discrete **digital** images with a slightly different definition: $I : [1, m] \times [1, n] \to [0, 255]$ where $[a, b]$ is the range of whole numbers from a to b, inclusive. A point x, y is called a **pixel**. A **color image** is a set of three grayscale images R, G, B giving the intensity of red, green and blue light respectively. A total of 256^3=16,777,216 colors can be represented in this way. Typical values of m and n are such that the **aspect ratio** $m/n \approx 4/3$ and m times n is about a few millions (megapixels).

2.2 Statistics

Computer vision aims to estimate the state of real-world objects given a set of observations. It is thus an example of parameter estimation or inductive reasoning. Due to the uncertainty inherent in such a process, a very natural idea is to try to apply statistical theory. We will therefore include a short introduction to some statistical concepts here. A basic statistical textbook such as [16] will expand endlessly upon these subjects; we shall therefore be brief.

2.2.1 Probability distributions

When we make a measurement of a real-world phenomenon we are making a **statistical experiment**. The set of all possible measurements is called the **sample space**. A **random variable** is a function that associates a real number with each element in the sample space. If the sample space has a finite or countably infinite number of elements, it is called a **discrete sample space**. Otherwise, it is a **continuous sample space**. A discrete random variable assumes each of its values with a certain probability. If the discrete random variable X takes the value x with probability $f(x) = P(X = x)$, the function $f(x)$ is called the **probability distribution** of X. It is nonnegative and sums to 1. A continuous random variable, by contrast, has a probability of zero of assuming *exactly* any of its values. Instead of exact values, we must deal with intervals. If a continuous random variable X assumes a value in the interval (a, b) with probability $\int_a^b f(x)dx = P(a < X < b)$, $f(x)$ is called the **probability density function** of X. It is nonnegative and $\int_{-\infty}^{\infty} f(x)dx = 1$. The function $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt$ is called the **cumulative distribution** of X.

2.2.2 Mathematical expectation

The **mean** or **expected value** of a continuous random variable X is

$$\mu = E(X) = \int_{-\infty}^{\infty} x f(x) dx.$$

Similarly, the **variance**

$$\sigma^2 = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x) dx$$

is simply the expected squared deviation from the mean. The positive square root σ of the variance is called the **standard deviation** of X. These two variables tell us a lot about a distribution. For instance,

$$\begin{aligned}\sigma^2 &= \int_{-\infty}^{\infty} (x-\mu)^2 f(x) dx \\ &\geq \int_{-\infty}^{\mu-k\sigma} (x-\mu)^2 f(x) dx + \int_{\mu+k\sigma}^{\infty} (x-\mu)^2 f(x) dx\end{aligned}$$

since the integrand is everywhere positive. Since $|x - \mu| > k^2\sigma^2$ in the remaining regions,

$$\begin{aligned}\sigma^2 &\geq \int_{-\infty}^{\mu-k\sigma} k^2\sigma^2 f(x)dx + \int_{\mu+k\sigma}^{\infty} k^2\sigma^2 f(x)dx\\ \frac{1}{k^2} &\geq \int_{-\infty}^{\mu-k\sigma} f(x)dx + \int_{\mu+k\sigma}^{\infty} f(x)dx.\end{aligned}$$

Hence

$$P(\mu - k\sigma < X < \mu + k\sigma) = \int_{\mu-k\sigma}^{mu+k\sigma} f(x)dx \geq 1 - \frac{1}{k^2}$$

This result is called Chebyshev's theorem, see for instance [16]. It tells us that we can use the mean and standard deviations to estimate probability without knowing the underlying distribution. In practice, however, it is useful to assume a known distribution.

2.2.3 Examples of distributions

We shall consider a few simple probability distributions. First off, what if we have no way of telling which parts of an interval are most likely? All parts must then be equally likely, yielding the **uniform distribution**

$$u(x; A, B) = \begin{cases} \frac{1}{B-A}, & A \leq x \leq B \\ 0 & \text{elsewhere} \end{cases}$$

with mean and variance

$$\mu = \frac{A+B}{2} \quad \text{and} \quad \sigma^2 = \frac{(B-A)^2}{12}.$$

A more common distribution is the **Gaussian** or **normal distribution**

$$n(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

which appears in many natural phenomena and the errors of scientific measurements. This is mainly due to the central limit theorem, which states that the sum of a large number of independent and identically-distributed random variables will be approximately normally distributed. The fact that this distribution is conceptually simple and easy to work with only increases its appeal.

2.2.4 Bayes' Law

The **conditional probability** of event A given event B is given as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

whenever $P(B)$ is nonzero. If $P(B) = 0$, $P(A|B)$ is not defined (nor would it be very useful if it were!). Rearranging the equation,

$$P(A \cap B) = \frac{P(A|B)}{P(B)}.$$

Applying the same reasoning to $P(B|A)$ yields

$$P(A \cap B) = P(B \cap A) = \frac{P(B|A)}{P(A)}.$$

hence

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{Bayes' Law}$$

which will prove useful in this thesis.

2.2.5 Maximum likelihood estimation

Often, we have a lot of data samples but are not privy to the nature of the stochastic process giving rise to them. Even if we guess at a particular distribution, we still need to estimate the relevant parameters. A principled way of doing this is **maximum likelihood estimation**. Simply put, we choose the parameters that maximize the probability of the observations we have made. For instance, let $\mathbf{X} = X_1, X_2, \ldots, X_n$ be a set of independent random variables taken from a probability distribution represented by $f(\mathbf{X}, \theta)$, where θ is a set of parameters of the distribution. Now let $\mathbf{x}$ be the actual observed values. The function

$$L(\mathbf{X}; \theta) = P(\mathbf{X} = \mathbf{x}|\theta)$$

is called the **likelihood function** of the sample. If we consider the sampled values $\mathbf{x}$ as constants and seek to find

$$\hat{\theta} = \max_{\theta} L(\mathbf{X}; \theta),$$

we obtain a logical estimate $\hat{\theta}$ of the distribution parameters. Note, however, that we need to know or guess the underlying distribution in order to use this method. We extend the concept discussed here to more general optimization problems, though it this no longer stricly a maximum likelihood estimation. The basic idea is the same: Given a set of observations $\mathbf{x}$, a model $M(\theta)$ dependent on model parameters θ and a probability function $P(M, \mathbf{X})$, we find the optimal model

$$\hat{M} = M(\hat{\theta}), \quad \hat{\theta} = \max_{\theta} P(M(\theta), \mathbf{X}).$$

2.3 Flow networks

A **network** $(\mathcal{N}, \mathcal{A})$ consists of a set of **nodes** $\mathcal{N}$ and a set of directed **arcs** $\mathcal{A} \in \{(i,j) | i \neq j,\ i,j \in \mathcal{N}\}$ connecting them. Suppose we identify a **source node** s and a **sink node** t, both in $\mathcal{N}$. Further, we define an upper bound u_{ij} on the flow through each arc (i,j) in $\mathcal{A}$. A **cut** C is then a set of nodes containing the source node but not the sink node. The **capacity** of a cut is defined as

$$\kappa(C) = \sum_{\substack{i \in C \\ j \notin C}} u_{ij}.$$

A **flow** $x : \mathcal{A} \mapsto \mathbb{R}^+$ is a function that details the transport x_{ij} along each arc (i,j) in $\mathcal{A}$. At each node i other than the source or sink the transport must sum to zero:

$$\sum_{j:(i,j) \in \mathcal{A}} x_{ij} - \sum_{j:(j,i) \in \mathcal{A}} x_{ij} = 0. \qquad \text{(flow balance)}$$

The magnitude $|x|$ of a flow is the total amount transported from the source node to the sink node. Due to the flow balance constraint, for an arbitrary cut C this is equivalent to:

$$|x| = \sum_{\substack{i \in C \\ j \notin C}} x_{ij} - \sum_{\substack{i \notin C \\ j \in C}} x_{ij}.$$

The **Maximum-Flow** problem is then to find the maximum feasible flow from the source node to the sink node:

$$\hat{x} = \arg\max |x|.$$

such that

$$0 \leq \hat{x}_{ij} \leq u_{ij}, \ \forall (i,j) \in \mathcal{A} \qquad \text{(feasibility)}$$

The **Minimum-Cut** problem is to find the cut C that minimizes $\kappa(C)$. An important result from duality theory is that these two problems are equivalent, i.e., the maximum feasible flow is equal to the minimum capacity. Intuitively, the min cut capacity $\hat{\kappa}$ is the *bottleneck* of the graph.

2.3.1 Ford-Fulkerson algorithm

Many algorithms for the solution of the maximum flow problem are descendants of the classical Ford-Fulkerson algorithm introduced in [6]. We shall outline it briefly before introducing a modification that is very relevant to our use. A **path** from node s to node t is a set of arcs forming a connection between those nodes. The direction of the arcs does not matter, only their connectedness. An **augmenting path** with respect to a flow x is a path from s to t such that $x < u$ on forward arcs of the path, and $x > 0$ on reverse arcs of the path. A flow x is maximal if and only if there is no augmenting path with respect to x. The Ford-Fulkerson has two steps: Finding an augmenting path, then pushing the maximum amount of flow through it. These two steps are repeated until no augmenting path is found, at which point the maximum flow has been found.

2.3.2 Boykov-Kolmogorov algorithm

The distinguishing factor between different augmenting path algorithms is the subalgorithm that chooses the order in which augmenting paths are saturated. The shortest-path criterion of the popular **Edmonds-Karp** or **Dinic algorithm** defines a **breadth-first** search. After all shortest paths of a fixed length k are saturated, the algorithm starts the search for augmenting paths of length $k + 1$ from scratch. The shortest-path method is crucial in ensuring a low worst-case bound on the running time: the Dinic algorithm has complexity $O(mn^2)$ where n is the number of nodes and m the number of edges in the graph. This is a useful algorithm for general problems, and in particular **sparse** graphs where the number of edges is low. However, the graphs that arise in image processing in general and in our problem in particular have a rather peculiar connectivity: Nodes are connected to their spatial neighbours as well as the source and/or sink nodes. The total number of edges is not very high, but if a large amount of nodes are connected to the source or sink, a breadth-first search will be very slow. The algorithm introduced in [2] is tailored to give good practical results on such **shallow** graphs. Note, however, that it has a *worse* theoretical running time than the Dinic algorithm. For integer flows, an augmenting path will increase the flow by at least 1. The worst-case scenario is that all augmenting paths yield no more than this, meaning that $\hat{\kappa}$ augmenting paths have to be found, where $\hat{\kappa}$ is the value of the maximum flow. This yields a complexity of $O(mn^2\hat{\kappa})$.

Outline

A **tree** is a connected set of nodes fulfilling certain criteria. It is either empty or consists of a **root** node and zero or more subtrees. Each node apart from the root is either a **leaf** or an **internal** node. An internal node has one or more **child** nodes and is called the **parent** of its child nodes. Leaf nodes have no children. To any node in the tree there is a unique path from the root node. The Boykov-Kolmogorov algorithm maintains two non-overlapping **search trees** S and T with roots at the source s and the sink t, respectively. In tree S all edges from each parent node to its children are non-saturated, while in tree T edges from children to their parents are non-saturated. The nodes that are not in S or T are called **free** nodes. Nodes are also classified as active or inactive. An **active** node is connected to a node not in the same tree by a non-saturated arc. A **passive** node is not, and thus cannot become a parent. When the two trees "touch" along a non-saturated edge, an augmenting path is found. The algorithm proceeds in three stages:

1. **growth** stage: Search trees S and T grow by adopting neighbouring free nodes (along non-saturated paths) until they touch, giving an augmenting path

2. **augmentation** stage: The maximum possible flow is pushed through the augmenting path, saturating one or several edges. Each saturated edge drops out of its tree, possibly splitting the tree into several trees.

3. **adoption** stage: we try to find a new path from the root to each node that lost its parent node in the augmentation stage. If this is not possible, that node is deleted and we try to find paths to the nodes "orphaned" by this deletion. When no orphans remain, single-tree structure has been restored and we can go back to the growth stage.

The algorithm terminates when the search trees S and T have no active nodes and the trees are separated by saturated edges. A maximum flow has been achieved, with cut sets equal to the sets S and T. Boykov and Kolmogorov have kindly made available a C++ software library that implements this algorithm, and we shall use this library to solve our max flow problems later in this thesis.

2.4 Discretization

Throughout this thesis we use fixed regular grids. The grid spacings $\triangle x, \triangle y$ and $\triangle z$ are in general not equal. In order to numerically solve PDEs we need to discretize them. The purpose of this thesis is not to test out exotic discretization schemes; rather, we use common finite-difference strategies. For time marching we use the standard forward Euler method

$$f'(t) = g(t) \rightsquigarrow f(t + \triangle t) = f(t) + \triangle t\, g(t)$$

and all spatial derivatives are first-order forward differences:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

2.5 Function spaces

In applied mathematics, many problems can be formulated as a search for a function satisfying certain constraints. To find the optimum such function we much search through the set of feasible functions. To this end, a rigorous theory of such **function spaces** is useful. We first introduce the concept of a vector space. A **real vector space** is a triple $(X, +, \cdot)$, in which X is a set, and $+$ and $\cdot$ are binary operators satisfying certain axioms (see, for instance, [3]):

1. If x and y belong to X then so does $x + y$.
2. $x + y = y + x$.
3. $x + (y + z) = (x + y) + z$.
4. X contains an element, 0, such that $x + 0 = x$ for all $x \in X$.
5. With each element x there is associated a unique element, $-x$, such that $x + (-x) = 0$.
6. If $x \in X$ and $\lambda \in \mathbb{R}$, then $\lambda \cdot x \in X$.
7. $\lambda \cdot (x + y) = \lambda \cdot x + \lambda \cdot y$.
8. $(\lambda + \mu) \cdot x = \lambda \cdot x + \mu \cdot x \quad (\lambda, \mu \in \mathbb{R})$.
9. $\lambda \cdot (\mu \cdot x) = (\lambda\mu) \cdot x$.
10. $1 \cdot x = x$.

