New PDE-based Methods for Surface and Image Reconstruction

Master Thesis in Applied Mathematics

Egil Bae

Department of Mathematics
University of Bergen

September 3, 2007
Preface

I want to thank all people who have been supportive during my master study.

Especially I thank my supervisor Joachim Weickert, and the people in his research group at Saarland University in Germany, where I stayed as a guest for most of the time. It has been a great pleasure to stay in your group. The great scientific atmosphere has brought up many new ideas.

I further want to thank my main supervisor Xue-Cheng Tai for convincing me that image processing is such an interesting application of mathematics.

I also would like to thank Tom Kavli from sintef IKT. Although I did not pursue the topic we planned, the applications we discussed have been an inspiration for the material in Chapter 4 of this thesis.

Egil Bae

Saarbruecken, August 2007
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>iii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Mathematical theory</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Differential geometry</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Tangent space and normal vectors</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 The first fundamental form</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 Differentiable functions and vector fields</td>
<td>6</td>
</tr>
<tr>
<td>2.1.4 The second fundamental form and curvature</td>
<td>7</td>
</tr>
<tr>
<td>2.1.5 Surface evolution</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Surface representation</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Level set functions</td>
<td>9</td>
</tr>
<tr>
<td>2.2.2 Triangulated surfaces</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Image representation</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Variational methods</td>
<td>12</td>
</tr>
<tr>
<td>3 Interpolation and compression</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Images</td>
<td>18</td>
</tr>
<tr>
<td>3.1.1 Interpolants</td>
<td>19</td>
</tr>
<tr>
<td>3.1.2 Encoding</td>
<td>24</td>
</tr>
<tr>
<td>3.1.3 Some results</td>
<td>25</td>
</tr>
<tr>
<td>3.1.4 Image approximation</td>
<td>26</td>
</tr>
<tr>
<td>3.1.5 Experiments</td>
<td>32</td>
</tr>
<tr>
<td>3.2 Surfaces</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1 Finite element discretization</td>
<td>38</td>
</tr>
<tr>
<td>3.2.2 Encoding</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3 Decoding</td>
<td>43</td>
</tr>
<tr>
<td>3.2.4 Some results and their shortcomings</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5 Surface approximation</td>
<td>44</td>
</tr>
<tr>
<td>3.2.6 Experiments</td>
<td>46</td>
</tr>
<tr>
<td>3.2.7 Other interpolants to consider</td>
<td>49</td>
</tr>
<tr>
<td>3.3 Conclusions</td>
<td>51</td>
</tr>
</tbody>
</table>
CONTENTS

4 2D-3D pose estimation ........................................ 53
  4.1 Rigid Transformations ........................................ 55
  4.2 Shape deformations .......................................... 57
  4.3 2D-3D surface reconstruction ................................ 60
    4.3.1 Pinhole camera model .................................. 61
    4.3.2 A variational method ................................... 61
  4.4 Our pose estimation method .................................. 64
    4.4.1 Experiments ............................................. 66
  4.5 Conclusions .................................................. 70

5 Summary and outlook ........................................ 73
  5.1 Summary and contributions .................................. 73
  5.2 Future research ............................................... 75

A Hopscotch iteration ........................................ 77
Chapter 1

Introduction

The purpose of this thesis is to propose new methods, based on partial differential equations (PDEs), for two traditional surface reconstruction problems, that have never been solved by PDEs before. We will also present a new improvement of an existing image compression method. The new methods are tested on various experiments.

The first of these problems is compression/decompression surface geometry represented by triangulated surfaces. This is typically approached by decorrelating the geometry data, today the wavelet transform is the most powerful and most widely used method. Our approach is very different: we have developed a PDE based method for reconstructing surfaces from point clouds, which we have applied as a way to compress the surface geometry, by deleting parts of the geometry that can easily be reconstructed by the PDE.

It must be noted that this is only the first step towards a practical method. There are still many open questions, and room for many further developments. Compression is generally a dangerous field to jump into, because the standards are so high, and existing methods have been so fine-tuned and perfected over the years. We still have a certain confidence in our method, due to recent success in a similar approach to image compression. Although we have not made any comparisons with other methods, the results look promising.

The second problem we consider is 2D-3D pose estimation. 2D-3D pose estimation means to estimate a known 3D object’s state in the 3D world, from a set of 2D images taken from different viewpoints. This state includes translation, rotation and possibly size and deformation. Existing methods typically searches for correspondences between points and lines in the image and points and lines on the 3D object.

Our approach is variational, formulated in the continuous setting by minimizing an energy functional defined over surfaces. Instead of searching for local correspondences, we are trying to fit the 3D object to the image data in a more
global way, as a result the method will be very robust to noise and other image distortions. Complex deformations can be incorporated in the model in a very natural way. The main ingredient in this method is an established PDE approach to classical 2D-3D surface reconstruction. This is combined with ideas we have borrowed from recent advances in image segmentation with a priori shape information.

These methods will be presented Chapter 3 and Chapter 4, respectively. More details will be given in the introduction to these chapters.
Chapter 2
Mathematical theory

This chapter introduces some mathematical concepts that will be needed throughout the thesis. Of course we have to make a decision what should be familiar to the reader. Since surfaces are so central in this work, a section will devoted to summarize some of the fundamentals of differential geometry. We will then proceed by showing two different ways of representing surfaces, either by level set functions or triangulations. Both these representations will be used extensively in the following chapters. Finally, in section 2.4 we discuss optimization problems involving functionals, particularly functionals with surfaces as arguments.

We assume the reader has some knowledge of fundamental functional analysis, basic probability theory and, of course, partial differential equations. The PDE discretization methods we consider are finite difference methods and finite element methods. In order to not drown the text completely, finite difference schemes are not written out explicitly unless they are very unstandard.

2.1 Differential geometry

In order to use PDEs for surface processing, we need to introduce the language of differential geometry, which extends differential calculus to spaces more general than \( \mathbb{R}^n \). In general, the mathematical spaces we will study are called differentiable manifolds, of which smooth surfaces are a special kind. We will start by the general definition of a manifold. After this we will mostly stick to surfaces when the theory is explained. For a thorough analysis of the subject see Do Carmo [9], and Bloch [28].

**Definition 2.1** A differentiable manifold of dimension \( n \) is a set \( M \) and a family of injective maps \( x_\alpha : U_\alpha \subset \mathbb{R}^n \rightarrow M \), where \( U_\alpha \) is open such that:

- \( \cup_\alpha x_\alpha(U_\alpha) = M \)
- for any pair \( \alpha, \beta \) with \( x_\alpha(U_\alpha) \cap x_\beta(U_\beta) = W \), the sets \( x_\alpha^{-1}(W) \) and \( x_\beta^{-1}(W) \) are open in \( \mathbb{R}^n \) and the map \( x_\beta^{-1} \circ x_\alpha \) is differentiable
The family \( \{ (U_\alpha, x_\alpha) \} \) is maximal relative to the conditions above

The simplest examples of manifolds are the euclidian spaces \( \mathbb{R}^n \). Another example is surfaces in \( \mathbb{R}^3 \), also called 2-manifolds in \( \mathbb{R}^3 \). Surfaces are topologically defined as 2-dimensional sets lying in \( \mathbb{R}^3 \). We call \( \mathbb{R}^3 \) the ambient space of the surface. In general manifolds may or may not be imbedded in such a higher dimensional manifold.

For convenience we also specialize this definition for smooth surfaces. For that we need a special map called a coordinate patch, which is an injective smooth map of an open subset of \( \mathbb{R}^2 \) onto the surface. In order to avoid degenerate maps, we also require its partial derivatives, denoted \( x_1 \) and \( x_2 \) (columns of Jacobian matrix), to be linearly independent. When these requirements are fulfilled the map \( x \) is called a coordinate patch.

**Definition 2.2** For an open set \( U \subset \mathbb{R}^2 \), the map \( x : U \to \mathbb{R}^3 \) is called a coordinate patch if it is smooth, injective and \( x_1 \times x_2 \neq 0 \) at all point in \( U \).

It is generally impossible to cover a whole surface with one such injective map from \( \mathbb{R}^2 \). However, it is possible to find finitely many coordinate patches \( x_\alpha : U_\alpha \to M \) such that their union cover \( M, \bigcup_\alpha x_\alpha(U_\alpha) = M \). When this is true \( \{(U_\alpha, x_\alpha)\} \) is called a parametrization of \( M \). A smooth (or regular) surface is now defined as

**Definition 2.3** \( M \subset \mathbb{R}^3 \) is a smooth surface if, for each point \( p \in M \), there exists a neighborhood \( V \) of \( p \) in \( \mathbb{R}^3 \) and a surjective (onto) map \( x : U \subset \mathbb{R}^2 \to V \cap M \), such that:

- \( x \) is a coordinate patch;

- The differential \( (dx)_q : \mathbb{R}^2 \to \mathbb{R}^3 \) is injective for all \( q \in U \).

In contrast to the general definition of a manifold, smooth surfaces are defined assuming they can be embedded in an ambient space. Note also that the definition do not require the change of coordinate formula to be smooth. For surfaces this can instead be proved as a theorem.

Before moving on we need to consider parameterized curves in \( \mathbb{R}^3 \). As opposed to surfaces, curves are defined as maps rather than sets.

**Definition 2.4** A parameterized differentiable curve is a differentiable map \( C : I \to \mathbb{R}^3 \) of an open interval \( I = (a,b) \subset \mathbb{R} \). If \( C''(p) \neq 0 \), for all \( p \in I \), we say \( C \) is regular.
2.1. DIFFERENTIAL GEOMETRY

2.1.1 Tangent space and normal vectors

For smooth surfaces there are at each point associated a uniquely defined tangent space, and infinitely many tangent vectors.

**Definition 2.5** We say $w$ is tangent vector to $M$ at point $p$, if there exists a regular curve $C : (-\epsilon, \epsilon) \rightarrow S$, with $C(0) = p$, such that $w = C'(0)$. Define further $T_pM$ as the collection of all tangent vectors at $p$. $T_pM$ is called the tangent space of $M$ at $p$.

For regular surfaces we have the important result that the collection of tangent vectors form a plane

**Theorem 2.6** For a smooth surface $M \in \mathbb{R}^3$, at each point $p \in M$, $T_pM$ is a two-dimensional vector space. Let $x : U \rightarrow M$ be a coordinate patch such that $p \in x(U)$, then $T_pM$ is the subspace of $\mathbb{R}^3$ spanned by the vectors $\{x_1, x_2\}$ evaluated at $x^{-1}(p)$.

We also define the tangent bundle $TM$ as the disjoint union of all tangent spaces

$$TM = \{(p, v) \text{ s.t } p \in M, \ v \in T_pM\}.$$ 

We proceed by defining unit normal vectors of surfaces as the vector orthogonal to the tangent space.

**Definition 2.7** Let $x : U \rightarrow M$ be a coordinate patch such that $p \in x(U)$. The unit normal vector function $n_p : U \rightarrow S^2$ is defined locally as

$$n(q) = \frac{x_1(q) \times x_2(q)}{||x_1(q) \times x_2(q)||} \quad \forall q \in x(U). \quad (2.1)$$

$S^n$ stands for the unit sphere in $\mathbb{R}^n$, $S^n = \{x \in \mathbb{R}^n \text{ s.t } ||x|| = 1\}$.

2.1.2 The first fundamental form

We will study geometric structures carried by surfaces. The first fundamental form allows us to use inner products of vectors on the tangent space as a basic tool for measurements on the surface.

**Definition 2.8** Assume $p \in M$. The first fundamental form at point $p$ of $M$, is the bilinear form $I_p : T_pM \times T_pM \rightarrow \mathbb{R}$ defined by

$$I_p(v, w) = \langle v, w \rangle$$

$\langle .., .. \rangle$ is the standard inner product in $\mathbb{R}^3$. The first fundamental form of $M$ is the collection $I$, of all bilinear forms $I_p$ at all points $p \in M$. 
CHAPTER 2. MATHEMATICAL THEORY

Note that this definition is made without reference to a particular parametrization. In order to do computations we need to express $I$ locally in terms of a given coordinate patch. Let $x : U \to M$ be such a coordinate patch with $p \in x(U)$. The matrix $g$ of the bilinear form $I_p$ with respect to the basis $\{x_1, x_2\}$ of $T_pM$ is defined as:

$$g = \begin{pmatrix}
I_p(x_1, x_1) & I_p(x_1, x_2) \\
I_p(x_2, x_1) & I_p(x_2, x_2)
\end{pmatrix} \quad \text{(2.2)}$$

The coefficients $g_{ij}$ are called the metric coefficients of $M$ with respect to the coordinate patch $x$.

With the first fundamental form at hand, we can define integration on $M$.

**Definition 2.9** Let $x : U \subset \mathbb{R}^2 \to M$ be a coordinate patch of a smooth surface $M$, and let $S \subset U$ be a set. The area of $S$ is defined as

$$\text{Area}(S) = \iint_{x^{-1}(S)} \sqrt{\det(g)} \, d\xi_1 d\xi_2. \quad \text{(2.3)}$$

We are now able to define integrals over $M$.

**Definition 2.10** Let $f : M \to \mathbb{R}^3$ be a function on $M$, and $\{(U_\alpha, x_\alpha)\}$ a parametrization of $M$. Let $\{\eta_\alpha\}$ be a finite partition of unity on $M$ with support($\eta_\alpha$) $\subset U_\alpha$. The integral of $f$ over $M$ is defined as

$$\int_M f \, dx = \sum_\alpha \iint_{U_\alpha} \eta_\alpha f(x_\alpha) \sqrt{\det(g)} \, d\xi_1 d\xi_2. \quad \text{(2.4)}$$

**2.1.3 Differentiable functions and vector fields**

In this thesis we are interested in PDEs defined on surfaces. We thus need to define differentiable functions on surfaces, and show how they can be differentiated.

**Definition 2.11** For a regular surface $M$, let $f : M \to \mathbb{R}$ be a scalar valued function defined on $M$. We say $f$ is $r$ times differentiable at $p \in M$ if there exists a coordinate patch $x : U \subset \mathbb{R}^2 \to \mathbb{R}$ with $p \in x(U) \subset M$ such that the composition $f \circ x : U \subset \mathbb{R}^2 \to \mathbb{R}$ is $r$ differentiable at $x^{-1}(p)$. If $f$ is $r$ times differentiable at all points of $M$, we say $f$ is $r$ times differentiable in $M$.

Similarly to the euclidian function spaces, we define the vector spaces $C^r(M) =$\{ All $f : M \to \mathbb{R}$ such that $f$ is $r$ times differentiable \}. $C^\infty(M)$ is the set of all infinitely differentiable (smooth) functions defined on $M$

Differentiable vector fields are defined as follows.
Definition 2.12 For a smooth surface $M \subset \mathbb{R}^3$, a differentiable vector field $W$ on $M$ is a differentiable function: $W : M \to \mathbb{R}^3$. If $W(p) \in T_pM$ for all $p \in M$, $W$ is called a tangent vector field.

We define the function spaces $C^r(TM)$ as the set of all tangent vector fields with each component belonging to $C^r(M)$.

Let $f$ and $g$ be two functions in $C^r(M)$, and $v, w$ two vector field in $C^r(TM)$. On the function spaces $C^r(M)$ and $C^r(TM)$ we define the following inner products

$$ (f, g)_M = \int_M fg\,dx \quad (2.5) $$

$$ (v, w)_TM = \int_M I(v, w)\,dx \quad (2.6) $$

where integration over $M$ is carried out according to equation (2.4).

First order differentials are defined in the following way.

Definition 2.13 Let $f$ be a differentiable function on a surface $M$. At the point $p \in M$, the first order differential of $f$ is the linear map $df_p : T_pM \to \mathbb{R}$ defined as follows: for any vector $w \in T_pM$ find a curve $C : (-\epsilon, \epsilon) \to M$ such that $C(0) = p$ and $C'(0) = w$ and define

$$ df_p(w) = (f \circ C)'(0). \quad (2.7) $$

The first order differential at $p$ is a map from acting on vectors in the tangent space of $M$ at $p$. We also define the differential $df$ as a linear functional on the tangent bundle.

Definition 2.14 Let $f$ be a differentiable function on a surface $M$. The differential $df : TM \to V$ is defined as: for any $q = (w, p) \in TM$, $df(q) = df_p(w)$

where $V$ is some function space on $M$.

2.1.4 The second fundamental form and curvature

We will introduce the second fundamental form as a tool to tell us how much a surface differs from its tangent plane. This will be used to define the fundamental geometric entities such as principal curvatures, Gauss curvature and mean curvature. We start by defining the shape operator of $M$ which is called the Weingarten map when $M$ is a surface. Note that the normal vector $n : M \to S^2$ may be considered a vector field on $M$. In the following definitions $M$ is considered a regular, compact, orientable surface.

Definition 2.15 For $p \in M$, the differential $dn_p : T_pM \to T_pM$ is a self-adjoint linear map called the Weingarten map.
CHAPTER 2. MATHEMATICAL THEORY

Definition 2.16 For \( p \in M \) the quadratic form \( II_p : T_p M \to \mathbb{R} \) defined by

\[
II_p(w) = -\langle d\pi_p(w), w \rangle \quad \forall w \in T_p M
\] (2.8)

is called the second fundamental form of \( M \) at \( p \).

The second fundamental form says very loosely speaking how much a surface bends in different directions at a certain point. For a point \( p \in M \) in \( M \), and a vector \( w \) in \( T_p M \), the normal curvature along the direction \( w \) is given by \( II_p(w) \).

The principal curvatures \( k_1 \) and \( k_2 \) at the point \( p \) are simply the maximum and minimum normal curvatures

\[
k_1 = \min_{w \in T_p M} II_p(w)
\] (2.9)

\[
k_2 = \max_{w \in T_p M} II_p(w).
\] (2.10)

Their corresponding minimizer and maximizer are called the principal directions of curvature. With the principal curvatures at hand we define the mean curvature \( H \), and the Gauss curvature \( K \):

\[
H = \frac{1}{2}(k_1 + k_2) \quad K = k_1 \cdot k_2.
\] (2.11)

In practice these entities are computed in the following way

Theorem 2.17 Let \( L \) be the Weingarten map of a surface \( M \) at a point \( p \in M \). The eigenvalues of \( L \) are the principal curvatures of \( M \) at \( p \), and the eigenvectors of \( L \) the corresponding principal directions of curvature. Furthermore:

\[
H = \text{trace}(L) \quad \text{and} \quad K = \text{det}(L).
\] (2.12)

2.1.5 Surface evolution

By introducing a time parameter \( t \), we can evolve surfaces in time according to certain differential equations. Let \( M(t) \) be the surface at time \( t \) with coordinates \( x(t) \), and let \( V : M(t) \times [0, \infty) \to \mathbb{R}^3 \) be a vector field on \( M(t) \). In general, surface evolution equations (also called surface flows) will have the form

\[
\frac{\partial x}{\partial t} = V(x(t), t) \quad t > 0
\] (2.13)

\( V \) is often called the speed function, since it specifies the speed for each point of the surface. An interesting result is that surface evolutions of this form can be transformed to a flow where all forces act in the normal direction
2.2. SURFACE REPRESENTATION

Theorem 2.18 The family of surfaces following the evolution rule (2.13) can be converted to the solutions of the following evolution

\[ \frac{\partial x}{\partial t} = \langle V(x, t), n \rangle n, \]  

(2.14)

where \( n \) is the surface normal vector and \( \langle .., \rangle \) is the euclidian inner product.

