Implicit Surfaces For Modelling Human Heads

Florian Steinke

Acknowledgements

The supervision of this diploma thesis by Prof. H. Ruder and Prof. B. Schölkopf is gratefully acknowledged.

Prof. H. Ruder opened up the possibility for me to discover the inspiring field of machine learning during my diploma thesis and supported my work in the physics department.

Prof. B. Schölkopf gave me the unique opportunity to learn so much during this year. I was invited to the machine learning summer school in Berder, France, which provided me with a quick and tremendously helpful introduction into the field of machine learning. Later on he continuously showed interest in my work and also proposed lots of ideas himself. The discussions with him have shaped the work presented here. He has also put great effort into publishing our results in the community. This has so far lead to two publications at internationally renowned conferences - ICML and EUROGRAPHICS [Schölkopf et al., 2005; Steinke et al., 2005]. I would also like to thank for the opportunity to attend these conferences which gave me a deeper insight into the topics of modern computer graphics and machine learning.

In this context, I would also like to thank Prof. Volker Blanz, who has contributed his profound knowledge of the graphics world. He has directed our focus into the promising as well as challenging direction of correspondence computations. Especially during the time before the deadline of EUROGRAPHICS, he put a great amount of personal effort and time into our work.

The atmosphere at the department of Empirical Inference at the Max-Planck-Institute for Biological Cybernetics was highly inspiring: Many co-workers and guests have given stimulating talks and were always open for questions and discussions. I want to acknowledge especially Christian Walder. During my numerous discussions with him about this thesis which is closely related to his work, I was able to gain many important insights. I also would like to especially thank my roommates Jan Eichhorn and Dr. Timothy Davison who always tried to help me with any technical or scientific problem that occurred.

Contents

Chapter 1

Introduction

To observe the world and to derive general rules from it, that lies at the heart of physics. To build general rules from a number of observations and to do so automatically, that lies at the heart of machine learning.

In recent years, the power of electronic information processing devices has increased dramatically. Simple computations can now often be done more or less instantaneously. But automatic inference and learning which seems so easy to do for humans is still a big problem for computers.

However, even this problem has started to shrink lately. The rise of kernel machines has provided the world with a powerful set of algorithms that perform very well in many real world applications and that at the same time can be used more or less as black-box machines. Support Vector Machines (SVM) and Support Vector Regression (SVR) now build the backbone of many learning machines in as different areas as medical diagnosis, remote sensing for satellites or financial time-series prediction.

This work will shortly review the theory behind kernel machines for regression tasks. Many of the tools on which this theory builds upon are well-known to physicists. It will include linear regularization operators, Hilbert spaces and Green's functions.

In the main part we will then focus on trying to apply these methods into an area that so far has seen few machine learning applications: computer graphics. We will demonstrate how noisy data acquisition from 3D scanners can be processed. A method that generalises discrete point measurements of a surface into a general smooth concept will be presented. Additionally, we will examine ways to improve the quality of the measurements by including a kind of model knowledge.

At last, we attack a problem that is closely related to acquiring general model knowledge: computing correspondences. They are the key step to build a very powerful class of models for real world objects - namely linear models. It has been shown that linear models can successfully be applied to solve recognition and synthesis tasks. However, the needed correspondence information is still hard to compute.

We thus hope to be able to contribute something new to the graphics community and also to provide a tool that will in future help to better model and understand the world in electronic computing devices.

Chapter 2

Introduction to Support Vector Regression

Support Vector Regression (SVR) is a non-parametric regression method. It generates a functional model from a finite number of given input-output examples. The presented introduction is based on the books [Schölkopf and Smola, 2002] and [Wahba, 1990].

Throughout this chapter we will use the Dirac notation that is so common-place in quantum mechanics. It is not typical in the machine learning literature, but we think it can ease the description. We will often work with Hilbert spaces and linear operators – a field in which the bra-ket notation is especially useful.

We denote the input domain as $\mathcal{X}$. In this work we typically assume $\mathcal{X} \subseteq \mathbb{R}^D$. $|f\rangle$ will denote a function in $L_2(\mathcal{X})$, i.e. a function $f : \mathcal{X} \to \mathbb{R}$. $\langle f|$ will denote the dual element of $|f\rangle$. Thus, $\langle .|. \rangle$ is the usual L_2 dot-product and $\|.\|$ the corresponding norm.

2.1 Introduction to non-parametric regression

Given N *training samples* $\{(x_i, y_i)\}_{i=1..m} \subseteq \mathcal{X} \times \mathbb{R}$, the regression task is to predict a value $y \in \mathbb{R}$ given some unseen $x \in \mathcal{X}$. One tries to find a suitable model function $|f\rangle$ and then predicts $y = \langle x|f\rangle$. In non-parametric regression we do not assume a specific (finite dimensional) model for $|f\rangle$ but consider it to be part of a rather general class of functions. We will write $\mathcal{H}$ for an appropriate function space. We assume that $\mathcal{H} \subseteq L_2(\mathcal{X})$ such that the dot-product $\langle .|. \rangle$ is also defined on $\mathcal{H}$.

A suitable model $|f\rangle \in \mathcal{H}$ should at least predict the correct values on the training points. If we have noisy data it may be enough if the $\langle x_i|f\rangle$ are close to the given training values y_i and not exactly equal. Thus, if we denote some *loss function* $\text{Loss} : (\mathcal{X} \times \mathbb{R} \times \mathbb{R})^m \to \mathbb{R}$ by $\text{Loss}\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\ldots,m}\right)$, it should be minimised over all functions in $\mathcal{H}$ yielding a small *training error*. Typical examples of a loss function are the quadratic loss $\frac{1}{m}\sum_i \left(\langle x_i|f\rangle - y_i\right)^2$ or a one-norm loss like $\frac{1}{m}\sum_i |\langle x_i|f\rangle - y_i|$.

If $\mathcal{H}$ is a rather general function class, e.g. $C^1(\mathcal{X})$ that is dense in $L_2(\mathcal{X})$, the problem of determining $|f\rangle$ just from the training set is ill-posed. The minimisation of the loss function has infinitely many solutions because the behaviour of functions between the points x_i is not restricted. Some of these functions may be extremely unstable between two points x_i which results in bad *generalization properties*: Given an unseen data point x the expectation of the

test error $E(\langle x|f\rangle - y)$ will be large. Statistical learning theory shows that the expected test error can be bounded if we do not choose our function arbitrarily from the minimisers of the loss function, but if we select the "smoothest" one. Smoothness being a relative term we will assume that it can be measured through a *regularization* functional $\|Pf\|$ where P is a linear *regularization operator*, e.g. $P = \partial_x$. Our objective is then to jointly minimize

$$\|Pf\|^2 + C \text{ Loss}\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\dots,m}\right) \tag{2.1}$$

over all functions in $\mathcal{H}$.

The *regularization parameter* C determines the relative weights of the two terms: For a small C the regularization term is dominating and the minimisation focuses on finding a smooth solution, almost neglecting the data term. For large C, the data is fitted very accurately but the resulting function may be less smooth. The choice of C therefore depends on how much we trust our training values. A standard procedure to chose C is *cross validation*: We estimate the regression function just from a part of the training data points and then test it on the rest. If this step is done several times with different splittings of the training set, the *empirical test error* is a measure on the expected test error for new unseen points x. We finally try to minimise the generalisation error through an appropriate choice of the regularization parameter C.

2.2 Representer theorem

Most interesting function spaces $\mathcal{H}$ are infinite dimensional, e.g. $C^1(\mathcal{X})$. So we cannot directly compute the minimizer $|f^*\rangle$ of (2.1). The following theorem will help us reduce the problem to a finite dimensional one, which can be solved efficiently.

Theorem 1 (Representer Theorem)**.** *The following two functions are identical:*

i)

$$|f^*\rangle = \underset{|f\rangle\in\mathcal{H}}{\text{argmin}} \|Pf\|^2 + C \; Loss\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\dots,m}\right) \tag{2.2}$$

ii)

$$|f^*\rangle = \sum_{i=1}^{m} \alpha_i^* \, |k_{x_i}\rangle \tag{2.3}$$

where

$$\alpha^* = \underset{\alpha\in\mathbb{R}^m,\, |f\rangle=\sum_{i=1}^m \alpha_i \left|k_{x_i}\right\rangle}{\text{argmin}} \alpha^T K \alpha + C \; Loss\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\dots,m}\right) \tag{2.4}$$

K is the kernel matrix with elements $K_{i,j} = \left\langle k_{x_i}\middle|k_{x_j}\right\rangle$.

Here, the kernel $k : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ with $k(x,y) = \langle x|k_y\rangle$ is a function that depends on both P and $\mathcal{H}$ if those are given. Conversely, if $k(x,y)$ is given, then P and $\mathcal{H}$ depend on it.

Note: If $\mathcal{H} \not\subseteq L_2(\mathcal{X})$ or $\mathcal{X} \notin \mathbb{R}^D$ the first expression does not make sense. However, one can still use the second expression to compute a model function $|f\rangle$. We only need that the kernel matrix K is always positive semi-definite. That is for example the case when one works with SVR or SVMs on graphs or texts.

Another note: The objective function in i) implies that we are approximately trying to find a function $|f\rangle$ satisfying the equation $P|f\rangle = 0$ with the boundary conditions given by the data. One could therefore also use the equivalent objective function (2.4) to solve linear differential equations in noisy environments [Wahba, 1977].

One more note: There are many methods to select the best kernel or the best kernel parameters for a regression or classification problem like (2.4) (termed "model selection"). These methods are equivalent to selecting the best regularization operators as implied through the above theorem. According to the last note, the first objective (2.2) is itself equivalent to solving a linear differential equation involving the thus found operator. So in total, model selection algorithms try to find the optimal form of a differential equation, a solution of which may have produced the data. For more details see [Steinke and Schölkopf, 2006] where this idea is exploited to solve a system identification task.

Proof. We will only sketch a proof here, but we will point at some of the mathematical problems that arise in the presented argumentation. The comments are marked with (Note ...) and refer to the list in the next section.

We think that this informal derivation still gives lots of insight into the relation between smoothing operators P and kernels k. It can also be used to compute some of the well known kernels as well as new ones.

i) $\rightarrow$ ii):

Let us define the kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ by

$$k(x, y) = \langle y|k_x\rangle \equiv \left\langle y\middle|(P^+P)^{-1}\middle|x\right\rangle$$

where P^+ is the adjoint operator of P. This implies that $k(x, y)$ is the *Green's function* of the operator P^+P (Note 1,2).

Next, we define a new dot-product on $\mathcal{H}$ by setting (Note 1,3)

$$\langle f|g\rangle_{\mathcal{H}} \equiv \left\langle f\middle|P^+P\middle|g\right\rangle.$$

Thirdly, we assume that $(\mathcal{H}, \langle .|.\rangle_{\mathcal{H}})$ is a Hilbert space (Note 3). It follows that every $|f\rangle \in \mathcal{H}$ can uniquely be written as

$$|f\rangle = \left|f^{\parallel}\right\rangle + \left|f^{\perp}\right\rangle$$

where $\left|f^{\parallel}\right\rangle \in V \equiv \underset{i=1,\ldots,m}{\text{span}}\{|k_{x_i}\rangle\}$ and $\left|f^{\perp}\right\rangle \in V^{\perp}$, i.e. for all $i = 1, \ldots, m$ we have $\left\langle f^{\perp}\middle|k_{x_i}\right\rangle_{\mathcal{H}} = 0$ (Note 2).

The following then holds for all $i = 1, \ldots, m$:

$$\begin{aligned} f(x_i) &= \langle f|x_i\rangle = \left\langle f\middle|(P^+P)(P^+P)^{-1}\middle|x_i\right\rangle = \langle f|k_{x_i}\rangle_{\mathcal{H}} \\ &= \left\langle f^{\parallel}\middle|k_{x_i}\right\rangle_{\mathcal{H}} + \underbrace{\left\langle f^{\perp}\middle|k_{x_i}\right\rangle_{\mathcal{H}}}_{=0} = f^{\parallel}(x_i) \end{aligned}$$

That is, the data term $\text{Loss}\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\ldots,m}\right)$ does not depend on $\left|f^{\perp}\right\rangle$.