We shall forgo the triple notation $(X, +, \cdot)$, instead referring to the vector space X for the rest of this thesis. A **norm** on a vector space X is a real-valued function, denoted by $\| \cdot \|$, that fulfills three axioms:

1. $\|x\| > 0, \quad 0 \neq x \in X.$
2. $\|\lambda x\| = |\lambda|\|x\|, \quad \lambda \in \mathbb{R}, x \in X.$
3. $\|x + y\| \leq \|x\| + \|y\|, \quad x, y \in X.$

A vector space with an associated norm is called a **normed linear space**. For instance, the space of continuous functions on a fixed compact interval $[a, b]$ with supremum norm:

$$C[a,b] = \{f : [a,b] \mapsto \mathbb{R} : f \,\text{continuous}\}$$
$$\|x\|_\infty \doteq \max_{s\in[a,b]} |x(s)|.$$

A normed linear space enables the study of **sequences** $x_1, x_2, \ldots$ whose convergence is measured by the norm:

$$x_n \to x \Rightarrow \lim_{n\to\infty} \|x_n - x\| = 0.$$

2.5.1 Completeness

When approximating some desired solution iteratively, we want to be able to make statements about the limit which our iteration is approaching. Even though every approximation we make has some desirable property (such as being a rational number), it is not certain that the limit has this property. For instance, it is possible to create a sequence of rational numbers that converges to π (just keep adding digits). Yet π itself is not a member of the set of rational numbers. What is missing from the space of rational numbers is **completeness**. A sequence $[x_n]$ in a normed linear space X is a **Cauchy sequence** if

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

If every Cauchy sequence in a space X is convergent, then X is said to be **complete**. A complete normed vector space is called a **Banach** space.

2.5.2 L^p spaces

For $1 \leq p \leq \infty$, $L^p(\Omega)$ denotes the space of all measurable functions on Ω that are p^{th}-power integrable on Ω. The norm on $L^p(\Omega)$ is given by

$$\|u\|_p = \left(\int_\Omega |u|^p dx\right)^{1/p}$$

for finite p or

$$\|u\|_\infty = \operatorname{ess\,sup} |\mathrm{u}|$$

for $p = \infty$. The term "function" is used slightly loosely here - members of L^p are **equivalence classes** of functions. In particular, $\|v - w\|_p = 0$ does *not* imply $v(x) = w(x), \forall x$.

2.6 Distributions

The theory of partial differential equations is, understandably, focused on the study of differentiable functions. On the other hand, functions represented on a computer are often piecewise constant on some appropriate grid. To reconcile these paradigms in a mathematically stringent way, we introduce the notion of distributions. To make this section as clear as possible, we need to introduce some tight notation. A **multi-index** is a n-tuple of non-negative integers

$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n).$$

Its order $|\alpha|$ is the sum of these integers. Given a multi-index α, we define the differential operator D_α as

$$D_\alpha = \prod_{i=1}^{n} \left(\frac{\partial}{\partial x_i} \right)^{\alpha_i}.$$

The vector space $C^\infty(\mathbb{R}^n)$ is defined as

$$C^\infty(\mathbb{R}^n) = \left\{ f : \mathbb{R}^n \to \mathbb{R} | D^\alpha f \in C(\mathbb{R}^n), \forall \alpha \right\}.$$

It is thus the space of all functions defined on $\mathbb{R}^n$ whose partial derivatives all exist and are continuous.

The **support** of a function f is the closure of $\{x : f(x) \neq 0\}$. The space $\mathcal{D}$ of **test functions** is the set of functions in $C^\infty(\mathbb{R}^n)$ having compact support. This means they are zero outside a finite region in $\mathbb{R}^n$.

A **distribution** is a continuous linear functional on $\mathcal{D}$. The space of such distributions is denoted $\mathcal{D}'$. We shall introduce some important distributions.

The **Dirac** distribution $\delta_x : \mathcal{D} \mapsto \mathbb{R}$ is given as

$$\delta_x(f) = f(x)$$

that is, it takes as its argument the *function* f and returns the *function value* $f(x)$. Simply writing δ without a subscript refers to δ_0.

The **Heaviside** distribution on $\mathcal{D}(\mathbb{R})$ is defined as

$$H(f) = \int_0^\infty f(x)dx$$

and similarly in n-D (integrating over the values of $x = (x_1, x_2, ..., x_n)$ where $x_i \geq 0, \forall i \in [1, n]$).

Let $f : \Omega \in \mathbb{R}^n \to \mathbb{R}$ be a **locally integrable** function, i.e. $\int_K |f|dx < \infty$ for all compact subsets K in Ω. To each such function, we can associate a distribution $\tilde{f}$:

$$\tilde{f}(g) = \int f(x)g(x)dx \quad (g \in \mathcal{D}).$$

This will be useful in extending the notion of the derivative to functions that are not differentiable in the classical sense.

2.6.1 Derivatives of distributions

We have seen that there is an injective mapping from the set of integrable function to the set of distributions. Our goal now is to extend the notion of differentiation to the latter, so that we may in a sense speak of PDEs operating on piecewise constant functions. First, we will need to lay some foundations.

If T is a distribution and α is a multi-index, the **distributional derivative** $\partial^\alpha T$ is the distribution defined by

$$\partial^\alpha T = T \circ (-D)^\alpha.$$

Let f be a function on $\mathbb{R}^n$ such that $D^\alpha f$ exists and is continuous. Define the corresponding distribution $\tilde{f}$. Then

$$\partial^\alpha \tilde{f} = \widetilde{(D^\alpha f)}.$$

That is, the notion of distributional and standard derivative coincide when both are meaningful. This is heartening.

We often say in a loose sense that the Dirac delta "function" is the "derivative" of the Heaviside function. In the distributional sense, we can make this rather handwaving argument a lot more precise: Let $\tilde{H}$ be the 1-D Heaviside distribution, and δ be the Dirac distribution at 0. Now for any test function g,

$$(\partial \tilde{H})(g) = -\tilde{H}(Dg) = -\int_0^\infty g' dx = g(0) - g(\infty) = g(0) = \delta(g),$$

i.e., $\partial\tilde{H} = \delta$.

A piecewise constant function f can be seen as a sum of shifted and scaled Heaviside functions and a constant:

$$f = \sum_{i=1}^{N} a_i H(x - b_i) + c.$$

Applying the distributional derivative of $\tilde{H}$,

$$\partial\tilde{f} = \sum_{i=1}^{N} a_i \delta_{b_i}(x).$$

This is not immediately useful unless we find a way to return it to the space of piecewise constant functions. To do so, we must approximate the Dirac distribution with some aptly chosen function.

2.7 Total Variation

It is often useful to define a measure of the complexity or tendency to oscillate of a function. An important tool to this end is the **total variation** (TV) norm. For a one-dimensional function $f(x)$, this is defined as

$$TV(f) = \sup \sum_{j=1}^{N} |f(x_j) - f(x_{j-1})|,$$

where the supremum is taken over all subdivisions of the real line $-\infty = x_0 < x_1 < \cdots < x_N = \infty$. For the total variation to be finite, f must approach a constant as $x \to \pm\infty$. Two simplifying cases are available. If f is differentiable, the TV norm becomes

$$TV(f) = \int_{-\infty}^{\infty} |f'(x)| dx \tag{2.1}$$

whereas for a piecewise constant grid function F we have

$$TV(F) = \sum_{i=-\infty}^{\infty} |F_i - F_{i-1}|.$$

Even if f is not differentiable in the classical sense, we can use (2.1) if we interpret $f'(x)$ as the distribution derivative, including delta functions at points where f is discontinuous. The multi-dimensional equivalent is

$$TV(f) = \int_{\Omega} |\nabla f(x)| dx.$$

2.7.1 Bounded variation spaces

A function $f \in L^1(\Omega)$ whose partial derivatives in the sense of distributions are measures with finite total variation in Ω is called a **function of bounded variation**. The class of all such functions is denoted $BV(\Omega)$. This space is endowed with the norm

$$\|f\|_{BV} = \|f\|_1 + \|f\|_{TV},$$

see, for instance, [18]. The space $BV_0(\Omega)$ is the restriction of $BV(\Omega)$ to functions that are zero on the boundary $\partial\Omega$.

2.8 The Fréchet derivative

Having defined spaces of functions, we would like extend our calculus toolbox to these spaces. First, we need a definition of the derivative. Let $f : D \to Y$ be a mapping from an open set D in a normed linear space X into a normed linear space Y. Let $x \in D$. If there is a bounded linear map $A : X \to Y$ such that

$$\lim_{h\to 0} \frac{\|f(x+h) - f(x) - Ah\|}{\|h\|} = 0$$

then f is said to be **(Fréchet) differentiable** at x. Furthermore, A is called the (Fréchet) **derivative** of f at x. The next theorem, as found in [3], links the idea of the Fréchet derivative to the gradient.

Theorem 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$. If each of the partial derivatives $D_i f = \partial f/\partial x_i$ exists in a neighborhood of x and is continuous at x then $f'(x)$ exists, and a formula for it is*

$$f'(x)h = \sum_{i=1}^{n} D_i f(x) \cdot h_i \quad h = (h_1, h_2, \ldots, h_n) \in \mathbb{R}^n$$

Speaking loosely, we say that the Fréchet derivative of f is given by the gradient of f.

Several results from classic calculus can be extended to functional analysis. For instance,

Theorem 2. *Let $f : D \to Y$, $\quad g : E \to Z$, where D is an open set in a normed linear space X, E is an open set in a normed linear space Y, and Z is a third normed space. If f is differentiable at x and g is differentiable at $f(x)$, then $g \circ f$ is differentiable at x, and*

$$(g \circ f)'(x) = g'(f(x)) \circ f'(x) \qquad \text{(Chain Rule)}$$

2.9 Integration by parts

We shall require the use of the general integration-by-parts formula. Turning to the excellent textbook [5], we find the classical theorem:

Theorem 3. *Let $f, g \in C^1(\bar{\Omega})$. Then*

$$\int_\Omega f_i g dx = -\int_\Omega f g_i dx + \int_{\partial\Omega} f g n(i) dS \tag{2.2}$$

where subscripts denote partial derivatives, dS is the surface element and $n(i)$ is the i^{th} part of the outward normal vector on $\partial\Omega$.

2.10 Constrained extremum problems

Elementary calculus deals extensively with extremum problems. Given a function $f(\mathbf{x})$ defined on some domain Ω, we wish to find the points where it takes its minimal and maximal values. If Ω is an open set, any extremum $\mathbf{x}_0$ must satisfy $f'(\mathbf{x}_0) = 0$, provided $f'(\mathbf{x}_0)$ exists. If $f'(\mathbf{x}_0) \neq 0$, we can move in/against the direction of the derivative and arrive at a point with a higher/lower value of f. If we move a sufficiently short distance, we will still be in Ω and f will be continuous, by the definition of open sets and derivatives. If we are not in an open domain, extrema can occur at the boundaries as well as in critical and non-differentiable points. This will tend to complicate matters somewhat, as we have to check the boundaries as well as the derivatives. We would like to handle this process in a more implicit manner. First, we will introduce some more precise definitions. Let Ω be an open set in a normed linear space, and let $f, g : \Omega \mapsto \mathbb{R}$ be continuously differentiable functions. We define the set $M = \mathbf{x} \in \Omega : g(\mathbf{x}) = 0$ and seek extrema of f restricted to M, $f|M$. A maximization problem can be viewed as a minimization problem of the negative of a function, thus we will simply look at minimization from now on. Our problem reads

$$\min f(\mathbf{x}) \quad \text{subject to} \quad g(\mathbf{x}) = 0. \tag{2.3}$$

To make our lives simple, we want to frame this as an unconstrained minimization instead. The toolbox for doing so is called the method of Lagrangian multipliers. We introduce the Lagrangian function Λ as:

$$\Lambda(x, \lambda) = f + \sum \lambda_k g_k$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is a vector of m scalars corresponding to the m constraints embodied in the vector-valued function $g = (g_1(x), g_2(x), \ldots, g_m(x))$.

We assert that solving the problem

$$\nabla\Lambda = 0$$

is equivalent to solving the constrained problem 2.3. Splitting the operator

$$\nabla = \nabla_x + \nabla_\lambda$$

so that ∇_x and ∇_λ represent the gradient with respect to x and λ, respectively, we have:

$$\nabla\Lambda = 0 \iff \nabla_x\Lambda = 0 \wedge \nabla_\lambda\Lambda = 0.$$

Now

$$\nabla_\lambda\Lambda = 0 \Rightarrow g = 0$$

and

$$\nabla_x\Lambda = 0 \Rightarrow \nabla_x f = -\sum \lambda_k \nabla_x g_k \Rightarrow \nabla_x f = 0$$

thus proving the assertion.

2.10.1 Extension to function spaces

Let X and Y be real Banach spaces. Let U be an open subset of X and introduce the functionals $f : U \to \mathbb{R}$ and $g : U \to Y$. We seek the minimum of f subject to g being the zero element in Y:

$$\min f(x) \quad \text{subject to} \quad x \in \{U|g(x) = 0\}. \tag{2.4}$$

We extend the notion of Lagrange multipliers to function spaces:

$$\lambda : Y \to \mathbb{R}$$

and

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

An element $\hat{x}$ minimizing (2.4) is then equivalent to

$$D\Lambda(\hat{x}, \lambda) = 0$$

where D is the derivative with respect to x and λ.

2.11 Calculus of variations

The **calculus of variations** is a natural extension of extremum problems to function spaces. Instead of finding the maxima and minima of functions over a domain of numbers such as $\mathbb{R}^3$, we look for critical points of *functionals* defined on function spaces. Define $f : U \to \mathbb{R}$ as in the previous section. We want to solve the minimization problem

$$\hat{x} = \underset{x \in U}{\arg\min}\, f(x).$$

For $v \in U$ such that $\|v\| = 1$, define the weak derivative of f in the direction v as

$$df(x; v) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon v) - f(x)}{\epsilon}.$$

For a function $\hat{x}$ to be a minimizer of f, a neccessary condition is the Euler-Lagrange Equation:

$$df(\hat{x}; v) = 0 \qquad \forall v \in U, \tag{2.5}$$

$\hat{x}$ is then a critical point of f.