Of special importance in this thesis is mean curvature flows. That is, flows where \( \langle V(x, t), n \rangle = H \)

\[ \frac{\partial x}{\partial t} = H(x, t) n. \]  

(2.15)

This surface flow is very often called mean curvature motion.

2.2 Surface representation

In chapter (2.1) we discussed surface evolutions by means of equations involving surface differentials. In order to implement this on a computer, we need some way to represent the surfaces and some way to discretize the differential equations. In this thesis PDE driven surface evolutions arise in two very different situations. As a consequence, we will need two very different discretization methods. Both of these have advantages and disadvantages compared to each other.

2.2.1 Level set functions

The level set method was invented by Osher and Sethian in 1987 [1]. They proposed to represent the surface as the zero-level of a function \( \phi \) of the three space variables. For a surface \( M \), let \( R \subset \mathbb{R}^3 \) be the region bounded by \( M \). \( \phi \) is a smooth function satisfying

\[
\begin{cases}
\phi(x) > 0, & \text{for } x \in R, \\
\phi(x) < 0, & \text{for } x \notin R, \\
\phi(x) = 0, & \text{for } x \in \partial R = M,
\end{cases}
\]

and is called the level set function. In order to get the best computational stability, \( \phi \) should not be too steep or too shallow. A normal choice of \( \phi \) is the signed distance function.

\[
\phi(x) = \begin{cases} 
\text{dist}(x, \partial R), & \text{for } x \in R, \\
-\text{dist}(x, \partial R), & \text{for } x \notin R, \\
0, & \text{for } x \in \partial R.
\end{cases}
\]

In the level set framework, evolution equations for surfaces are transformed into PDEs in the ambient space \( \mathbb{R}^3 \). Discretization is achieved by for instance finite difference schemes in \( \mathbb{R}^3 \). The advantages of the level set method are as follows:
• Since the grid points of the discretization are fixed, the surface can go through large deformations without degenerating.

• During evolution, the surface $M$ can undergo topological changes which do not need to be modeled explicitly, because topological changes of $M$ does not imply topological changes of $\phi$.

• Very easy implementation.

The main disadvantage is the higher computational cost of discretizing in the one dimension higher ambient space. For $n$ grid point in each direction of the 3D volume, the complexity of one time step of surface evolution is $O(n^3)$. This can be speeded up considerably by solving the PDE only in a narrow band of width $B$ near the zero-level of the level set function. This will reduce the complexity of one time step to $O(Bn^2)$. In our experiments, we have not implemented this. Note that even with the narrow band approach, the level set method is considered expensive.

Various geometric entities can be described in terms of level set functions. Let $x : U \subset \mathbb{R}^2 \to M$ be a coordinate patch. Let $(\xi_1, \xi_2)$ be the coordinate variables of $U$. By the definition of level set function we have

$$\phi(x(\xi_1, \xi_2)) = 0 \quad \forall (\xi_1, \xi_2) \in U.$$  \hspace{1cm} (2.16)

Differentiating with respect to $\xi_1$ and $\xi_2$ we get:

$$\phi_{\xi_1} = \langle \nabla \phi, x_{\xi_1} \rangle = 0,$$
$$\phi_{\xi_2} = \langle \nabla \phi, x_{\xi_2} \rangle = 0.$$  \hspace{1cm} (2.17)

Since $x$ was a coordinate patch, $x_{\xi_1}$ and $x_{\xi_2}$ are two linearly independent vectors in $T_pM$. Since $\nabla \phi$ is perpendicular to both of them, it must be parallel to $n_p$. Normalizing and choosing the right signs we obtain the inward and outward normal vectors expressed in $\phi$:

$$n_{in} = \frac{\nabla \phi}{|\nabla \phi|},$$
$$n_{out} = -\frac{\nabla \phi}{|\nabla \phi|}.$$  \hspace{1cm} (2.18)

Other geometric entities such as mean curvature can also be obtained by a bit lengthier computation:

$$H = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}.$$  \hspace{1cm} (2.19)

In practise we limit the domain of definition of $\phi$ to a bounded box $\Omega$ in $\mathbb{R}^3$.
2.2. SURFACE REPRESENTATION

Figure 2.1: Low resolution triangulated surface consisting of 501 vertexes and 998 faces.

2.2.2 Triangulated surfaces

Triangulated surfaces are special kinds of surfaces that is very easy to represent on a computer. The surface geometry is piecewise linear consisting of a finite number of triangles.

Definition 2.19 Let \( T = \{T_1, ..., T_m\} \) be a set of triangles in \( \mathbb{R}^3 \). \( T \) is said to be a manifold triangulation if:

- \( T_i \cap T_j \) is either empty, a common vertex (triangle corner) or a common edge for \( i \neq j \).
- \( M_T = \bigcup_{i=1}^{m} T_i \) is an orientable 2-manifold (surface).

We are interested in the case that \( M_T \) is a closed orientable surface.

Triangulated surfaces are represented on a computer by storing the vertex positions with high precision floating points, and the triangles as a graph over the collection of vertexes. A very common representation is by polygonal meshes. A polygonal mesh consists of three kinds of elements: mesh geometry, edges and faces. The data used to describe these mesh elements consists of the mesh connectivity and the mesh geometry.

- The mesh geometry is defined as the collection of all vertexes positions.
• The mesh connectivity is defined as the incidence relations between the mesh elements. That is, for each vertex its incident edges and faces, for each face the vertexes and edges bounding it and for each edge the end vertices and incident faces.

A triangulated surface can be evolved over time by evolving its vertex positions while keeping the mesh connectivity fixed. The danger is that the mesh can degenerate and no longer satisfy the conditions of definition (2.19). If the evolution is over short distances this is usually not a problem.

2.3 Image representation

In the continuous case, a gray valued image is simply a function \( f : \Omega \rightarrow \mathbb{R} \) defined over a (usually rectangular) domain \( \Omega \subset \mathbb{R}^2 \). The function value \( f(x,y) \) specifies the brightness value at point \((x,y)\). The brightness value may be any real number between 0 (the darkest) and 1 (the lightest). A digital image is the discrete counterpart of this representation, where \( \Omega \) is replaced by a square grid \( \Omega^h \), and \( f \) is replaced by a quantized version \( f^h : \Omega^h \rightarrow \mathbb{Z} \) over \( \Omega^h \). Typically \( f^h \) range between 0 and 256, but when discretizing PDEs over \( \Omega^h \), the function \( f^h \) could take any floating point value. Discrete points in \( \Omega^h \) are usually called pixels.

2.4 Variational methods

Throughout this thesis, surface reconstruction models will be formulated as minimization problems of certain energy functionals. These minimization problems will be transformed into an equivalent PDE formulation, and then be solved using ordinary discretization methods. We will present some tools which is needed in order to solve these minimization problems.

Let's start by studying an ordinary function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \). We want to minimize

\[
\min_{x \in \mathbb{R}^n} f(x). \tag{2.20}
\]

A local minimizer of \( f \) is defined as a point \( x^* \) at which there exists a neighborhood \( N_\varepsilon(x^*) \), such that for any \( y \in N_\varepsilon(x^*) \), \( f(y) \geq f(x^*) \).

A necessary condition for \( x^* \) being a local minimizer for \( f \) is the following

\[
\nabla f(x^*) = 0.
\]

Note that this is not a sufficient condition. Apart from being a local minimizer, \( x^* \) could be a local maximizer, or a saddle point when this condition holds. We have so far only discussed local minimizers of \( f \). A global minimizer of \( f \), is a
point \( x^* \) such that for any \( y \in \mathbb{R}^n \), \( f(y) \geq f(x^*) \). The optimization methods we will discuss searches for local minimizers only. To tell whether minimizer we have found is in fact a global minimizer, is a difficult task.

The gradient decent method is such a method that searches for local minimizers. It is defined by introducing the artificial time parameter \( t \), and looking for the steady state solution of the system of ODEs

\[
\begin{align*}
\frac{dx}{dt} &= -\nabla f \\
x |_{t=0} &= x_0
\end{align*}
\]

Note that this method is only able to find local minimizers. One way to make sure the local minimizer we find is a global minimizer, is to initialize the vector \( x_0 \) close to the global minimizer.

These ideas for minimizing functions can be generalized to functionals over any vector space. Let \( E : V \to \mathbb{R} \) be a functional on a Banach space \( V \). We want to solve the minimization problem

\[
u^* = \arg \min_{u \in V} E(u).
\]

For some \( v \in V \) with \( ||v|| = 1 \), we define the weak derivative of \( E \) in the direction \( v \) as (see [2])

\[
dE(u; v) = \langle \frac{\partial E}{\partial u}, v \rangle = \lim_{\epsilon \to 0} \frac{E(u + \epsilon v) - E(u)}{\epsilon}.
\]

Vectors \( u^* \in V' \) that satisfies \( \frac{\partial E}{\partial u} |_{u=u^*} \cdot v = 0 \) for all \( v \in V \), are called a critical points of \( V \). The condition

\[
\langle \frac{\partial E}{\partial u}, v \rangle = 0 \quad \forall v \in V.
\]

is a necessary condition for \( u^* \) being local minimizer of \( E \), and is called the Euler-Lagrange condition.

Let us now concentrate on the case that \( V \subset L^2(\Omega) \), where \( \Omega \) is some bounded set in \( \mathbb{R}^n \). Using the natural inner product on \( L^2(\Omega) \), the weak derivative in the direction \( v \) is simply

\[
\lim_{\epsilon \to 0} \frac{E(u + \epsilon v) - E(u)}{\epsilon} = \int_{\Omega} \frac{\partial E}{\partial u} v \, dx.
\]

\( \frac{\partial E}{\partial u} \) is said to be the Gateaux derivative of \( E \) if this integral converges for all \( v \in V \) with \( ||v|| = 1 \).

This method for finding minimizers of energy functionals will be illustrated by an example energy that will be useful for us throughout the whole thesis.
Example 2.20 We wish to minimize energies of the form

\[ \tilde{E}(S) = \tilde{f}(S) + \int_S dA \]  

(2.26)

where \( S \) is a regular surface in \( \mathbb{R}^3 \). \( \tilde{f} \) is an interior energy that is trying to fit the surface \( S \) to some known data. The last term is called a regularizer, and penalizes surfaces \( S \) of large surface area. Note that regular surfaces are not members of a vector space. In order to use the theory we need to transform (2.26) so that the argument belongs to a vector space. This is achieved by representing \( S \) by a level set function. We obtain

\[ E(\phi) = f(\phi) + \int_\Omega |\nabla H(\phi)| dx = f(\phi) + \int_\Omega \delta(\phi)|\nabla \phi| dx \]  

(2.27)

where \( E \) and \( f \) the equivalents of \( \tilde{E} \) and \( \tilde{f} \) with level set arguments. The function \( H \) is the heaviside function defined as

\[ H(x) = \begin{cases} 
1, & \text{if } x \geq 0, \\
0, & \text{if } x < 0.
\end{cases} \]  

(2.28)

In this case the vector space \( V \) will be \( V = L^2_\Omega(\Omega) \cap C^\infty_\Omega(\Omega) \). The subscript \( n \) denotes the Neumann boundary condition

\[ \frac{\partial \phi}{\partial n} = 0 \quad \partial \Omega. \]  

(2.29)

We want to find the Gateaux derivative of \( E \). The Gateaux derivate is the function \( \frac{\partial E}{\partial \phi} \) satisfying

\[ \int_\Omega \frac{\partial E}{\partial u} v dx = \lim_{\epsilon \to 0} \frac{E(u + \epsilon v) - E(u)}{\epsilon} = \int_\Omega \frac{\partial f}{\partial u} v dx \]

\[ + \int_\Omega \delta'(\phi)|\nabla \phi| dx + \int_\Omega \delta(\phi) \lim_{\epsilon \to 0} \left( \frac{|\nabla(\phi + \epsilon v)| - |\nabla \phi|}{\epsilon} \right) dx \]

for all \( v \in V \) with norm 1. Let us have a look at the last of these terms. This can be written

\[ \int_\Omega \delta(\phi) \lim_{\epsilon \to 0} \left( \frac{|\nabla(\phi + \epsilon v)| - |\nabla \phi|}{\epsilon} \cdot |\nabla(\phi + \epsilon v)| + |\nabla \phi| \right) dx \]

which is, by evaluating the limit

\[ \int_\Omega \delta(\phi) \frac{\nabla \phi}{|\nabla \phi|} \nabla v dx. \]
2.4. VARIATIONAL METHODS

We can now use Green’s identity. For a function $g$ and a vector field $v$ we have

$$
\int_{\Omega} w \cdot \nabla g \, dx = \int_{\partial \Omega} g w \cdot n \, dS - \int_{\Omega} g \nabla \cdot w \, dx.
$$

By putting $w = \frac{\delta(\phi)}{|\nabla \phi|} \nabla \phi$, and $g = v$ we get

$$
\int_{\Omega} \delta(\phi) \frac{\nabla \phi}{|\nabla \phi|} \nabla v \, dx = \int_{\partial \Omega} \delta(\phi) \frac{\partial \phi}{|\nabla \phi|} v \, dS - \int_{\Omega} \nabla \cdot \left( \delta(\phi) \frac{\nabla \phi}{|\nabla \phi|} \right) v \, dx.
$$

Because of the boundary condition (2.29), the first term vanishes. We now put this into the expression for the weak derivative of $E$

$$
\int_{\Omega} \frac{\partial E}{\partial \phi} v \, dx = \int_{\Omega} \frac{\partial f}{\partial \phi} v \, dx + \int_{\Omega} \left[ \delta'(\phi) |\nabla \phi| - \nabla \cdot \left( \delta(\phi) \frac{\nabla \phi}{|\nabla \phi|} \right) \right] v \, dx
$$

$$
= \int_{\Omega} \frac{\partial f}{\partial \phi} v \, dx + \int_{\Omega} \left[ \delta'(\phi) |\nabla \phi| - \delta'(\phi) \frac{|\nabla \phi|^2}{|\nabla \phi|} - \delta(\phi) \nabla \cdot \left( \frac{\nabla \phi}{|\nabla \phi|} \right) \right] v \, dx
$$

$$
= \int_{\Omega} \frac{\partial f}{\partial \phi} v \, dx - \int_{\Omega} \delta(\phi) \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right) v \, dx.
$$

We are interested in minimizers for $E$. Candidates for these minimizers are vectors satisfying the Euler lagrange condition (2.23). Putting into the left hand side we must have

$$
\int_{\Omega} \frac{\partial E}{\partial \phi} v \, dx = \int_{\Omega} \left[ \frac{\partial f}{\partial \phi} - \delta(\phi) \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right) \right] v \, dx = 0.
$$

Since this holds for all $v \in L^2_n(\Omega) \cap C^\infty_n(\Omega)$, we must have

$$
\frac{\partial E}{\partial \phi} = \frac{\partial f}{\partial \phi} - \delta(\phi) \nabla \cdot \left( \frac{\nabla \phi}{|\nabla \phi|} \right) = 0
$$

Equation (2.30) is called the Euler-Lagrange equation for of the minimization problem (2.22). It is a partial differential equation with solutions being local minimizers and maximizers of $E$. In order to find the local minimizers in practice, we introduce an artificial time parameter $t$ and look for the steady state of the evolution

$$
\frac{\partial \phi}{\partial t} = - \frac{\partial E}{\partial \phi} = - \frac{\partial f}{\partial \phi} + \delta(\phi) \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right),
$$

$$
\phi|_{t=0} = \phi_0.
$$

This equation is called the gradient decent equation for the minimization problem (2.22). Similar to the euclidian case, we may have to initialize $\phi_0$ well in order to
find the global minimizer. In order to solve such a PDE in practice, $\delta(x)$ needs to be replaced by a smooth approximation. We have used
\[
\delta_\varepsilon(x) = \frac{1}{\pi} \left( \frac{\varepsilon}{\varepsilon^2 + x^2} \right)
\] (2.33)
for some small $\varepsilon$, throughout the thesis.

An interesting situation to consider, is the case when the functional $f$ is identically zero, which would give us the flow:
\[
\frac{\partial \phi}{\partial t} = \delta(\phi) \left( \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right).
\] (2.34)
From Section 2.2.1 we recognize $\nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}$ as the mean curvature. In fact, equation 2.34 is the level set representation of mean curvature motion, see equation (3.58). This further implies that mean curvature motion is the gradient descent equation for the minimization problem
\[
\min_M \int_M dA.
\]
Chapter 3

Interpolation and compression

In this chapter we will present new methods for surface compression, surface interpolation, surface approximation and give some contributions to image compression. Let us start by giving some definitions to make these new terms absolutely clear.

Definition 3.1 Interpolation is the term used for methods that construct new data points from a discrete set of data points.

Usually this means to construct a continuous function from a discrete set of function values.

One example would be to use polynomials of varying degree as interpolant. In figure 3.1 we see a set of scattered data points in 1d, and the result of interpolating with linear polynomials.

Approximation (curve fitting in 1D) is similar to interpolation, with the exception that we now require the interpolant to closely approach the data points, but not necessarily pass through them. The benefit of this depends on what method one is using, but it often results in a smoother reconstruction. The drawback is usually worse accuracy.

Definition 3.2 In general data compression is the process of encoding data using

Figure 3.1: Left: Data points, right: interpolation using linear polynomials.
as few information bearing units (usually bits) as possible, such that the inverse process, called decoding, will return all original information.