The regularization term yields

$$\|Pf\|^2 = \langle Pf|Pf\rangle = \langle f|f\rangle_{\mathcal{H}} = \left\langle f^{\parallel}\middle|f^{\parallel}\right\rangle_{\mathcal{H}} + \underbrace{\left\langle f^{\perp}\middle|f^{\perp}\right\rangle_{\mathcal{H}}}_{\geq 0} + 2\underbrace{\left\langle f^{\parallel}\middle|f^{\perp}\right\rangle_{\mathcal{H}}}_{=0}.$$

Thus the minimizer of this term will always have $\left|f^{\perp}\right\rangle = 0$ regardless of $\left|f^{\parallel}\right\rangle$. The two above results together show that the minimizer of the objective function (2.2) has to satisfy $\left|f^{\perp}\right\rangle = 0$.

ii) $\rightarrow$ i):

Let us define a kernel operator K as

$$K = \int_{\mathcal{X}} dx \int_{\mathcal{X}} dy \, |x\rangle \, k(x,y) \, \langle y| \, .$$

Then we set the regularization operator P to be (Note 4)

$$P = \sqrt{K^{-1}}.$$

Having defined this operator, one can follow the above forward proof to show that ii) is equivalent to i). The kernel constructed there will equal the here given one. □

2.2.1 Some problems and potential solutions

Here, we discuss some of the mathematical difficulties of the derivation above. For some of them we hint at possible solutions, for others open questions remain.

1. P and P^+P may not have full rank, usually they do not (e.g. $P = \partial_x$ maps all constant functions to 0). Then P^+P is not invertible and the dot-product $\langle .|. \rangle_{\mathcal{H}}$ is not positive definite.

   Let N_P be the null space of the operator P^+P, i.e. the space of all functions which are mapped to 0 by P^+P. We can then define the dot-product $\langle .|. \rangle_{\mathcal{H}}$ as $\langle .|. \rangle_{\mathcal{H}} \equiv \langle .|P^+P + \mathbf{1}_{N_P}|. \rangle$ using the identity operator $\mathbf{1}_{N_P}$ on N_P. This dot-product is positive definite.

   Next, we can set $|k_x\rangle = (P^+P)^{\dagger} \, |x\rangle$ where $(P^+P)^{\dagger}$ is the Moore-Penrose pseudo-inverse of (P^+P). Thus, $(P^+P) \, |k_x\rangle = |x\rangle$ and $|k_x\rangle \in N_p^{\perp}$.

   The argumentation now holds as above, we just have to include a null space part into the regression function, i.e. the solution will be given in the form $|f\rangle = \sum_i \alpha_i \, |k_{x_i}\rangle + \sum_j \beta_j \, |u_j\rangle$. The $|u_j\rangle$ are supposed to span N_P.

   For further details see [Schaback, 2000; Wahba, 1990]. In this context also conditionally positive definite (CPD) functions are involved. However, it remains somewhat unclear how exactly these arise.

2. $|x\rangle$ is a distribution, not a function. Therefore it is not so clear how $(P^+P)^{-1} \, |x\rangle$ is defined or whether $|k_x\rangle$ is not a distribution itself. If so, then $\mathcal{H}$ must be a Hilbert space of distributions (?).

   The very same question seems to appear in standard quantum mechanics, where often the base system of the Dirac delta functions is used.

   A keyword in the search for explanations seems to be "Rigged" Hilbert spaces.

3. Which Hilbert space is $\mathcal{H}$?

It should be a subset of $L_2(\mathcal{X})$. Unlike $L_2(\mathcal{X})$ functions in $\mathcal{H}$ should be point-wise defined. $\mathcal{H}$ should allow derivatives up to a given order to be computed (e.g. if P is ∂_x). It has to be complete.

Could $\mathcal{H}$ be the Sobolev space $W^{m,2}(\mathcal{X})$? If so, $\mathcal{X}$ has to be bounded and we cannot do inference on whole $\mathbb{R}$. Or is $\mathcal{H}$ a Hilbert space of distributions?

4. Is K always invertible? Under which conditions on k? Does the kernel operator K have a spectral decomposition? This is not necessary for deriving P from it, but helps computing the root of the inverse of K.

   By the use of Mercer's theorem (see [Schölkopf and Smola, 2002, p.39]) we know that if $k(x, y)$ is positive definite, bounded, and $\mathcal{X}$ is compact, then K has a spectral decomposition. We can then write $K = \sum_{i=1}^{\infty} |u_i\rangle \, \lambda(i) \, \langle u_i|$ and $P = \sum_{i=1}^{\infty} |u_i\rangle \, \frac{1}{\sqrt{\lambda(i)}} \, \langle u_i|$.

2.2.2 Example: linear splines

Let $P = \partial_x$, $\mathcal{X} = [0, \pi]$ and $\mathcal{H}$ the space containing all twice differentiable functions which vanish on the boundaries (Note 3).

There are two ways to compute the kernel $k(x, y)$:

- analytic Green's functions: We saw that $k(x, y)$ is the Green's function of the operator $P^+P = -\partial_x^2$. Following [Roach, 1982] the current problem yields

  $$k(x, y) = (x - y)H(x - y) - \frac{x}{\pi}(\pi - y) \tag{2.5}$$

  with H being the Heaviside step function.

- Spectral representation: We can write $P^+P = \sum_{w=1}^{\infty} |w\rangle \, (-w^2) \, \langle w|$ with $|w\rangle$ being the functions $\left\{ \sqrt{\frac{2}{\pi}} \sin wx : w = 1, 2, \ldots \right\}$. The kernel then is

  $$k(x, y) = -\sum_{w=1}^{\infty} |w\rangle \, \frac{1}{w^2} \, \langle w| = -\sum_{w=1}^{\infty} \frac{\sin wx \sin wy}{w^2} \tag{2.6}$$

We are not able to explicitly show that the two functions are identical. However, it is easy to show that the series (2.6) is uniformly convergent and thus converges to a continuous function. Test experiments strongly support the proposition that the series (2.6) equals the first expression (2.5).

The result is a linear spline, i.e. a piecewise polynomial of degree 1 which is continuous. This is a well-known result. For $P = \partial_x^2$ we would similarly get cubic splines.

2.2.3 Inverse example: the Gaussian kernel

Suppose we are given the widely used Gaussian kernel $k(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$. Which regularization operator P corresponds to it?

We have $K = \int dxdy\, |x\rangle\, e^{-\frac{\|x-y\|^2}{2\sigma^2}}\, \langle y|$. This operator is diagonal in the momentum eigensystem $|k\rangle$ with $\langle x|k\rangle = e^{ikx}$. This is because all radial basis kernels are translation-invariant. Then

$$\begin{aligned}
K &= \int dxdy\, |x\rangle\, e^{-\frac{\|x-y\|^2}{2\sigma^2}}\, \langle y| \\
&= \int dk \int dxdy\, |k\rangle\, \langle k|x\rangle\, e^{-\frac{\|x-y\|^2}{2\sigma^2}}\, \langle y|k\rangle\, \langle k| \\
&= \int dk\, |k\rangle \left(\int dxdy e^{ik(y-x)} e^{-\frac{\|x-y\|^2}{2\sigma^2}} \right) \langle k| \\
&= \int dk\, |k\rangle \left(\int dx \right) \underbrace{\left(\int dw e^{ikw} e^{-\frac{\|w\|^2}{2\sigma^2}} \right)}_{\text{Fourier trafo}} \langle k| \\
&= const \int dk\, |k\rangle\, e^{-\frac{\sigma^2 k^2}{2}}\, \langle k| \\
P &= \sqrt{K^{-1}} = \int dk\, |k\rangle \left(e^{-\frac{\sigma^2 k^2}{2}} \right)^{-\frac{1}{2}} \langle k| = \int dk\, |k\rangle\, e^{\frac{\sigma^2 k^2}{4}}\, \langle k| \\
&= \sum_{n=0}^{\infty} \frac{1}{n!} (\int dk\, |k\rangle\, \frac{\sigma^2 k^2}{4}\, \langle k|)^n = \sum_{n=0}^{\infty} \frac{\sigma^{2n}}{4^n n!} (-\partial_x^{2n})
\end{aligned}$$

This result is identical to the one derived by [Girosi et al., 1993].

2.2.4 Example: harmonic oscillator

Apart from these standard results one could also think of new kernels implied by regularization operators which may sometimes be useful:

Suppose our data points are centred around $x = 0$ and we think that a good regression function should also be localised apart from being smooth. We could define a penalty like

$$P^+ P = -\frac{\hbar}{2m}\partial_x^2 + \frac{1}{2} m w^2 x^2,$$

which corresponds to the harmonic oscillator (HO) Hamiltonian. Then, letting $|n\rangle$ be the eigen-functions of the HO we could write

$$k(x, y) = \langle x|k_y\rangle = \sum_n \langle x|n\rangle\, \frac{1}{\hbar\omega(n + 1/2)}\, \langle n|y\rangle\,.$$

If the data points were for example drawn from a physical experiment involving a harmonic oscillator, this kernel function might lead to more plausible results than a standard Gaussian kernel.

2.3 Support Vector Regression (SVR)

A special case of non-parametric regression is ϵ-*insensitive* Support Vector Regression (SVR) introduced by Vapnik [Vapnik, 1995]: As a loss function we choose

$$\text{Loss}\left(\{x_i, y_i, \langle x_i|f\rangle\}_{i=1,\ldots,m} \right) = \sum_{i=1}^{m} |f(x_i) - y|_\epsilon\,.$$

The ϵ-insensitive loss function $|.|_\epsilon$ takes the values $\max\{|x| - \epsilon, 0\}$. Errors smaller than ϵ are not penalised and larger ones are linearly measured.

We can compute the regression function $|f\rangle$ using the kernel objective (2.4). We additionally include a constant offset b into the regression function, i.e. $f(x) = \sum_i \alpha_i k(x, x_i) + b$. We then solve the problem

$$\begin{aligned} \min_{\alpha,\xi^{(*)},b} \frac{1}{2}\alpha^T K\alpha + C \sum_i \xi^{(*)} & \qquad (2.7) \\ \text{s.t.} \quad \forall i = 1,\ldots,m: \sum_j \alpha_j k(x,x_j) + b - y_i &\leq \epsilon + \xi_i \\ -(\sum_j \alpha_j k(x,x_j) + b - y_i) &\leq \epsilon + \xi_i^* \\ \xi^{(*)} &\geq 0 \end{aligned}$$

$\xi^{(*)}$ always refers to all the ξ_i as well as to the ξ_i^*. This problem is a convex, quadratic optimisation problem.

Using the ϵ-insensitive loss function, the solution expansion often is relatively sparse, i.e. many of the coefficients α_i are zero. The training examples x_i, which have a non-zero coefficient α_i and thus support the solution expansion, are often termed *Support Vectors*.

Computing the Lagrange dual and using the Karush-Kuhn-Tucker (KKT) optimality conditions, we can rewrite the above problem to yield the system

$$\begin{aligned} \min_{\alpha^{(*)}} \quad & \frac{1}{2}(\alpha - \alpha^*)^T K(\alpha - \alpha^*) + \sum_i (\alpha_i - \alpha_i^*)y_i + \epsilon \sum \alpha^{(*)} \qquad (2.8) \\ & \text{s.t.} \quad \sum (\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i^{(*)} \leq C \end{aligned}$$

where we have split the coefficients α_i into a positive and a negative part, α_i and α_i^* respectively. Thus,

$$f(x) = -\sum_i (\alpha_i - \alpha_i^*)k(x_i, x) + b.$$

This is the form that is typically used to compute the regression function f in standard algorithms like libSVM [Chang and Lin, 2005] or SMO [Platt, 1998].

2.4 Reproducing kernel Hilbert spaces (RKHS)

The space $(\mathcal{H}, \langle .|. \rangle_\mathcal{H})$ is often termed a *reproducing kernel Hilbert space* (RKHS). This is due to the fact that the kernel associated with it is the representer of evaluation, i.e.

$$\forall\, |f\rangle \in \mathcal{H},\ \forall x \in \mathcal{X} : \langle k_x | f \rangle_\mathcal{H} = \big\langle x \big| (P^+P)^{-1}(P^+P) \big| f \big\rangle = \langle x | f \rangle = f(x).$$

Especially we have the *reproducing property* for kernels

$$\forall x, y \in \mathcal{X} : \langle k_x | k_y \rangle_\mathcal{H} = \big\langle x \big| (P^+P)^{-1}(P^+P) \big| k_y \big\rangle = k(x,y).$$

Let us imagine that we map all points $x \in \mathcal{X}$ into $\mathcal{H}$ using the *kernel mapping* $\phi : \mathcal{X} \to \mathcal{H},\ \phi(x) = |k_x\rangle$. Often $\mathcal{X}$ is called the *input space* and $\mathcal{H}$ the *feature space*. Even though $\mathcal{H}$ is an infinite dimensional space we can still work geometrically with these vectors $\phi(x) \in \mathcal{H}$

because the dot-products can be computed with a simple kernel function evaluation in the input space, namely

$$\langle \phi(x) | \phi(y) \rangle_{\mathcal{H}} = k(x, y). \tag{2.9}$$

This "kernel trick" (2.9) implies that every algorithm that can be conducted using only inner products and norms in the input space $\mathcal{X}$ can also be computed in the feature space $\mathcal{H}$. If an algorithm in input space $\mathcal{X}$ is linear its feature space extension is non-linear which can sometimes add additional power to a method.