Chapter 3

Surface Reconstruction

Scene reconstruction is the problem of inferring the scene that has resulted in a set of observations. A **scene** consists of some **incident light** and a number of **objects** interacting with the light. Interactions can be reflection, refraction, occlusion, diffraction and so on. All of these interactions are well understood and the problem of generating images of a given scene, known as **computer-generated imagery** (CGI), is rather simple. The **inverse problem** that we study, however, is not. In fact it is by nature ill-posed, and additional conditions must be applied to remedy this.

3.1 Well-posedness

A problem is **well-posed** if it satisfies these criteria (see [5]):

1. There must **exist** a solution
2. The solution must be **unique**
3. The solution must be **stable** under small perturbations of the data

If our data is aquired from the real world, it would seem obvious that condition 1 is fulfilled. After all, *some* scene must have yielded the data. However, even if a real answer exists, an algorithm designed to attain that answer may not be solvable. If our method seeks the minimum of some functional, we should first ascertain that such a minimum exists. As for the other conditions, the nature of the problem gives cause for concern. Anyone who has been to a house of mirrors or been fooled by an optical illusion knows that a given observation can give rise to different interpretations. Also, matching points between images is very sensitive to errors in the positions and values of the points. In order to find a meaningful solution to our problem, we must first make it well-posed. We do this by imposing additional **constraints**.

3.1.1 Constraints

A very natural candidate is the celebrated **Occam's Razor**. It states that when several hypotheses explain our observations equally well, we should select the *simplest* one. In our case this means we should seek scenes made of *few and simple* objects. Instead of finding scenes that *exactly* match our observations, we seek the scene that maximizes a combined score of data fidelity and object simplicity. **Data fidelity** is measured as the distance between the observed images and those predicted by our reconstruction. **Object simplicity** is enforced by minimizing some relevant measure on the surfaces, such as total variation (TV) or surface area. Judiciously chosen, the simplicity constraint will make the problem satisfy both the latter well-posedness criteria. Several scenes may fit the data, but selecting the simplest such scene yields a unique solution. Stability comes from the neighbourhood effects of the constraint. For instance, minimizing surface area will penalize outliers, thus making the reconstruction less sensitive to noise. Once we know that our problem is well-posed, we can get down to the business of solving it.

3.2 Forward model

The key to any inverse problem is an understanding of the processes that formed the observations we have made. We thus need a model of the formation of images from a scene. As mentioned earlier, this is a very common problem and many different models exist. At the most general level, we assume that the scene has an **impulse response function** and that this function is linear. What this means is that a ray of light of a certain wavelength and incident direction into the scene yields a certain illumination of pixels in the observed images. Scaling the input intensity changes output intensities by the same factor. Different wavelengths may have different impulse response functions (for instance, the angles of refraction are different). Knowing the impulse response of the scene thus amounts to knowing the response at every pixel (2D) to incident light from any direction (2D) and any wavelength (1D) for a total of 5 dimensions of data needed. We would like to simplify our model greatly to make the inverse problem tractable. A model can be formed from three parts: **Reflectance**, **light** and **camera** models.

3.2.1 Reflectance models

Bi-directional reflectance distribution function

If we disregard transparent volumes, the interaction between light and objects is simply a matter of reflection and absorption at each point of the surfaces in the scene. Ideal, mirror-like surfaces will simply reflect light as a ray at the same angle to the surface normal. However, most real-world surfaces reflect light in a more complex manner. In general, for every incident angle, the light will be reflected in several directions. To completely describe the reflectance of a surface point, we need to know the amount of light reflected in every direction for every possible incident direction. Both incident and reflected ray directions are described by two angles (**azimuth** and **elevation**). A total description of the reflectance properties of a surface is called the **Bi-directional reflection distribution function** (BRDF). This function defines the amount of reflected light in a given direction (2D) as a result of incident light from a given direction (2D) at a certain point on the surface (2D). Thus, the BRDF is a 6-dimensional function. Depending on the discretization size of surface and angle elements, storing the BRDF on a computer can take up a prohibitive amount of space. Further simplification is required.

Specular+diffuse

A popular model for the BRDF is the **Specular+Diffuse** model. The reflectance of a surface is the sum of a **specular** (mirror-like) and a **diffuse** (cloth-like) term. Part of the light is absorbed, part spread equally in all directions, and part reflected like from a mirror. At each point we now have five scalar quantities to keep track of: Diffuse reflectance, specular reflectance and RGB absorption values. This BRDF model is thus only 2-dimensional, realizing huge storage savings. In his Ph.D. thesis, Hailin Jin[8] performs surface reconstruction using this BRDF. However, this model may be overly complicated in the case of Lambertian surfaces.

Diffuse only

A **Lambertian** surface exhibits only diffuse reflectance. This means that any point on the surface has the same appearance from all angles. Obviously, this property is helpful when trying to match points in different images of the object. We now only need to store the colour (RGB) or intensity (grayscale) of each point. This is known as a **texture map**.

3.2.2 Light models

Raytracing

A classic but computationally expensive lighting model, raytracing is the process of tracing rays of light from visible points to light sources to determine illumination. Intuitively, one might want to reverse this, tracing the rays *from* the light sources, but a large number of these rays never hit a visible surface element and should be ignored. Using a raytracing model is very ambitious - if successful, we would reconstruct not only the solids, but also the light sources themselves. On the other hand, we would have to select our constraints very carefully to ensure a well-posed problem. The use of raytracing in scene reconstruction is a tempting avenue for future work, but for now we focus on simpler models.

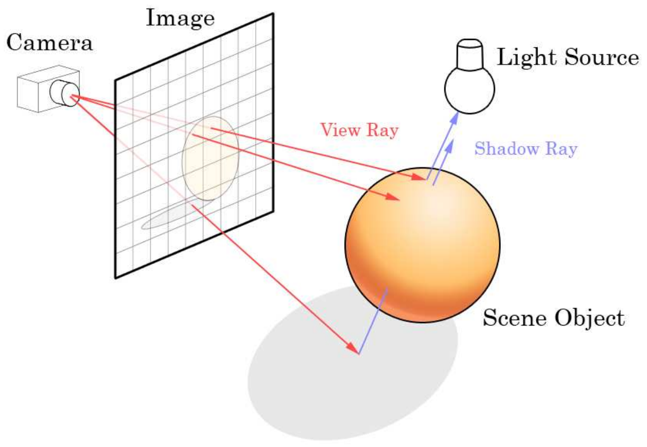


Figure 3.1: *Rays are traced from the camera centre through every pixel. When a ray hits a scene object, new rays are traced toward light sources to determine illumination. Image: Wikimedia Commons.*

Light map

If our surface is Lambertian, we can simplify our light model greatly. We simply specify the total intensity of the incident light at each point. The surface now supports a texture map and an intensity map. The observable intensity at a point is the product of these functions. One challenge is now to separate light effects from texture effects. For instance, a picture of a globe could either be a globe-textured ball in neutral light, a white ball with globe-textured light projected onto it, or anything inbetween. To make our problem well-posed, we need to impose constraints on our maps. In typical real-world scenarios, light sources are relatively smooth, while fine detail comes from surface texture. To capture such scenes, we require the light map to be smooth.

Intensity map

A less ambitious route is to combine the light and reflectance map into a single intensity map. We consider only the intensity of a point, and not whether it is due to light or texture. This simplified model has been very popular, see for instance [8] and [4]. Once a surface model and intensity map have been reconstructed, we can approximate the original scene by reprojecting the intensity map onto the surface.

Statistical distribution

Finally, we can abandon all thoughts about texture and instead see the intensity values on a surface as samples from a stochastic function. The surface and background support distinct stochastic functions, giving the likelihood of observing each intensity value. These functions can form the basis for an a posteriori maximum likelihood estimate. We outline the use of this model in surface reconstruction in Chapter 5.

3.2.3 Camera models

Having chosen our model for the interaction of objects and light, we need to understand the process of image formation. In this section we will first introduce a simplified but unrealistic model, and then build upon it to correct for its shortcomings.

Pinhole camera

The word *camera* means "room". The origin comes from *camera obscura*, a "dark chamber" illuminated by a small aperture into a neighbouring room. Light passing through the aperture would form an image of the next-door scene on a sheet of canvas, and an artist could use this image as a guide to more easily paint realistic portraits. Reducing the size of the chamber and substituting photosensitive film for the painter yields the pinhole camera. In the ideal case, the pinhole is just a point about which the scene is reflected (Fig. 3.2) . In reality, pinhole sizes are restricted by **diffraction**. Light, being an electromagnetic wave, spreads when passing through small openings. See, for instance, the groundbreaking double-slit experiment of Thomas

Young[17]. We appear to be at an impasse. Large apertures allow light from a single point in the scene to illuminate a large region of the sensor, smearing the image. On the other hand, small apertures spread the light, causing the same problem. How can we remedy the situation? A humble piece of glass may be the answer.

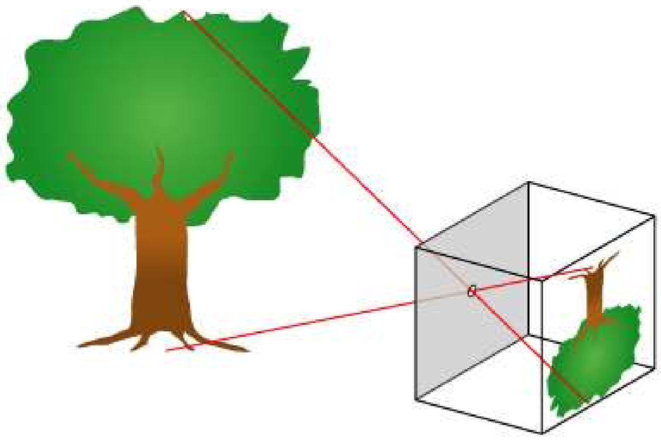


Figure 3.2: *A pinhole camera. Light rays pass through the aperture, creating an upside-down image of the object on the opposite wall. Image: Wikimedia Commons.*

Lens-based imaging

A convex lens focuses light, sending all rays of a given wavelength through one point (called the **optical centre**). Points in the scene are mapped to points on the sensor, creating sharp images. The downside is that only points at a certain range are in focus. By adjusting the aperture we can change the size of this depth of field (DOF). As long as the entire region of interest is within the DOF, we can ignore the effects of lens focus. When our scene is in focus, it is possible to create a one-to-one correspondence between camera pixels and the closest surface points. This is needed in reconstruction. Although the lens helps us get sharper images than with a pinhole camera, it introduces some new problems of its own, all maneagable:

- **Spherical aberration.** When using spherical lens elements, light hitting the edge of the lens has a different optical centre than light hitting the centre. More advanced aspherical lenses seek to correct this.

- **Image distortion.** Some lenses distort the geometry of scenes, mapping lines to curves. This distortion can be modelled and corrected in post-processing.

- **Chromatic aberration.** Lenses use refraction to focus light. Unfortunately, the refractive index of a lens is dependent of the wavelength of light passing through it, causing the red, blue and green parts of an image to be differently focused and shifted in relation to each other. This can be fixed by the use of an *apochromatic* lens, by using only one of the RGB channels (grayscale imaging), or in post-processing.

The process of determining an imaging system's distortion parameters and correcting for these is called **camera calibration**. When this has been done, it should be possible to model the imaging system's projection from scene coordinates to camera coordinates by a **projection matrix**, and to account for camera positions by a **rotation matrix** and a **translation vector**. Finding these values is called **camera resectioning**. For a description of camera calibration and resectioning, see [7]. A useful tool is the Camera Calibration Toolbox for Matlab by Klaus Strobl et al. However, for the numerical experiments in this thesis, we prefer to use datasets where camera calibration and resectioning have already been performed, both to save time and to avoid introducing additional error sources in our tests.

3.3 Summary

Having thus outlined the available models for reflectance, lighting and imaging, we are ready to embark on the problem of scene reconstruction. Having restricted ourselves to opaque objects, so that a scene can be described as a collection of surfaces, our first order of the day is to find a suitable way to mathematically describe surfaces. Our solution is the use of **level sets**, defined in Chapter 4.

Chapter 4

Level Set Methods

In image processing, we often want to track moving regions and boundaries such as curves and surfaces. This can be done explicitly, by defining and tracking control points on the region and using some form of interpolation to define the area between control points. However, this can require rather complicated housekeeping (adding and removing control points as needed, checking for self-intersections and so on). If the object changes significantly, or undergoes topological changes such as merging or splitting, there are much simpler options.

4.1 Level sets

For the above reasons, a static grid is desirable. However, we are trying to track a moving region. If our object is an N-dimensional object embedded in $N+1$-dimensional space, our grid must now cover at least those parts of $\mathbb{R}^{N+1}$ through which the object is expected to move. This added dimension is the price we must pay for not having to track regions explicitly. Following Osher and Sethian[12],we track our object as the zero level set of some function defined in our region of interest $\Omega \in \mathbb{R}^{N+1}$. I will outline two candidates for such a function.

4.1.1 Signed distance function

A classical function used in level set methods is the **signed distance function**, defined as

$$\phi = \begin{cases} dist(\xi, S) & \xi \text{ outside } S \\ 0 & \xi \in S \\ -dist(\xi, S) & \xi \text{ inside } S \end{cases},$$

for $\xi \in \Omega$. The volume of the region enclosed by S is defined as

$$\int_{\Omega} 1 - H(\phi) d\xi \ ,$$

where H is the Heaviside function. To define a measure on S itself, we differentiate the Heaviside function with respect to spatial coordinates.

$$\frac{d}{d\xi} H(\phi(\xi)) = \frac{d}{d\phi} H(\phi) \nabla\phi(\xi) = \delta(\phi(\xi)) \nabla\phi(\xi)$$

in the distributional sense. The surface area of S is then

$$|S| = \int_{\Omega} \delta(\phi(\xi)) \nabla\phi(\xi) d\xi \ .$$

The good news is we're still integrating over our entire grid and don't have to trouble ourselves with changing regions. The bad news is that we have to make a discrete approximation to the impulse function δ, see for instance [15]. Typically, these approximations tend to smear boundaries somewhat. Another challenge is the fact that evolutions of the function tend to make it stray from the ideal signed distance function, and it must be reinitalized. As evidenced by the very common usage of this formulation, these problems are manageable. But is it possible to introduce a framework that avoids these issues altogether?