This is opposed to lossy compression where we, as the name implies, loose information while going through the encoding/decoding process. The art of lossy compression is thus to encode the data with as few information bearing units as possible, while making the data loss during decoding as “insignificant” as possible. The word “insignificant” is very significant, it may have different meaning depending on what kind of data we are processing. In image compression, the decoded image should give a visual impression as close as possible to the original image. For surfaces it is a bit more complicated. Our method is specialized for triangulated surfaces. This is a very common surface representation in computer graphics. In this kind of application the important thing is again visual impression when the surface is being hit by different light sources. On the other hand, triangulated surfaces are also quite common in engineering. Here the reconstruction quality may have a different meaning. We will do not wish to limit ourself to one of these applications, but try to be as general as possible. Still surface reconstruction quality will only be measured visually throughout this thesis. We have tried to considered various norms, but since we are not comparing directly with other methods, these numbers don’t really tell us anything. When comparing with other methods in the future, they should of course be considered.

The reader may now perhaps see a possible relationship between interpolation and lossy compression. By storing only a small set of the data points, the original data can be reconstructed, up to some error, by interpolating between the known points. This idea has recently been shown to be very promising in image compression (see [14]), and is the motivation for our surface compression method. Two major questions are what interpolant to use, and how to decide what parts of the surface data we wish to store. This will be addressed in Chapter 3.2.

Before moving on to surfaces we will give a summary of PDE-based image compression, which our method is inspired from. In fact, while working with surfaces we have made a very interesting discovery that turns interpolation into approximation and improves the results significantly. This geometric idea can also be transferred back to images. It will be introduces in Chapter 3.1.4.

3.1 Images

PDE-based image compression is a recent method that is still under development, but so far with very promising results. The idea is as follows: Encode the image by sparsifying the pixels in a very clever way. Decoding is achieved by interpolating between the sparse set of pixels. The interpolant is a differential operator acting on the image domain, with the sparse set of gray values as Dirichlet boundary conditions. We will later come back to what we mean by a clever sparsification. What can be said for now, is that it almost always implies a scattered set of single
3.1. IMAGES

pixels. We will formulate the problem in a continuous setting, since this allows us to better analyze the equations. In practice, of course discretized versions of the differential operators will be used. Let \( \Omega \) be the image domain, and \( \hat{\Omega} \subset \Omega \) the domain we want to interpolate. Let \( I \) be the true image. We want to find a function \( u \) approximating \( I \) in \( \hat{\Omega} \) such that

\[
Lu = 0 \quad \text{in} \quad \hat{\Omega}
\]

\[
u = I \quad \text{on} \quad \partial \hat{\Omega}
\]

where \( L \) is some elliptic operator. We will also use Neumann boundary conditions on the surrounding image boundary in all our equations

\[
\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad \partial \Omega.
\]

Note that if the boundary domain consists of a scattered set of points of measure zero, these equations are not well formulated. When we talk about interpolation points \( x_i \), we shall thus mean small balls \( B_\epsilon(x_i) \) of radius \( \epsilon \) around \( x_i \). When the equations are discretized, the balls are replaced by single pixels \( \hat{x}_i \), which always are squares of nonzero areas.

What we do, can also be treated as an image inpainting problem. Inpainting is the problem of filling "holes" of missing information in a image. Often this is approached by solving a suitable PDE in the "hole", with data from the surrounding border as Dirichlet/Neumann boundary conditions. However, the situation is more difficult for us, since the boundary pixels are usually scattered throughout the whole image domain. Ironically we are not inpainting in a "hole", but in a domain filled with holes.

3.1.1 Interpolants

It's now time to specify reasonable interpolants. When interpolating between function values, we are interested in a function that smoothly connects the function values. Natural options for our differential operator are thus smoothing operators. When used for compression there are also another very important factor we need to consider, performance. The discrete operator must be as easily computable as possible, or else the method will be very impractical. The simplest example is linear diffusion. The diffusion equation will be extremely important for us throughout the whole thesis, so we will take some time to summarize its physical interpretation and some useful properties.

Linear diffusion

The diffusion equation describes the spatial distribution of some physical quantity such as heat, chemical concentration etc. over time. We denote this quantity with
CHAPTER 3. INTERPOLATION AND COMPRESSION

Let $V \subset \Omega$ be an arbitrary subset of some global region $\Omega$. The rate of change of $u$ within $V$, is caused by the negative of the total flux (denoted $F$) through $\partial V$.

\[
\frac{d}{dt} \int_V u \, dx = - \int_{\partial V} F \cdot n \, dS = \int_V \nabla \cdot F \, dx.
\]

Since $V$ was arbitrary, we get

\[
\frac{\partial u}{\partial t} = -\nabla \cdot F.
\]

Very often $F$ is proportional to the negative of $\nabla u$

\[
F = -c \nabla u.
\]

This implies

\[
\frac{\partial u}{\partial t} = \nabla \cdot (c \nabla u) = c \Delta u \tag{3.4}
\]

which is known as the diffusion equation. $c$ is called the diffusion coefficient. Our problem is thus

\[
\frac{\partial u}{\partial t} = \Delta u \quad \text{in} \quad \tilde{\Omega} \times [0, \infty) \tag{3.5}
\]

\[
u = I \quad \text{on} \quad \partial \tilde{\Omega} \times [0, \infty) \tag{3.6}
\]

\[
u|_{t=0} = I_0 \quad \text{in} \quad \tilde{\Omega} \times \{t = 0\} \tag{3.7}
\]

where $I_0$ is some initialization image in the unknown domain. $u^* = \lim_{t \to \infty} u$ will be reconstructed image. We can immediately deduce some useful properties of this reconstruction by reviewing some of the mathematical theory for the diffusion equation. The proofs of these results can, for instance, be found in Evans [15].

**Theorem 3.3** There exists at most one solution eq. (3.5) and (3.7), with the mixed Dirichlet/Neumann boundary conditions (3.6) and (3.3).

In addition to uniqueness of solution when going forward in time, we have a similar result when going backward in time. Given the solution of the diffusion equation at time $T$, the solution at any time $t<T$ can be retrieved.

**Theorem 3.4** Let $I_1$ and $I_2$ be arbitrary initializations over $\tilde{\Omega}$. Let $u$ be the solution of (3.5)-(3.7) with $I_0 = I_1$, and $v$ be the solution of (3.5)-(3.7) with $I_0 = I_2$, then

\[
\lim_{t \to \infty} u = \lim_{t \to \infty} v.
\]

This means that the problem is convex, its solution does not depend on the initialization. In the following results we assume the boundary conditions (3.5)-(3.7) hold.
Theorem 3.5 let $u$ be the solution of eq. (3.5)-(3.7). For any time $T$
\[
\max_{\Omega} u\big|_{t=T} = \max_{\partial\Omega} u\big|_{t=T}
\]
, and
\[
\min_{\Omega} u\big|_{t=T} = \min_{\partial\Omega} u\big|_{t=T}.
\]
Furthermore these conditions hold when $T \to \infty$.

The max-min principle says that the maximum and minimum values of $u$ are are always attained on the boundary. This is important for us, since it means the solution will locally be bounded by its nearest interpolation values. No over and under shoots will be created.

The desired smoothness can be proved everywhere except at the boundary.

Theorem 3.6 For any time $T$, $u\big|_{t=T} \in C^\infty(\tilde{\Omega})$, furthermore
\[
 u^* = \lim_{t \to \infty} u \in C^\infty(\tilde{\Omega}).
\]

At the boundary the solution is usually not even differentiable, especially when the boundary only consists of a single point. We will show this phenomena in a 1D example and a 2D example.

Example 3.7 We have used linear diffusion in 1d to interpolate the data points in figure (3.1). Figure (3.2) shows the solution of (3.5)-(3.7). As we can see, interpolation with linear diffusion in 1d, is exactly the same as interpolation with linear polynomials. Note also the discontinuity of derivatives at each boundary point.

In 2d the situation is very different. While in 1d, any PDE-interpolant will only depend on the interpolation points to the right and to the left, in 2d the interpolant will in theory be influenced by all interpolation points in the set. Additionally, a logarithmic singularity will arise at each interpolation point. In order to explain this phenomena, we want to investigate the time independent

![Figure 3.2: Left: Data points, right: interpolation with linear diffusion.](image)
formulation (3.1) - (3.2), with $L$ being the laplace operator. We have until now assumed the interpolation points $x_i$ are not actual points, but rather small balls $B_\epsilon(x_i)$ of radius $\epsilon$. When the radius $\epsilon$ goes to zero, the formulation (4.21) - (3.2) breaks down, because the differential equation does not makes sense for boundaries consisting of infinitely small points. A good illustration is to compare $u$ with the gravitation field that is created when stars are present in the universe. If the radius of these stars goes to zero, black holes arises and the gravitation goes to infinity close to the stars. The limiting case of the interpolation ball radius going to zero, can be regarded as such ”black holes” arising in the interpolation points.

For now we will speak of functions in the distributional sense. In order to not complicate things too much we will not be 100% mathematically correct. Instead of assigning values to the interpolation points we will assign them weights. This can be done by rewriting (3.1) - (3.2) in the following way

$$-\Delta u = f \quad \text{in} \quad \Omega \quad (3.8)$$

where

$$f(x) = \sum_i I(x_i) \delta(x_i). \quad (3.9)$$

We have in words moved the boundary conditions directly to the differential equation by introducing the force term $f$. Consequently, we are now solving the PDE in the whole image domain. For simplification we assume $\Omega = \mathbb{R}^2$.

Let us now have a look at the fundamental solution of Laplace equation $\mathbb{R}^2$.

**Definition 3.8** The function

$$\Phi(x) = -\frac{1}{2\pi} \log(|x|)$$

defined for $x \in \mathbb{R}^2$, is called the fundamental solution of Laplace’s equation.

$$\Delta u = 0 \quad \text{in} \quad \mathbb{R}^2. \quad (3.10)$$

The fundamental solution of Laplace’s equation can be used to construct the solution of Poisson equation in $\mathbb{R}^2$.

**Theorem 3.9** Define $u$ as

$$u(x) = \int_{\mathbb{R}^2} \Phi(x - y)f(y) \, dy$$

then

$$-\Delta u = f \quad \text{in} \quad \mathbb{R}^2. \quad (3.10)$$
In our case that means
\[
u(x) = \sum_i I(x_i) \int_{\mathbb{R}^2} \Phi(x-y) \delta(x_i) dy = \sum_i I(x_i) \Phi(x-x_i) = \frac{1}{2\pi} \sum_i I(x_i) \log(|x-x_i|).
\]

This equation is simply a superposition of the fundamental solution of Laplace equation with weights \(I(x_i)\) at the "interpolation points" \(x_i\). As we can see, a logarithmic singularity arises at each "interpolation point", where the solution \(u\) goes to infinity.

When discretizing the equations this scenario does not occur, because discrete points will always be regarded as having a nonzero area. So why is this continuous situation important to us? Because the discrete equations are an approximation of the continuous equations. In the discrete case we are always able to fix the interpolation values, they will never go to infinity, but as the discretization becomes finer, the slope of the solution near these interpolation points gets steeper and steeper. This is in fact a discrete approximation of the singularity. The phenomena will be shown in experiments in a moment.

Despite of its simplicity it turns out linear diffusion is pretty good for the purpose of compression. The drawback is artifacts like these singularities. Our own method, to be presented in section 3.1.4 is also based on linear diffusion. First we mention some other operators that we will compare our results with.

**Nonlinear diffusion**

Linear diffusion is a process inspired by nature. However, image processing is not a study of nature, but belongs to a universe where we are the gods and can invent our own physical laws. While diffusion has the effect of smoothing, which is desirable in for instance noise removal, it also destroys sharp edges which we normally want to keep sharp. This has motivated researchers to develop nonlinear diffusion "laws". These are a hybrids between forward and backward diffusion, and work by modifying the diffusion coefficient at edges. Perona and Malik \cite{16} proposed the following model
\[
\frac{\partial u}{\partial t} = \nabla \cdot (G(||\nabla u||)\nabla u). \tag{3.11}
\]

\(G\) is a function that suppresses diffusion at sharp edges, indicated by large gradients. For instance \(G\) may be chosen as
\[
G(s) = (1 + \frac{s^2}{\lambda^2})^{-1} \tag{3.12}
\]

where \(u_\epsilon = K_\epsilon * u\) is a prefiltered image using the convolution kernel \(K_\epsilon\). Without this prefiltering the problem is ill-posed.

Since \(G\) depends only on the magnitude of the gradient of \(u\), the diffusion is suppressed equally in all directions. We would ideally like to allow smoothing
along edges, while suppressing it across edges. For this we first generalize by replacing the diffusion coefficient with a diffusion tensor $D$

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\epsilon u, \nabla u). \quad (3.13)$$

Perona and Malik diffusion is now the special case where $D\epsilon = G(||\nabla u||)I$. Because of the isotropic nature of $D\epsilon$, this process is often called isotropic nonlinear diffusion.

Weickert instead proposed an anisotropic diffusion tensor [17], which is constructed by letting the first eigenvector be parallel to $\nabla u\epsilon$, and the second eigenvector orthogonal to $\nabla u\epsilon$. Suppression of diffusion across edges, is achieved by letting the first eigenvalue be decreasing function of $||\nabla u||$, for instance by choosing $\lambda_1 := G(||\nabla u||)$ for $G$ defined above. Similarly we allow diffusion along edges by putting $\lambda_2 := 1$. Once we know eigenvectors and eigenvalues, the tensor can be constructed in explicit form. This process is called anisotropic nonlinear diffusion, or just anisotropic diffusion.

**Higher order diffusion**

Bilinear and trilinear diffusion is defined as:

$$\frac{\partial u}{\partial t} = -\Delta^2 u \quad (3.14)$$

$$\frac{\partial u}{\partial t} = \Delta^3 u \quad (3.15)$$

respectively. These equations have not shown to be good for interpolation between a scattered set of points. A lack of max-min principle leads to undesirable over-under shoots and oscillations, see [14]. Some recent fourth order nonlinear diffusion processes has recently been proposed by Tai et. al. [37, 38, 39]. These have not been never tested for scattered point interpolation yet. They may give good result, but it is unclear whether the extra computational burden is worth it.

Of all operators we have discussed so far, only linear diffusion and anisotropic diffusion has shown to be useful for interpolation. Even more so anisotropic diffusion, but at a higher computational cost. Higher order diffusion fails because of oscillations. Isotropic nonlinear diffusion fails miserably in that it hardly respects the boundary conditions at all.

**3.1.2 Encoding**

Encoding of the image means to find a sparsification that results in a good reconstruction when going through the decoding step. This can be formulated as an
3.1. IMAGES

optimization problem. Lets say we want to store $N$ number of the pixels. Denote such a set of pixels as $S$, and let $u_S$ denote the corresponding solution of (3.1) - (3.2) using this set as boundary pixels. We can obtain the optimal set $S^*$ as

$$S^* = \arg\min_{S \subseteq \Omega | |S| = N} ||u_S - I||_E$$

(3.16)

where $|| \cdot ||_E$ is some norm. As we have mentioned, a good reconstruction is one that gives a visual impression close to the original image. It is of course impossible to find a norm that accurately measures visual quality. Good options may be $L^1$, $L^2$ or various Sobolev norms.

Now comes the question of how to solve this optimization problem. Intuitively $S$ should contain pixels where a lot "is happening" in the image. That is, locations of large gradient magnitude. There are some speculation that $S$ may be obtained by thresholding the Laplacian of the image. However, no proof has been made for the simple norms above.

A stochastic approach can instead be used to solve (3.16) in the $L^1$ norm. The bad news is that this method is very computationally demanding. It can only be used for scientific purposes, as a way to demonstrate the concept of the compression method. Here is a sketch of the algorithm. Define the set $Q \subseteq \Omega$. $Q$ will be initialized as the empty set and successively be filled with the least important pixels in the image. Finally $S$ is obtained as the remaining pixels. That is, $S = \Omega/Q$.

Initialize $Q = \emptyset$.

do while ($|\Omega| - |Q| > |N|$)

Pick $S \subseteq \Omega/Q$ randomly such that $|S| = p|\Omega/Q|$.

Solve (3.1) - (3.2) with $\bar{\Omega} = \Omega\setminus S$.

Update: $Q = Q + \arg\min_{x \in \Omega/S} |u_S(x) - I(x)|$.

$p$ may typically be chosen as 0.9. To speed up the algorithm, usually $Q$ is filled with several of the minimizers in the update step. This is a tradeoff between speed and accuracy.

3.1.3 Some results

Before moving on to our own methods we show some results obtained with the operators we have discussed. Figure (3.3) shows the reconstruction from 5% of the pixels using linear diffusion. Despite of its simplicity, as we can see, the operator gives pretty good results. One disadvantage is the singularities at the interpolation points. Results of anisotropic diffusion will be presented in the
CHAPTER 3. INTERPOLATION AND COMPRESSION

Figure 3.3: Left to right: (a) Ground Truth. (b) Boundary set containing 5% of all pixels. (c) Interpolation from the 5% of pixels if figure b using linear diffusion.

3.1.4 Image approximation

We will now propose our image approximation method, which is also based on linear diffusion. The basic idea is very simple: iteratively switch the roles of inpainting domain and boundary domain. This turns out to be very superior to interpolation with linear diffusion. We will not only avoid singularities, but a further modification will yield a much better overall reconstruction of the image.

We will start to illustrate our method in its simplest form. Let \( \Omega = [\Omega_1 \Omega_2] \) denote a decomposition of the image domain \( \Omega \), let further \( u = [u_{\Omega_1} \ u_{\Omega_2}] \) denote the corresponding decomposition of gray values into each of the domains. Similarly for the true image \( I \). We choose as \( \Omega_1 \) the set of pixels with known gray values \( u_{\Omega_1} \). \( \Omega_2 \) contains the pixels of unknown gray values. Let \( u_0^{\Omega_2} \) be an arbitrary initialization over \( \Omega_2 \). We propose the following method for approximating the function \( u \) over \( \Omega \).

1) Initialize \( u_0 = [u_0^{\Omega_1} \ u_0^{\Omega_2}] \), where \( u_0^{\Omega_1} = I_{\Omega_1} \).

2) Solve for \( n = 1, 2, ..., N \)

\[
\Delta u^n_{\Omega_2} = 0 \quad \text{in} \quad \Omega_2 \quad (3.17)
\]

\[
u^n_{\partial \Omega_2} = u^{n-1}_{\Omega_1} \quad \text{on} \quad \partial \Omega_2 \quad (3.18)
\]

\[
\Delta u^n_{\Omega_1} = 0 \quad \text{in} \quad \Omega_1 \quad (3.19)
\]
For increasing $n$, $u^n$ will be a smoother and smoother approximation of the true image $I$, but also with increasingly worse accuracy. We will justify this statement more mathematically in a moment. First we will show some results.

Our situation is opposite of what we are used to in numerical analysis. We have numerical method, but are not really sure what underlying equation we are solving. That is what we want to investigate. We have not come up with any exact answer yet but very strong speculations. Further work would be complete mathematical proofs of the statements to follow.