2.4.1 SVR and its RKHS interpretation

We can now give a geometrical interpretation to the SVR. If we set $|w\rangle = \sum_{i=1}^{m} \alpha_i |k_{x_i}\rangle$ then

$$\begin{aligned} f(x) &= \sum_i \alpha_i k(x_i, x) + b = \sum_i \alpha_i \langle k_{x_i} | k_x \rangle_{\mathcal{H}} + b \\ &= \langle w | \phi(x) \rangle_{\mathcal{H}} + b. \end{aligned}$$

This is a linear function in the reproducing kernel Hilbert space $\mathcal{H}$. Thus, one can interpret SVR as searching a linear function in $\mathcal{H}$ which "best" approximates our data points mapped into $\mathcal{H}$ by the kernel mapping $\phi(x) = |k_x\rangle$. This way, we have reduced a non-linear function approximation task in $\mathcal{X}$ into a linear problem in $\mathcal{H}$.

The regularization term can be written as $\|w\|_{\mathcal{H}}^2$. Components of $|w\rangle$ that are not necessary to fit the data are set to zero yielding an as simple as possible vector w. Thus, there is a nice duality between a geometric meaning in the RKHS $\mathcal{H}$ and a functional smoothness criterion in the input space $\mathcal{X}$.

Chapter 3

Implicit Surface Reconstruction

3.1 Introduction

One way to represent the surface of an object in $\mathcal{X} \subseteq \mathbb{R}^D$ is as an *implicit surface*. The surface is then defined as

$$\{x \in \mathcal{X} : f(x) = 0\},$$

i.e. the zero level-set of an appropriately chosen *implicit surface function* $f : \mathcal{X} \to \mathbb{R}$.

The surface reconstruction problem which we will try to solve in this chapter is stated as follows: Given N points $x_1, \ldots, x_N \in \mathcal{X}$ sampled from the surface of some object construct an implicit surface function $f : \mathcal{X} \to \mathbb{R}$ which describes the surface of the data generating object. One can either interpolate the given points, i.e. $f(x_i) \equiv 0,\ i = 1, \ldots, N$, or just approximate them. In this case, the implicit surface function does not necessarily have exact zero values on the given surface points. Yet, it might be a better approximation of the true surface because the given surface points are often distorted by noise.

A special implicit surface function $f : \mathcal{X} \to \mathbb{R}$ is the *signed distance function* for which the following properties are fulfilled:

- $|f(x)|$ equals the distance to the surface and
- $f(x) > 0$ if x inside the object and $f(x) < 0$ otherwise.

This way, if one moves from a given surface point inward along its surface normal, the function value will increase linearly with slope one, i.e. $\|\nabla f(x)\| = 1$. The functions f which we will construct here will approximate this behaviour in the neighbourhood of the surface.

The surface points typically are acquired from 3D scanners, such as laser or structured light scanners. A scan will normally contain on the order of 10^5 3D points. This high number makes efficient algorithms mandatory. Algorithms with quadratic or higher scaling behaviour are prohibitive as they would need more than 10^{10} computations which is too much even for simple operations. We will put a lot of effort into achieving an overall O(N log N) runtime. Memory requirements will also be bounded by O(N log N).

The algorithm presented in this work will construct the implicit surface function as the result of a machine learning regression task (cf. Chapter 2). The surface points are used to create the input (Section 3.3) into a Support Vector Regression (SVR). For speed reasons we do not

use the standard ϵ-insensitive SVR but some modification thereof (Section 3.5). Additionally, a compactly supported kernel, the Wu function [Schaback, 1995]

$$k(r) = (1-r)_+^4(4 + 16r + 12r^2 + 3r^3), \tag{3.1}$$

with $r = \frac{\|x-y\|}{\sigma}$ is applied. We will finally employ a multi-scale scheme in order to adapt the complexity of the implicit representation to the intrinsic shape complexity (Section 3.4). In Section 3.6 we will describe how implicit surfaces can be displayed, and we will show experiments in Section 3.7. We will also describe a method to automatically determine all necessary parameters in Section 3.8, such that only a fitting accuracy ϵ needs to be chosen in advance.

3.2 Related work

The idea of representing surfaces as implicit functions has been around for a long time, but only gained broader interest more recently as the computational power to deal with large point sets became available. An early theoretical overview without any experiments is given by [Schaback, 1995].

A first experimental approach is described in [Turk et al., 2001]. The implicit surface function is approximated as a linear sum of fully supported radial basis functions centred at the given surface points. To compute the coefficients a linear system of the size of the number of given surface points needs to be solved. This limits the approach to about 2,000 - 3,000 points. The linear system does not just include the zero constraints for the surface points as this would lead to the trivial solution $f \equiv 0$. [Turk et al., 2001] make use of additionally given surface normals and create new points shortly off the surface where they set the desired function value to match the signed distance function. This way triviality is avoided and normal information can be included. Our approach will use a similar technique that is explained in more detail in Section 3.3.2.

[Carr et al., 2001] extend the above approach to a state-of-the-art method that is able to deal with hundreds of thousands of points. They use a radial spline basis and solve the linear system by applying a sophisticated approximation scheme which is similar to the Fast-Multipole-Methods used in astronomy [Beatson and Newsam, 1998; Yang et al., 2003].

Another method to deal with a large linear system is to try to sparsify it: [Morse et al., 2001] introduce compactly supported radial basis functions into the fitting process. The method of [Ohtake et al., 2003b] applies several scales of fitting to improve the interpolation properties of the implicit surface function.

An alternative to these global techniques, where only one $f : \mathcal{X} \to \mathbb{R}$ for the whole object exists, is given by [Alexa et al., 2003]: Given a query point x the Moving-Least-Squares (MLS) method approximates the implicit surface function through a linear function, i.e. a plane, which is determined by a locally weighted least squares fit of adjacent points. The function value is just the distance to the plane. Similarly, one can construct a corresponding point on the surface by projecting x on the thus constructed plane approximation. [Amenta and Kil, 2004] give a mathematical proof that such surfaces are well defined. For computing the locally weighted sums one needs a good distance measure of points close to a surface; [Klein and Zachmann, 2004b] propose to use the geodesic distance instead of the Euclidean distance which takes possible bending of the surfaces into account.

[Shen et al., 2004] also compute the implicit surface function only locally. Based on a least squares fit they are able to include exact normal and polygonal constraints. The polygonal constraints help to avoid small bumps which often occur in other methods that just build on point constraints.

[Ohtake et al., 2003a] mix the global and the local approaches: They try to approximate the surface with a simple base function. If this fails for the given set of points, the volume is subdivided into an octree and the simple approximation is done on the subsets. In order to avoid non-differentiable transitions between the pieces, they introduce weighting functions which overlap and combine the simple approximation pieces in to a globally defined implicit surface function. Using this method the complexity of the surface description nicely adapts to its intrinsic shape complexity. They are able to work with huge models almost in real time ($> 10^6$ points).

A completely different approach is taken by [Museth et al., 2002]: Here the implicit surface function is the discretised solution of a partial differential equation (PDE). The signed distance function f optimally has the property $\|\nabla f(x)\| = 1$ and the boundary values $f|_{Surf} = 0$.

Estimating a smooth from a given set of point samples is a regression problem. First steps to use the machine learning perspective in this application were given by [Walder and Lovell, 2003], yet this approach was limited in size and was only applied to 2D shapes in computer vision. Another machine learning method was described by [Schölkopf et al., 2004], which will be closer looked upon in Section 3.3.1.

What are the typical applications that make use of implicit surfaces?

The standard problem is the construction of a smooth surface from noisy point sampled data acquired through 3D scanners. All above described methods are able to deal with this, but [Carr et al., 2003] give an especially nice treatment of how the data can be quickly smoothed at the user's wishes. [Shen et al., 2004] are able to include polygonal constraint into the fitting which may lead to very smooth and exact results if not just points but also surface triangles etc. are given. With the exception of the locally supported approach of [Morse et al., 2001] all approaches are also able to smoothly fill moderately sized holes in the dataset.

Implicit surface functions are naturally smooth unlike discretised meshes. Therefore, they can be used to extract differential geometric properties of the surfaces such as curvature or normal directions in a very stable way [Thirion and Gourdon, 1995]. [Ohtake et al., 2004] use an implicit surface function to extract crest lines and other points of interest from a surface to yield a sketch of an 3D object. These sketches can be used for object recognition.

Given an implicit surface function which approximates the signed distance, it is easy to perform inside/outside tests and to determine the distance to the surface. Therefore some papers mention constructive solid geometry (CSG) as possible fields of application [Museth et al., 2002]. One can determine the volume of an union or dissection of two 3D objects by just looking at the signs of the individual implicit surface functions. [Klein and Zachmann, 2004a] use this property of implicit surfaces to achieve fast collision detection for 3D objects.

Depending on the method used to construct the implicit descriptions they can also be used to compress 3D models. A smooth part of the surface can be approximated with a very simple implicit surface function. If the function model adapts to the local surface complexity, an implicit description can yield a much sparser representation than an uniformly sampled point set [Ohtake et al., 2003a].

Given two implicit surface functions, one can naturally blend one into the other by taking the convex combination of the two. This approach is able to transform topologically different

objects into each other [Cohen-Or et al., 1998]. Yet, if the two surfaces are not similar to each other, the interpolation may introduce artifacts such as spurious surfaces. This transformation also does not solve the correspondence problem (see Chapter 6), therefore the interpolation may not make sense even if it is smooth. Another idea to use an implicit description for morphing is taken up by [Turk and O'Brien, 1999]: Two objects are represented by a single hyper-surface in 4D (3D data + 1D interpolation time). To get intermediate objects one can just take a slice of the 4D surface and plot it in 3D. Like above, this method does not solve the correspondence problem.

When working with deforming meshes, one often needs to find a new mesh with better geometric properties (say almost equally angled triangles, etc.) to increase the numerical stability of some computations. One way to re-sample the object is to first fit an implicit surface representation to the given input mesh and then extract a new mesh with the desired properties from it.

3.3 Setting up the regression problem

The approach presented here aims at using machine learning regression techniques to solve the reconstruction problem. The preferred choice of algorithm is Support Vector Regression (SVR) as described in Chapter 2. The estimated functions can be written as

$$f(x) = \sum_i \alpha_i k(x_i, x) + b,$$

thus, the implicit surfaces $\{x \in \mathcal{X} : f(x) = 0\}$ correspond to hyper-planes in a reproducing kernel Hilbert space $\mathcal{H}$.

Initially, we are given N surface points which are supposed to have an implicit surface function value zero. If we use these points and the value zero as the training input, then for any choice of loss function and any regularizer, the simplest and optimal solution of the regression will just be $f \equiv 0$. This function interpolates the data points exactly and is the smoothest function under any linear smoothness measure. However, this function does not define any surface.

There are two ways to get around this problem: One is to modify the regression task and include new terms in the objective function. Equivalently one could constrain b to some non-zero value. The other idea is to include training points which are not on the surface. Here the training values are set to match the signed distance. The second class of approaches usually requires information about the surface normal to be given.

3.3.1 Changing the objective function

Initially, we experimented with the first option, namely changing the objective function. The so-called "SlabSVM" [Schölkopf et al., 2004] sets $f(x) = \sum_i \alpha_i k(x, x_i) + b$ and includes $-b$ into the SVR objective function (see Chapter 2). Additionally, instead of fitting one hyper-plane to the training points, two parallel planes at distance δ to each other are fitted to the dataset, such that as many points as possible fall into the space between the two planes, i.e. the "slab". This way, one does not actually define a two dimensional manifold in input space, but a thin shell which includes the given surface points.