4.1.2 Piecewise constant function

In [11],the idea of using a **piecewice constant** function $\phi(\xi)$ to track regions was introduced. The interior of each region is assigned a unique constant value. A single function can define any given number of regions. For a set of regions $\{\Omega_i\}, i = \{1, 2, \dots, n\}$, each region is defined as follows:

$$\Omega_i = \{\xi : \phi(\xi) = i\}.$$

To each region we associate a characteristic function

$$\psi_i(\xi) = \left\{ \begin{array}{ll} 1 & \xi \in \Omega_i \\ 0 & \text{elsewhere} \end{array} \right. .$$

These functions are defined as

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

Integrals over subregions can now be taken over the entire domain:

$$\int_{\Omega_i} f(\xi)d\xi = \int_{\Omega} \psi_i f(\xi)d\xi.$$

The task of tracking the changing regions has been delegated to the basis functions, which in effect become masks limiting the region of our integrals. The area of the surfaces bounding the regions is

$$|S_i| = \int_{\Omega} |\nabla \psi_i| d\xi. \tag{4.1}$$

The volume of an object is easy to find:

$$|\Omega_i| = \int_{\Omega} \psi_i d\xi.$$

Having defined our objects implicitly, verifying that basic properties such as area and volume are readily available, we have developed a natural framework for studying evolving surfaces. We will be testing this framework on a real-world problem in Chapter 5.

Chapter 5

Probabilistic Surface Reconstruction

Having laid out our toolbox in the previous Chapters, we are ready to begin the task of computer vision proper. We want to test the PCLSM on the task of surface reconstruction, using the pinhole camera model and stochastic surface texture. Kolev[9] implemented such a scheme using signed distance level sets; we will modify his algorithm to use the PCLSM instead.

5.1 Probabilistic functional

Our scene Ω consists of a solid region Ω_{obj} and a transparent region Ω_{bck}. A background, which we shall not treat explicitly, exists outside Ω and is visible through Ω_{bck} whenever it is not occluded by Ω_{obj}. A closed surface S bounds Ω_{obj}, separating it from Ω_{bck}. A set of images $\{I_1, \dots, I_n\}$ with associated perspective projections $\{\pi_1, \dots, \pi_n\}$ detail the scene. We want to find the surface $\hat{S}$ that best describes what the cameras see. This optimality is framed as a maximum a posteriori likelihood:

$$\hat{S} = \arg\max_{S \in \Lambda} P(S|I)$$

where Λ is the set of closed surfaces in Ω, $P(S|I)$ is the conditional probability of S given I, and $I \doteq \{I_1, \dots, I_n\}$. From Bayes' formula we have

$$P(S|I) = \frac{P(I|S)P(S)}{P(I)}.$$

$P(I)$ does not depend on S, and we can ignore it in the arg max problem. We now need explicit functions $P(I|S)$ and $P(S)$. If we assume that the voxels are independent, we arrive at

$$P(I|S) = \left[\prod_{\xi\in\Omega} P(I(\pi(\xi))|S)\right]^{d\xi}$$

where $I(\pi(\xi)) \doteq \{I_i(\pi_i(\xi))\}, i = 1 : n$ and $d\xi$ is one divided by the number of voxels in Ω. Since a surface S defines a partitioning of Ω, we see that

$$P(S|I) \propto \left[\prod_{\xi\in\Omega_{obj}} P_{obj}(\xi) \cdot \prod_{\xi\in\Omega_{bck}} P_{bck}(\xi)\right]^{d\xi} \cdot P(S)$$

where $P_{(\cdot)}(\xi) \doteq P(I(\pi(\xi))|\xi \in \Omega_{(\cdot)})$. If we could assume independence between images, we could further specify these functions:

$$\begin{aligned} P_{obj}(\xi) &= \prod_{i=1}^{n} P(I_i(\pi_i(\xi))|\xi \in \Omega_{obj}) \\ P_{bck}(\xi) &= 1 - \prod_{i=1}^{n} [1 - P(I_i(\pi_i(\xi))|\xi \in \Omega_{bck})]. \end{aligned}$$

The assymetry between the expressions is because a point on the object will be observed as on the object by all cameras, whereas an point in the transparent region around the object will be observed as background by at least one camera, provided we have enough cameras to fully resolve the object. These equations look promising, but they are based on a flawed assumption. The images are all of the same object, and will obviously be highly correlated. In fact, our problem would be impossible to solve otherwise! Instead, we take the geometric mean of the individual probabilities, in effect giving each image an equal "vote" on the true grayscale value of a point:

$$\begin{aligned} P_{obj}(\xi) &= \sqrt[n]{\prod_{i=1}^{n} P(I_i(\pi_i(\xi))|\xi \in \Omega_{obj})} \\ P_{bck}(\xi) &= 1 - \sqrt[n]{\prod_{i=1}^{n} [1 - P(I_i(\pi_i(\xi))|\xi \in \Omega_{bck})]}. \end{aligned}$$

For the conditional probabilities of intensity observations, we assume the object and background support distinct stochastic functions. More precisely, we posit Gaussian distributions of greyscale values on each region:

$$P(I_i(\pi_i(\xi))|\xi \in \Omega_{(\cdot)}) = f(I_i(\pi_i(\xi)); \mu_{(\cdot)}, \sigma_{(\cdot)})$$

where f is the probabiltity density function of the normal distribution:

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The mean and standard deviation parameters of the distributions can be set manually or updated adaptively. If we are using an iterative algorithm to approach $\hat{S}$, we can update the values from the actual observations of the current regions:

$$\mu_{(\cdot)} = \underset{\xi\in\Omega_{(\cdot)}}{mean}(I(\pi(\xi))),\ \sigma_{(\cdot)} = \sqrt{\underset{\xi\in\Omega_{(\cdot)}}{var}\,(I(\pi(\xi)))}.$$

Having defined a model for $P(I|S)$, we choose a measure for $P(S)$ that maintains well-posedness (see Chapter 3). We consider complicated surfaces to be less likely than simple ones:

$$P(S) = e^{-\nu|S|}$$

where ν is a positive constant that can be tuned to produce smoother surfaces at the expense of poorer data fitting. Putting the pieces together, our problem reads

$$\hat{S} = \arg\max_{S\in\Lambda}\left[\prod_{\xi\in\Omega_{obj}} P_{obj}(\xi)\cdot\prod_{\xi\in\Omega_{bck}} P_{bck}(\xi)\right]^{d\xi}\cdot P(S).$$

Since all probabilities are nonnegative and the logarithm function is monotonically rising, maximizing this functional is equivalent to maximizing its logarithm. We take its negative to convert it into an energy minimization problem:

$$\hat{S} = \arg\min_{S\in\Lambda} E_d(S) \doteq -\sum_{\xi\in\Omega_{obj}}\log(P_{obj}(\xi))d\xi + \sum_{\xi\in\Omega_{bck}}\log(P_{bck}(\xi))d\xi + \nu|S|. \tag{5.1}$$

In the continuum limit the sums become integrals, and we will use this notation for the rest of the Chapter, though the calculations are of course on a finite grid.

5.2 Implementation

We wish to formulate (5.1) in the PCLSM framework. Following our discussion in Chapter 4, we get:

$$E(\phi) = -\int_\Omega [\psi_{obj}\log P_{obj}(\xi) + \psi_{bck}\log P_{bck}(\xi)]d\xi + \nu\int_\Omega |\nabla\psi_{obj}|d\xi.$$

Since we are only dealing with two regions, we can simplify our level set framework by apt choices of the constant values:

$$\phi(\xi) = \begin{cases} 1, & \xi \in \Omega_{obj} \\ 0, & \xi \in \Omega_{bck} \end{cases}$$

The final version of the functional reads

$$E(\phi) = -\int_\Omega [\phi(\xi)\log P_{obj}(\xi) + (1-\phi(\xi)\log P_{bck}(\xi)]d\xi + \nu\int_\Omega |\nabla\phi(\xi)|d\xi$$

subject to

$$K(\phi) \doteq \phi(\phi-1) = 0$$

to keep the piecewise constant level set function ϕ valued either 0 or 1 at each point. We recognize this as a constrained minimization problem amenable to the method of Lagrangian multipliers. We augment the functional with a Lagrangian term:

$$\Lambda(\phi,\lambda) = E(\phi) + \tilde{\lambda}(K(\phi))$$

where

$$\tilde{\lambda}(K) = \int_\Omega \lambda K(\phi)d\xi$$

is the distribution corresponding to $\lambda \in BV_0(\Omega)$. To find a minimum point we need to solve the Euler-Lagrange equation:

$$d\Lambda(\phi,\lambda;u,v) = 0 \qquad \forall u,v \in BV_0(\Omega),$$

cf. (2.5). Splitting the operator, we have

$$dE(\phi;u) + d\left[\int_\Omega \lambda K(\phi)d\xi\right](\phi;u) = 0$$

and

$$d\left[\int_\Omega \lambda K(\phi)\right](\lambda;v) = 0.$$

Solutions of the augmented functional correspond to saddle points. We need to minimize $\Lambda(\phi,\lambda)$ with respect to ϕ and maximize it with respect to λ. To find this critical point, we use a gradient descent method. The derivatives of our augmented functional with respect to ϕ and λ give gradient directions in function space. Locally, moving ϕ in the negative of this direction will decrease the functional value. Similarly, we move λ in the positive gradient direction. If we repeat this iteratively, using sufficiently small steps, we will move toward local minima. Starting with a sensible initialization ϕ_0, λ_0, we hope to converge to a global minimizer for the constrained problem. To this end, we introduce an **artificial time** parameter t, arriving at these PDEs:

$$\frac{\partial\phi}{\partial t} = \frac{\partial\Lambda}{\partial\phi} \qquad \phi|_{t=0} = \phi_0$$
$$\frac{\partial\lambda}{\partial t} = \frac{\partial\Lambda}{\partial\lambda} \qquad \lambda|_{t=0} = \lambda_0$$

To find the derivatives of Λ we need to employ weak differentiation. We shall show the process for the most difficult part of our functional; the other parts are similar but easier. We need to find the weak derivative $d[\int_\Omega |\nabla\phi|d\xi](\phi;u)$ for an arbitrary $u \in BV_0(\Omega)$.

$$\begin{aligned}
d[\int_\Omega |\nabla\phi|d\xi](\phi;u) &= \lim_{\epsilon\to 0}\int_\Omega \frac{|\nabla(\phi+\epsilon u)| - |\nabla\phi|}{\epsilon}d\xi \\
&= \int_\Omega \lim_{\epsilon\to 0}\frac{|\nabla(\phi+\epsilon u)| - |\nabla\phi|}{\epsilon}\cdot\frac{|\nabla(\phi+\epsilon u)| + |\nabla\phi|}{|\nabla(\phi+\epsilon u)| + |\nabla\phi|}d\xi \\
&= \int_\Omega \lim_{\epsilon\to 0}\frac{(\nabla(\phi+\epsilon u))^2 - (\nabla\phi)^2}{\epsilon(|\nabla(\phi+\epsilon u)| + |\nabla\phi|)}d\xi \\
&= \int_\Omega \lim_{\epsilon\to 0}\frac{\epsilon\nabla\phi\cdot\nabla u + \epsilon^2(\nabla(u))^2}{\epsilon(|\nabla(\phi+\epsilon u)| + |\nabla\phi|)}d\xi \\
&= \int_\Omega \frac{\nabla\phi}{2|\nabla\phi|}\nabla u d\xi
\end{aligned}$$

We now turn to the integration by parts formula (Eq. 2.2). Define the vector

$$w = \frac{\nabla\phi}{2|\nabla\phi|} = (w^{(1)}, w^{(2)}, w^{(3)}).$$

For each i,

$$\int_\Omega u_{x_i}w^{(i)}d\xi = -\int_\Omega uw^{(i)}_{x_i}d\xi + \int_{\partial\Omega} uw^{(i)}n(i)dS.$$

Since $u \in BV_0(\Omega)$, the last term is zero. Summing over i, we get:

$$\int_\Omega \frac{\nabla\phi}{2|\nabla\phi|}\nabla u d\xi = -\frac{1}{2}\int_\Omega \nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)ud\xi.$$

Putting this part in with the rest of the terms in our weakly differentiated functional, we get that

$$-\int_\Omega \left[\log P_{bck} - \log P_{obj} - \frac{\nu}{2}\nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right) + \lambda(2\phi-1)\right]ud\xi = 0$$

for all $u \in BV_0(\Omega)$. For this to be true for any u, the functional in the brackets must be everywhere zero. Together with a similar condition from the weak derivative with respect to λ, this the **Euler-Lagrange condition** for our algorithm:

$$\log P_{bck} - \log P_{obj} - \frac{\nu}{2}\nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|}\right) + \lambda(2\phi - 1) = 0 \tag{5.2}$$

$$K(\phi) = 0 \tag{5.3}$$

yielding the evolution equations

$$\frac{\partial \phi}{\partial t} = -\log P_{bck} + \log P_{obj} + \frac{\nu}{2}\nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|}\right) - \lambda(2\phi - 1) \tag{5.4}$$

$$\frac{\partial \lambda}{\partial t} = K(\phi). \tag{5.5}$$

To solve these coupled PDEs, we adapt the algorithm from the original PCLSM paper [11]:

Algorithm. Initialize ϕ^0 and λ^0. `for` $k = 1$ `do`:

- Find μ_{obj}, σ_{obj}, μ_{bck} and σ_{bck} from

  $$\mu_{(\cdot)} = \text{mean}\left(\sum_{i=1}^{n} \int_\Omega I_i(\pi_i(\xi))\psi_{(\cdot)} d\xi\right)$$

  and

  $$\sigma_{(\cdot)} = \text{std}\left(\sum_{i=1}^{n} \int_\Omega I_i(\pi_i(\xi))\psi_{(\cdot)} d\xi\right).$$

- Update the level set:

  $$\phi^k = \phi^{k-1} + \triangle t \frac{\partial L(\phi^{k-1}, \lambda^{k-1})}{\partial \phi}$$

  given the statistical parameters.