Let us first have a look at equation (3.17) - (3.20). The reader may recognize this as the domain decomposition method for the PDE

$$\Delta u = 0 \quad \text{in} \quad \Omega$$

with two subdomains. Domain decomposition is usually a method for discretizing a PDE so that it can be solved by parallel processors in the most efficient way. It is a method that has been deeply analyzed since its introduction in the eighties.
This is treated extensively in Bjoerstad et. al. [34], a book that is highly recommendable. We will instead make use of this method in a completely different way. The main difference in our situation, is that we are not interested in the solution at convergence, but rather at the intermediate step $N$. We also have a very strange decomposition into one large subdomain, and one small subdomain consisting of a scattered set of single grid points (pixels).

Equations (3.17) - (3.20) may also be interpreted in a different way. For that we will need some results from an old numerical PDE solver called Hopscotch iteration developed by Gordon and Gourlay in the sixties. Today this method is almost forgotten, and is not included in any modern textbooks. We have thus given a short summary in appendix A.1. For more info see [20, 18, 19, 21].

In order to see the connection between Hopscotch iteration and our method, we treat the solution in each subdomain as the steady state of the diffusion equation in the domain

$$\frac{\partial u^n_{\Omega_2}}{\partial t} = \Delta u^n_{\Omega_2} \quad \text{in} \quad \Omega_2 \times [0, \infty)$$  \quad (3.21)

$$u^n_{\partial \Omega_2} = \lim_{t \to \infty} u^{n-1}_{\Omega_1} \quad \text{on} \quad \partial \Omega_2 \times [0, \infty)$$  \quad (3.22)

$$u^n_{\Omega_2} |_{t=0} = u^{n-1}_{\Omega_2} |_{t=0} \quad \text{in} \quad \Omega_2.$$  \quad (3.23)

$$\frac{\partial u^n_{\Omega_1}}{\partial t} = \Delta u^n_{\Omega_1} \quad \text{in} \quad \Omega_1 \times [0, \infty)$$  \quad (3.24)

$$u^n_{\partial \Omega_1} = \lim_{t \to \infty} u^n_{\Omega_2} \quad \text{on} \quad \partial \Omega_1 \times [0, \infty)$$  \quad (3.25)

$$u^n_{\Omega_1} |_{t=0} = u^{n-1}_{\Omega_1} |_{t=0} \quad \text{in} \quad \Omega_1$$  \quad (3.26)

We discretize explicitly in time with time steps $\tau$, by letting $u^{n,m}_{\Omega_i}$ denote the approximation to $u_{\Omega_i} |_{t=\tau m}$.

$$u^{n,m+1}_{\Omega_2} = u^{n,m}_{\Omega_2} + \tau \Delta u^{n,m}_{\Omega_2} \quad \text{in} \quad \Omega_2 \times [0,1,\ldots)$$  \quad (3.27)

$$u^{n,m}_{\partial \Omega_2} = \lim_{m \to \infty} u^{n-1,m}_{\Omega_1} \quad \text{on} \quad \partial \Omega_2 \times [0,1,\ldots)$$  \quad (3.28)

$$u^{n,0}_{\Omega_2} = u^{n-1,0}_{\Omega_2} \quad \text{in} \quad \Omega_2$$  \quad (3.29)

$$u^{n,m+1}_{\Omega_1} = u^{n,m}_{\Omega_1} + \tau \Delta u^{n,m}_{\Omega_1} \quad \text{in} \quad \Omega_1 \times [0,1,\ldots)$$  \quad (3.30)
3.1. IMAGES

\[ u_{\partial \Omega_1}^{n,m} = \lim_{m \to \infty} u_{\Omega_1}^{n,m} \quad \text{on} \quad \partial \Omega_1 \times [0, 1, \ldots) \quad (3.31) \]

\[ u_{\Omega_1}^{n,0} = u_{\Omega_1}^{n-1,0} \quad \text{in} \quad \Omega_1. \quad (3.32) \]

This situation is very similar to Hopscotch discretization of the diffusion equation

\[ \frac{\partial u}{\partial t} = \Delta u \quad \text{in} \quad \Omega \times [0, \infty) \quad (3.33) \]

but instead of alternately taking one time step in each domain, we take as many time steps as needed for convergence before switching to the other domain. Thus it doesn’t matter whether the time discretization is implicit or explicit. Despite of this difference we have a strong feeling that our method is still a discretization of the the diffusion equation, with \( n \) being the time parameter. Let us summarize this:

**Postulate 3.10** Let \( u \) be the solution of \((3.33)\), and \( \{u^n\}_{n=0}^\infty \) the solution of \((3.17)-(3.20)\). Given \( n \), for some \( t_n \in \mathbb{R} \), \( u^n \) approximates \( u|_{t_n} \).

We believe a proof can be made by combining the theory of domain decomposition and Hopscotch iteration. Both methods have been analyzed deeply. If this holds true the interesting thing is that any errors in the reconstruction are numerical errors from the PDE discretization. The underlying continuous model is perfectly exact. We thus hope that analyzing the numerical method can directly be used to say something qualitatively about the approximation. Of course we are reconstructing a diffused image, but because of backward uniqueness we can in theory retrieve the original.

However, this is not possible in practice. Backward diffusion is extremely unstable, so any small error in the diffused image will become catastrophal after only a few time steps. The method in its current form is thus not very useful except for avoiding singularities.

We will instead take eq. \((3.21)-(3.26)\) one step further. What we do now will at first seem very strange, the method we present is based on a geometric discovery we have made on surfaces. Its interpretation may become more clear when the reader is familiar with Chapter 3.2

Solve for \( n = 1, \ldots, N \)

\[ \frac{\partial u^n_{\partial \Omega_2}}{\partial t} = \Delta u^n_{\Omega_2} \quad \text{in} \quad \Omega_2 \times [0, \infty) \quad (3.34) \]

\[ u^n_{\partial \Omega_1} = u^{n-1}_{\Omega_1}|_{t=T} \quad \text{on} \quad \partial \Omega_2 \times [0, \infty) \quad (3.35) \]

\[ u^n_{\Omega_2}|_{t=0} = u^{n-1}_{\Omega_2}|_{t=0} \quad \text{in} \quad \Omega_2 \quad (3.36) \]
\[
\frac{\partial u^n_{\Omega_1}}{\partial t} = (-1)^n \Delta u^n_{\Omega_1} \quad \text{in} \quad \Omega_1 \times [0, T) \tag{3.37}
\]

\[
u^n_{\Omega_1} = \lim_{t \to \infty} u^n_{\Omega_2} \quad \text{on} \quad \partial\Omega_1 \times [0, \infty) \tag{3.38}
\]

\[
u^n_{\Omega_1}\big|_{t=0} = u^{n-1}_{\Omega_1}\big|_{t=0} \quad \text{in} \quad \Omega_1. \tag{3.39}
\]

In words, we are iteratively applying backward diffusion every second iteration in \(\Omega_1\) (the domain of known pixel values), and forward diffusion in \(\Omega_2\). Choosing the end iteration \(N = 2\) will give the optimal results. After this the reconstruction quality tends to get worse.

It is now getting increasingly more difficult to say what underlying equation we are discretizing. Since \(\Omega_2\) coincides with locations of high gradients, the answer may be some sort of edge enhancing diffusion of the original image. Again the theory of domain decomposition and Hopscotch iteration may give an answer. What we know is that the results are pretty remarkable. This will be shown in the results section in a moment. First a few words about the implementation. Backward diffusion is as we mentioned extremely unstable, however we are only applying it on one pixel, with the surrounding pixels acting as Dirichlet boundary conditions. This means that backward diffusion can be performed by first solving forward diffusion to convergence, and then invert the displacement and multiply it by a factor. Let \(v\) be the solution of the diffusion equation, and \(u\) the solution of the backward diffusion equation. We assume the domain \(P\) consists of one single pixel.

\[
\frac{\partial v}{\partial t} = \Delta v \quad \text{in} \quad P \times [0, \infty] \tag{3.40}
\]

\[
v|_{t=0} = f \quad \text{at} \quad P \tag{3.41}
\]

\[
v = g \quad \text{in} \quad \partial P \times [0, \infty) \tag{3.42}
\]

\[
\frac{\partial u}{\partial t} = -\Delta u \quad \text{in} \quad P \times [0, \infty] \tag{3.43}
\]

\[
u|_{t=0} = f \quad \text{at} \quad P \tag{3.44}
\]

\[
u = g \quad \text{in} \quad \partial P \times [0, \infty). \tag{3.45}
\]

Let \(v^*\) be the solution of \((3.40) - (3.42)\) at convergence

\[
v^* = \lim_{t \to \infty} v.
\]
For some time $T_{\lambda}$, the solution of (3.43) - (3.45) can now be obtained as

$$u|_{t=T_{\lambda}} = g + \lambda(g - v^*)$$

where $\lambda$ is some parameter. The problem is we cannot control exactly at what time we want to stop the backward diffusion, instead we control the size of the displacements more directly. This is anyway what we want in our application. In fact, putting the parameter $\lambda$ to 1 has been the optimal choice in all our experiments so far.
3.1.5 Experiments

We have done experiments comparing linear diffusion, anisotropic diffusion and our method \((3.34)-(3.39)\). Figure (3.5) shows two test images as well as sparsifications down to 2.5\% of all pixels, computed by the algorithm in Section 3.1.2.

Figure (3.6) shows the reconstructions using only these 2.5\% of pixels as input. Comparing our method to linear diffusion we obviously get a much sharper, more correct reconstruction of the gray values. This is also clear from the average absolute errors in table (3.1.5). The singularities are reduced, but not completely removed as was the case for the lower compression rate in figure (3.4). Points in the neighborhood of the singularity seems to also get effected somewhat. A further idea would be to use overlapping domains. It could be interesting to see how this affects the singularities and overall reconstruction.

While our method is superior to linear diffusion at high compression rates, it is interesting to compare it with anisotropic diffusion. Irena showed in [14] that anisotropic diffusion in the compression setting is highly competitive to methods such as the widely used JPEG format. Considering the average absolute error, our method and anisotropic diffusion are about equal. We must remember that our method is an approximation method, thus potential errors at the interpolation points will add up the overall error. From visual impression we note that anisotropic diffusion also leads to singularities, but they are much less obvious. As expected it also reconstructs sharp edges pretty well. On the other it gives a piecewise constant, cartoonish like appearance to the image destroying some of the depth impression. Since smoothing along edges are allowed, corners also tend to be smoothed out. Another thing to consider is performance. As we mentioned our method is optimal with 2 iterations. By neglecting the computation on the small domain of known pixels values, that means our method is twice as expensive as linear diffusion. Anisotropic diffusion is generally 30 \(-\)40 times as expensive as linear diffusion, even for the most clever implementation.

<table>
<thead>
<tr>
<th>Average absolute error (L1)</th>
<th>Linear diffusion</th>
<th>Our method</th>
<th>Anisotropic diffusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trui</td>
<td>14.614</td>
<td>8.037</td>
<td>7.697</td>
</tr>
<tr>
<td>Peppers</td>
<td>12.967</td>
<td>9.471</td>
<td>10.591</td>
</tr>
</tbody>
</table>
Figure 3.6: Reconstructions from 2.5 % of pixels Top: Linear diffusion. Middle: Our method with $N = 2$. Bottom: Anisotropic diffusion.
3.2 Surfaces

We will now start our focus on surfaces. Our aim is two-fold. First we would like to develop a PDE-based method for shape reconstruction from point clouds. Secondly, we will apply this as a new compression method for triangulated surfaces. Triangulated surfaces are a very popular surface representation in fields such as computer graphics and engineering. We will start to formulate the method in a continuous setting, the continuous equations are then discretized by the finite element method directly on the manifold. It turns out this fits very nicely with triangulated surfaces, if we choose our finite element space in the right way. Lets first give a brief overview of existing compression methods for triangulated surfaces.

Other compression methods

By compression of triangulated surfaces we shall mean two different things, either compression of mesh connectivity or mesh geometry (see chapter (2.2.2). Our focus will be on the latter. For a nice overview of methods for compression of mesh geometry, see [35]. These methods typically exploit the fact that 3D coordinates of vertices are not independent. A very effective method for decorrelating data in such a way is the wavelet transform. Wavelets is to current date probably the most powerful method for compressing mesh geometry. The idea is to approximate the surface at different levels of resolution \( V_0 \subset V_1 \subset \ldots \subset V_n \). Only the coarsest mesh \( V_0 \) is represented. The wavelet coefficients express the differences between the levels, and can thus be used to approximate the finer levels of resolution. If there is a strong correlation between the vertex coordinates, a lot of the wavelet coefficients are close to zero. Compression is achieved by storing the surface in the frequency domain (in terms of wavelet coefficients), and simply ignoring the coefficients that are close to zero. For more details, see for instance [36].

Now lets move on to our method, which is inspired by PDE based image compression presented in the previous chapter. First of all, we would like to know what connections there are between surface processing and image processing. There are no exact answer to this question, but rather different ”philosophies”

Let us have a look at a typical surface. What we perceive as edges in an image are locations of high gradients. Similarly locations of high curvature are perceived as edges on a surface. In some sense one could say curvature plays the same role for surfaces as gradient does for images. This is a fundamental difference as gradients are a first order entity, whereas curvature is a second order entity. There has been some speculation that the natural approach to surface processing, is through processing of the normal vectors. Among those
3.2. SURFACES

are Osher, who recently proposed a fourth order PDE for surface denoising that act by smoothing the normal map \([10]\). Unfortunately, processing of the normal vectors leads to PDEs of higher order, which is computationally very bad news.

Alternatively, we can treat the surface as a union of finitely many coordinate patches. Each of these coordinate patches are, as we know, a map \(U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3\). Thus the surface geometry is locally very similar to a vector valued image, such as a colour image. Interpolation between a sparse set of points is also a very local operation, so differential operators that act directly on the surface geometry may not be a bad choice for us.

As in the image interpolation we have to find suitable differential operators. Pursuing the last alternative, we are interested in using a sparse set of the surface as Dirichlet boundary condition, and apply a smoothing operator in "between".

In chapter 1 we defined differentiable functions on a manifold \(M\). Also recall how we defined inner products on \(C^0(M)\) and \(C^0(TM)\)

\[
(f, g)_M = \int_M fgdx
\]

\[
(v, w)_{TM} = \int_M I(v, w)dx
\]

where \(I\) is the first fundamental form of \(M\). Lets proceed by defining some fundamental differential operators on \(M\). Let \(f \in C^1(M)\), and let \(x\) be the coordinate map of \(M\). We want to define gradients on a surface \(M\). In the euclidian space \(\mathbb{R}^n\), the gradient of a function \(f \in C^1(\mathbb{R}^n)\) is defined as the vector function \(\nabla f\) satisfying

\[
\langle \nabla f, w \rangle = df(w) \quad \forall w \in T\mathbb{R}^n
\]
where $df$ is the differential of $f$, and $\langle ., . \rangle$ is the standard euclidian inner product. The gradient of a function $f$ on a surface $M$ is defined in a similar way by using the surface metric for measurements. The gradient $\nabla_M f$ of a function $f$ on $M$ at the point $p$ in direction $v$ is defined as

$$I_p(\nabla_M f, v) = df_p(v) \quad \forall v \in T_p M.$$  \hfill (3.46)

Since $df_p$ is linear and $(\frac{\partial x}{\partial \xi_1}, \frac{\partial x}{\partial \xi_2})$ evaluated at $x^{-1}(p)$ is a basis for $T_p M$, this is equivalent to

$$I_p(\nabla_M f, \frac{\partial x}{\partial \xi_i}) = df_p(\frac{\partial x}{\partial \xi_i}) \quad \text{for } i = 1, 2.$$  \hfill (3.47)

This definition can be extended to the whole of $M$, by representing the differential $df$ with respect to the first fundamental form

$$I(\nabla_M f, \frac{\partial}{\partial \xi_i}) = \langle \frac{\partial}{\partial \xi_i}, df \rangle.$$  \hfill (3.48)

In local coordinates we get

$$\nabla_M f = \sum_{i,j} g^{-1}_{ij} \frac{\partial (f \circ x)}{\partial \xi_j} \frac{\partial}{\partial \xi_i} \quad \text{(3.49)}$$

where, as we know, $g$ is the matrix of the metric coefficients. We can now further define the divergence $\text{div}_M v$ for a vector field $v \in TM$ as the dual operator of the gradient (see [8])

$$\int_M \text{div}_M v \phi dx = -\int_M I(v, \nabla_M \phi) dx \quad \text{(3.50)}$$

for all $\phi \in C^\infty_0(M)$.

The Laplace operator was used extensively in the last chapter. We are now able to generalize this to 2-manifolds in $\mathbb{R}^3$. The operator $\Delta_M : C^2(M) \to \mathbb{R}$ is defined as

$$\Delta_M = \text{div}_M \circ \nabla_M \quad \text{(3.51)}$$

and is called the Laplace-Beltrami operator. The Laplace-Beltrami operator applied to a function $u \in C^2(M)$, can also be written out in local coordinates

$$\Delta_M u = \sum_{i,j} \frac{1}{\sqrt{\det g}} \frac{\partial}{\partial \xi_i} (g_{ij}^{-1} \sqrt{\det g} \frac{\partial u}{\partial \xi_j})$$  \hfill (3.52)

however, this explicit form will not be used in the finite element discretization.

An analogue of Greens formula can be proved for surfaces with boundaries

$$\int_M \Delta u \phi dx = -\int_M I(\nabla_M u, \nabla_M \phi) dx + \int_{\partial M} \nu \cdot u \phi d\sigma \quad \text{(3.53)}$$
where integration is carried out according to (2.4). \( n^c \) at \( p \) is the co-normal, which is the normal vector of the curve \( \partial M \) at \( p \) lying in the plane \( T_p M \). We are now able to define the diffusion equation for a function \( u \in C^2(M) \)

\[
\frac{\partial u}{\partial t} = \Delta_M u \quad \text{on} \quad M \times [0, \infty)
\]  

(3.54)

\[
u|_{t=0} = u_0 \quad \text{on} \quad M \times \{t = 0\}.
\]  

(3.55)

However, as we discussed, we want to apply this operator directly on the surface geometry. The equation then reads

\[
\frac{\partial x}{\partial t} = \Delta_{M(t)} x \quad \text{on} \quad M(t) \times [0, \infty)
\]  

(3.56)

\[
M(0) = M_0
\]  

(3.57)

where \( x \) denotes the surface coordinates (a vector of three components), and \( M_0 \) is the initial surface. This is often called the geometric diffusion equation.