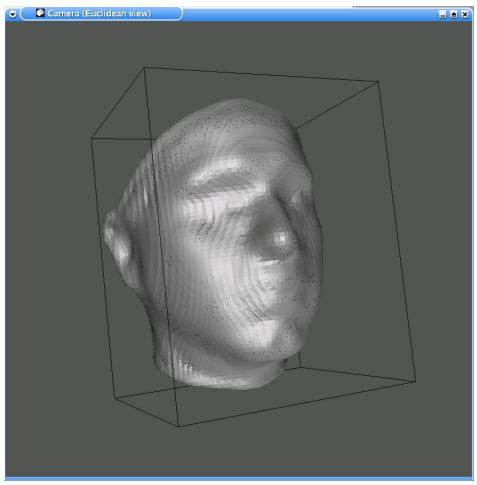


Figure 3.1: *Image of a head reconstructed with the SlabSVM. For computational reasons the slab width δ was set to zero which initially yields an unconnected surface. In order to being able to plot the surface, the whole function was shifted by a positive constant. This way, many structure details are lost.*

Using this method we got some primary results (see Figure 3.1). But there are some fundamental problems which cannot be solved in that framework:

As stated by the representer theorem (see Section 2.2), the constructed function will consist of a number of kernel functions which are centred on the given surface points. If we use a radial basis kernel, e.g. a Gaussian, one can image that the kernels are "bumps" sitting on each data point. The level-set zero of the implicit surface function is just lying near the top of these bumps (see Figure 3.2). Several problems arise:

- If computed exactly and $\delta = 0$, the surface only consists of the training points and is not connected. If on the other hand $\delta \neq 0$, the surface is not determined exactly in input space. In general, it is hard to interpolate the surface into regions without training points, because the surface is very close to the top of the kernel "bumps".
- The representation is very unstable with respect to small changes in the kernel coefficients. Small deviations may even let the surface disappear completely.
- If we do ray tracing or employ iso-surfacing using the marching cubes algorithm (see Section 3.6), we test the function in certain intervals for change in sign. It is very easy to miss parts of the surface because the area where the function has positive value - given a negative offset b - is very small. It will become even smaller when we want to model more details and thus reduce the kernel width and decrease δ.
- We also miss our goal of approximating the signed distance function which is useful for many applications such as inside-outside tests, constructive solid geometry, or shape matching (see Chapter 6).

There is a second method that can work without normal information [Walder et al., 2005]: The idea is to aim for quadratically large function values everywhere except on the surface. If the function is very smooth at the same time the optimal solution will have one connected region of positive sign (e.g. inside) and another region of opposite sign (outside). However, this approach yields a non convex optimisation problem and is difficult to handle numerically.

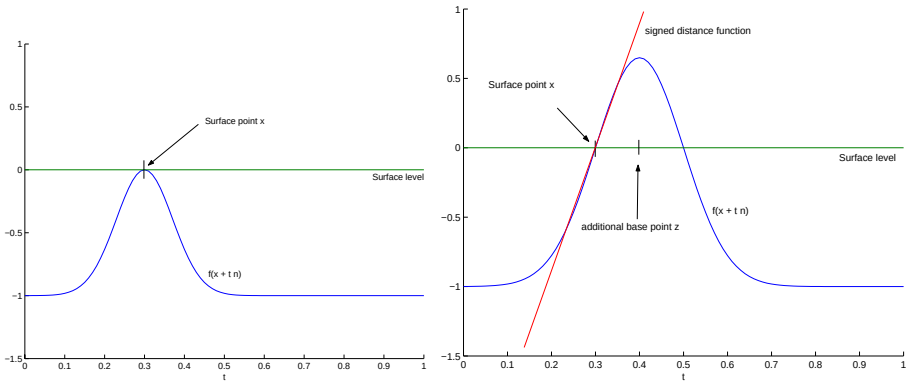


Figure 3.2: *Two cuts along the surface normal. The implicit surface function is depicted in blue. The resulting surface is its cut with the zero surface level. The left image shows the situation for the SlabSVM: The bumps are centred at the surface points. Small changes in the height of the bump will let the surface disappear completely. On the right hand side we see the situation with carefully created additional training points. The surface is stable with respect to small changes in the height of the kernel bump and the implicit surface function has a non vanishing gradient around the surface point x. If adjusted correctly the function is able to approximate the signed distance function in the neighbourhood of the surface.*

3.3.2 Training point generation

It turns out that a better method is to use training points off the surface with training values matching the signed distance function (see Figure 3.2). This has the following advantages:

- Triviality is avoided.
- The surface is defined in a part of the kernel "bumps" where the gradient is high. This way our representation becomes more stable concerning changes in the coefficients. It also makes it more attractive for all kinds of fast approximation algorithms for evaluation and regression.
- The function has now larger areas where it has a function value greater than zero. We therefore cannot miss it with a grid, even for larger grid spacings (used e.g. for ray tracing or Marching Cubes).
- The implicit surface function can now be made a local approximation of the signed distance function if we adjust the training values of the new training points accordingly.

However, in order to construct these off-surface training points we need surface normals given at all the surface points. It is reasonable to assume that some idea of a normal is available. They can easily be estimated from nearest neighbour information if the objects are not sampled too sparsely. Suppose a 3D scanner acquires a depth image r_{xy} line by line. We can then approximate a normal by $n_{xy} = (r_{x(y+1)} - r_{xy}) \times (r_{(x+1)y} - r_{xy})$. Our fitting algorithm will use a linear loss function, so it should be robust to small errors in this approximation.

Our method for creating the off-surface training points is very similar to the approaches of [Turk et al., 2001] or [Carr et al., 2001]. [Shen et al., 2004] present a method which includes exact normal information into the fitting process instead of the "pseudo" normals implied

by creating off-surface training points. However, their approach is based on local function approximations and does not easily transfer to our global regression problem.

The additional training points x_i are generated by displacing the given surface points along their surface normals by a distance d. This distance is chosen such that the kernel function $k(x) = k(x_i, x)$ has a stable, non vanishing gradient on the supposed surface. It turned out that $\frac{\sigma}{3}$ is a good choice for d. For each surface point we generate one inside and one outside training point. The original point directly on the surface is dropped from the training set, as the neighbouring points already constrain the surface enough.

If we set the training value y_i of our new training points x_i to equal the supposed signed distance, i.e. $y_i = \pm d$, the implicit surface function may not exactly equal the signed distance function in the close neighbourhood of the surface, because the kernel function (3.1) is not linear in radial direction. In order to get a close approximation of the signed distance, we could determine the training values y_i differently: One could select some points along the surface normal and demand correct signed distances there. One can then determine the necessary training values y_i through a least squares fit of the two kernels involved. However, the experiments did not show any significantly different results compared to the straight forward method, i.e. setting $y_i = \pm d$. We therefore abandoned the proposed procedure in favour of computational simplicity and speed.

When deciding whether to include a thus constructed off-surface training point in the final optimisation, we use a heuristic to ensure that its target value y_i is approximately within 10% of the correct signed distance. Clearly, the true distance of the point is at most $|y_i|$. However, it may happen that the surface bends around, and it in fact is closer than this. We discard the point if there exists a surface point at a distance of less than $0.9|y_i|$. This method worked well in all our experiments.

3.4 Multi-scale scheme

The proposed function fitting algorithm uses compactly supported radial basis kernel (3.1) having an intrinsic length scale σ, the kernel width. In order to be able to extrapolate over holes in the data, we need large kernel widths and in order to optimise the reconstruction of fine details we also should include small length scales. We therefore apply a multi-scale scheme where we fit starting from a coarse level and successively refine with smaller kernel widths. The final function then is the sum of the functions computed at each level. Thus, at each refinement level we just approximate the remainder of the above computed function and the desired one.

As kernel widths σ we propose to use a cascade of $\sigma_0, \frac{\sigma_0}{2}, \frac{\sigma_0}{4}, \ldots$ where σ_0 is chosen on the order of $\frac{1}{2}$ times the object diameter. In our experiments we found 5 to 6 scales to be sufficient for fitting accuracies ϵ of about 10^{-4} of the object diameter.

In general a SVR solution using the ϵ-insensitive loss function will be a relatively sparse kernel expansion $f(x) = \sum_i \alpha_i k(x, x_i)$, i.e. many α_i will equal zero. Training points with nonzero coefficients are called *support vectors* (SV). In general it is not known before the regression problem is solved which of the training points will become a support vector. In our application it is critical to yield optimal speed. We therefore try to extract probable SVs before actually computing the regression. This sub-sampling together with the compactly supported kernel yields a sparse kernel matrix K, $K_{ij} = k(x_i, x_j)$, simplifying the computations. However, we take care to keep the reduced functional model flexible enough to being close to the optimal solution which would use all training points.

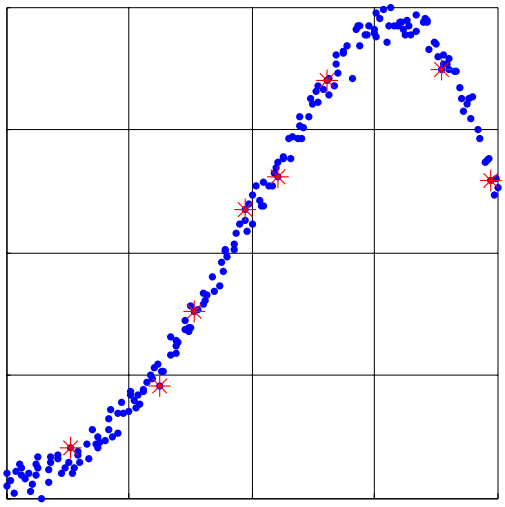


Figure 3.3: *Some surface points (blue) and the points which are extracted by the subdivision scheme (red). The minimal boxes are plotted as black lines.*

The subset extracted is to lie approximately equally spaced on the surface (see Figure 3.3). We therefore apply a simple box subdivision scheme, which starts with a bounding box of the given surface points, and subdivides the box recursively as long as there are still points inside the box and as long as a minimum box size ($\frac{\sigma}{2}$) has not been reached. Once the subdivision process has stopped we extract the centre most points from each box on the finest level (see Figure 3.3). We thereby construct an approximately equally spaced subsampling of all points in O(N log N) runtime (N being the number of given surface points).

Using this subsampling procedure, the condition of the kernel matrix K can be controlled effectively. A small condition renders the resulting optimisation problem of the regression amenable to greedy optimisation techniques. The condition is mainly dependent on how strongly the training points interact through the matrix. As our subsampling stops at a scale proportional to the kernel support size σ this number is held constant over all scales (see Figure 3.4). It also turns out to be independent from the number of given surface points N.

There is another mechanism to reduce the problem size in the refinement scales: If at a proposed training point the implicit surface function computed at higher levels already gives a value closer to the proposed training value than a given fitting accuracy ϵ, then we discard this training point as nothing more has to be done in this region. Thereby we can efficiently adapt the number of kernel centres to the local complexity of the surface.

3.4.1 RKHS interpretation

The above constructed multi-scale functions are sums of functions each of which is a linear function in a reproducing kernel Hilbert space (RKHS). Is it possible to interpret the combined function as a single linear function in a RKHS as well?

For simplicity, we just use two scales here. An extension to more scales is straight forward.

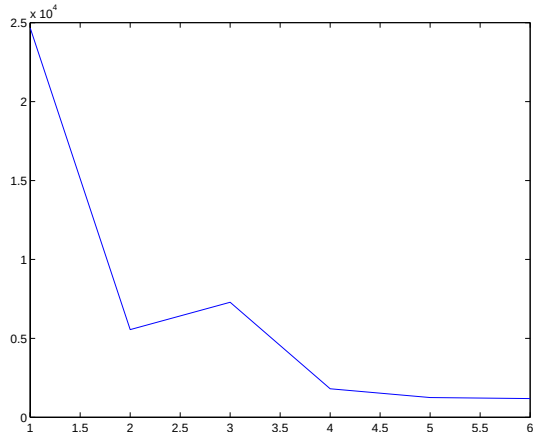


Figure 3.4: *The condition of the kernel matrix for a head dataset on 6 scales ($\sigma = \frac{1}{2} \dots \frac{1}{64}$ of the object diameter. Note that the condition is higher for bigger kernel support sizes and converges for the small scales. The condition is generally proportional to the number of points which interact though the matrix. For small kernels the surface region within the support of one kernel function is approximately flat. For the larger scales the surface may still have some bending resulting in more points within the support of the kernel function and thus a higher condition of K.*

The multi-scale implicit surface function takes the form

$$\begin{aligned} f(x) &= \sum_{i=1}^{N_1} \alpha_{i,1} k_1(x, x_{i,1}) + \sum_{i=1}^{N_2} \alpha_{i,2} k_2(x, x_{i,2}) + b \\ &= \langle w_1 | \phi_1(x) \rangle_{\mathcal{H}_1} + \langle w_2 | \phi_2(x) \rangle_{\mathcal{H}_2} + b, \end{aligned}$$

with $w_{1(2)} \in \mathcal{H}_{1(2)}$ and $\phi_{1(2)} : \mathcal{X} \to \mathcal{H}_{1(2)},\ \phi_{1(2)}(x) = k_{1(2)}(x, .)$. Can we write this function as a linear function in another RKHS $\mathcal{H}$, i.e.

$$f(x) = \langle w | \phi(x) \rangle_{\mathcal{H}} + b \tag{3.2}$$

with $w \in \mathcal{H}$ and $\phi : \mathcal{X} \to \mathcal{H}$?