- Update the Lagrangian multiplier by

  $$\lambda^k = \lambda^{k-1} + rK(\phi).$$

- `if not` converged: Set $k = k + 1$ and return to step 1.

A few hitches are apparent in this algorithm. First, the means and standard deviations take into account all points in the relevant volumes, not only points that are visible from a given camera. This is the best we can do when not dealing with visibility issues. The effect of this solution will be to give extra weight to the values of points on the surface where the volume is thick. We do not suspect that this will be a large problem, but there are other reasons to consider visibility as well. We will discuss this in the future work section. Another problem is the fact that while ϕ will be close to 0 and 1 at convergence, it may stray far from this ideal during iterations. This may affect the accuracy of the surface measure. We would do well to monitor the magnitude of $K(\phi)$ during iteration.

Chapter 6

Graph Cuts

In Chapter 5 we have seen how our functional can be minimized using an augmented Lagrangian/steepest descent method. While this method imposes a heavy computational burden, it is conceptually intuitive and suitable to a very general class of problems. However, as is often the case in computational mathematics, we can trade simplicity for computational efficiency. An emerging field in the image processing community is the use of so-called graph cuts. Many problems can be interpreted as a labeling problem, i.e. we want to classify points into one of two groups. To each such assignment is associated a certain cost, and we want to globally minimize the cost of our assignments. If this cost functional satisfies certain criteria, it is possible to establish a one-to-one correspondence between this labeling problem and the min cut/max flow problem in graph theory. This is a much studied problem, and highly efficient algorithms exist for its solution. Typically, a speed increase of several orders of magnitude can be achieved when the graph cut method is employed. In addition, the method is guaranteed to achieve a global minimum, and requires less human attention due to the lack of tuning parameters. Though the speed of computers is still on the rise, we should not expect the emphasis on efficient algorithms to abate. Typically, faster machines allow the solution of more difficult problems that were practically impossible before. Also, many image processing algorithms have their utility greatly increased if they can be run in real-time on commonplace hardware. This steady “mission creep” ensures that there will always be a need for efficiency.

6.1 Relation to reconstruction problem

Due to the speed and guaranteed global minimization of graph cut algorithms, we wish to formulate our reconstruction functional as a max flow problem. Single-object reconstruction can be viewed as a **voxel labelling** problem. We seek to label each volume element (voxel) in our region of interest Ω as part of the object or the background. To each voxel in Ω we assign a node in the graph. A cut C defines a labelling of all voxels, with the nodes in the source set corresponding to foreground voxels. If we are able to define edge weights that make the cost of any cut equal to the energy of the corresponding reconstruction functional, the min-cut algorithm will solve our problem. The definition of edge weights is the main challenge in graph cut formulations. A significant part of this chapter will deal with this.

6.2 Edge weights

Let us first review the Kolev functional:

$$E(S) = -\int_{V_{obj}} \log P_{obj}(\xi)dx - \int_{V_{bck}} \log P_{bck}(\xi)d\xi + v|S|.$$

If we fix the mean and standard deviation, the first two integrals are separable in the sense that the classification of any single point into background or foreground does not affect the contribution of any other points. This allows a simple representation in graph cut form. We add an edge from the source node to each voxel node, and one from each voxel node to the sink node. The capacities of the source and sink edges is equal to the cost of labelling the node as background or foreground, respectively. If not for the regularization term, we would be done now. Adding edge weights that represent this term will prove to be more difficult.

6.2.1 Representing surface area

We have the choice of two paradigms for surface area representation. Recall from (4.1) that the area of the surface enclosing a volume is given by the total variation of its characteristic function:

$$|S_i| = \int_{\Omega} |\nabla \psi_i| dx.$$

Ideally, we would like to use the TV_2-norm, as it is anisotropic. In our finite difference framework this becomes

$$\sum_{i,j,k} \delta \sqrt{|\psi_{i+1,j,k} - \psi_{i,j,k}|^2 + |\psi_{i,j+1,k} - \psi_{i,j,k}|^2 + |\psi_{i,j,k+1} - \psi_{i,j,k}|^2}$$

where δ is the volume of a grid cell. However, to be graph-representable, the function must be separable into a sum of functions of two points at a time. These simple functions can then be represented by weighted edges between the nodes corresponding to these points. A discussion of this fact and an extension to functions of three points at a time (though with added 'dummy' nodes) can be found in [10]. Obviously, the TV_2-norm can not be separated in this way. We could fall back to the TV_1-norm

$$\sum_{i,j,k} \delta(|\psi_{i+1,j,k} - \psi_{i,j,k}| + |\psi_{i,j+1,k} - \psi_{i,j,k}| + |\psi_{i,j,k+1} - \psi_{i,j,k}|),$$

but this is anisotropic and would favour blocky surfaces. A solution might be to use a weighted sum of rotated total variation norms, as outlined for the 2D case in [13]:

$$TV_{1,\frac{\pi}{4}}(u) = \frac{1}{2}(TV_1(u) + TV_1(R_{\frac{\pi}{4}}(u)))$$

where $R_{\frac{\pi}{4}}$ is the operator which rotates its argument $\frac{\pi}{4}$ radians. We need to generalize this result to three dimensions, and will give more precise definitions in that discussion.

$\frac{\pi}{4}$-isotropic TV_1-norm in 3D

In the 2D case, it is possible to make the norm isotropic under any rotation that is a multiple of $\frac{\pi}{4}$. In 3D, this is impractical because there is an infinite number of rotation directions. We must settle for making our norm invariant under a subset of the possible rotations. We base our weights on the discussion in [1]. This paper discusses norms in rather general 3-D Riemannian spaces and is a bit too involved to summarize here. We resign ourselves to merely simplifying their result to our regular 3-D grid:

$$w_k = \frac{\triangle x \triangle y \triangle z \triangle \theta_k}{\pi |e_k|^3}.$$

w_k is the flow constraint on the arc from a node to its neighbour with relative position e_k. $\triangle x, \ldots$ are the grid size parameters. $\triangle\theta_k = \triangle\psi_k \triangle\phi_k$ is the angular discretization step of the neighbourhood system. To make this slightly less obscure, we will take as an example a 6-neighbourhood system on a Cartesian grid. In this case, $\triangle x = \triangle y = \triangle z = \delta$, all e_k are canonical unit vectors, and $\triangle\psi_k = \triangle\phi_k = \pi/2$ for all k. All edge weights are thus equal to $w_k = \frac{\delta^3\pi^2/4}{\pi\delta^3} = \frac{\pi}{4}$. Since we multiply the surface measure by a smoothing factor, we can meld it with this factor and thus need not pay much attention to the discussion here. For other neighbourhood systems, we are not so lucky. However, if we restrict ourselves to Cartesian grids and discretize the sphere with equal steps, the non-constant part of the edge weights is simply a division by the cube of the length of the vector e_k. To save ourselves from headaches, we shall put all other terms into the smoothing parameter w.

6.3 Graph construction

We construct a graph $G = (\mathcal{N}, \mathcal{A})$ thus:

- $\mathcal{N}$ consists of a source node s, a sink node t and a node n_{abc} for each voxel $\xi_{abc} = \xi_0 + a \cdot dx\vec{i} + b \cdot dy\vec{j} + c \cdot dz\vec{k}$ in the grid.
- $\mathcal{A}$ has three types of members:
  - An arc (s, n_{abc}) from the source node s to each interior node with associated flow constraint
$$u_{sn_{abc}} = P_{air}(\xi_{abc}) = 1 - \sqrt[n]{\prod_{i=1}^{n}[1 - P(I_i(\pi_i(\xi_{abc}))|\xi_{abc} \in V_{air})]}.$$
  - An arc (n_{abc}, t) from each interior node to the sink node t with associated flow constraint
$$u_{n_{abc}t} = P_{obj}(\xi_{abc}) \sqrt[n]{\prod_{i=1}^{n} P(I_i(\pi_i(\xi_{abc}))|\xi_{abc} \in V_{obj})}. \tag{6.1}$$
  - Neighbourhood arcs. In a 6-neighbourhood, arcs from each interior node n_{abc} to $n_{(a-1)bc}, n_{(a+1)bc}, n_{a(b-1)c}, \ldots, n_{ab(c+1)}$, each with weight w. For other neighbourhoods, see the discussion in Section 6.2.1.

6.4 Pseudocode

```
read images and coordinates
```

```
create grid
while not converged do
    create graph
    for all voxels in grid do
        add node to graph
        add edges to source and sink node
        add edges to nodes corresponding to neighbourhood
        add weight to source node edge: $P_{bck}$
        add weight to sink node edge: cost of classifying as $P_{obj}$
        add weights to neighbourhood edges: surface area representation
    end for
    solve min-cut problem
    update mean and deviation from new partitioning
end while
```

6.5 Implementation

We define the flow constraint in Section 6.3. We use a combination of MATLAB and C++ for this part. MATLAB is useful for its intuitive and powerful handling of 3D graphics, but is rather slow when running nested `for` loops like our code does. Our code constructs the graph whose maximum flow solves the minimization problem; for the task of solving the max-flow problem itself we gratefully use the library provided by Boykov and Kolmogorov [2]. Camera parameters and images are read in MATLAB and processed to form an $k \times l \times m \times n$ array of intensity values, where k, l and m is the grid size and n is the number of cameras. For each point x, y, z in the grid and each camera i, we find the value $I_i(\pi_i(x, y, z))$ "seen" by camera i at that point. This array is then written to a file in a format amenable to being read by our C++ program. Once read, we use these values in the calculation of the P_{obj} and P_{bck} values central to the Kolev algorithm. Finally, we construct a graph with appropriate weights and solve it using the Boykov-Kolmogorov library. Since the graph-cut solver is not iterative, we cannot update the statistical parameters continually like in the gradient descent method. Instead, we run the graph cut solver, calculate new parameters and feed them back into the solver. We hope to show experimentally that this method quickly converges and is still faster than the gradient descent method.

Chapter 7

Results

Having defined and implemented our surface reconstruction algorithms, the time has come to put them to the test. For the test we need a suitable dataset: A collection of pictures of an object, with known camera locations. Generating usable datasets is a meticulous task and well beyond the scope of this thesis. We have used a dataset provided by the vision group at Middlebury College as part of a project comparing stereo reconstruction algorithms[14]. This dataset is a plaster reproduction of the temple of the Dioskouroi in Agrigento, Sicily. We use the 16-image "sparse ring" set; some views are included in figure 7.1. Since the model has a relatively uniform hue, little information is lost by using greyscale versions of the images. We choose the red channel.

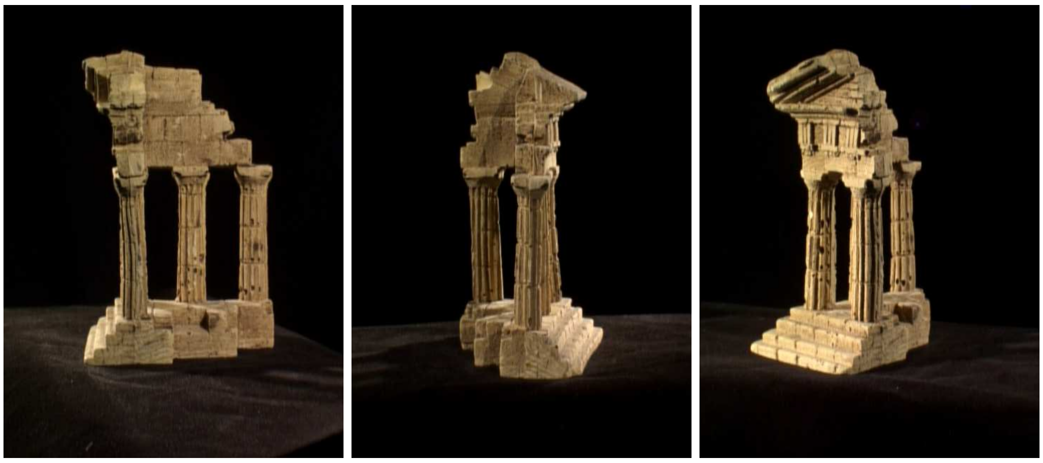


Figure 7.1: *3 out of 16 views from the sparse ring temple dataset. The images have been rotated to be upright.*

The camera perspective projection is modelled by a matrix K, equal for all images in the set. Each camera view i is also represented by a rotation matrix R_i and translation vector t_i.

t_1	R_1			K		
-0.0727	0.0219	0.9833	-0.1807	1520.4	0	302.3
0.0223	0.9986	-0.0127	0.0520	0	1525.9	246.9
0.6146	0.0488	-0.1816	-0.9822	0	0	1

Table 7.1: *Example translation vector, rotation and perspective projection matrices corresponding to image 1 in the sparse ring temple dataset.*