Note that the differential operators are now time dependent, since the domain where they are defined is time dependent. In the past, this equation has been used for the purpose of geometry denoising. It is a second order equation, and is thus very computationally efficient compared to Osher’s method, but unfortunately it also has some disadvantages. One of these being shrinkage. It can be shown that (3.56) - (3.57) is equivalent to mean curvature motion (see Willmore [5] page 151)

\[
\frac{\partial x}{\partial t} = 2H(x, t)n.
\]  

(3.58)

In Chapter 2.4 we saw the shrinkage effects of mean curvature motion. In fact one can show (see Clarenz [3] and Sapiro [4])

\[
\frac{d}{dt} \text{Area}(M(t)) = - \int_{M(t)} H^2 \, dx
\]

\[
\frac{d}{dt} \text{Volume}(M(t)) = - \int_{M(t)} H \, dx.
\]

When used for compression, the volume of \( M^{true} \) is known. This has led us to consider volume preserving flows as interpolant to counteract these shrinking effects. This can be achieved by introducing a forcing term as follows

\[
\frac{\partial x}{\partial t} - \Delta_{M(t)} x = fn
\]  

(3.59)

where \( f \) is defined as

\[
f = \frac{1}{\int_{M(t)} dx} \int_{M(t)} H \, dx.
\]
Huisken [6] showed that the volume enclosed by the surface do not change under this evolution. However this means that the surface must be initialized with the correct volume, a task that may not be very easy. We have not pursued this idea any further. The simplest methods should always be tested first, in our case that means linear geometric diffusion. In fact we found a very different way of preventing the shrinking effects, which will be shown in the end of the chapter.

Let \( M^{\text{known}} \) denote the known part of the surface. In general \( M^{\text{known}} \) will be a disconnected set. Our aim is to find a closed surface containing \( M^{\text{known}} \), that closely resembles the true surface \( M^{\text{true}} \). Our interpolation method with geometric diffusion is thus the following

\[
\frac{\partial x}{\partial t} = \Delta_{M(t)} x \quad \text{on} \quad M(t) \times [0, \infty) \tag{3.60}
\]

\[
M(0) = M^{\text{true}} \tag{3.61}
\]

\[
x = x_0 \quad \text{on} \quad \partial M(0) \times [0, \infty). \tag{3.62}
\]

### 3.2.1 Finite element discretization

We will discretize (3.60)-(3.62) by the finite element method directly on the surface geometry. Finite element discretization was first proposed for geometric diffusion by Dziuk in [30]. In contrast, the level set method is instead discretized in the surface embedding space. This has many advantages as was stated in Chapter 1, but there are many reasons why the level set approach was not an option for us. First of all is the performance issue. Compression need to be relatively fast in order to be practical. Since the surface geometry itself is 2 dimensional, our computations only need to be done in 2d. The level set method is discretized in 3d, thus its main disadvantage will always be expensive computation. Secondly, from a global viewpoint we are always able to initialize the surface well. This means we can avoid degenerations of the surface mesh, which is typical when evolving a mesh through large deformations. And lastly, our main focus will be on triangulated surfaces. As we will show, the finite element method fits very natural with this kind of surface representation.

We have tried to make this section as complete as possible. Still it is recommendable that the reader has some knowledge about the finite element method in normal euclidian spaces.

As a first step of the discretization, we express (3.56) - (3.57) in variational form. Find \( x(t) \) with \( x_i(t) \in C^\infty_b(M(t)) = V \) (subscript \( b \) means satisfying the boundary conditions), for \( i = 1, 2, 3 \) such that

\[
\int_{M(t)} \frac{\partial x_i}{\partial t} \phi \, dx = \int_{M(t)} \Delta_{M(t)} x_i \, \phi \, dx = \int_{M(t)} I(\nabla_{M(t)} x_i, \nabla_{M(t)} \phi) \, dx \quad t > 0 \tag{3.63}
\]
3.2. SURFACES

for all $V$. Note that the boundary term in Greens formula vanishes, because the boundary is held fixed, see [32]. We will from now on use the inner product notation for this equation

$$\left( \frac{\partial x_i}{\partial t}, \psi \right)_{M(t)} = -\left( \nabla M(t)x_i, \nabla M(t)\psi \right)_{TM(t)} \quad t > 0 \quad (3.64)$$

$$M(0) = M_0 \quad (3.65)$$

for $i = 1, 2, 3$ and all $\psi \in V$.

Discretization is achieved by replacing the infinite dimensional space $V$, with a finite dimensional subspace $V_h \subset V$. The surface coordinate maps are thus reduced to a finite dimensional subspace. This will result in a discrete surface which we call $M_h$. Let $X(t)$ denote the discrete coordinate map, we then obtain

$$\left( \frac{\partial X_i}{\partial t}, \Psi \right)_{M_h(t)} = -\left( \nabla M_h(t)X_i, \nabla M_h(t)\Psi \right)_{TM_h(t)} \quad t > 0 \quad (3.66)$$

$$M_h(0) = M_{ho} \quad (3.67)$$

for $i = 1, 2, 3$ and all $\Psi \in V_h$.

We also want to discretize in time. Let $\tau$ denote a small time step, we are interested in discrete values $X^n$ such that $X^n \approx X(n\tau)$. Time derivative can be approximated by a backward Euler scheme

$$\frac{\partial X((n+1)\tau)}{\partial t} = \frac{X^{n+1} - X^n}{\tau} + O(\tau).$$

Let further $M^n_h$ denote the discrete surface at time step $n$. Equations (3.64)-(3.65) can now be discretized in time by the following implicit scheme

$$\left( \frac{X^{n+1}_i - X^n_i}{\tau}, \Psi \right)_{M^n_h} = -\left( \nabla M^n_h X^{n+1}_i, \nabla M^n_h \Psi \right)_{TM^n_h} \quad t > 0 \quad (3.68)$$

$$M^n_h = M_{ho} \quad (3.69)$$

Finally, we choose a basis $\{\Phi_j\}_{j=1}^m$ for $V_h$, where $m$ is the dimension of $V_h$. Now we can express $X^n_i$ in terms of the basis as

$$X^n_i = \sum_{k=1}^m c_i^n(k)\Phi_k. \quad (3.70)$$

By putting into (3.64)-(3.65), rearranging and using the axiom that inner products are linear we arrive at

$$\sum_{k=1}^m c_i^{n+1}(k)(\Phi_k, \Phi_j)_{M^n_h} + \tau \sum_{k=1}^m c_i^{n+1}(k)(\nabla M^n_h \Phi_k, \nabla M^n_h \Phi_j)_{TM^n_h} = \sum_{k=1}^m c_i^n(k)(\Phi_k, \Phi_j)_{M^n_h}. \quad (3.71)$$
Define the matrices $M_{kj} = (\Phi_k, \Phi_j)_{M_k}$ and $L_{kj} = (\nabla M_n^k \Phi_k, \nabla M_n^k \Phi_j)_{TM_n^k}$. Since $c_i^n(k)$ is known at step $n + 1$, (3.71) is a linear system that can be written

$$(M + \tau L)c_i^{n+1} = Mc_i^n$$

(3.72)

where $c_i^n$ is a vector of components $c_i^n(k)$. $(M + \tau L)$ is known as the stiffness matrix. Note that it is independent of the spatial coordinate $i$.

The $c$ coefficients are the parameters that control the shape of the surface. Their interpretation may become more apparent when we specify the subspace $V_h$ with a particular basis.

$V_h$ is called the finite element space. In our case it will consist of piecewise linear polynomials defined on a subdivision of the surface $M_h$. As a result the discrete surface $M_h$ will itself be piecewise linear. We will concentrate on a special kind of piecewise linear surfaces, namely triangulated surfaces. This surface representation was discussed in chapter 1. As we know it consists of a set points in 3d space, and a set of triangles having elements of the point set as corners. We choose this set of triangles as subdivision for our finite element space. Let us state this a little more formally. Define

$$T_h = \{T_i\}_{i \in I} \text{ triangulation of } M_h$$

$$V_h = \{\theta \in C^0(M_h) \text{ s.t. } |\theta|_{T_i} \in P_i \forall i \in I\}.$$ 

$P_n$ is a notation for the space of polynomials of order $n$. Let us also formally define a finite element.

A finite element is a triple $(K, P, \Sigma)$, where

- $K$: Geometric object
- $P$: Space of functions defined on $K$
- $\Sigma$: Basis for dual space of $P$

In our case, as we mentioned, $K$ is a triangle in $\mathbb{R}^3$, $P = P_1(K)$, and $\Sigma = \text{coordinates of corners in } K$. Now, lets suppose we are given a triangle $T \subset M_h$. $T$ may be considered the image of a coordinate patch $X : \hat{T} \rightarrow M_h$, where $\hat{T} \subset \mathbb{R}^2$ is a reference triangle. In particular, we choose

$$\hat{T} = \{ (\xi_1, \xi_2) \in \mathbb{R}^2 \mid \xi_1, \xi_2 > 0, \xi_1 + \xi_2 < 1 \}.$$ 

For each coordinate patch $X_\alpha : U_\alpha \subset \mathbb{R}^2 \rightarrow T_\alpha$, we define the nodal basis for $P_1(T_\alpha)$ in the following way

$$\Phi_1(X_\alpha(\xi_1, \xi_2)) = \xi_1$$

$$\Phi_2(X_\alpha(\xi_1, \xi_2)) = \xi_2$$

$$\Phi_3(X_\alpha(\xi_1, \xi_2)) = 1 - \xi_1 - \xi_2.$$
Once the basis has been chosen, the stiffness matrix can be explicitly formed. Furthermore, any function $\theta$ in $P_1(T_\alpha)$ can now be expressed as

$$\theta = \sum_{k=1}^{3} \bar{X}(k) \Phi_k$$

where $\bar{X}(k)$ is the coordinates of corner (node) $k$ in $T_\alpha$. Inserting in \((3.71)\), this leads to the following update equation for all point coordinates in the mesh

$$(M + \tau L)\bar{X}_{i}^{n+1} = M\bar{X}_i^n \quad i = 1, 2, 3,$$

\((3.73)\)

where $\bar{X}_i^n$ is simply the vector of coordinate $i$ of all vertices at time step $n$. Because of the implicit time discretization, we can take relatively large time steps. Usually we can get convergence in 5-10 time steps.

**Solving the linear system of equations**

The linear system of equation that arises in each timestep needs to be solved in the fastest possible way. The stiffness matrix has as many rows and columns as there are vertices in the mesh, which is usually hundreds of thousands. We note the following about this system of equations:

- The stiffness matrix $(M + \tau L)$ is symmetric.
- $(M + \tau L)$ is very sparse.
- An approximate solution of vertex coordinates $\bar{X}_i^{n+1}$ at time step $n+1$ exist, namely the coordinates at the present time step $\bar{X}_i^n$.

From these facts it is clear that an iterative method is the best option. Iterative methods involves only matrix vector multiplications and are thus of order $O(n^2)$, where $n$ is the number of rows and columns. In contrast direct methods are of order $O(n^3)$. If we have a good initialization, which is the case for us, only a few iterations are necessary. In fact our situation is even simpler. $(M + \tau L)$ has its main concentration of nonzeros as a band around the diagonal. The reason is because of the very local basis we have chosen for $V_h$. This situation of many nonzeros can be exploited further by using a sparse matrix representation. This is a way to represent the matrix by a pair of vectors of the same lengths as there are number of nonzeros. The first vector specifies locations of the nonzeros, and the second specifies their values.

In the sparse matrix representation, matrix vector multiplications can be computed in $O(Bn)$ where $B$ is the average number of nonzeros in each column. In our case $B$ usually averages at about 6-7.

This way of representing the matrices are also crucial for us since it is impossible to declare enough memory to store a full matrix object of hundreds of thousands rows and columns.
We have used a combination of Gauss-Seidel and the successive overrelaxation method (SOR) for solving the linear system. Other iterative methods to consider are the conjugate gradient method.

3.2.2 Encoding

Recall from Chapter 2.2.2 that a triangulated surface consists of a set of point coordinates, and a set of triangles. For polygonal meshes these are represented by mesh geometry and mesh connectivity.

For a typical surface mesh, there are twice as many triangles as points. In our experience the amount of data for storing mesh geometry (vertices) and mesh connectivity are about equal.

Decoding can be performed in the same manner as for images. We want to store only a small subset of the surface, denoted $S$, such that interpolating with this subset as boundary condition yields a reconstruction as close as possible to the true surface $M^{true}$. As was the case for images, this can be formulated as an optimization problem. Let $M^S$ denote the solution of (3.60)-(3.62). In the continuous setting we want find the subset $S^*$ such that

$$S^* = \arg \min_{S \subset M^{true}, |S|=A} ||M^S - M^{true}||_E$$

(3.74)

in some norm $|| \cdot ||_E$. When switching to the discrete setting we want to store a small set of the coordinates $\{c_i\}$ specifying the surface. In our case that means we want to store a small set of vertex positions. All other points and triangles are deleted. Let $V$ denote the set of all vertices. Reconstruction from a given subset $v \subset V$, will be denoted $M^v_h$. Compression down to $N$ vertices, means to find to find $v^*$ such that

$$v^* = \arg \min_{v \subset V, |v|=N} ||M^v_h - M^{true}_h||_E.$$  

(3.75)

We do not yet know how to solve this optimization problem efficiently. An idea would be thresholding of the magnitude of the laplace beltrami operator applied to the surface. However this is only guessing. In this work we want to prove the concept and leave to future research direction the problem of solving (3.75) efficiently. We will instead use a similar stochastic approach as we did for image coding. Although slow, we can make sure to find the right solution up to some statistical error. In the coding step $M^{true}_h$ is known, and can thus be used as initial surface for the evolution. Comparing $M^v_h$ with $M^{true}_h$ can be done by comparing vertex positions. We would like to know vertex displacements in the normal direction. In fact we know from mean curvature motion that all forces act in the normal direction. However this is not completely true, since additional forces are introduced through incorporations of the boundary conditions. We will nevertheless simply compare the euclidian distance between nodes in $M^v_h$.
and $M_h^{true}$, neglecting these potential tangential forces. As for images we define the set $Q$, that will now successively be filled with the least important vertices.

Initialize $Q = \emptyset$

do while ($|V| - |Q| > |N|$)

Pick $v \subset V/Q$ randomly such that $|v| = p|V/Q|$.

Solve (3.60)-(3.61) with $\partial M = \partial(V/Q)$.

Update: $Q = Q + \arg \min_{x \in V/v} ||M^v_h - M^{true}_h||_2$.

We have chosen $p = 0.9$.

### 3.2.3 Decoding

Decoding for triangulated surfaces means to solve the linear system (3.73) to convergence. However, since the points we wish to process are deleted in the coding step, we need to create a new points in some initial locations. We further need to create a triangulation on all points that are now in the mesh. There are many fast algorithms for creating triangulations given a set of points. One idea would thus be to first create a triangulation on the sparse set of known points. This triangulation could then be refined multiple times, until one gets the desired resolution. This also provokes a multiresolution strategy where we iteratively, after each refinement, solve the interpolation problem on the current mesh.

This has not been our focus however. Since the problem is convex, we have simply used the original surface and triangulation as initialization, just to get the results as fast as possible.

### 3.2.4 Some results and their shortcomings

Figure (3.8) shows a surface mesh of Moai on the left, which was compressed down to 10% of all points. On the right, we see the corresponding reconstruction after decoding with geometric diffusion. The first thing you notice, is probably the disturbing singularities at the interpolation points. While singularities were hardly noticeable in an image, they are catastrophic when appearing on the surface geometry. On the other hand, the fact that these singularities appear is a sign that we may be on the right track. Apart from them, the overall reconstruction is actually pretty good. The volume gets reduced, but only slightly. These facts has motivated us to develop the surface approximation method which we will present in the next chapter.
3.2.5 Surface approximation

The idea for our approximation method in Chapter 3.1.4 was originally discovered while working with surfaces. The singularities are a much bigger disadvantage on a surface, because they are a lot more obvious. We observed that by iteratively changing the role of inpainting domain and boundary domain, we could reconstruct a diffused version of the surface and avoid these singularities. We were somehow trading smoothness for accuracy. However, since the interpolation points were now longer fixed, they were not able to hold on to the surface preventing it from shrinking. This led us to the idea of instead solving the backward problem on the interpolation points. This would exaggerate their role of controlling the shape of the surface, somewhat related to control points in spline representation for curves and surfaces. In fact, not only shrinkage effects were prevented, but the overall reconstruction became sharper and more realistic. This inspired us take these geometric ideas back to the image setting. We have discussed this extensively in Chapter 3.1.4. In the image setting, we tried to search for a mathematical meaning of the method, as a combination of domain decomposition and Hopscotch iteration. The reader is advised to have a look at Chapter 3.1.4 before proceeding. Our two methods for surface approximation are basically the same as (3.17)-(3.20) and (3.34)-(3.39) from Chapter 3.1.4 but instead with operators we have discussed in this chapter. In the following we let $M_1 = M/\partial M$, and $M_2 = \partial M$. For simplicity we use the continuous representation of the operators.
Method I:
Solve for \( n = 1, \ldots, N \)

\[
\frac{\partial x_1^n}{\partial t} = \Delta_{M_1^n(t)} x_1^n \quad \text{on} \quad M_1^n(t) \times [0, \infty) \tag{3.76}
\]

\[ M_1^n(0) = M_1^{n-1}(0) \tag{3.77} \]

\[ x_1^n = \lim_{t \to \infty} x_1^{n-1} \quad \text{on} \quad \partial M_1^n(0) \times [0, \infty) \tag{3.78} \]

\[
\frac{\partial x_2^n}{\partial t} = \Delta_{M_2^n(t)} x_2^n \quad \text{on} \quad M_2^n(t) \times [0, \infty) \tag{3.79}
\]

\[ M_2^n(0) = M_2^{n-1}(0) \tag{3.80} \]

\[ x_2^n = \lim_{t \to \infty} x_1^n \quad \text{on} \quad \partial M_2^n(0) \times [0, \infty). \tag{3.81} \]

Method II:
Solve for \( n = 1, \ldots, N \)

\[
\frac{\partial x_1^n}{\partial t} = \Delta_{M_1^n(t)} x_1^n \quad \text{on} \quad M_1^n(t) \times [0, \infty) \tag{3.82}
\]

\[ M_1^n(0) = M_1^{n-1}(0) \tag{3.83} \]

\[ x_1^n = x_2^{n-1} |_{t = T} \quad \text{on} \quad \partial M_1^n(0) \times [0, \infty) \tag{3.84} \]

\[
\frac{\partial x_2^n}{\partial t} = (-1)^n \Delta_{M_2^n(t)} x_2^n \quad \text{on} \quad M_2^n(t) \times [0, T) \tag{3.85}
\]

\[ M_2^n(0) = M_2^{n-1}(0) \tag{3.86} \]

\[ x_2^n = \lim_{t \to \infty} x_1^n \quad \text{on} \quad \partial M_2^n(0) \times [0, \infty). \tag{3.87} \]

For method II, \( N = 2 \) seems to give the optimal results. By neglecting the computation on the small domain, this means method II is twice as expensive as linear geometric diffusion.
The implementation of backward diffusion is similar to the approach in chapter (3.1.4) for images. Since only a single vertex is being processed with its neighbors as boundary vertices, we can simply solve the forward diffusion problem, and then invert the displacements.