[Aronszajn, 1950, Sec. 6] states that the space $\mathcal{H} = \left\{ f_1 + f_2 : f_{1(2)} \in \mathcal{H}_{1(2)} \right\}$ is a RKHS with the kernel $k(x, y) = k_1(x, y) + k_2(x, y)$ if endowed with the dot product

$$\langle f | g \rangle_{\mathcal{H}} = \min \left(\langle f_1 | g_1 \rangle_{\mathcal{H}_1} + \langle f_2 | g_2 \rangle_{\mathcal{H}_2} \right). \tag{3.3}$$

The minimisation is taken over all decompositions $f = f_1 + f_2,\ g = g_1 + g_2$ with $f_{1(2)}, g_{1(2)} \in \mathcal{H}_{1(2)}$.

This allows to set $w = w_1 + w_2$ and $\phi(x) = k_1(x, .) + k_2(x, .)$. The result is then almost of the desired form (3.2), we only need to avoid the minimisation when computing the dot product $\langle . | . \rangle_{\mathcal{H}}$ in (3.3). There are two possible events when the minimisation does not have any effect and can be omitted:

1. The decomposition could be unique. This is equivalent to $\mathcal{H}_1 \cap \mathcal{H}_2 = \{0\}$.
2. The value of $\langle f_1 | g_1 \rangle_{\mathcal{H}_1} + \langle f_2 | g_2 \rangle_{\mathcal{H}_2}$ could be independent of the decomposition into $f_{1(2)}$, $g_{1(2)}$. This would imply that $\|h\|_{\mathcal{H}_1} = \|h\|_{\mathcal{H}_2}$ for all $h \in \mathcal{H}_1 \cap \mathcal{H}_2$.

So the key question seems to be the following: Are there any $h \neq 0$ in $\mathcal{H}_1 \cap \mathcal{H}_2$, if $\mathcal{H}_1$ and $\mathcal{H}_2$ are the RKHSs corresponding to Wu kernels (3.1) (or similarly to a Gaussian kernels) of different widths σ? Both spaces $\mathcal{H}_{1(2)}$ are dense in the $C(\mathcal{X})$ with respect to the infinity norm $\|.\|_\infty$. We also have that $\mathcal{H}_{1(2)} \subseteq C(\mathcal{X})$. Thus, for each element in $h \in \mathcal{H}_1$ there exists a sequence of elements in $\mathcal{H}_2$ that converges point-wise to h. But does there exist such a sequence that is also a Cauchy sequence in the RKHS norm $\|.\|_{\mathcal{H}_2}$? If yes, then the limiting element h would also be in $\mathcal{H}_2$. However, this is not obvious. Convergence in the norm $\|.\|_{\mathcal{H}_{1(2)}}$ implies convergence in the norm $\|.\|_\infty$, but not the other way around.

It therefore remains an open problem whether one can interpret the multi-scale implicit surface function as a single linear function in a combined RKHS.

3.5 Efficient fixed bias ϵ-SVR

Having set up the individual regression problems for the different scales we now have to describe a fast algorithm to compute them. We do not use standard ϵ-SVR (see Chapter 2) but an adaptation thereof using a fixed offset b (see Section 3.5.1). The adapted version turns out to yield a simpler optimisation problem than regular SVR.

The training time of standard (non sparse) implementations of ϵ-SVR like libSVM [Chang and Lin, 2005] or SMO [Platt, 1998] scales approximately between $O(N^{1.5})$ and $O(N^3)$. First experiments with libSVM and a Gaussian kernel were limited to less than 5000 points and required hours of computation time. We then tried to compute the simplified sparse optimisation problem (see Section 3.5.1) with a modified version of libSVM, but the implementation was difficult. We therefore constructed a new optimiser that applies a simple but efficient coordinate descent scheme (see Section 3.5.2). It turns out that the update rules exactly equal a standard sparse optimiser – namely the Gauß-Seidel-method.

3.5.1 Fixed offset SVR and derivation of the dual problem

In our case it does not make sense to optimise over b. We construct as many training points with training value $-d$ as points with training value $+d$. The computed offset b will therefore always be close to zero. If b is close to zero, than the implicit surface function will also take on this value far away from the surface. A ray tracing algorithm then cannot know whether the surface is close or not and therefore has to make many small steps. Thus, a better choice is to fix b to the value $-d$, i.e. the negative training value of training points on the first scale. This way, the implicit surface function has considerable distance to zero in areas far apart from the surface.

We can write the primal regression problem as follows (compare to Chapter 2)

$$\begin{aligned} \min_{w,\xi^{(*)}} \quad & \frac{1}{2}\|w\|_{\mathcal{H}}^2 + C\sum \xi^{(*)} \\ \text{s.t.} \quad & \langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i \leq \epsilon + \xi_i \\ & -(\langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i) \leq \epsilon + \xi_i^* \\ & \xi^{(*)} \geq 0 \end{aligned}$$

where $f(x) = \langle w|\phi(x)\rangle_{\mathcal{H}} + b$ is a linear function in the reproducing kernel Hilbert space $\mathcal{H}$. $\xi^{(*)}$ stands for all ξ_i and ξ_i^*. The above problem equals the ϵ-insensitive SVR, except that we do not optimise over the constant parameter b.

Dualizing this problem yields a finite problem which can be optimised efficiently. The Lagrange function with the dual variables $\alpha^{(*)}, \beta^{(*)}$ is

$$\begin{aligned} L(w, \xi^{(*)}, \alpha^{(*)}, \beta^{(*)}) = \\ \frac{1}{2}\|w\|^2_{\mathcal{H}} + C\sum \xi_i^{(*)} - \sum \beta_i^{(*)}\xi_i^{(*)} \\ + \sum \alpha_i(\langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i - \epsilon - \xi_i) \\ + \sum \alpha_i^*(-(\langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i) - \epsilon - \xi_i^*) \end{aligned}$$

Following the standard dualization procedure described in [Schölkopf and Smola, 2002], we can now write the problem in the form

$$\begin{aligned} \max_{\alpha^{(*)},\beta^{(*)}} \min_{w,\xi^{(*)}} \quad & L(w, \xi^{(*)}, \alpha^{(*)}, \beta^{(*)}) \\ \text{s.t.} \quad & \alpha^{(*)} \geq 0 \quad \beta^{(*)} \geq 0. \end{aligned}$$

This problem is unconstrained and differentiable in the primal variables $w, \xi^{(*)}$ yielding the optimality conditions

$$\begin{aligned} w &= -\sum_i (\alpha_i - \alpha_i^*)\phi(x_i) \\ \beta^{(*)} &= C - \alpha^{(*)} \\ 0 \leq \alpha_i^{(*)} &\leq C \end{aligned}$$

The dual problem is then given as

$$\begin{aligned} \min_{\alpha^{(*)}} \quad & \frac{1}{2}\left\|\sum_i (\alpha_i - \alpha_i^*)\phi(x_i)\right\|^2_{\mathcal{H}} + \sum_i (\alpha_i - \alpha_i^*)y_i + \epsilon\sum \alpha^{(*)} \\ & \text{s.t.} \quad 0 \leq \alpha_i^{(*)} \leq C \end{aligned}$$

or in matrix notation with the kernel gram matrix K ($K|_{ij} = k(x_i, x_j)$) as

$$\begin{aligned} \min_{\alpha^{(*)}} \quad & \frac{1}{2}(\alpha - \alpha^*)^T K(\alpha - \alpha^*) + y^T(\alpha - \alpha^*) + \epsilon(\alpha + \alpha^*) \\ & \text{s.t.} \quad 0 \leq \alpha_i^{(*)} \leq C \end{aligned} \tag{3.4}$$

The solution function is $f(x) = -\sum_i (\alpha_i - \alpha_i^*)k(x_i, x) + b = K(\alpha^* - \alpha) + b$. This result is almost the same as for the standard ϵ-insensitive SVR, except that the dual problem does not have a linear constraint $\sum_i(\alpha_i - \alpha_i^*) = 0$. This will simplify the optimisation process as will be shown below.

We do not use C/N in the objective function but just C as a regularization parameter. Because we are constructing our training points equally spaced, the term $\|w\|^2_{\mathcal{H}} = \alpha^T K\alpha$ is growing more or less linearly. If the mean error per point is also constant, we have to chose C instead of C/N to keep the balance between the two terms, i.e. to make the regularization parameter independent of the number of training points.

3.5.2 Efficient optimisation

The dual form of the fixed offset SVR (3.4) has a structure like

$$\begin{aligned} \min \quad & x^T K x + l^T x \\ \text{s.t.} \quad & 0 \leq x \leq C \end{aligned} \tag{3.5}$$

This is a convex quadratic optimisation problem with the only constraint, that the x are within an axis-aligned box of size C. K is a symmetric, positive semidefinite matrix. By construction (compactly supported kernel and subsampling procedure) K is sparse and has a limited condition. One can therefore imagine the optimisation problem as a descent in a quadratic valley which is more or less spherical (see Figure 3.5).

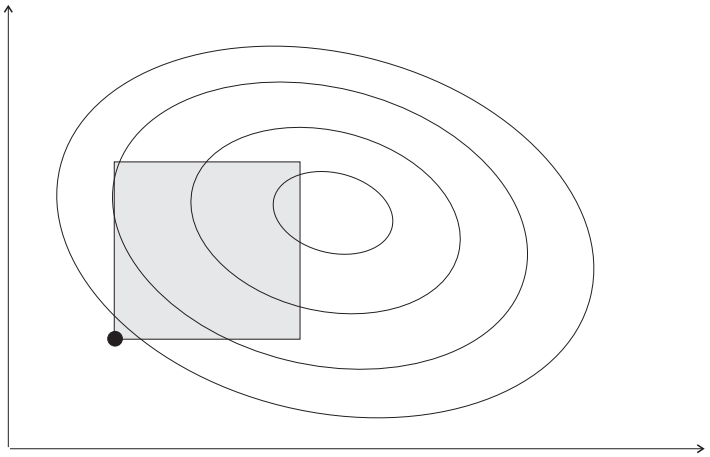


Figure 3.5: *Sketch of the simplified optimisation problem: The grey region shows the valid coefficients, the ellipses depict the contour lines of the quadratic objective function. Note that for the standard ϵ-insensitive SVR the valid region for the coefficients would reduce by one dimension due to the linear constraint. This renders that problem somewhat more complex then the proposed one. In our method we can solve subproblems of dimension one at a time instead of having to solve two dimensional subproblem in each step like in SMO [Platt, 1998].*

The fist choice of optimiser for quadratic problems is often a conjugated gradient (cg) method. However, in the present setup this method would not make sense. Because we do not want to leave the box shaped feasibility region we would have to limit the step size of the cg iteration or change the direction of an update. In both cases subsequent directions of the optimisation would not be conjugated any more. However, this property is the key to why cg methods typically perform so well.

For the specific problem (3.5) we propose to use a very simple coordinate descent scheme, i.e. we optimise the function in one axes aligned direction at a time and then cycle through all dimensions. This method is trivial to implement.

Let us now look at one dimensional optimisation step more closely. The objective function as it depends only on one coefficient looks like

$$F(x_i) = \frac{1}{2} x_i K_{ii} x_i + x_i \sum_{j \neq i} K_{ij} x_j + l_i x_i + const(x_{j \neq i})$$

The optimality condition gives

$$\begin{aligned} \frac{\partial F(x_i^{new})}{\partial x_i} = 0 &= x_i^{new} K_{ii} + \sum_{j \neq i} K_{ij} x_j + l_i \\ \Rightarrow x_i^{new} - x_i &= \frac{-l_i - \sum_{j \neq i} K_{ij} x_j}{K_{ii}} \end{aligned} \tag{3.6}$$

If x_i^{new} lies outside the box constraints, we take the closest feasible point to be the new coefficient.