7.1 Gradient descent version

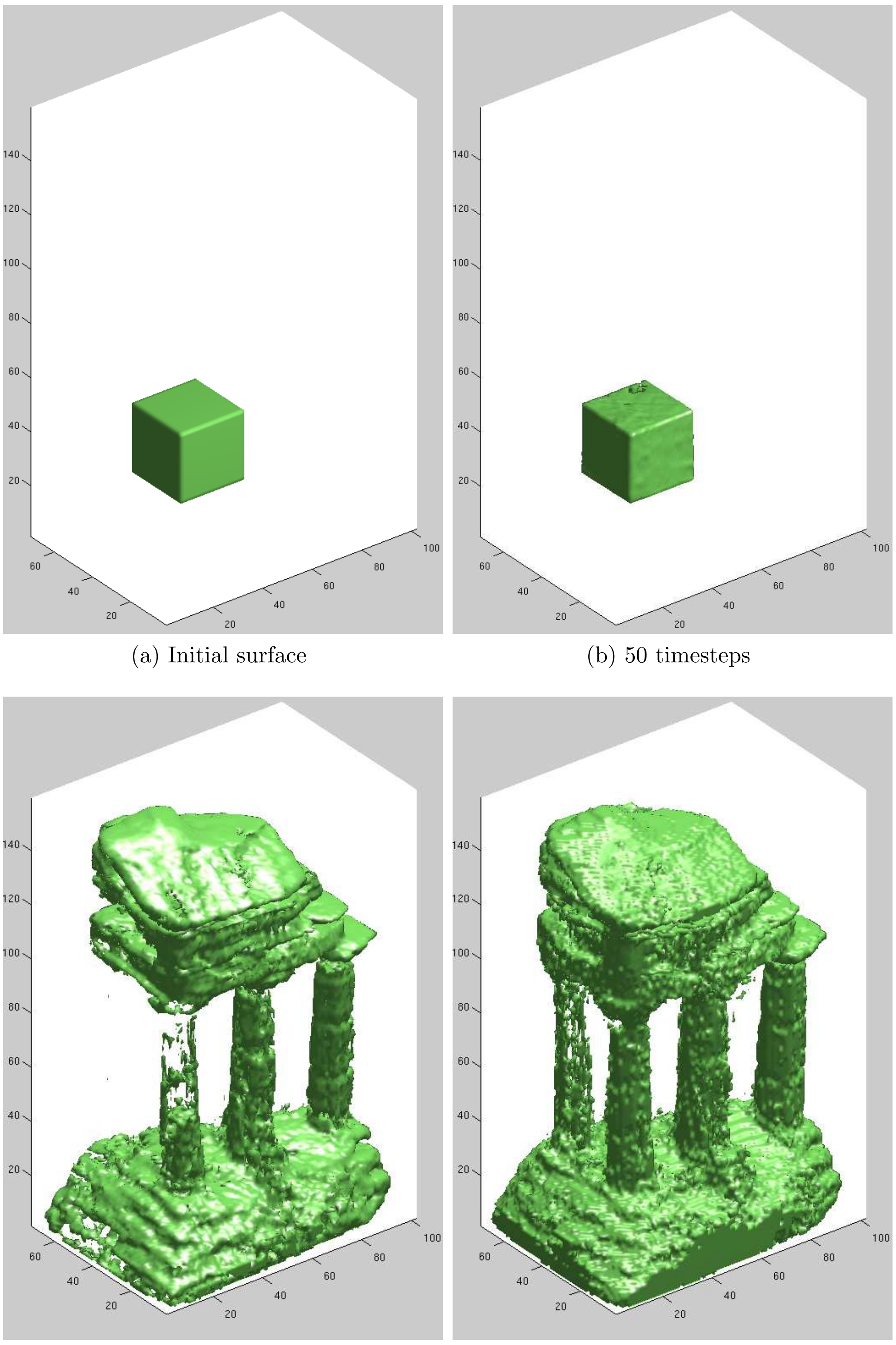
In this section we test the gradient-descent augmented Lagrangian solver for the PCLSM version of the Kolev algorithm. We want to test the visual quality of a high-resolution reconstruction, its tolerance to noise and robustness when faced with sparse datasets.

7.1.1 PCLSM test 1: Temple dataset, high resolution

We define a grid yielding a tight bounding box around the temple. The relatively high resolution of 160x100x70 is chosen, yielding 1,12 million volume elements ("voxels"). These numbers correspond roughly to the aspect ratios of the bounding box, yielding roughly cubic voxels. This is not required by our algorithm, but there seems no reason to use a different resolution in each principal direction. The initial surface is selected rather carelessly as a prism at an arbitrary location in the interior of the grid. It is our working hypothesis that the algorithm is quite robust in the face of poor initializations, as long as we do not create deliberately pathological cases. We run 50 iterations at a time, recording the updated statistical values and evolved surfaces. As the evolution of the surface seems to slow down, we increase the value of the term r pulling the solution toward ± 1.

Grid size	dx	dy	dz	dt	n
$160 \times 100 \times 70$	0.00103	0.00101	0.00109	0.1	16
μ_{obj}	σ_{obj}	μ_{bck}	σ_{bck}	ν	r
0.4538	0.3091	0.3896	0.3215	10^{-6}	10^{-8}

Table 7.2: *Initial values for running example 1. The averages and standard deviations have been calculated from the initial surface.*



(a) Initial surface

(b) 50 timesteps

(c) 100 timesteps

(d) 750 timesteps

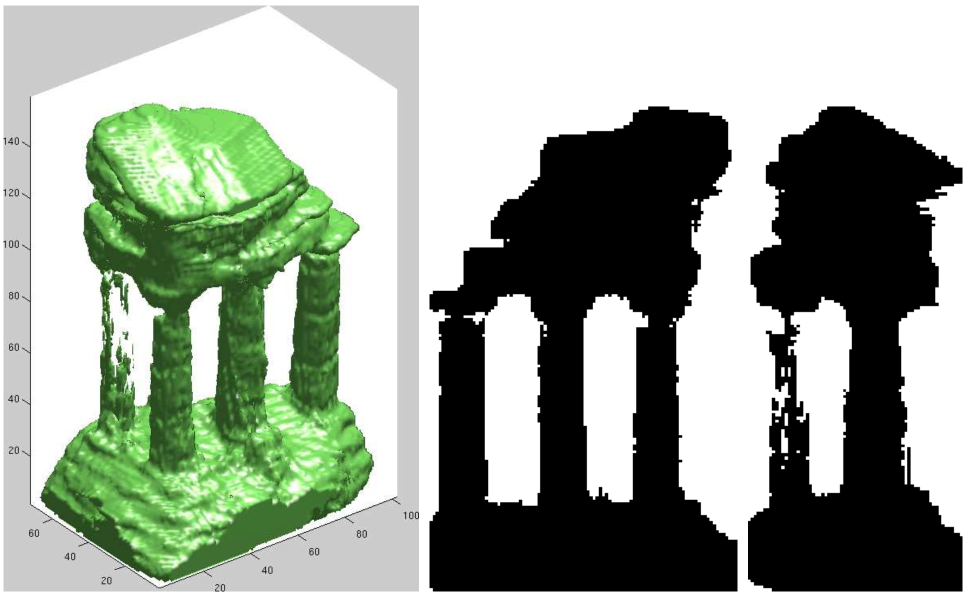
Figure 7.2: *The evolving surface at $t = 0$, $t = 50$, $t = 100$ and at convergence ($t = 750$). Sharp edges near the edge of the grid are due to truncation.*

We run the algorithm in chunks of 50 timesteps and evaluate the results. Isosurfaces of the level set function at 0 is shown in 7.2. We see that there is little change from $t = 0$ to $t = 50$ and a comparatively large leap from $t = 50$ to $t = 100$. This is because a point in the background set, initialized to -1, will still register as a background point until it passes 0. We do not know what values are the best for monitoring convergence; we stopped iterations when $\max dL < 0.1$, the number of new object voxels after 50 timesteps was less than 1/1000 the total number of grid cells, and the object showed no visible evolution. This is probably a conservative estimate. The converged surface is encouragingly life-like, but we observe a few spurious free-floating voxel clusters. We know that the temple model is a single compact object, and we would like our reconstruction to reflect that. Taking the converged surface as the initial surface of a new evolution, we increase the smoothing parameter ν from 10^{-6} to 10^{-4}. As seen in Figure 7.3b, this successfully removes most outliers, but also seems to have an adverse effect on the reconstruction of one of the columns. Increasing ν to 10^{-3} removes remaining outliers, but also the mentioned column in its entirety.

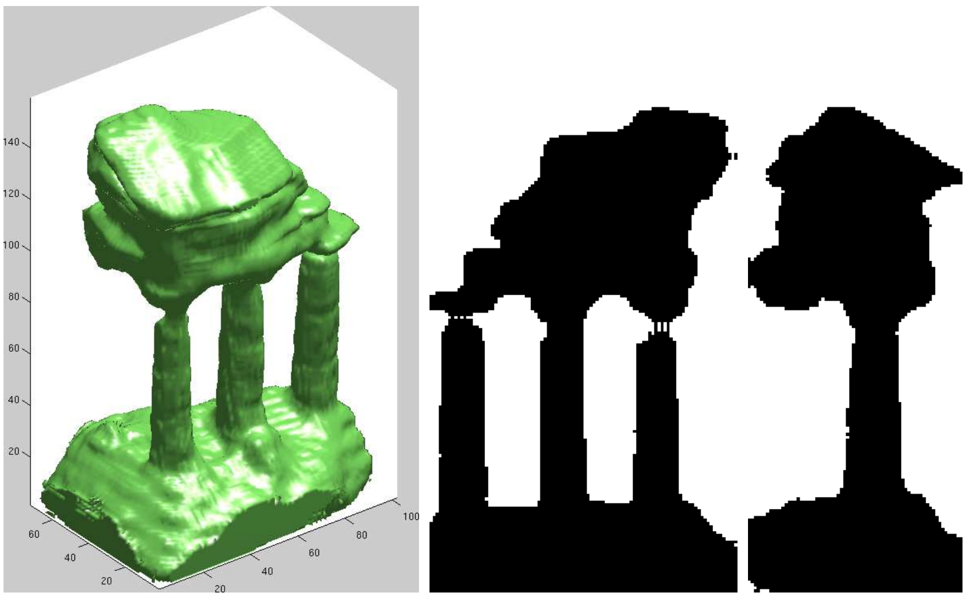
7.1.2 PCLSM test 2: Heavy Gaussian noise

An advantage of the probabilistic approach as opposed to local correspondence methods is robustness to noise. We want to test the ability of our algorithm to recover a surface from images that have been heavily affected by noise. First, we find the average of the standard deviations of our original images, $\bar{\sigma_I} = 0.2608$. We corrupt each image with additive Gaussian noise of the same standard deviation, for a signal-to-noise ratio of $1 : 1$. This is a significant amount that would be devastating to any algorithm depending on gradients in the image domain.

Our first attempt is to use the same naïve initialization we employed in example 1. However, the algorithm fails to converge, probably due to the similarity of the means in the initialized regions. We try another approach. We manually select two regions in one of the images, one representing the background and one representing the object, see Figure 7.4a. We calculate the statistics of these regions and fix them as inputs for our algorithm. We set the smoothing term ν and constraint term r to zero, allowing us to do large timesteps. Taking three steps with $dt = 1$ yields a very rough initial guess as seen in Figure 7.4b. The method outlined here is akin to a simple tresholding, which is sometimes used as the initial guess for 2-D segmentation. Since we want a smooth result, we apply smoothing to the level set function using MATLAB's `smooth3` and halve the resolution in each direction. With this degree of noise any high resolution details will be spurious



(a) $\nu = 10^{-4}$



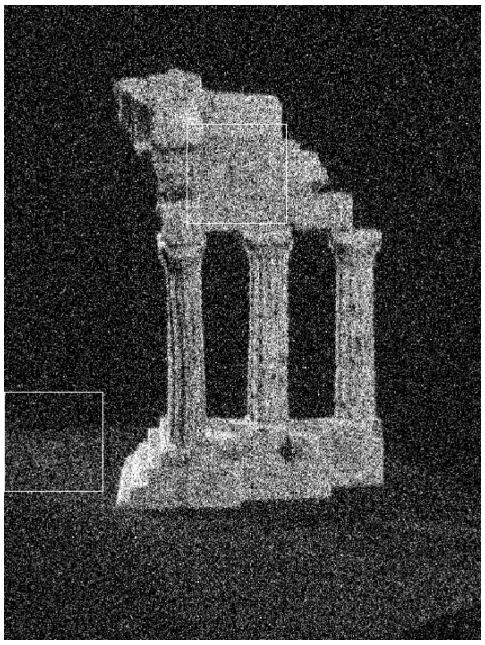
(b) $\nu = 10^{-3}$

Figure 7.3: *Reconstructed surface using gradient descent/PCLSM method. 3-D view and principal silhouettes for two different values of smoothing parameter ν.*

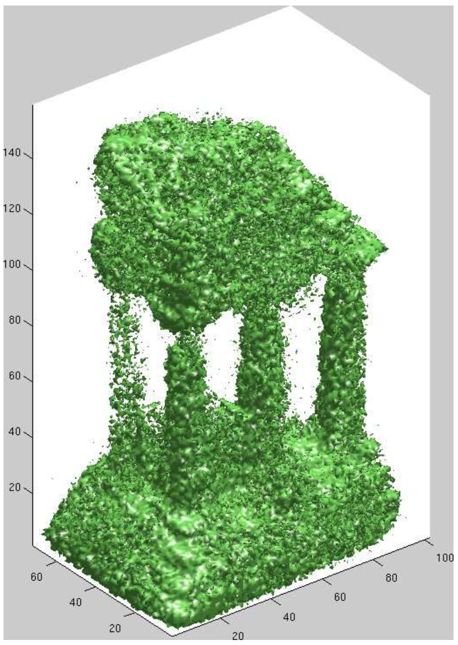
at any rate. Finally, we constrain this smoothed function to the values ± 1 and use this as our initial surface. Since we are dealing with very noisy data, we set our smoothing term to $\nu = 10^{-3}$. This requires a short time step, but since we have a reasonably good initial guess, we hope to see swift convergence nevertheless. We set $r = 10^{-8}$ and $dt = 10^{-3}$. In this experiment our algorithm faces a problem. At low smoothing settings, we do not get a continuous object. At higher settings, the surface shrinks away. We suspect this is due to the high standard deviations making it easier for the background set to aquire the darkest parts of the object set; this in turn increases the mean and standard deviation of the background set and so on. This feedback loop eventually causes the surface to shrink away altogether. To remedy this, we try fixing the statistical parameters at the initial values.This has the added advantage of increasing the running speed by about 70%, since the statistics updating subroutine does not fully take advantage of the array optimizations in MATLAB. By making this change we are straying away from our true algorithm. Future versions may be made adaptive, updating statistical parameters less often when noise is high. This will make convergence slower but increase stability.