3.2.6 Experiments

![Figure 3.9: Left: Moai ground truth, 10002 vertices, 20000 triangles. Right: Armadillo ground truth, 165954 vertices, 331904 triangles.](image)

We have done experiments comparing linear geometric diffusion, method I and method II. The experiments are performed on one low resolution surfaces and two high resolution surfaces. Figure (3.9) right shows the low resolution surface, and figure (3.9) left shows a high resolution surface. In figure (3.10) we see the reconstructions of the low resolution surface after compression down to 5% and 10% of the vertices. As we can see the surface approximation methods yields much more visually pleasing reconstructions without singularities. Method I gives a slightly shrinked and deformed reconstruction, while using method II we are able to prevent these characteristic effects of geometric diffusion, and get an overall more correct reconstruction.

We also note that the surface is reconstructed in a piecewise linear like fashion. This is especially noticeable on the slightly curved frontal part of the object.
Figure 3.10: TOP: Reconstructions from 10% of all points. BOTTOM: Reconstructions from 5% of all points
Left: Linear geometric diffusion. Middle: 1 iteration of method I. Right: 2 iterations of method II.
Figure 3.11: LEFT: reconstruction from 10% of vertices, RIGHT: reconstruction from 5% of vertices.
Top: Linear geometric diffusion. Bottom: 2 iterations of method II.
3.2. SURFACES

Figure 3.12: Reconstructions from 10% of all points. Total number of vertices: 199169, triangles: 398043
Left: Ground truth. Middle: linear geometric diffusion. Right: 1 iterations of method I.

at 5% compression rate. As a result, linear parts of the object gets reconstructed well. At higher resolution, interpolation becomes more of a local operation and less of a global, which means such piecewise linear reconstruction will be more ideal. Figure (3.11) and figure (3.12) shows such high resolution examples. Reconstruction from 10% and 5% of the vertices gives pretty good results. Some very high frequencies with low magnitude, such as some wrinkles in (3.12), tends to be washed out.

3.2.7 Other interpolants to consider

There are particularly two other differential operators we believe should be tested for compression in the future. Due to its success in image compression, an anisotropic nonlinear geometric diffusion process should be considered. Secondly a higher order diffusion process, smoothing the normal map should be tested.

Anisotropic geometric diffusion

We remember from images that we could formulate a nonlinear diffusion process that suppresses smoothing in areas of sharp edges, indicated by large gradient. A similar process can be formulated for surfaces, but we must remember sharp edges are now indicated by high curvature. An anisotropic approach was suggested for geometry denoising by Rumpf in [5]. The diffusion coefficient is replaced by a
diffusion tensor $D_\epsilon : TM_\epsilon \to TM_\epsilon$ as follows

$$\frac{\partial x}{\partial t} = \text{div}_{M(t)}(D_\epsilon \nabla_{M(t)} x). \quad (3.88)$$

Let $\lambda_1$ be smallest principle curvature, and $\lambda_2$ the largest principal curvature. $D_\epsilon$ is constructed so that its eigenvectors are parallel to the principal directions of curvature. We want to allow smoothing in the direction corresponding to the smallest principal curvature, and suppress smoothing in the direction corresponding to the largest principal curvature. This can be achieved by constructing the eigenvalues of $D_\epsilon$ as $G(\lambda_1)$ and $G(\lambda_2)$, for some decreasing function $G$. For instance $G$ may be chosen according to (3.12).

Also this process can be shown to have a characteristic surface and volume minimization effect. It should therefore be used in combination with the approximation method we just have presented.

Similar to the image case, the subscript $\epsilon$ means $D_\epsilon$ is not computed on the actual surface $M$, but a preprocessed surface $M_\epsilon$ where very high frequencies are filtered out. For instance one may use a time step of size $\epsilon$ of mean curvature motion.

**Minimizing total curvature**

Interpolation with linear geometric diffusion is the gradient descent equation of the minimization problem

$$\min_{M \in \Xi} \int_M dA. \quad (3.89)$$

Where $\Xi$ now is the set $\Xi = \{M \mid M$ is connected and closed, $M^{\text{known}} \subset M\}$. A more intuitive approach would be for instance to minimize the total curvature

$$\min_{M \in \Xi} \int_M k_1^2 + k_2^2 \, dA \quad (3.90)$$

which we expect would yield a smooth reconstruction without minimizing volume. This would lead to a 4th order gradient descent equation. For the problem to be well-posed, not only positions but also 1st order differentials, such as normal vectors should be specified.

The experience from images is that these higher order diffusion processes results in oscillations when the boundary consists of points of very small diameter, no max/min principle can be proved. This does not necessarily mean the same would happen for surfaces. This equation do fit with the proposition of transferring image processing ideas to surfaces by processing the normal map. However, even if good results are achieved, one would always have the problem of high computational cost associated with such higher order equations. Numerically it is also not clear whether it is possible to solve using linear elements. It may have more relevance for classical inpainting (hole filling) of surface geometry, since performance is not such a big issue there.
3.3 Conclusions

We have presented a new method for compression of surface geometry based on partial differential equations.

It must be noted that this work is only a proof of concept, and can only be regarded as the first step towards a practical method. There are still many open questions and room for many further developments. Judging by the results of experiments, we are confident that PDE-based geometry compression may become competitive to existing methods. We particularly believe it will be very competitive in case of surfaces consisting of sharp edges, and otherwise a piecewise linear nature.

The compression method was based on a geometric PDE for reconstructing (interpolating) surfaces from a scattered set of points. Compression was achieved by storing only the set of points that could not easily be reconstructed by the PDE. In this thesis we only implemented a linear geometric diffusion process. While geometric diffusion may not be the best candidate for interpolation because of singularities and unwanted shrinking effects, it could be used as the basis in a new approximation method with much greater success. The benefits of this simple interpolant, were existence of a max-min principle and very fast computation. In fact, using finite element discretization and iterative methods for solving its linear system of equations, the surface could be reconstructed in \(O(\text{number of vertices})\) complexity.

While other methods typically reduces the number of bits for storing each vertex coordinate, we instead delete a large number of the coordinates completely while storing the remaining coordinates with high precision. As a result, the mesh connectivity (triangulation) is also deleted. The effect of this is both positive and negative:

- Positive: Since the mesh connectivity does not need to be stored, the effective compression rate is much higher than simply the fraction of stored vertices over total vertices

- Negative: In the decoding step, new vertices need to be initialized, and a triangulation must be defined over this new vertex set before the PDE can be solved (see discussion in 3.2.3)

In this thesis, this issue has simply been ignored. We have just used the original vertex set and triangulation for initialization. There is no guarantee that the same results can be obtained from different triangulations, however this is just a minor issue. As we discussed in 3.2.3, we suggest a multiresolution approach to this problem. Many fast algorithms exist for creating triangulations and/or remeshing.

Because of this uncertainty about the actual compression rate and the actual reconstruction quality, we have decided not make comparison with existing methods yet. This should be one of the first things to be done in the future.
Surface compression was in this thesis developed by transferring ideas from recent PDE-based image compression. In the surface setting we came up with an idea that could vice-versa be transferred back to images. This idea, that involved iteratively changing the role of inpainting domain and boundary domain, improved results significantly both for surfaces and for images. In the image domain, we tried to interpret this as a discretization method of a yet unknown underlying PDE. It has a striking similarity with two established PDE solvers, namely domain decomposition and Hopscotch iteration. Further investigations should be made in order to get a greater understanding of this method.
Chapter 4

2D-3D pose estimation

2D-3D pose estimation is the problem of estimating a known 3d object’s location in the world coordinate system from a set of 2d images shot from different viewpoints. For example, for rigid objects there are 6 degrees of freedom to describe this location in the 3d world, 3 for translation and 3 for rotation. Pose estimation in the rigid case thus means to find a map that translates and rotates the object such that it fits the 2d data in the best possible way.

We will propose a variational approach for pose estimation, by combining recent advances in 2D-3D surface reconstruction and image segmentation. This method has certain advantages and disadvantages compared to existing methods. Several approaches to the pose estimation problem exists. It is beyond the scope of this thesis to review this literature. An overview can be found in [12, 11]. What most existing methods have in common, is to search for correspondences between points in the image and points in the 3D object. This point concept can also be generalized to correspondences of 2D point to 3D line, or 2D line to 3D line. Since these methods rely on identification of such points or lines in the image, they are usually very sensitive to noise or other image distortions such as occlusions. The input images typically have to be presegmented, unless there is a clear distinction between image object and background.

In our method, we instead try to fit the 3D object to the images in a more global fashion, by minimizing a certain energy functional over surfaces. As a result, our method is robust to noise, and there is no requirement that the image objects should be presegmented. In fact, it works well even if the input images are so bad, that a correct segmentation is impossible without any prior shape information. This means we are dually solving a completely different problem, namely that of segmenting the 2D images using a priori information about all views of all 3d shapes in a certain shape subspace (to be specified later). These are in fact direct consequences of each other.

Another problem with existing methods is dealing with complex deformations. When deformations are involved, the number of pose parameters goes well beyond 6. Kinematic chains have been used to model piecewise rigid objects. That is,
CHAPTER 4. 2D-3D POSE ESTIMATION

objects consisting of rigid pieces tied together in single points. Deformations of for instance the human body can be approximated well by kinematic chains. For more complicated deformations it has to our knowledge not been any good solutions so far.

Again, since our method is based on minimizing an energy functional over surfaces, a generalization to rigid transformations and deformations can be made very simple and elegant, as we will see. Our method is level set based. The level set method is well known for flexibility in allowing large shape deformations. This is our main motivation for using it, and we hope this is enough compensation for its main disadvantage, namely expensive computation.

Most related to our work, is probably a method proposed by Brox et. al. in 2005. He proposed to combine a traditional pose estimation method with variational image segmentation. He was able to simultaneously segment the image object, and estimate the pose parameters, by alternating between image segmentation and pose estimation every second iteration. Of course any deformations in his approach are also restricted to kinematic chains. It is also not clear under which criteria it converges, very good initializations are usually necessary. In our method, the entire model is formulated in a single energy functional. As a result, both pose parameters and segmentation contour can be estimated by the gradient descent method. It is thus clear that our method is much easier to analyze, we can for instance simply explain fail of convergence by local minima in the energy functional.

The input images used for pose estimation contain in some sense a 2 dimensional projection of the object in its correct pose. The human brain can easily create a 3 dimensional abstraction of the object using clues such as lightning reflection and contour. Existing methods takes only the contour into account when estimating the pose. Our method is also contour based, but a very interesting future work would be to generalize this to integrate shading information.

There are two main disadvantages to our method. First, it does not work in case of one single view. However, we are interested in difficult scenarios of deformations and image occlusions. Two or more views are usually necessary for the pose estimation problem to be well posed in these situations. Secondly, since our method is level set based, it is computationally expensive compared to other methods.

Traditional 2D-3D surface reconstruction will be the heart of our method. 2D-3D reconstruction is the problem of estimating the shape of a scene from a set of images taken from different viewpoints. This is very related to pose estimation. We will particularly focus on recent variational approaches to the 2D-3D reconstruction problem. These methods estimate the 3d surface as the minima of certain energy functionals taking surfaces as argument. These energy functionals seem to work well for measuring similarity between surface and image, which
is our motivation for using them for pose estimation. Simply put, our method consists of solving these minimization problems with appropriate constraints on the surface.

So far we have been very loosely speaking. We now need to put a little more meat on the bones.

In Section 4.1 we explain how to represent the map that translates and rotates rigid bodies, often called rigid body motion. We will in Section 4.2 further introduce a shape deformation model, through a process called principle component analysis on a set of training shapes. This model imitates the human brain in learning from examples, and is in one of the most elegant and efficient ways to model shape deformations. It has been used in a number of applications, including image segmentation with a priori shape information.

In Section 4.3, we will start by defining operators such as perspective projections, discuss camera setup and the relationship between 2d images and the 3d world. We will then derive a recently proposed 2D-3D reconstruction model, and show how it can be solved by the level set method.

In Section 4.4 we present our own pose estimation method.

### 4.1 Rigid Transformations

We will define operators that rotate and translate points in the euclidian space. Let's start by pure rotation about the origin. Such operators belong to the group \( SO(n) \) defined as

\[
SO(n) = \{ M \in \mathbb{R}^{n \times n} \mid MM^T = I, \ det(M) = 1 \}.
\]  

(4.1)

\( n \) is the dimension of the euclidian space. When \( n = 2 \), any element in \( SO(2) \) can be represented by the matrix

\[
M(\theta) = \begin{pmatrix}
\cos(\theta) & \sin(\theta) \\
-\sin(\theta) & \cos(\theta)
\end{pmatrix}
\]

(4.2)

for some \( \theta \in [0, 2\pi) \). When applied to a point in \( \mathbb{R}^2 \), the point gets rotated \( \theta \) degrees counterclockwise about the origin. In case \( n = 3 \), the situation is a bit more difficult. We will start by defining special elements of \( SO(3) \), that rotates points counterclockwise about the \( x, y \) and \( z \) axes respectively

\[
M_x(\omega) = \begin{pmatrix}
\cos(\omega) & \sin(\omega) & 0 \\
-\sin(\omega) & \cos(\omega) & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

(4.3)

\[
M_y(\beta) = \begin{pmatrix}
\cos(\beta) & 0 & \sin(\beta) \\
0 & 1 & 0 \\
-\sin(\beta) & 0 & \cos(\beta)
\end{pmatrix}
\]

(4.4)
\[ M_z(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & \sin(\gamma) \\ 0 & -\sin(\gamma) & \cos(\gamma) \end{pmatrix} \]  \hspace{1cm} (4.5)

Any element in \( SO(3) \) can now be represented by the matrix

\[ M(\omega, \beta, \gamma) = M_x(\omega)M_y(\beta)M_z(\gamma) \]  \hspace{1cm} (4.6)

for \((\omega, \beta, \gamma) \in [0, 2\pi)\). Note that \( SO(3) \) is not commutative, so the order in which \( M_x, M_y \) and \( M_z \) are multiplied does matter.

We can now define the group of rigid transformations

\[ SE(n) = \{ R \mid R(x) = Mx + T, \ M \in SO(n), \ T \in \mathbb{R}^n \}. \]  \hspace{1cm} (4.7)

In case of \( n = 3 \), the elements of this group can also be represented by a single matrix \( R \)

\[ R(\omega, \beta, \gamma, t_1, t_2, t_3) = \begin{pmatrix} M_{3 \times 3} & t_{1 \times 3} \\ 0_{3 \times 1} & 1 \end{pmatrix} \]  \hspace{1cm} (4.8)

This matrix is applied to points in \( \mathbb{R}^3 \) of the form: \((x, y, z, 1)\). The 4th coordinate is called the homogenous coordinate, and is used to distinguish points from vectors in \( \mathbb{R}^3 \). Vectors will in constrast have the form \((x, y, z, 0)\).

In other words \((\omega, \beta, \gamma, t_1, t_2, t_3)\) are 6 variables that can identify any rigid transformation in \( \mathbb{R}^3 \). We define these variables as the pose parameters, and will denote them by vector \( p \) throughout the rest of this thesis.

Let’s say \( S \) is a shape in \( \mathbb{R}^3 \) represented by the level set function \( \psi \). We would like to apply a rigid transformation to \( S \). This can be done by transforming the coordinate system where \( \psi \) is defined

\[ \psi_{\text{rigid}}(x) = \psi(R(p)x) \quad \forall x. \]  \hspace{1cm} (4.9)

In the discrete case \( \psi \) is only defined on a scattered set of grid points. In general, by transforming the coordinate system, we may need function values outside this grid. This requires a suitable interpolation method. We have used trilinear interpolation in all our experiments. More accurate methods to consider are tricubic interpolation, but at a higher computational cost.

Before moving on to the next section where we discuss non-rigid shape deformation, we show how scaling can be incorporated in the level set framework. Let \( \psi \) represent some surface in initial pose. \( \psi \) can be scaled and rigidly transformed according to the equation

\[ \psi_{\text{scale+rigid}}(x) = \psi(sR(p)x) \quad \forall x \]  \hspace{1cm} (4.10)

where \( s \) is the scale parameter. Note that that this level set function is not not invariant under scaling. This can be counteracted by multiplying with the factor \( \frac{1}{s} \): \( \psi_{\text{scale+rigid}}(x) = \frac{1}{s}\psi(sR(p)x) \). On the other hand this will complicate the expressions, especially under differentiation. As long as scaling is not extreme, our experience is that formula (4.10) works fine.
4.2 Shape deformations

Shape deformations can arise in a number of different ways. For instance we may be interested in finding the pose and shape of a specimen of a known species. Allowable shape deformations in this case must include both the natural variation within the species, and natural variations within each specimen. By natural variations within the specimen, we mean all possible ways the specimen can bend and deform. When doing pose estimation it is important to be able to deal with shapes with such a large amount of freedom. So far kinematic chains have been used to approximate shape deformations to a certain degree. As we mentioned in the introduction, this may work well in the right situation, but for complicated deformations it is extremely clumsy and inelegant.

When the human brain recognizes a shape within a certain ”class”, we do not visualize a reference shape being stretched and bend in a certain way. Instead we have learned how shapes within this class may look like through a number of observations. In our deformation model we will imitate this learning process, by using principal component analysis on a set of ”training shapes”. This deformation model has been used in many mathematical applications. Most closely related to us is probably its use in image segmentation. Including such prior shape knowledge in image segmentation was first proposed by Leventon in [25], and has later been improved by Cremers [22, 24] and Tsai [26].

Let \( \{T_i\}_{i=1}^n \) be a set of aligned training shapes. By aligned, we mean the shapes are rotated, translated and scaled so that they fit each other in the best possible way. We will come back to this in more detail later. Each \( T_i \) is represented by level set functions on a discrete grid, in particular we use signed distance functions. Let \( X, Y, Z \) be the number of grid point in the \( x, y \) and \( z \) direction of our coordinate system respectively. Each \( T_i \) is furthermore rearranged in lexicographic order in a long vector in \( \mathbb{R}^{XYZ} \).