This method is in fact almost identical with the symmetric Gauß-Seidel method for solving the linear system $Kx = -l$. Solving that linear system is equivalent to minimizing $\frac{1}{2}x^T Kx + l^T x$ which is almost what we do. The only difference is our box constraint which limits the update steps.

The method converges at least asymptotically: In each step we are potentially decreasing the objective value, but never get worse. At least once in a cycle through all directions, we will make a move if we are not at the minimum yet. There are no local minima, so eventually the algorithm will converge. For more details see [Luo and Tseng, 1992]. Because the condition of the problem is quite good, the algorithm will converge very fast. In our experiments typically 10 to 20 rounds over all coefficients already gave an optically good result.

The single update steps (3.6) can be computed very efficiently, with an effort just depending on the number of entries per line in the kernel matrix. If m is the maximal number of nonzero entries in a line of K, we get a single optimisation in $O(m)$ and a whole round over all directions in $O(Nm)$. If we run just a fixed number of rounds this algorithm has linear running time. In practice, even problems with N = 10^5 and $m = 200$ could be solved in few seconds.

Additional improvement for real time computations could be achieved by using an active set shrinking technique (see e.g. [Chang and Lin, 2005]). But for our purpose a training time of a couple of seconds was already good enough.

How to choose a good stopping criterion? We could say that we are close to the optimum if the improvement in a whole optimisation round over all coefficients is smaller than some threshold ϵ_s. The improvement in a single step (3.6) is given by

$$\Delta F = F(x_i^{new}) - F(x_i) = \frac{1}{2} K_{ii} (x_i^{new,2} - x_i^2) + (x^{new} - x_i)(\sum_{j \neq i} K_{ij} x_j + l_i)$$

This value can be computed for free as the potentially costly part, the kernel summation, is already done in order to obtain x_i^{new}.

What is a good value for the stopping threshold ϵ_s? The value of the objective function scales proportional to the number of training points N and to (C + 1), i.e. the loss function and the regularization term. Therefore ϵ_s should be proportional to $N(C + 1)$ as well. It should also depend on the desired fitting accuracy ϵ as well as on the scale of the kernel σ. We therefore propose to set $\epsilon_s \equiv 0.05N(C + 1)\epsilon\sigma$. The factor 0.05 is chosen empirically to give best results. The method proposed results in about 15 iterations of the coordinate descent method independent of the number of training points. This seams reasonable when comparing it to the by construction good condition of the kernel matrix. We additionally limited the number of iterations to range from 5 to 200.

In general, the first few iterations seem to be the most important ones, as the visual results look very similar once the round count has passed a small fixed number. Therefore it is not specifically important to chose the "correct" stopping criterion. Also the run-times of the whole fitting process depend only to a small part on the thus optimised regression computation itself (see Section 3.7).

Although this training algorithm is linear in the number of training points N, we still need to construct the kernel matrix which is itself a $O(N^2)$ process if implemented naively. We apply a fast nearest neighbour searcher [Merkwirth et al., 2000] that after a preparation time of $O(N \log N)$ can extract the m neighbours of a training point that have nonzero kernel values in time $O(m \log N)$. Constructing the kernel matrix therefore takes only $O(mN \log N)$ time.

3.6 Visualizing the implicit surface

There are basically two ways to visualise an implicit surface: One is the extraction of a surface mesh from the implicit representation. The mesh can then be plotted with standard tools. The other method is direct ray tracing. In our work we focused on the former.

3.6.1 Fast evaluation of the kernel expansions

Whatever we do, we have to compute many function evaluations on the way. The trivial implementation thereof has a runtime of $O(N)$ where N is the number of expansion coefficients of the implicit surface function. We will show how to reduce the work to $O(\log N)$.

We are working with compactly supported kernels, so most of the kernel summands will be zero for a given query point. In order not to compute all distances which would be order $O(N)$ work, we apply a fast nearest neighbour search [Merkwirth et al., 2000]. It has a running time of $O(m \log N)$ where m is the number of neighbours within the kernel support. By construction, m is bounded by a small constant as the training points lie approximately evenly distributed on the surface due to the subsampling process. Having collected all possible non zero kernel expansion points, it is then straight forward to compute the kernel expansion in $O(m \log N)$ time.

3.6.2 A fast marching cubes algorithm

The standard algorithm for mesh extraction from implicit surface functions is the marching cubes algorithm [Lorensen and Cline, 1987]. It subdivides the whole plotting volume into small cubes of equal size. For each cube it computes the sign of the implicit surface function on all 8 corners. If not all of them share the same sign, then a piece of the surface is passing through that cube. Triangles describing the geometry are extracted from a precomputed hash table where the 8 sign bits of the corner function values serve as the key. The exact positions of the vertices on the edges of the cube are determined by linear interpolation of the corner function values.

Let the plotting box be subdivided into M pieces in each direction, giving M^3 cubes in total. A straight forward implementation of the marching cubes algorithm would then use $O(M^3)$ function evaluations. However, the surface constructed will only contain $O(M^2)$ triangles, i.e. a number roughly proportional to the surface area.

Improvement is achieved with a surface following version. Given a starting cube, we examine the neighbouring cubes if and only if our starting cube contains a piece of surface. We now have to keep track of the cubes which have already been visited. Let their number P be of order $O(M^2)$. We could either store a flag for every possible cube providing a fast $O(1)$ lookup but an $O(M^3)$ memory requirement. Alternatively, we could store the visited cubes in a list which requires just $O(P)$ memory but also $O(P)$ time for a lookup. A compromise can be achieved by using sorted lists for the storage of visited cubes. The memory requirement remains linear in the number of visited cubes and the insertion and lookup can be done in $O(\log P)$ [STL, 1994]. The whole improved marching cubes algorithm then has a runtime scaling like $O(2M^2 \log M \log N)$.

There are other more sophisticated proposals in the literature which produce an optimal number of triangles with specific properties. In regions of high curvature one can for example produce more triangles than in flat regions which results in a geometry adaptive surface extraction [Karkanis and Stewart, 2001].

3.7 Experimental results

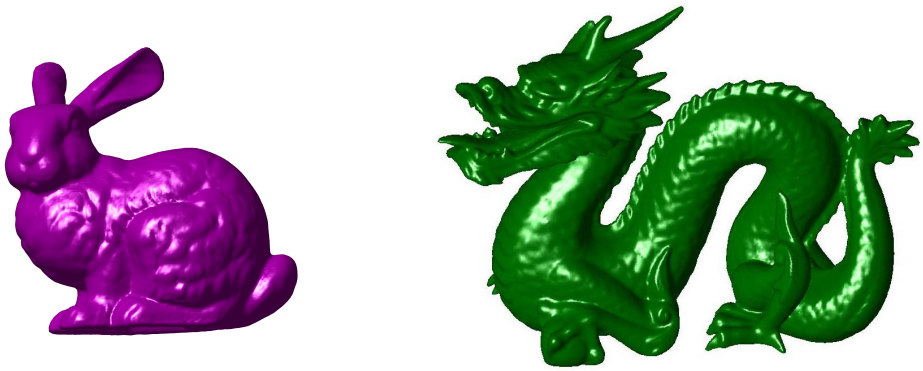


Figure 3.6: *An implicit surface reconstruction of the bunny and the dragon of the Stanford 3D Scanning Repository. The original mesh of the dragon has more than 437k data points. Our reconstruction uses 280k kernel centres.*

We have tested our method for surface reconstruction mainly on head datasets (see Figure 3.7, 3.8, 3.10, 3.11) but also applied it to other 3D objects, such as models of the Stanford 3D Scanning Repository (see Figure 3.6) or especially created objects from psychophysical research (see Figure 3.9).

Denoising The standard reconstruction problem includes noise on the data points. All surface reconstruction methods in the literature use a quadratic loss function to evaluate the fit between the model and the data. This implicitly assumes Gaussian noise. The here proposed method on the other hand works with an ϵ-insensitive loss function which is optimal for a noise model with heavier tails and in general more robust to some outliers. Large deviations are weighted less strongly by the linear error measure. Therefore, the objective is not completely dominated by a few outlier data points.

To demonstrate the effect of such a loss function we reconstructed a head from a dataset with huge outliers (see Figure 3.7). The ϵ-insensitive loss function yielded a good result with almost standard parameters. We also implemented our code with a quadratic loss function.

For this setting we could not find a any parameter setting which yielded comparable results. Spurious surfaces caused by the outlier noise could not be removed in this setup.

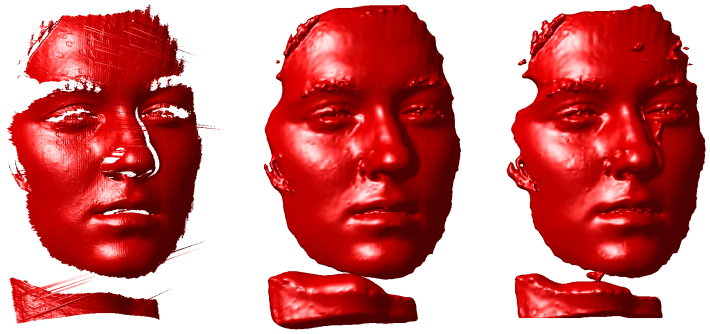


Figure 3.7: *The outlier noise in the left scan was automatically removed with our method using a linear loss function (centre). For the squared loss function (right) even the best parameter set produced some spurious surfaces.*

Smoothing If the data term is weighted highly, our method allows for a very exact reconstruction (Figure 3.8). In order to smooth the results one can either refit the whole surface with a regularization parameter putting less weight on the data term or just leave the more detailed scales aside when plotting the implicit surface function.

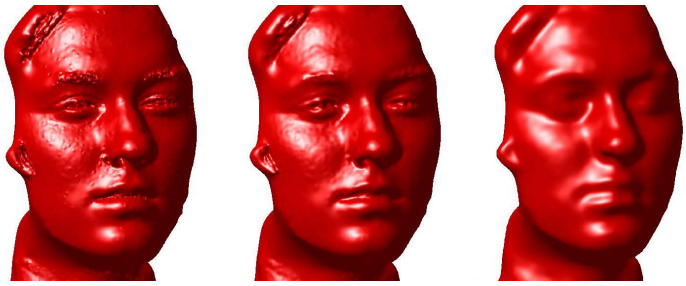


Figure 3.8: *A head model reconstruction with different regularization parameters. The left image uses the parameters chosen by the automatic validation procedure proposed in Section 3.8, which leads to accurate results. For some applications, a smoother surface (centre) may be preferable. Additional smoothing without re-fitting can be obtained by just excluding the finest scales during evaluation as demonstrated in the right figure.*

The smoothness properties of the surface can also be used to determine differential geometric features from the surface (see Figure 3.9). Given a discrete mesh we can fit an implicit representation to it and compute e.g. the mean or Gaussian curvature from it.

Smooth, structure less surface parts of the original data often show some bumpiness in the reconstructions which may be due to the fact that we are just using point constraints. This is a rather common phenomenon also to other approaches. For an in-depth review and possible ways to fix the problem see [Shen et al., 2004].

Hole filling The reconstructed surfaces nicely fill holes in the original dataset (see Figure 3.10). The extrapolated surface parts look similar to what a human would have inserted intuitively (see also Figure 3.11).

Other researchers (cf. [Carr et al., 2001]) have discarded compactly supported base functions because of their supposedly bad extrapolation properties. However, in a multi-scale frame-

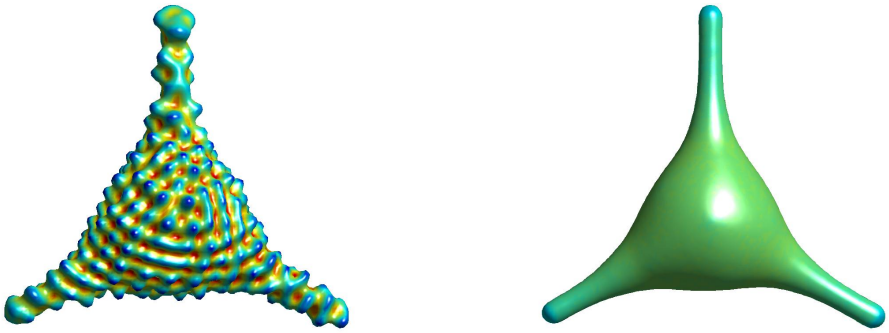


Figure 3.9: *Two different 3D objects where the colour codes the mean curvature. These objects were used for psychophysical research [Cooke et al., 2005].*

Figure 3.10: *Holes due to occlusions in the scanning process (left) are filled by the implicit surface (right).*

work these effects are limited and we can enjoy the intrinsically fast training and evaluation resulting from sparse computations. In our work, the influence of one point is only local, which seems to be a reasonable assumption. If one uses thin-plate splines like [Carr et al., 2001] do, the effect of a point close to the left ear might heavily influence the implicit surface function in the area of the right ear or near the nose. It is not clear why that should be sensible and how this can lead to a stable implicit surface function.