7.1.3 PCLSM test 3: 4 temple images

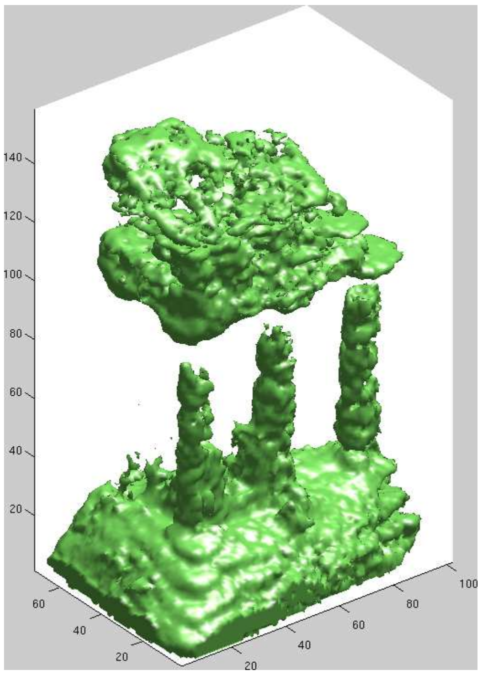
In real scenarios, the number of input images might be small. We want to test the performance of our algorithm in this case. We manually select 4 out of the 16 images from the temple sparse ring dataset. We try to select images with clear silhouettes and from views that are far apart. Our algorithm does not apply local correspondence and needs such cues to give acceptable results. With so little data to work with, we need a relatively large smoothing parameter to make the algorithm stable. The result of running the algorithm to convergence with the parameter ν set to $2 \cdot 10^{-4}$ is shown in Figure 7.5a. The result looks fairly decent, but we see that the temple now has 6 columns instead of 4! This is because the locations of the spurious extra columns were hidden behind or in front of other columns in all our views, and thus were “seen” as object points by all cameras. Apart from this the reconstruction has some outliers and artifacts, but is still surprisingly good.



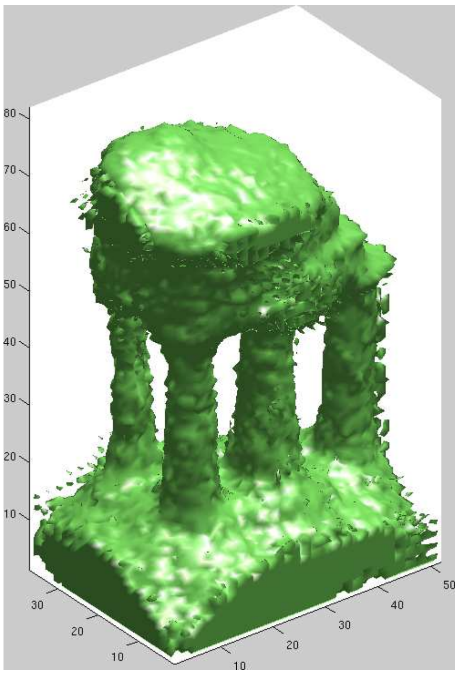
(a) Regions representative of object and background.



(b) Crude reconstruction based on statistics from regions in Figure 7.4a.

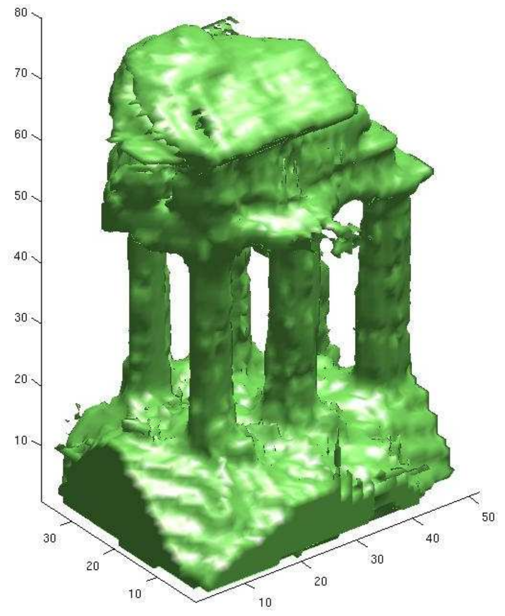


(c) Smoothed version of 7.4b

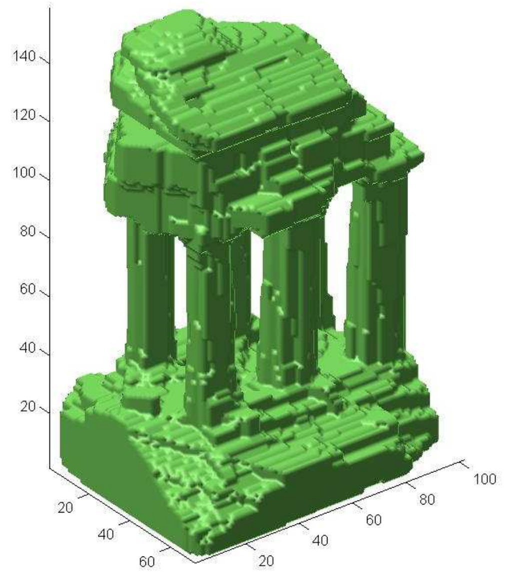


(d) Surface at convergence of fixed-statistics iteration

Figure 7.4: *Running example with noisy input images.*



(a) Gradient descent, $\nu = 2 \cdot 10^{-4}$.



(b) Graph cuts, $w = 5 \cdot 10{-}4$.

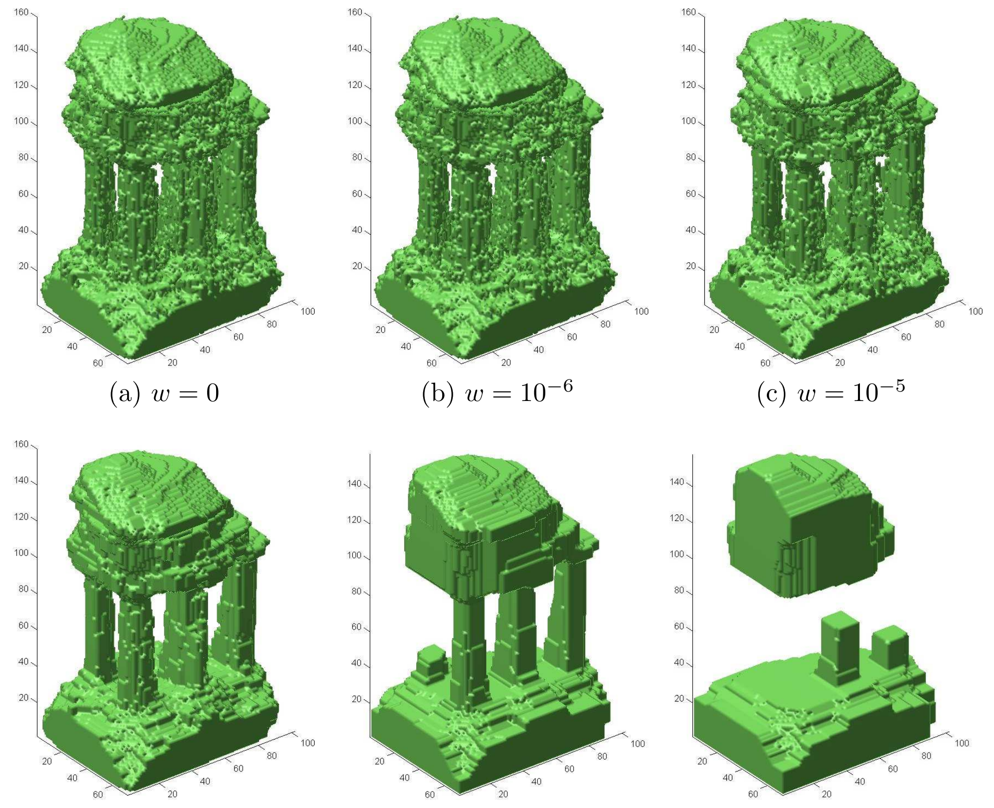
Figure 7.5: *Recovered surfaces from 4 images.*

7.2 Graph cut version

Here we test the graph cut version of our algorithm, using the temple dataset. The graph cut method is popular mainly for its running speed, but due to the surface measure challenges outlined in Section 6.2.1, we suspect that smoothed surfaces will have anisotropic artifacts. We will run a test with several different values of the smoothing parameter w. As w is increased we expect to see more anisotropic artifacts as well as slightly increased running time, since the algorithm is $O(mn\hat{k})$ where $\hat{k}$ is the cost of the minimum cut. Testing shows insignificant change after the second iteration, we shall use 5 iterations in our running examples to be safe.

7.2.1 GC test 1: 6-n surface measure, varying w.

We apply the graph cut solver to the temple sparse ring dataset, using a 6-neighbourhood surface norm and varying values of the smoothing term w. Reconstructed surfaces for various values of w are shown in Figure 7.6. We see that this algorithm tends to include more voxels in the object, making it thicker. Also note the severe anisotropic artifacts as the smoothing parameter is increased. Reconstructions (e) and (f) are probably unusable for most applications. On the bright side, note that the algorithm converges to a stable result even when $w = 0$. Given more noise or aberrations in the input images, this might not be the case.



(a) $w = 0$ (b) $w = 10^{-6}$ (c) $w = 10^{-5}$

(d) $w = 10^{-4}$ (e) $w = 10^{-3}$ (f) $w = 2 \cdot 10^{-3}$

Figure 7.6: *Reconstructed surfaces using graph cut solver with 6-neighbourhood surface measure and varying smoothing parameter w.*

7.2.2 GC test 2: Gaussian noise

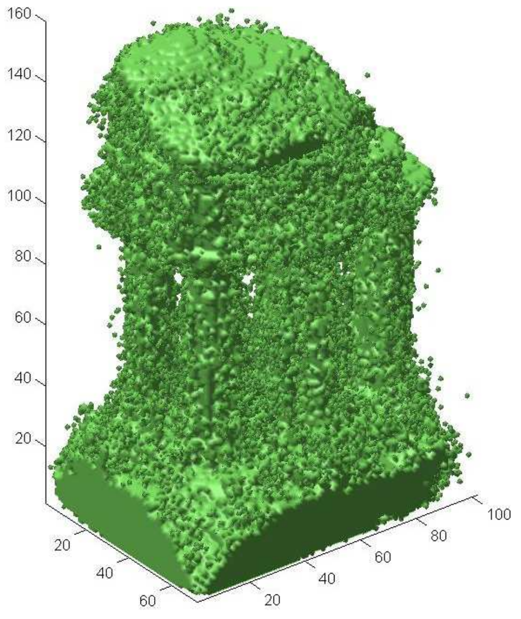
To check the performance of the graph cut algorithm in the presence of noise, we run an experiment on the Gaussian-corrupted input images from running example 2. We test different values of the smoothing parameter w. We hope that sufficient smoothing will be attained before anisotropic artifacts appear. The results are shown in Figure 7.7. As can be seen, the last two values yield decent results, though (c) still has some outliers and (d) is showing mild anisotropic artifacts. By contrast, the result in Figure 7.4d is pleasingly smooth but shows some residual noise. Recall that to even get that result we had to fix the statistics at reasonable values. The graph cut method is commendably stable even in the presence of noise.

7.2.3 GC test 3: 4 temple images

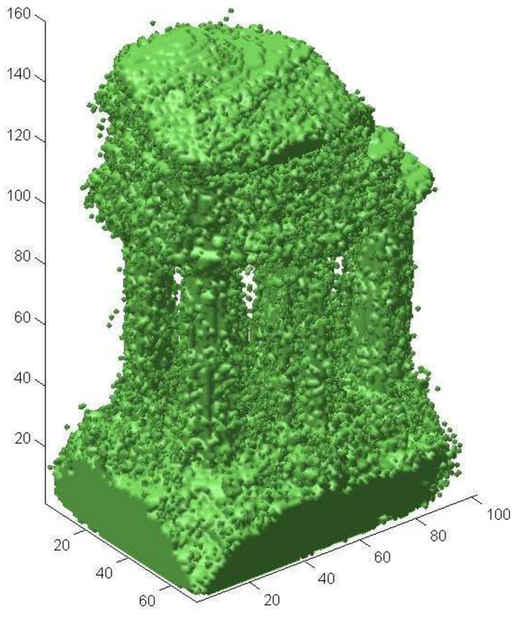
Having tested the gradient descent method on a sparse set of images, we want to compare its performance to the graph cut method. We use the same input images, and set the smoothing parameter w to $5 * 10^{-4}$. To try to minimize the anisotropy artifacts, we use a fairly high resolution of $160 \times 100 \times 70$. As can be seen in Figure 7.5b, the result is fairly similar. The graph cut method is a lot more blocky, but it is also more resistant to noise (fewer outliers). It is also a lot faster.

7.2.4 GC test 4: 26-n surface measure

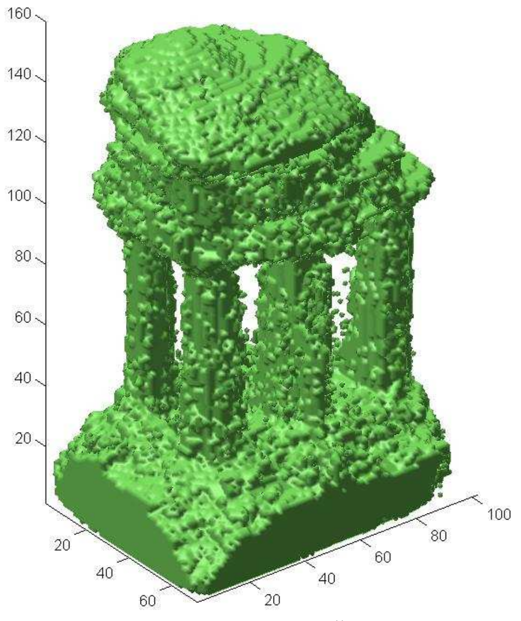
The blockiness of the graph cut method is a severe disadvantage. In an effort to remedy the situation, we extend the surface measure from a 6-neighbourhood to a full 26-neighbourhood. This will make the surface measure invariant under some fractional rotations, though not fully isotropic. Also we can expect the method to be a lot slower, since it is of order $mn\hat{k}$ and we increase the number of edges per node from 8 to 28. We define edge weights as per the discussion in Section 6.2.1, using all the simplifying assumptions we can. As a result, the value for w here is not directly comparable to the w-values of the 6-neighbourhood system. The results of this experiment is shown in Figure 7.8. Surprisingly, only the two largest tested values of w show significant increase in running time over the 6-n system. This may be because these are the only reconstructions to deviate seriously from the ground truth; this might make the solutions less “obvious” and prolong running times.