We are interested in the subspace \( T = \text{Span}\{T_i\}_{i=1}^n \) of all shapes. We will create a very nice basis for \( T \), having the property that the first basis vector corresponds to the direction of largest variability in \( T \), the second basis vector the second largest variability in \( T \) etc. To illustrate the method we have created a example in 2d. The situation in 2d is exactly the same, except the level set functions are now functions of two variables. Figure [4.1] shows four selected shapes from a training set of 12. This example will also be used in the experiment section.

Lets first compute the mean shape \( \bar{T} \) as

\[
\bar{T} = \frac{1}{n} \sum_{i=1}^{n} T_i.
\]

We want to somehow extract the shape variabilities from this mean shape. Define
the mean offset functions
\[ \hat{T}_i = T_i - \bar{T} \quad i = 1, \ldots, n \]
and define further the matrix \( M = [\hat{T}_1 \hat{T}_2 \ldots \hat{T}_n] \), having each \( \hat{T}_i \) as columns. We can now do a singular value decomposition on the covariance matrix \( \frac{1}{n} MM^T \)
\[ U \Sigma U^T = \frac{1}{n} MM^T. \] (4.11)
Here \( \Sigma = \text{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\} \), where \( \lambda_1 > \lambda_2 \ldots \lambda_n \) are the singular values arranged in decreasing order. The columns of \( U = [U_1 U_2 \ldots U_n] \) form an orthogonal basis for \( \mathcal{T} \). For each \( i \), \( \sqrt{\lambda_i} \) is by construction the standard deviation in the direction \( U_i \). This means that \( U_1 \) is the direction of largest variation corresponding to the largest singular value \( \lambda_1 \), \( U_2 \) is the direction of second largest shape variation corresponding to the second largest singular value \( \lambda_2 \) etc. Any shape \( \Psi \in \mathcal{T} \) can now be represented as
\[ \Psi = \bar{T} + \sum_{i=1}^{n} \alpha_i U_i \] (4.12)
but because of the decreasing importance of the orthogonal directions, the following approximation to \( \Psi \) can be made
\[ \hat{\Psi} = \bar{T} + \sum_{i=1}^{k} \alpha_i U_i \] (4.13)
where \( k < n \). Choosing \( k = 5 \) or \( 6 \) we are often able to catch the most important variations in \( \mathcal{T} \). Note that \( \hat{\Psi} \) is uniquely defined by the \( k \) scalars \( \{\alpha_i\}_{i=1}^{k} \).

By using the standard deviations \( \sqrt{\lambda_i} \) we can furthermore define a probability function on our subspace of shapes \( \mathcal{T} \). Let \( \Psi \) be any shape in \( \mathcal{T} \) represented as in equation (4.12). Assuming the training shapes are distributed according to a Gaussian probability density, the probability of \( \Psi \) can be computed as
\[ P(\Psi) = \frac{1}{2\pi|\Sigma|} \exp(\alpha^T \Sigma^{-1} \alpha) \] (4.14)
where $\alpha$ is the vector of components $\alpha_i$.

An important question is how to compute the singular value decomposition in equation (4.11). Typically the matrix $\frac{1}{n}MM^T$ is dense with tens of millions of rows and columns. Computing the SVD of such a matrix is extremely impractical. Fortunately there is a trick that reduces the computational efforts to something trivial.

**Theorem 4.1** Let $u$ be an eigenvector of $M^T M$ with corresponding eigenvalue $\lambda$, then $Mu$ is eigenvector of $M M^T$ with corresponding eigenvalue $\lambda$.

**Proof:** From the definition of eigenvector/eigenvalue we have

$$M^T M u = \lambda u.$$ 

Multiplying from the left by $M$, and using associativity of matrix multiplication we get

$$(M M^T)Mu = \lambda Mu.$$ 

We can now compute the SVD of the much smaller $n \times n$ matrix $\frac{1}{n} MM^T$ as $\tilde{U} \Sigma \tilde{U}^T = \frac{1}{n} MM^T$, and use this find $U$ as $U = M \tilde{U}$. Typically $n$ will be a couple of hundreds.

We are finally able to find the level set representation of any allowable shape transformation using the $m + 7$ parameters $(p, s, \alpha)$

$$\psi(sR(p)x; \alpha) = \frac{1}{s} \tilde{T}(sL(p)x) + \frac{1}{s} \sum_{i=1}^{k} \alpha_i U_i(sL(p)x).$$  \hspace{1cm} (4.15)

One question that remains is how to align the training shapes. As we mentioned, each shape needs to be rotated, translated and scaled with respect to each other so that they are the most similar. This needs to be done before the set can be analyzed. Tsai [26] proposed a variational approach to the problem, which we have implemented. We will very briefly explain this method.

Assume now that each shape is represented by a binary level set function. That is, a function that is 1 interior to the surface and 0 exterior to the surface. Denote this level set function $I_i$. Let $p^i$ and $s^i$ be the pose parameters and scale parameters of shape $I_i$. For simplicity we denote $\tilde{I}_i = I_i(s^IL(p^i)x)$. We want to find $(p^i, s^i)$ minimizing the energy function

$$E(p^i, s^i) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \frac{\int_{\Omega} (\tilde{I}_i - \tilde{I}_j)^2 dx}{\int_{\Omega} (\tilde{I}_i + \tilde{I}_j)^2 dx}.$$  \hspace{1cm} (4.16)

In words, we find the rigid transformations that simultaneously minimize the $L^2$ difference between any binary level set functions. Normalization of each term by $\int_{\Omega} (\tilde{I}_i + \tilde{I}_j)^2 dx$ is done to prevent all the shapes to shrink.
CHAPTER 4. 2D-3D POSE ESTIMATION

Figure 4.2: Training shapes from fig (4.1) after alignment

The gradient decent equations for this minimization problem reads

\[
\frac{\partial p_i}{\partial t} = - \sum_{j=1, j \neq i}^{n} \left\{ \frac{2 \int_{\Omega} (\tilde{I}_i - \tilde{I}_j) \nabla_{\rho} \tilde{I}^i dx}{\int_{\Omega} (\tilde{I}_i + \tilde{I}_j)^2 dx} - \frac{2 \int_{\Omega} (\tilde{I}_i - \tilde{I}_j)^2 (\tilde{I}_i + \tilde{I}_j) \nabla_{\rho} \tilde{I}^i dx}{\left( \int_{\Omega} (\tilde{I}_i + \tilde{I}_j)^2 dx \right)^2} \right\}.
\]

(4.17)

4.3 2D-3D surface reconstruction

2D-3D reconstructions deals with the problem of reconstructing 3d information of the real world from a set of 2d-images of the real world. Obviously this is an impossible task because 2d images are of a dimension lower and thus contain much less information. However, we can hope for quite good approximations. The human brain is extremely good at this. Our visual system is purely two dimensional, yet the brain is able to easily transform this information to 3d. When observing an object we make use of several different clues to create a 3d impression of this object. These are mainly

1. the object silhouette
2. lightning reflections and shading on the surface
3. combination of visual impressions from different viewpoints (our two eyes)
4. texture

Many mathematical models have been proposed that tries to mimic the brain in forming 3d information from these clues. We will focus on some recent variational models. In our implementation and experiments we have considered only methods taking the first and third of these clues into account, the so called "shape from contour" methods. However methods using also the second clue, the so called "shape from shading" methods, are a relatively short step forward in the variational framework. In order to explain these methods further, we need some concrete ideas of how images are formed. This is not the focus of the thesis, so we will try to be as brief as possible.
4.3. 2D-3D SURFACE RECONSTRUCTION

4.3.1 Pinhole camera model

Let \( \{C_1, ..., C_m\} \) be the coordinate systems of \( m \) different cameras located in the 3d world. The origin of \( C_i \) will be the focal point of camera \( i \). Each of these coordinate systems may be considered as a rigid transform of a certain reference coordinate system, and thus be complete described by an element in \( SE(3) \). The image domain \( \Omega_i \), corresponding to camera \( i \), is now defined as a subset of the plane \( z = 1 \) in \( C_i \). On these domains we define the actual images \( \{I_1, ..., I_m\} \) observed by the different cameras.

The pinhole camera model is the ideal camera model where the focal point is an infinitesimally small point. In this camera model the relationship between points in 3d and points in the image can be described by the so called perspective projections \( \pi_i \). \( \pi_i \) is a map \( \mathbb{R}^3 \rightarrow \Omega_i \) defined as

\[
\pi_i \left( \begin{array}{c} x \\ y \\ z \end{array} \right) = \left( \begin{array}{c} x/z \\ y/z \\ 1 \end{array} \right)
\]

where \( (x, y, z) \) are the coordinates of an arbitrary 3d point expressed in coordinate system \( C_i \).

4.3.2 A variational method

The motivation for our pose estimation method has been some recent variational approaches to the 2D-3D reconstruction problem. These methods find the recon-
structured surface as the argmin of a certain energy functional acting on surfaces, and are typically solved by the gradient decent method. Yezzi and Soatto proposed "stereoscopic segmentation" [27], that evolves a surface so that its perspective projections better and better fit the 2d images. Interestingly this method will in theory also segment the 2d images, distinguishing object from background. One drawback is the definition of the energy in the image domain, resulting in very local evolutions. This is not ideal for us, since operations such as rotations and translations are very global in nature. We have instead made use of a very similar method proposed by Kolev et. al. [29], that estimates the surface as the most probable given the set of input images.

Let $V$ be a subset of $\mathbb{R}^3$ with the property that $\pi_i V \subset \Omega_i$ for all $i$. We want to find

$$\hat{S} = \arg \max_{S \in \Xi} P(S|\{I_1, \ldots, I_m\}).$$

(4.18)

$\Xi$ is the set of all closed surfaces lying in $V$. By using the Bayes rule this probability can be expressed as

$$P(S|\{I_1, \ldots, I_m\}) \propto P(\{I_1, \ldots, I_m\}|S)P(S).$$

(4.19)

We now discretize in space and make the assumption that voxels (discrete 3d points) are independent. This leads to

$$P(S|\{I_1, \ldots, I_m\}) \propto \left[ \prod_{x \in V} P(\{I_k(\pi_k(x))\}_{k=1, \ldots, m}|S) \right] dx P(S)$$

(4.20)

where $dx$ is the discretization step. For a specific surface $S$, the volume $V$ may be partitioned in two, $V = V_{obj}^S \cup V_{bck}^S$. $V_{obj}^S$ consists of $S$ and its interior points, $V_{bck}^S$ of the points lying on the background. Using this partition we may express the probability as

$$P(S|\{I_1, \ldots, I_m\}) \propto \left[ \prod_{x \in V_{obj}^S} P_{obj}(x) \right] dx \left[ \prod_{x \in V_{bck}^S} P_{bck}(x) \right] dx P(S)$$

(4.21)

where we have made the simplification of notation

$$P_{obj}(x) = P(\{I_k(\pi_k(x))\}_{k=1, \ldots, m}|x \in V_{obj}^S)$$

(4.22)

$$P_{bck}(x) = P(\{I_k(\pi_k(x))\}_{k=1, \ldots, m}|x \in V_{bck}^S).$$

(4.23)

If we could make an assumption about independence between image observations, we could make the expression even more concrete. In this case we would have

$$P_{obj}(x) = \prod_{k=1}^m P(I_k(\pi_k(x))|x \in V_{obj}^S)$$

(4.24)
4.3. 2D-3D SURFACE RECONSTRUCTION

\[ P_{\text{bck}}(x) = 1 - \prod_{k=1}^{m} [1 - P(I_k(\pi_k(x)) | x \in V^S_{\text{bck}})]. \]  

(4.25)

However this would lead to a bias towards the probabilities 0 and 1. Instead we use the geometric mean of the probabilities of each observation

\[ P_{\text{obj}}(x) = \sqrt[m]{\prod_{k=1}^{m} P(I_k(\pi_k(x)) | x \in V^S_{\text{obj}})} \]  

(4.26)

\[ P_{\text{bck}}(x) = 1 - \sqrt[m]{1 - \prod_{k=1}^{m} P(I_k(\pi_k(x)) | x \in V^S_{\text{bck}})}. \]  

(4.27)

\( P(I_k(\pi_k(x)) | x \in V^S_{\text{obj}}) \) is the probability of observing intensity \( I_k(\pi_k(x)) \), given that \( x \) is interior to, or on the surface. Similar for \( P(I_k(\pi_k(x)) | x \in V^S_{\text{bck}}) \). These probability distributions can be modeled by a Gaussian densities

\[ P(I_i(\pi_i(x)) | x \in V^S_{\text{obj}}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{obj}}} \exp \left( -\frac{(I_i(\pi_i(x)) - \mu_{\text{obj}})^2}{2\sigma^2_{\text{obj}}} \right) \]  

(4.28)

\[ P(I_i(\pi_i(x)) | x \in V^S_{\text{bck}}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{bck}}} \exp \left( -\frac{(I_i(\pi_i(x)) - \mu_{\text{bck}})^2}{2\sigma^2_{\text{bck}}} \right). \]  

(4.29)

The means and variances (\( \mu \) and \( \sigma \)) are updated during the surface evolution, by projecting the current surface down to the image planes, and then computing the mean and variance of gray levels in the corresponding image partitions.

We finally need to specify \( P(S) \), the surface probability distribution. We are interested in smooth, connected surfaces. We thus consider surfaces of large surface area less likely. This can be achieved by letting

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

(4.30)

where \( |S| \) is a notation for surface area. This maximum a-posteriori estimation can be converted to an energy minimization problem by taking the negative logarithm. In the continuum limit this gives us the following energy

\[ E(S) = - \int_{V^S_{\text{obj}}} \log P_{\text{obj}}(x) \, dx - \int_{V^S_{\text{bck}}} \log P^S_{\text{bck}}(x) \, dx + \nu |S|. \]  

(4.31)

The integrand can of course also be written out as

\[ \log P_{\text{obj}}(x) = - \frac{\log(\sqrt{2\pi}\sigma_{\text{obj}})}{m} - \frac{\sum_{k=1}^{m} (I_k(\pi_k(x)) - \mu_{\text{obj}})^2}{2\sigma^2_{\text{obj}} m} \]  

(4.32)
and similarly for \( P_{\text{bck}} \). We want to solve
\[
\hat{S} = \arg\min_{S \in \Xi} E(S).
\] (4.33)

In order to do this, we start by formulating (4.31) using level set functions
\[
E(\phi) = -\int_V [H(\phi) \log P_{\text{bck}}(x) + (1 - H(\phi)) \log P_{\text{obj}}(x)] \, dx + \nu \int_R |\nabla H(\phi)| \, dx.
\] (4.34)

\( H \) is the heavyside function defined in Chapter 2.4, and \( \nu \) is some parameter. In order to use this functional for a visually acceptable surface reconstruction, as many as 16-20 view are necessary. When used for pose estimation, we are only interested in using 2-3 views. In these cases of few input images, it turns out the model favors 3D objects of large volume. We have therefore included a term penalizing large volumes as follows
\[
E(\phi) = -\int_V [H(\phi) \log P_{\text{bck}}(x) + (1 - H(\phi)) \log P_{\text{obj}}(x)] \, dx + \sigma \int_V H(\phi) \, dx + \nu \int_R |\nabla H(\phi)| \, dx
\] (4.35)

for some parameter \( \sigma \). Note that this functional has exactly the same form as the example functional in Chapter 2.4. The first two terms constitutes the internal energy \( f \). By using a similar technique as in the example we can compute the Euler-Lagrange equation of \( E \)
\[
\frac{\partial E}{\partial \phi} = -\delta(\phi(x)) [\log P_{\text{bck}}(x) - \log P_{\text{obj}}(x) - \sigma] - \nu \delta(\phi(x)) \nabla \cdot \left( \frac{\nabla \phi(x)}{|\nabla \phi(x)|} \right).
\] (4.36)

This is solved by the gradient descent method in the standard way by looking for steady state of the evolution
\[
\frac{\partial \phi}{\partial t} = \delta(\phi(x)) [\log P_{\text{bck}}(x) - \log P_{\text{obj}}(x) - \sigma] + \nu \delta(\phi(x)) \nabla \cdot \left( \frac{\nabla \phi(x)}{|\nabla \phi(x)|} \right).
\] (4.37)

### 4.4 Our pose estimation method

We are now finally able to present our pose estimation method. As we have talked about, this consists of solving (4.35) under the constraint that \( \phi \) must belong to scaled, rigid transformed elements of \( T \). As usual in optimization there are two ways to integrate such constraints, either by adding a penalty term in the objective function, or as a more direct integration. As a first attempt we tried adding a forcing term to (4.35) penalizing deviations from the allowable shapes in the sense of \( L^2 \) distance between level set functions.
\[
E_{\text{penalty}}(\phi, p, s, \alpha) = \int_V (\phi(x) - \psi(sR(p)x; \alpha))^2 \, dx
\] (4.38)
where $\psi(sR(p)x;\alpha)$ is defined in (4.15). Our pose estimation energy is thus simply

$$E_{\text{pose}}(\phi,p,s,\alpha) = E(\phi) + \gamma E_{\text{penalty}}(\phi,p,s,\alpha)$$  \hspace{1cm} (4.39)

for some parameter $\gamma$. $E_{\text{pose}}$ is a functional depending on level set functions $\phi$ and the $m + 7$ pose parameters $(p, s, \alpha)$. Minimization can be interpreted as simultaneously drawing $\phi$ toward the subspace $T$, and drawing the pose parameters to the element in $T$ best approximating $\phi$ in the $L^2$ norm.

Minimization is achieved by solving the gradient decent equations for all unknowns. We get the system of equations

$$\frac{\partial \phi(x)}{\partial t} = -\frac{\partial E}{\partial \phi}(x) - 2\gamma[\phi(x) - \psi(sR(p)x;\alpha)]$$  \hspace{1cm} (4.40)

$$\frac{\partial p_i}{\partial t} = -2\gamma s \int_V [\phi(x) - \psi(sR(p)x;\alpha)] \nabla \psi \cdot \frac{\partial R}{\partial p_i} x \, dx \quad i = 1, \ldots, 6$$  \hspace{1cm} (4.41)

$$\frac{\partial \alpha_i}{\partial t} = -2\gamma \int_V [\phi(x) - \psi(sR(p)x;\alpha)] U_i(sR(p)x) \quad i = 1, \ldots, m$$  \hspace{1cm} (4.42)

$$\frac{\partial s}{\partial t} = -2\gamma \int_V [\phi(x) - \psi(sR(p)x;\alpha)] \nabla \psi \cdot Rx \, dx.$$  \hspace{1cm} (4.43)

These equations are discretized by the finite difference method in a standard way. We have used forward Euler scheme for the time discretization.