Signed distance function Certain applications rely on the fact that our implicit representation locally approximates the signed distance function. We evaluated the quality of this approximation by plotting the function value for points shifted along the surface normals (Figure 3.12). The diagram indicates that the function values are a good approximation to the signed distance up to about 15% of the object's diameter. The validity range can be increased by including larger kernels into the multi-scale iteration.

Speed We reconstructed the head in Figure 3.8 from 180k input points on a 2.2 GHz AMD CPU with 1 GB memory. We achieved the timings shown below when using 6 scales and starting with σ_0 equal to half of the object diameter. The fitting accuracy was set to 10^{-3} of the object diameter.

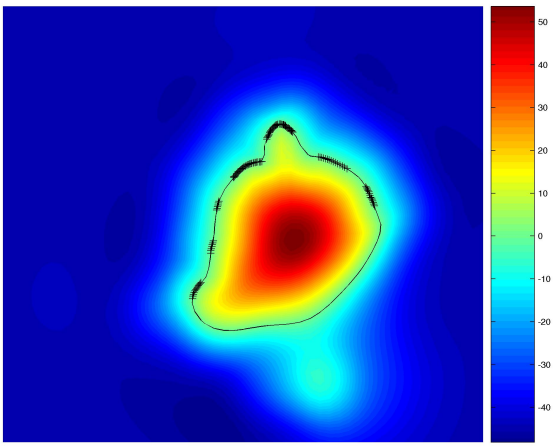


Figure 3.11: *A horizontal cut through the head (Figure 3.10) at the height of the nose (nose pointing upwards). The function values are coded in colours. The black contour depicts the estimated surface and the black crosses symbolise the original training data points. The surface extrapolates nicely into regions with few or no surface points.*

Scale	1	2	3	4	5	6
Sigma [max object width]	1/2	1/4	1/8	1/16	1/32	1/64
N						
extracted	104	380	1368	4991	17917	60128
discarded because function value good enough	0	4	111	877	7359	39707
discarded because normal invalid	76	169	352	844	1867	3297
training points	132	583	2162	7384	19249	37545
rounds until stopping criterion reached	12	4	6	3	3	3
Time [s] for						
subset extraction	0.18	0.25	0.34	0.36	0.46	0.67
training point generation	0.15	0.34	0.69	1.76	4.87	12.86
comp. kernel matrix	0.06	0.08	0.26	0.83	1.78	2.33
optimisation	0.04	0.03	0.07	0.08	0.13	0.15
Sum times [s]	0.44	0.70	1.36	3.03	7.25	16.01

The total combined reconstruction time is about 30 sec for the 180k point model including all steps like subset extraction, training point generation, kernel matrix computation and regression. Our implementation of the marching cubes algorithm needed about 21 sec for this model while producing 44k vertices and 89k faces for a plotting accuracy - i.e. minimal cube size - of $\frac{1}{150}$ of the object diameter.

We tried to compare the implementation of our method with the state-of-the-art FastRBF code of [Carr et al., 2001]. Trial versions of this system are available in public domain. The authors of [Carr et al., 2001] have told us that there is a commercial multi-scale version of their system, but the code available to us did not include that. It produced large numbers of RBF centres which made a fair comparison difficult. For a head model with 50k input points the runtime was more than 7 min using more than 100k RBF centres. In cases where both methods used the same number of kernel centres for optimisation, our code ran about two to three times faster than the FastRBF code. This is probably due to the fact that our

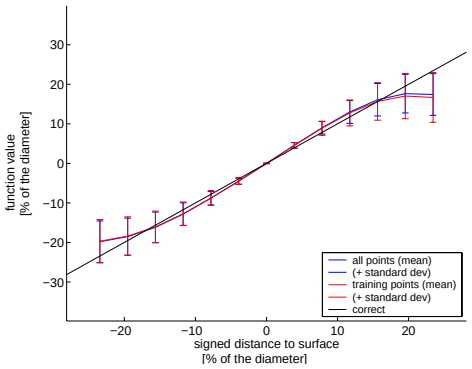


Figure 3.12: *The implicit surface function f of Figure 3.8, plotted along the surface normals, reproduces the signed distance function within a range of 15% of the object diameter. Red: values for training points, blue: generalization to all surface points. Both lines are almost identical. The black line shows the true signed distance.*

optimisation procedure is extremely simple (in fact, the core procedure is only 15 lines of C++ code) whereas [Carr et al., 2001] use sophisticated approximation schemes to handle non-compactly supported RBF functions.

Adaptivity The number of kernel centres adjusts nicely to the intrinsic complexity of the surface. In the table below we reconstructed different sub-samplings of the head model in (Figure 3.8) with an accuracy of 10^{-3}. Flat regions can already be approximated with a few large kernels, so a larger number of input points does not increase precision. Thus, in our experiment the number of used kernel centres saturates at some point while the input point density is still increasing but the implied geometry stays the same. If, on the other hand, the structures of the object are more complex (e.g., the dragon model in Figure 3.6), then the number of required kernel centres increases significantly.

Object	# Points	Time [s]	# Kernel centres
Head	2509	3.4	7797
Head	15k	12.4	20297
Head	51k	23.3	25563
Head	178k	34.9	30162
dragon	437k	289	284003

3.8 Automatic parameter selection

We propose an automated hierarchical cross validation procedure to select the necessary parameters. The only parameter to be chosen manually is the fitting accuracy ϵ.

In order to chose the regularization parameters – one on each scale – we randomly select 20% of the given surface points as validation data and construct the implicit surface function based on the other 80%. The quality of a fit is evaluated through the mean absolute function value on the validation set. For a good fit this measure should be small, close to the noise level. If the data term is weighted too heavily we encounter over-fitting. If on the other hand the regularization term is weighted too much, the surface points are not approximated well.

In order to judge whether this error measure agrees with visual inspection we constructed one head with different regularization weightings (see Figure 3.13). The minimum of the loss

function nicely corresponds to the visually most appealing candidate.

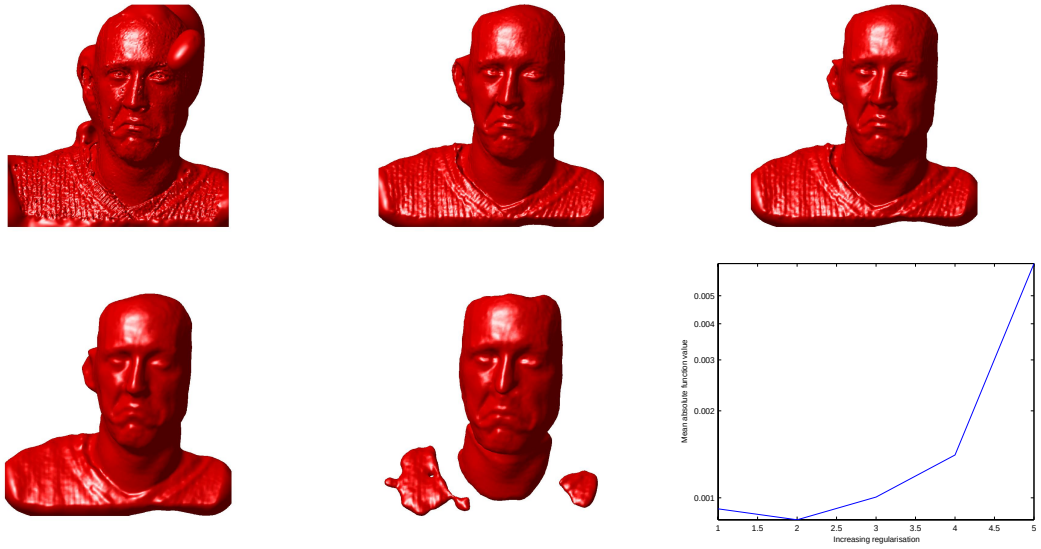


Figure 3.13: *The same head fitted with increasing weight on the regularization term. The diagram in the last figure shows that the mean absolute function value corresponds to the visual rating.*

We have to determine one regularization parameter C for each scale. We propose to use a hierarchical selection scheme. For each scale in order of decreasing size, we test different regularization parameters and take the best parameter from that scale for the computations of the next refined scales. This way, the number of necessary trials is linear in the number of scales. If we selected all parameter in the same step, we would have to test exponentially many parameter combinations. Even though the hierarchical approach is not guaranteed to find the globally optimal parameters, it seems a reasonable assumption that a good fit performs well after each scale, not just after the final one.

We applied the proposed parameter selection method to the dataset in Figure 3.13. We computed the parameter-error plots depicted in Figure 3.14 from which the optimal parameters were chosen. The finally constructed surface is shown in Figure 3.15. The automatic parameter selection procedure tends to yield rather exact reconstructions of the given dataset.

The starting kernel width σ_0 determines how far out from the surface the signed distance field still makes sense. Larger σ_0 also help to increase the hole-filling capacities of the implicit surface function. We typically chose it to be one half of the object diameter.

There are two ways to chose the number of used scales: One can either compute the regression on finer and finer scales, until all surface points – or at least a large fraction of them – are within the desired fitting accuracy ϵ. The other possibility is to choose a fixed number of scales. The kernel width of the finest scale should match the smallest surface features that we want to recover. We typically chose the latter approach. For a head model around 6 scales seemed to be optimal.

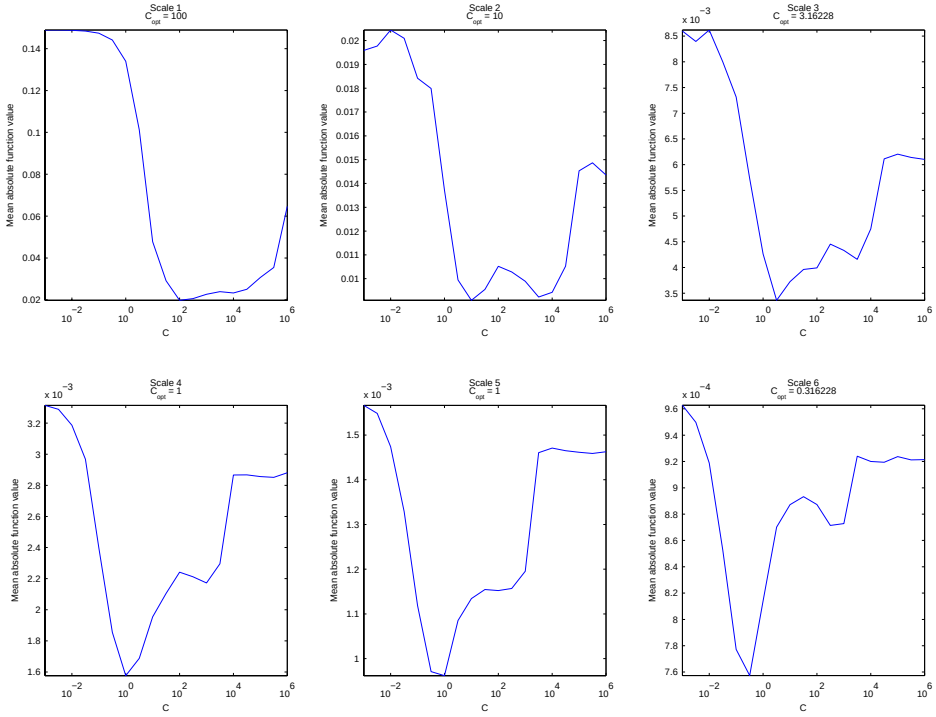


Figure 3.14: *The mean absolute function value in units of object diameter vs. the regularization parameter C. High C values put the weight of the objective function onto the data dependent term, low values onto the regularization.*

Figure 3.15: *The head reconstructed with the parameters chosen by the validation process and* $\sigma_0 = 0.5$ *in units of the object diameter.*

3.9 Conclusions

The presented method to reconstruct implicit surfaces is a rather straight forward application of Support Vector Regression onto the implicit surface reconstruction problem.