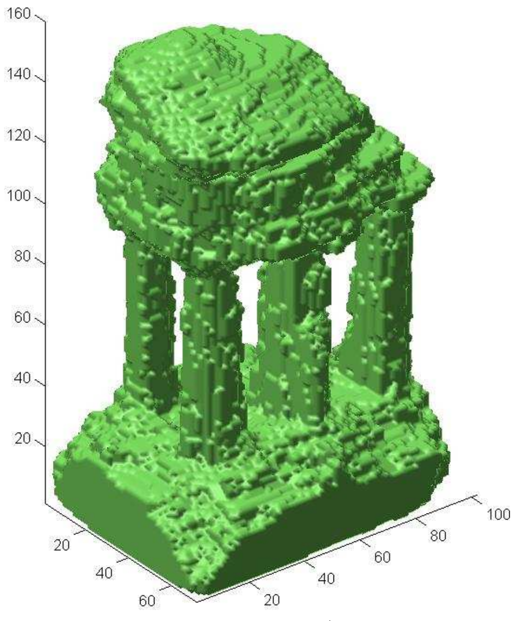
(a) $w = 0$.



(b) $w = 10^{-5}$.

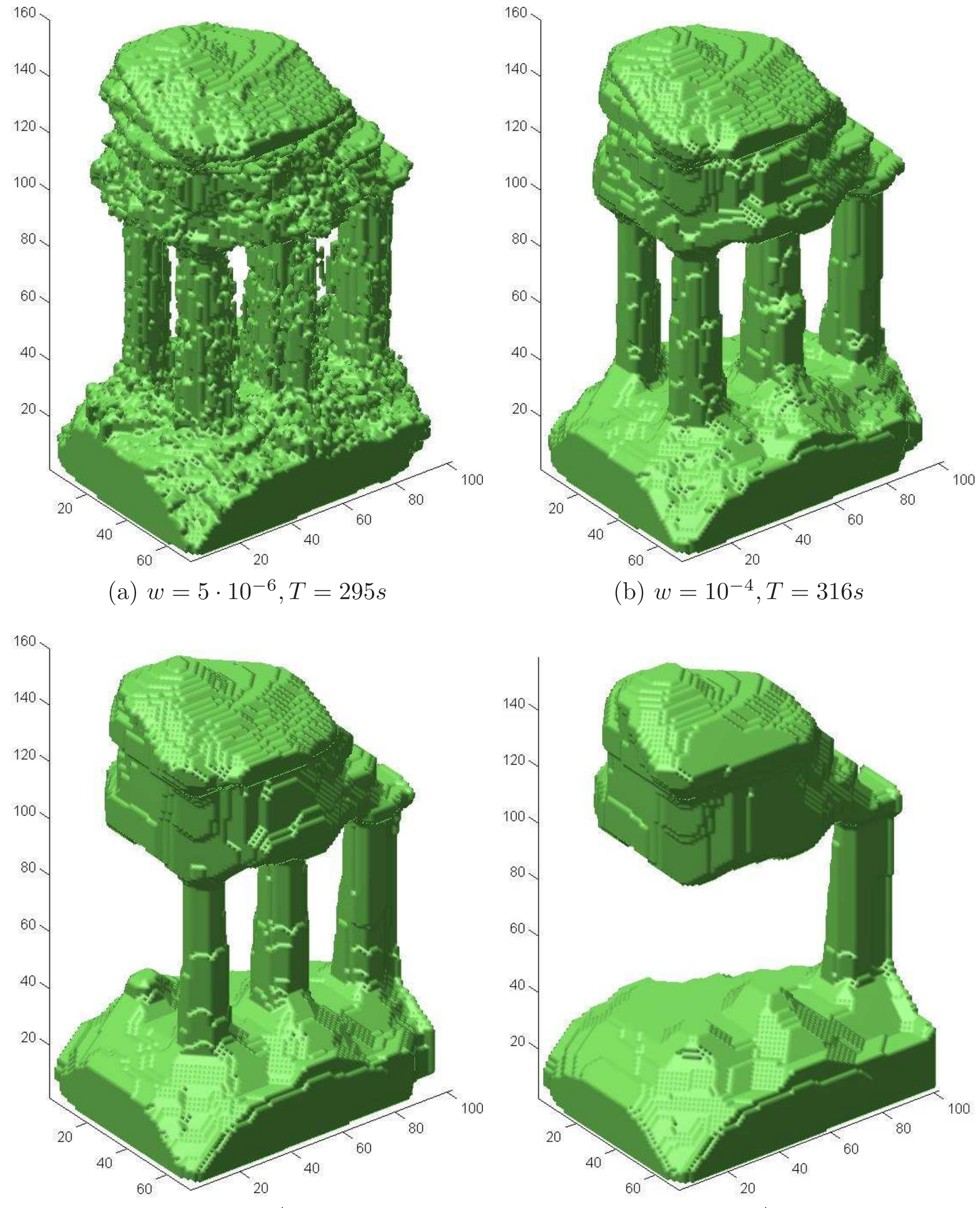


(c) $w = 5 \cdot 10^{-5}$.



(d) $w = 10^{-4}$

Figure 7.7: *Graph cut reconstruction of 1:1 SNR Gaussian-corrupted images.*


(a) $w = 5 \cdot 10^{-6}, T = 295s$

(b) $w = 10^{-4}, T = 316s$

(c) $w = 2.5 \cdot 10^{-4}, T = 404s$

(d) $w = 5 \cdot 10^{-4}, T = 712s$

Figure 7.8: *26-n reconstructed surfaces for different edge weights w, with running times T.*

Chapter 8

Discussion

In this thesis we have looked at two interesting frameworks for representing surfaces for the purpose of functional minimization. We have implemented the classic **Kolev** algorithm using both these frameworks; this allows us to compare them and elucidate their strengths and weaknesses. To wit:

- The PCLSM is a very general representation that allows very many different questions to be asked. Combined with the gradient descent method it shows somewhat slow convergence; this is exacerbated by factors that require small timesteps, such as a large smoothing term. On the plus side, it allows for a very natural extension to multiple volumes.

- The graph cut method is highly specialized; functionals to be minimized must be framed in a very precise manner to be graph representable. Many questions can not be posed as graphs at all. The algorithm is very fast, especially when combined with max flow solvers that exploit the peculiarities that image processing graphs often share. When a problem is represented as a graph, it has a very attractive guaranteed global minimum property. The lack of tuning or "fudge" parameters means less interaction from the user is needed, which saves work and increases the repeatability of experiments. Traditional isotropic surface area measures are not graph representible, and we must resign ourselves to anisotropic norms. We have tested a simple 6-neighbourhood and a more complex 26-neighbourhood norm; the latter yields better results but is slower and requires more thought in its implementation.

It is clear that both of these tools have a future, as they are suited to different tasks. Though the graph cut solver is attractive in problems where speed is essential, it is unable to handle global effects such as rotations of objects and visibility constraints.

8.1 Future work

Though the imagination knows no bounds, a M.Sc. project has a deadline and we have not been able to pursue all ideas sparked by our research. In this section we will outline some of the avenues we chose not to go down.

8.1.1 Visibility

Perhaps the greatest flaw in our algorithms is the simplifying disregard for visibility considerations. When classifying a voxel as object or background, this is not a big problem - interior voxels are seen as "object-colored" by all cameras, and are correctly identified as part of the object. However, in the statistics-updating step, failing to consider visibility means "thick" parts of the object are given undue weight. Ways of calculating visibility exist - rays traced through each pixel from the optical centre can find the closest surface voxel, yielding both a one-to-one correspondence between camera pixels and object voxels and a **depth map** for each image. A **visibility function** $\chi_i(\xi) : \Omega \rightarrow [0, 1]$ can be associated with each camera i, defined as

$$\chi_i(\xi) = \begin{cases} 1 & \text{if } \xi \text{ is visible from camera } i \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

This function would have to be updated after every iteration of the graph-cut algorithm or every few iterations of the level set algorithm. Handling visibility will also allow some other extension to our algorithms.

8.1.2 Multiple object reconstruction

The idea of a single, reasonably homogeneous object surrounded by air and a simple background is fine in laboratory conditions, but in real-life scenarios we may need more complex models. If we assume that our object consists of several object regions Ω_j, $j = \{1, 2, \ldots, m\}$ as well as the background region Ω_0, we can define different intensity distributions on each object and thus capture more complicated scenes. In order to do this, however, we need visibility tracking. If an object is occluded by another in one image, we can't use information from that image to say anything about the occluded object. We use (8.1) to restrict the images used in the labelling of a voxel. For instance, refer back to (6.1). Adding a visibility constraint, we get

$$u_{n_{abc;m}t} = P_m(\xi_{abc}) = \sqrt[\tilde{n}]{\prod_{\substack{i=1 \\ \chi_i(\xi)=1}}^{\tilde{n}} P(I_i(\pi_i(\xi_{abc}))|\xi_{abc} \in V_m)}$$

where $\tilde{n} = \sum_i \chi_i(\xi_{abc})$. What we have done here is to take the product and root over only the pixels where the object is visible. Note also that we speak of node $n_{abc;m}$ instead of node n_{abc} - in the multi-object case, each voxel needs to be represented by one node for each object. Instead of an edge from the source s to n_{abc} and one from n_{abc} to the sink t, we now have a succession of edges from s to $n_{abc;1}$, $n_{abc;1}$ to $n_{abc;2}$ and so on until the sink. Each edge has a constraint equal to the cost of assigning the voxel to the relevant object. Using the PCLSM method, multi-object handling is easier, as has been noted in Chapter 4.

8.1.3 Automatic camera calibration

One of the greatest problems we faced in testing our algorithm was finding datasets with calibrated camera coordinates. In many situations these data are incomplete or unavailable. We would like to extend our algorithm to automatically find camera positions. Given a known object, this would perhaps not be very difficult. However, since the reconstruction of the object depends on the calibration and the calibration depends on the object, we have to be very careful. A joint minimization may be our best bet. In addition to minimizing with respect to the surface S, we now also need to minimize with respect to camera parameters K_i, R_i and t_i for each camera i. We could make our functional explicitly dependent on these parameters and try to find an Euler-Lagrange condition that would minimize the energy. The PCLSM method may be able to solve this, but we remain pessimistic that this can be done using graph cut methods. This is because a small change in camera coordinates will affect all voxels at the same time, thus violating the graph representability demands.

8.1.4 Volumetric reconstruction

Since both our algorithm reconstruct complete volumes in order to find surfaces, it would be possible to expand them to include volume effects. This could range from simple things like transparency and different intensities at different depths to complicated things like sub-surface scattering and refraction inside the volumes. To deal with transparency we would need to classify the depth of each voxel relative to a surface, this would be analogous to a “partial” visibility function that is continuously graded between 0 and 1.

8.1.5 Moving volume tracking

If our dataset is a movie rather than a set of images taken at the same time, we need to track object movement and deformation. To make the problem tractable, we would need to assume that the object is relatively constant from one frame to the next - that its movement and deformation speed has a given maximum. The recovered surface from one frame can be used as the initialization for recovering the surface in the next, and deviations from this shape and position can be penalized. A simplifying assumption could be that the camera and background are immobile. Simple image subtraction could then recover moving objects and give our algorithm a boost.

Bibliography

[1] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. *International Conference on Computer Vision*, 2003.

[2] Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[3] Ward Cheney. *Analysis for Applied Mathematics.* Springer-Verlag, 2001.

[4] Carlos Hernández Esteban and Francis Schmitt. Silhouette and stereo fusion for 3d object modeling. Technical report, Ecole Nationale Supérieure des Télécommunications, 2004.

[5] Lawrence C. Evans. *Partial Differential Equations.* American Mathematical Society, 1998.

[6] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[7] Janne Heikkilä and Olli Silvén. A four-step camera calibration procedure with implicit image correction. Technical report, University of Oulu, Finland, 1997.

[8] Hailin Jin. *Variational Methods for Shape Reconstruction in Computer Vision.* PhD thesis, Washington University, 2003.

[9] Kalin Kolev, Thomas Brox, and Daniel Cremers. Robust Variational Segmentation of 3D Objects from Multiple Views. Technical report, CVPR Group, University of Bonn, 06.

[10] Vladimir Kolmogorov and Ramin Zabih. What Energy Functions Can Be Minimized via Graph Cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

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

[12] Stanley Osher and James A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 1988.

[13] F. Ranchin, A. Chambolle, and F. Dibos. Total variation minimization and graph cuts for moving objects segmentation. Technical report, Université Paris Dauphine; Ecole Polytechnique; Université Paris 13, 08.

[14] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. Technical report, University of Washington; Stanford University; Middlebury College; Microsoft Research, 2006.

[15] Xue-Cheng Tai and Tony Chan. A Survey on Multiple Level Set Methods with Applications for Identifying Piecewise Constant Functions. *International Journal Of Numerical Analysis and Modeling*, 04.

[16] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & Statistics for Engineers & Scientists.* Prentice Hall, 2002.

[17] Thomas Young. *A Course of Lectures on Natural Philosophy and the Mechanical Arts.* Taylor and Walton, 1807.

[18] William P. Ziemer. *Weakly Differentiable Functions.* Springer-Verlag, 1989.