However, after doing some experiments it seems these equations have a lot of difficulty converging. They often get stuck in local minima, and we need very small time steps in order to get a stable evolution.

We have instead used a different strategy that is a lot more efficient. Let’s have a look at (4.35) once again. This minimization problem can be transformed to a constrained minimization problem by simply replacing $\phi$ with $\psi(sR(p)x,\alpha)$

$$E(p,s,\alpha) = \int_V [H(\psi(sR(p)x,\alpha))\log P_{\text{bck}}(x) + (1-H(\psi(sR(p)x,\alpha)))\log P_{\text{obj}}(x) + \sigma] \, dx.$$  \hspace{1cm} (4.44)

Note that we have omitted the surface regularization term, since the shape subspace automatically excludes uninteresting noisy and disconnected surfaces. By going back to formula (4.21), the shape probability $P(S)$ can instead be replaced by the probability distribution on the subspace $T$ defined in equation (4.14). This leads to the following energy

$$E(p,s,\alpha) = \int_V [H(\psi(sR(p)x,\alpha))\log P_{\text{bck}}(x)$$
for some parameter $\lambda$. This is in fact an energy function over $\mathbb{R}^{m+7}$. The gradient descent equations are in this case just a system of ODEs, which can be derived using the chain rule and product rule.

\[
\frac{\partial p_i}{\partial t} = s \int_V \delta(\psi(sR(p)x; \alpha))[\log P_{obj}(x) - \log P_{bck}(x) - \sigma] \nabla \psi \cdot \frac{\partial R}{\partial p_i} x \, dx \\
\quad i = 1, \ldots, 6
\]  

(4.46)

\[
\frac{\partial \alpha_i}{\partial t} = \int_V \delta(\psi(sR(p)x; \alpha))[\log P_{obj}(x) - \log P_{bck}(x) - \sigma] U_i \, dx \\
\quad + \frac{\lambda}{\pi |\Sigma|} (\Sigma^{-1} \alpha)_i \\
\quad i = 1, \ldots, m
\]  

(4.47)

\[
\frac{\partial s}{\partial t} = \int_V \delta(\psi(sR(p)x; \alpha))[\log P_{obj}(x) - \log P_{bck}(x) - \sigma] \nabla \psi \cdot Rx \, dx.
\]  

(4.48)

These equations can be discretized with simple finite differences. We have used explicit Euler time discretization.

In this approach the shape is completely constrained to subspace $\mathcal{T}$, this may be considered a drawback. By using a penalty term as in (4.39), we allow slight deviations from $\mathcal{T}$ which is more realistic since we are never able to catch all possible variations in 100% detail from a finite training set. However, the errors in surface energy are usually much more serious than limitations of the training subspace, so deviations from the training set could as well be due to errors in the reconstruction model. This is particularly true when only a small number of views are available. In case of many views the first approach may be of greater interest, since the surface estimation is expected to be quite good.

### 4.4.1 Experiments

We will do various experiments on estimating pose parameters. However we have not been able to create a suitable experiment where we use the deformation model in 3D. For this we need a large number of 3D training shapes, which require a lot of time and resources to acquire. The time constraints for a master thesis is very limited, and I chose not to spend a lot of time on this. Instead we have used the deformation model in a 2d experiment. The equivalence in 2d is to find pose and shape from a set of 1d signals. We will come back to this in a moment. First we show how the method performs for traditional pose estimation in 3d. This is a special case of our method where we omit the shape variables $\alpha$, and scale variables $s$. Since the object is not allowed to change size, the volume parameter $\sigma$ can be set to zero.
4.4. OUR POSE ESTIMATION METHOD

Figure 4.4: Perspective projections of Armadillo from 3 different viewpoints.

Table 4.1: Estimated pose parameters, angles are in radians.

<table>
<thead>
<tr>
<th>p</th>
<th>True</th>
<th>2 views</th>
<th>3 views</th>
<th>occlusion</th>
<th>initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>ω</td>
<td>0</td>
<td>-0.0058</td>
<td>0.0030</td>
<td>0.0091</td>
<td>0.6</td>
</tr>
<tr>
<td>β</td>
<td>0</td>
<td>-0.00079</td>
<td>0.0021</td>
<td>-0.0217</td>
<td>0.9</td>
</tr>
<tr>
<td>γ</td>
<td>0</td>
<td>-0.0148</td>
<td>-0.0119</td>
<td>-0.0460</td>
<td>0.7</td>
</tr>
<tr>
<td>t₁</td>
<td>50</td>
<td>50.4849</td>
<td>50.4964</td>
<td>50.5084</td>
<td>55</td>
</tr>
<tr>
<td>t₂</td>
<td>50</td>
<td>50.3350</td>
<td>50.4674</td>
<td>50.6709</td>
<td>55</td>
</tr>
<tr>
<td>t₃</td>
<td>50</td>
<td>51.0725</td>
<td>50.5468</td>
<td>51.1859</td>
<td>45</td>
</tr>
</tbody>
</table>

We invite Armadillo from Chapter 3 for our first experiment. Figure 4.4 shows 2d images of Armadillo artificially shot from 3 viewpoints with added Gaussian noise. Figure 4.5 shows the results of estimating pose from the first 2 views, all 3 views as well as some other experiments we will talk about in a moment. The exact numbers for the estimations are summarized in table 4.1.

What can we conclude from this? From two and three views we are able estimate the pose with good accuracy from a decent initialization. Especially the rotation parameters are very accurate. However, if the initialization is far from ground truth, the evolution gets trapped in local minima and fails to converge to the right solution. Brox’ method also failed to converge for bad initializations, however it is easier for us to explain the phenomena in a mathematical way. There are also a number of strategies to avoid convergence to local minima for the gradient descent method. One of these being multiresolution. The idea is to solve the equations on a coarse space discretization, and successively use the solution as initialization on finer and finer space discretizations. Coarser space discretizations will act by smoothing out the energy function, and thus hopefully eliminating some of the local minima. Another advantage of this is a much faster
Figure 4.5: Estimations using input images from figure 4.4.
(a) Ground Truth, (b) initialization, (c) estimation from 3 views, (d) estimation from the first two views only, (e) reconstruction from 3 views without constraints, (f) reconstruction from 2 views without constraints, (g) bad initialization (h) steady state from bad initialization, (i) estimation from the two occluded images in figure 4.6.)
4.4. **OUR POSE ESTIMATION METHOD**

algorithm. We have not implemented this however.

Note that the translation estimation is a bit inaccurate. A possible explanation for this, is the low resolution on the artificially created 2d images. This leads to big roundoff errors when projecting the surface down to these images. In practice we would want a lot higher resolution on the images, than the surface ambient space (in our case they are the same). We believe this will improve the translation estimation significantly.

Increasing the number of views beyond 2 does not seem to make a large difference. Note that for one single view the method does not work, simply because it is not able to estimate depth good enough. This is usually possible for existing methods, if the input image has a good quality. Our aim is not to compete with these methods. As we have mentioned in the introduction, we will study situations where existing methods fails, or have a very limited performance. The next example shows such a situation, see figure (4.6). In addition to strong Gaussian noise, both images contain occlusions, so that only part of the object is visible. Figure (4.5) shows the estimated pose, see table (4.4.1) for the exact numbers. As we can see, the estimations are still good, although slightly more inaccurate than the previous example. In figure (4.6) the white line represents the silhouette of the estimated object projections. This shows the very interesting dual problem. We are able to correctly segment the 2d images even in case of these strong occlusions, by using all views of the 3D shape as a priori information. In situations where deformations and/or occlusions are allowed, the pose estimation problem is generally ill-posed from one view. It does not have a unique solution. Existing methods may need up to 4 views or more when deformations are introduced through kinematic chains. Our method has been developed with deformations in mind from the very beginning, thus its one view performance is not a big issue for us. Unfortunately, we can not demonstrate the deformation model in 3d for the reasons stated above. We have instead created a 2d example, where we have used the fish training set from Section 4.2. In 2d, the matrix $R$ is a member of $SE(2)$. Our aim is to reconstruct $R$ and shape parameters $\alpha$ from a set of 1d perspective projections from different viewpoints. The perspective projection operator of "camera" $i$ is now simply defined as

$$
\pi_i \left( \begin{array}{c} x \\
 y 
\end{array} \right) = \left( \begin{array}{c} x/y \\
 1 
\end{array} \right)
$$

for $x, y$ expressed in the 2d coordinate system $C_i$.

Figure (4.7) shows a fish shape not belonging to the training shapes, and its 1D projections from 4 viewpoints. From these projections we wish to estimate pose and shape. Figure (4.8) shows ground truth, estimated pose and shape, initialization and finally overlap between estimation (green) and ground truth (red).

Bear in mind that reconstruction from 1d to 2d is more difficult than 2d to 3d because relatively a lot more information gets lost. From a single 1d signal it
is impossible to say anything about the shape of the object projecting it, which is usually possible for a 2D image. These results in 2D should thus be a good motivation for testing 3D deformations in the future.

4.5 Conclusions

A new variational method was presented for the pose estimation problem. The idea was to solve traditional 2D-3D surface reconstruction problem under certain constraints. Experiments showed that the method is robust to noise and low quality input images with the image object partially occluded. When no deformations were involved, the method gave good results from 2 views. The method could also be treated from a different perspective, namely that of segmenting 2D images with a priori shape information about the 3D object projecting it.

Because of its strong connection to 2D-3D surface reconstruction, object deformations could be integrated in the model in a very natural way. We incorporated deformations through principal component analysis on a set of training shapes, an idea that is borrowed from recent advances in image segmentation. Experiments in 2D showed that the method is thus able to handle complex object deformations, however a larger number of views were necessary. The situation of
4.5. CONCLUSIONS

Fig. 4.7: Left: Ground truth shape and pose. Right: Perspective projections onto 4 1d images

Fig. 4.8: From left to right: (a) Ground truth (b) estimated pose and shape from 4 views (c) initialization (d) red: ground truth, green: estimation.

Going from 2d to 3d is probably a little easier than 1D to 2D, but it remains to see how this turns out.

No direct comparisons with other methods were made. Our aim were not to introduce a new competitor in the ideal pose estimation scenario, but to develop a method that could handle situations where current methods have very limited success. These include complex 3d deformations that are not easily modelled by kinematic chains, and more realistic input images where the image object blends in with, or is occluded by the background.

Further work would be testing of the deformation model in 3D, and testing with real images. A comparison should be made with Brox’ method [23], although he is focusing more on the segmentation part than the pose estimation part. Some other pose estimation methods that aims to handle more realistic input images, are currently under development, but have not been published yet. Comparison with these should also be made.

The following are the main drawbacks of our method:

1) while existing methods converges in a couple of seconds, our method generally takes a couple of minutes. We believe this can be speeded up by

- using narrow band for level set functions
- multiresolution

2) In case of no deformations and good input images, most existing methods are able to estimate pose from one view. The minimum number of views for our method to converge is two.

3) If the pose is initialized far from the correct state, the method tends to get trapped in local minima. However, good initialization is also a requirement for most other methods.
Chapter 5

Summary and outlook

5.1 Summary and contributions

Summary

In this thesis we have proposed new methods for two different surface reconstruction problems. For both of these methods, our aim have just been to prove the concept. More experiments need to be done, and comparison with other methods have to be made in the future.

The first problems we considered were compression of surface geometry, and reconstruction of surface geometry from point clouds. The numerical surface representation we used, was triangulated surfaces. Inspired from recent advances in image compression, we proposed a new method for solving these problems using partial differential equations. In the implementation and experiments our focus were on a linear geometric diffusion equation acting on the surface geometry. This equation was discretized by the finite element method directly on the surface. When used alone this equation led to undesirable singularities and volume shrinkage in the reconstructed surface. However we proposed to use this equation as basis for a new approximation method, by alternating the role of inpainting domain and boundary domain. When combined with a suitable inverse evolution step, this improved results significantly.

We further brought this geometric idea back images, where it was shown to greatly improve reconstruction quality in PDE-based image compression compared to linear diffusion. We tried to interpret the method as a discretization method of an underlying PDE, because of its strong connections to domain decomposition and Hopscotch iteration.

Our conclusion is that more research should absolutely be put into this surface compression method. Based on the experiments, we believe this has good potential for becoming competitive to other methods. In the "future research" section we will point out some of the open problems, and suggest some further developments.
The second problem we considered were 2D-3D pose estimation in Chapter 4. We proposed a variational method for pose estimation that is designed to handle difficult input images, and capture complex 3D deformations. Experiments showed that the method is robust to noise and situations where the image object is partially occluded. Since the entire model was formulated in a single energy functional, it was much easier to analyze than other methods.

The main ingredient was a variational method for reconstructing surface from a collection of 2D images. We transformed this into a pose estimation method by introducing strong constraints on the surface, forcing it to belong to rigidly transformed elements of a certain deformation subspace.

Although the method gave good results, we were a bit concerned about the computational complexity involved. It remains to see if its advantages are enough compensation. This should be made clear by creating suitable experiments with 3D deformations.

For more detailed summaries, see the "Conclusions"-section at the end of each chapter.

Contributions

This section will once and for all distinguish my own scientific contributions in this work from what has been done before. Note that the words "our" and "we" are used throughout thesis with the meaning "me with support from my supervisors".

- PDE-based image interpolation and compression as discussed in Section 3.1 is method developed by Weickert, Irena et. al. [14], my own contribution in this section has been the image approximation method in Subsection 3.1.4 and the attempts to analyze this.

- In Section 3.2 my contribution has been the actual PDE-based methods for reconstructing surface from point clouds, and the method for compressing surface geometry. The finite element method for discretizing linear geometric diffusion has been proposed by Dzuik [30], but implemented from scratch by me in C++.

- In Chapter 4, my contribution has been the actual new variational pose estimation method. However, this is based on an established method for 2D-3D surface reconstruction, presented in Section 4.3.2. The deformation model, presented in 4.2, has earlier been used for image segmentation with a priori shape information.

Implementation:
5.2. **FUTURE RESEARCH**

- All methods used for experiments in this thesis, have been implemented from scratch by me with the following exceptions: *linear and anisotropic diffusion for image compression, the coding algorithm for image compression.*

  The methods in Chapter 3 have been implemented in C++, and the methods in Chapter 4 have been implemented in matlab.

## 5.2 Future research

After finishing this thesis there are still some open questions, and room for many further developments. We would especially like to point out 4.

Let's start with chapter 3: compression and interpolation.

- We have in this thesis only considered linear geometric diffusion for geometry compression. This works very well when used as basis for the approximation method. In chapter 3.2.7 we discussed some other differential operators that should be tested. Of the ones we mentioned, we have most belief in anisotropic geometric diffusion. Anisotropic diffusion should also be used in combination with the approximation method, because of its shrinking effects.

- It is not clear how to solve to minimization problem in the compression step efficiently. We have a feeling a solution close to the minimizer may be obtained by thresholding the magnitude of the Laplace-Beltrami operator applied to the surface. This must either be proved mathematically or tested through numerical experiments. Another option may be B-Tree triangular coding as proposed by Distasi et. al [33] for images. This approach would fit very nicely with triangulated surfaces.

- We do not completely understand the approximation method mathematically. When something improves results significantly, attempts should be made to understand why. We have tried to regard the method as a discretization method of an unknown underlying PDE. Since it has so strong connections with domain decomposition and Hopscotch iteration, we believe the deep theory of these could be used to give some answers. Future research would be to investigate these connections further. Our hope is to find bounds for the actual reconstruction errors. This idea of connecting function approximation with PDE discretization methods should also be of interest from a wider perspective than just image processing.

and Chapter 4: Pose estimation
The pose estimation method we presented is contour based. A very interesting future work would be to generalize this to a method taking shading information into account. This can be done very elegantly in the variational framework, by simply replacing the underlying energy functional with a shape from shading functional (see [31], for an example). Lightning reflections and uneven shading could thus be transformed from drawbacks to advantages when estimating 3D pose. To current date we are not aware of any "pose-from-shading" methods.
Appendix A

Hopscotch iteration

Hopscotch iteration is a numerical method for solving time dependent partial differential equations of the form

\[
\frac{\partial u}{\partial t} = Lu + f(x, y, t) \quad \Omega \times [0, T)
\]  \hspace{1cm} (A.1)

where \( L \) is some second order linear differential operator, and \( \Omega \) is some domain in \( \mathbb{R}^2 \). Let \( \Omega^h \) denote a square grid discretization of \( \Omega \) with mesh spacing \( k \), and \( t_m \) a time discretization of mesh spacing \( \tau \). Let further \( L_h : \Omega^h \rightarrow \Omega^h \) be a discretization of \( L \) on \( \Omega^h \). \( U^n \) will denote the approximate solution of \( u(ki, kj, n\tau) \) at \((ki, kj, n\tau)\). The idea behind Hopscotch iteration is to split the domain \( \Omega^h \) in two, \( \Omega^h_1 \) and \( \Omega^h_2 \), and then iteratively take one timestep in each domain at a time with the solution in the other domain held fixed. The time discretization is explicit in \( \Omega_1 \) and implicit in \( \Omega_2 \).

\[
U^{n+1}_{\Omega_1^h} = U^n_{\Omega_1^h} + \tau (L_h U^n_{\Omega_1^h} + g^n_{\Omega_1^h}) \quad \text{in} \quad \Omega_1^h
\]  \hspace{1cm} (A.2)

\[
U^{n+1}_{\Omega_2^h} = U^n_{\Omega_2^h} \quad \text{on} \quad \partial\Omega_1^h
\]  \hspace{1cm} (A.3)

\[
U^{n+1}_{\Omega_2^h} = U^n_{\Omega_2^h} + \tau (L_h U^{n+1}_{\Omega_2^h} + g^{n+1}_{\Omega_2^h}) \quad \text{in} \quad \Omega_2^h
\]  \hspace{1cm} (A.4)

\[
U^{n+1}_{\Omega_2^h} = U^{n+1}_{\Omega_1^h} \quad \text{on} \quad \partial\Omega_2^h
\]  \hspace{1cm} (A.5)

The actual decomposition of the domain were originally proposed by Gordon as follows (see \[18, 20\]):

\[
(x, y) = (ki, kj) \in \left\{ \begin{array}{ll}
\Omega_1^h, & \text{if } i + j \text{ is even} \\
\Omega_2^h, & \text{if } i + j \text{ is odd}
\end{array} \right.
\]

Later, more general decompositions were considered, see \[19\] for a summary. What they have in common is decompositions into rather scattered domains. For more details, analysis of convergence, numerical error etc., see the mentioned references \[20, 18, 19, 21\].
Bibliography


[22] D. Cremers Statistical shape knowledge in variational image segmentation Department of Mathematics and Computer Science, University of Mannheim, Germany, 2002