By exploiting the special structure of this low dimensional regression problem, we are able to build a very efficient algorithm. The performance is similar to the state-of-the-art in computer graphics. As a special feature our approach uses the ϵ-insensitive loss function which performs superior to standard quadratic losses if the dataset contains outlier noise.

Future work may look at applications of implicit surfaces in higher dimensional spaces. In many control applications variational optimisation problems on high dimensional manifolds occur. One could try to model these manifolds implicitly and solve the corresponding optimisation problems efficiently with algorithms that make use of the special structure of such a manifold description.

Chapter 4

Denoising

4.1 Introduction

How can one use an implicit surface function for denoising? We have already looked at how to construct a smooth surface from noisy surface points in Chapter 3. This surface is assumed to be similar to the original data generating surface. We will now use it to reduce the acquisition noise in the given surface points.

Formally stated, we will try to solve the following problem: Given m points $z_i \in \mathcal{X} \subseteq \mathbb{R}^D$ which are generated from true surface points x_i like $z_i = x_i + \epsilon_i$, with the ϵ_i drawn iid from some noise model. We want to reconstruct the original surface points x_i. As this might not be uniquely possible, e.g. shifts along a surface cannot be reversed, we will construct m points $\hat{x}_i$ such that they are close to the original surface and $\|\hat{x}_i - x_i\|$ is small.

For many applications we just need surface points at more or less arbitrary positions. For example we need to extract uniformly spaced surface points for a regular mesh construction. If this is the case, then we can neglect the input points z_i and just construct new ones from the estimated implicit surface using for example the marching cubes algorithm (see Section 3.6.2). However, if the original positions of the input points z_i carry more information than just where the surface should be, then they should be retained as much as possible while still filtering out the acquisition noise.

In order to reduce the noise of the input points $z_i \in \mathcal{X}$, we will project them onto the estimated implicit surface. The surface is a non-linear manifold in the input space $\mathcal{X}$, however, in the reproducing kernel Hilbert space $\mathcal{H}$, the implicit surface function $f : \mathcal{X} \to \mathbb{R}$, $f(x) = \langle w|\phi(x)\rangle_{\mathcal{H}} + b$ can be interpreted as hyper-plane (see Chapter 2). Thus, it may be easier to project the points z_i onto a plane in $\mathcal{H}$ than projecting directly in the input space $\mathcal{X}$.

We will examine methods of both kinds as well as other kernel methods – namely kernel principal component analysis (KPCA). We will first describe the theory for each method in the following sections and then give results in Section 4.6.

Experiments in this chapter were conducted at an early stage of this thesis' work. At that time, we were still working with Gaussian kernels, which do not yield sparse kernel matrices. The experiments therefore were limited to smaller sample sizes (up to $m \approx 1000$). We also just used one scale for reconstruction. However, we are convinced that the findings of this chapter will also transfer to the new, more powerful description introduced in Chapter 3 without significant differences.

4.2 Projection to hyper-plane in $\mathcal{H}$

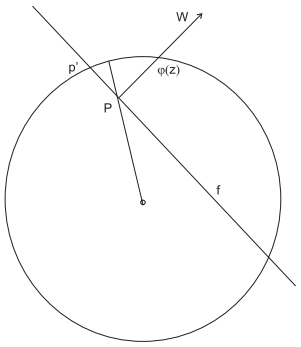


Figure 4.1: *The figure shows the unit sphere in $\mathcal{H}$ as well as a hyper-plane $f(\phi(x)) = \langle w|\phi(x)\rangle_{\mathcal{H}} + b = 0$ representing an implicit surface. The orthogonal projection of $\phi(z)$ onto the implicit surface hyper-plane is p, the spherical projection (see below) is p'.*

We first build a smooth implicit surface, i.e. a hyper-plane in $\mathcal{H}$, using all the given points (see Chapter 3 or [Schölkopf et al., 2004]). Then for any noisy point $z \in \mathcal{X}$ we proceed as follows to get a reduced noise version $\hat{x}$:

- Map z into $\mathcal{H}$ using the kernel mapping $\phi : \mathcal{X} \to \mathcal{H},\ \phi(z) = |k_z\rangle$.
- Project the $\phi(z)$ onto the hyper-plane in $\mathcal{H}$ to get $p \in \mathcal{H}$ (orthogonal or spherical projection, explained below).
- Find an (approximate) pre-image $\hat{x}$ for the projected point p, i.e. find that $\hat{x} \in \mathcal{X}$ for which $\|\phi(\hat{x}) - p\|_{\mathcal{H}}$ is minimal. Thus, we have to (approximately) invert the kernel mapping $\phi : \mathcal{X} \to \mathcal{H}$.

4.2.1 Orthogonal projection

For projecting $\phi(z)$ on to the hyper-plane $f(\phi(x)) = \langle w|\phi(x)\rangle_{\mathcal{H}} + b$ to get point $p \in \mathcal{H}$, we have the following conditions:

$$\begin{aligned} p = &\quad \phi(z) + \lambda w \qquad \text{(orthogonal projection)} \\ 0 = f(p) = &\quad \langle w|p\rangle_{\mathcal{H}} + b \qquad \text{(new point } p \text{ should be on the plane)} \end{aligned}$$

They yield

$$p = \phi(z) - \frac{f(z)}{\|w\|^2_{\mathcal{H}}} w. \tag{4.1}$$

This formula defines a vector $p \in \mathcal{H}$ which may be infinite dimensional and we therefore cannot compute all its components. But as we will see below, we will just need to evaluate dot-products between p and other vectors $\phi(x)$ in order to extract all the needed information. As w is given as a finite sum $w = \sum_i \alpha_i \phi(z_i)$, so is p. Therefore we can evaluate these dot-products as finite kernel expansions using the linearity of the inner product and the kernel equation (see Chapter 2)

$$\langle \phi(x)|\phi(z)\rangle_{\mathcal{H}} = k(x, z).$$

A similar argumentation will also hold for other equations below that deal with vectors in $\mathcal{H}$.

4.2.2 Pre-imaging

Inverting the implicit kernel mapping ϕ is a non-trivial problem [Schölkopf and Smola, 2002, p.544]: Not every point in $\mathcal{H}$ has a pre-image in the input space $\mathcal{X}$ because the dimension of $\mathcal{H}$ is always much higher than the dimension of the input space, potentially even infinite.

As we most often cannot invert the kernel mapping exactly, we are searching for an approximate pre-image, i.e. a point $\hat{x} \in \mathcal{X}$ for which the distance $\|\phi(\hat{x}) - p\|_{\mathcal{H}}$ is minimal. Thus we define

$$\hat{x} = \underset{y \in \mathcal{X}}{\operatorname{argmin}} \|\phi(y) - p\|_{\mathcal{H}} = \underset{y \in \mathcal{X}}{\operatorname{argmax}} \langle \phi(y) | p \rangle_{\mathcal{H}} \,. \tag{4.2}$$

Here, we have assumed a radial basis kernel with $k(y, y) = k(0) = 1$ for all $y \in \mathcal{X}$.

We compute $\hat{x}$ using gradient descent to minimise (4.2). This method has no global convergence property. But it works quite well, if we start with a reasonable initial guess. One choice for a starting point is the the noisy point z which we expect not to be too far from the noise free version x and therefore also not from $\hat{x}$.

An alternative method to find the pre-image $\hat{x}$ is the fix point iteration proposed in [Schölkopf and Smola, 2002, p.547]. However, the region of convergence for this method seemed to be smaller in our experiments than for directly minimizing (4.2).

4.2.3 Spherical projection

If a radial basis kernel is used – we typically do so –, all points in $\mathcal{H}$ that potentially could have an exact pre-image are lying on the unit sphere, because for all $x \in \mathcal{X}$ we have $\|\phi(x)\|^2_{\mathcal{H}} = k(x, x) = k(0) = 1$. The orthogonally projected point p is not on the unit sphere, so it cannot possibly have an exact pre-image.

In order to be able to find a good pre-image for p, one could instead of the orthogonal projection use a projection along the unit sphere: As the spherically projected point p' we use the closest point to $\phi(z)$ that is on the unit sphere and also on the hyper-plane (see Figure 4.1). This yields the following conditions:

$$\begin{aligned} p' &= \mu\phi(z) + \nu w && \text{(closest point)} \\ \|p'\|^2_{\mathcal{H}} &= 1 && \text{(on the sphere)} \\ f(p') &= \langle w | p' \rangle_{\mathcal{H}} + b = 0 && \text{(on the plane)} \end{aligned}$$

One can solve this system for the two coefficients μ, ν yielding

$$\begin{aligned} \mu &= \pm\sqrt{\frac{\|w\|^2_{\mathcal{H}} - b^2}{\|w\|^2_{\mathcal{H}} - \langle w | \phi(z) \rangle^2_{\mathcal{H}}}} \\ \nu &= \frac{-b - \mu \langle w | \phi(z) \rangle_{\mathcal{H}}}{\|w\|^2_{\mathcal{H}}} \end{aligned}$$

For $\|w\|^2_{\mathcal{H}} >> \langle w | \phi(z) \rangle^2_{\mathcal{H}}, b^2$ which is normally fulfilled we can simplify it to $\mu \approx 1 - \frac{b^2}{2} + \frac{\langle w | \phi(z) \rangle^2_{\mathcal{H}}}{2}$.

We have to keep in mind that the unit sphere in a high dimensional space $\mathcal{H}$ is itself a high dimensional manifold. Therefore, not even all the points on the unit sphere will have an exact pre-image. Yet, one might hope that using the spherical projection method the chances are higher.

4.3 KPCA

As an alternative method to the above feature space projection methods we propose *kernel principal component analysis* (KPCA). KPCA works without the implicit surface functions.

Principal component analysis (PCA) has been successfully used in all kinds of denoising applications: The given data points are supposed to lie normally distributed in some space. The idea is to find a small set of orthogonal principle component vectors (PCs) that optimally span the data space. The empirical correlation matrix is diagonalised to yield these vectors. Typically they are ordered by descending eigenvalue. The hope is that the first PCs span the space of real data variation and later PCs just describe additional noise. A noisy point can then be projected onto that span of the first PCs to filter out the noisy components. More detailed descriptions can be found in any machine leaning book, e.g. [Schölkopf and Smola, 2002, p.427].

KPCA is very similar to PCA. The points are first projected into the reproducing Hilbert space $\mathcal{H}$ and only then PCA is applied. In total, the PCs are then vectors in $\mathcal{H}$, i.e. non-linear features.

Say we are given points in three dimensions that are sampled from a plane. If we compute the PCA, the first two eigenvalues will be big, the third one will be small. The first two PCs will span the plane and the third one will point into an orthogonal direction. Here we assume, that our the surface points z_i lie on a plane if mapped into the RKHS $\mathcal{H}$. Therefore, it may be reasonable to try KPCA to find the subspace, that corresponds the hyper-plane. We can then denoise points, by projecting them in $\mathcal{H}$ onto the first PCs.

KPCA restricts the $\phi(z)$ to lie in a finite dimensional subspace of $\mathcal{H}$. Thus, it restricts the $\phi(z)$ much stronger than the hyper-plane projection methods explained above. There, only one degree of freedom is constrained. Using the hyper-plane projection methods the denoised points $\hat{x}_i$ are often identical to the noisy ones z_i as can be seen in the experimental Section 4.6. One reason may be that the $\phi(z)$ are not constraint enough in $\mathcal{H}$. KPCA therefore seems a good way to proceed.

Our implementation follows directly the method explained in [Schölkopf and Smola, 2002, p.429]: We use a shifted version of the KPCA as our data points are not centred in $\mathcal{H}$. We plug $\hat{\phi}(z_j) = \phi(z_j) - \frac{1}{m}\sum_{i=1}^{m}\phi(z_i)$ into a normal KPCA and diagonalise the matrix

$$\hat{K} = K - 1_m K - K 1_m + 1_m K 1_m$$

where $K_{ij} = k(z_i, z_j)$ and $(1_m)_{ij} = 1$. The eigenvectors u_k then yield the principal components

$$C_k = \sum_{i=1}^{m} u_{k,i}\hat{\phi}(x_i) = \sum_{i=1}^{m}\left(u_{k,i} - \sum_{j=1}^{m} u_{k,j}\right)\phi(x_i).$$

In order to denoise a point $z \in \mathcal{X}$, we first map it to $\mathcal{H}$ yielding $\phi(z)$, and then project it onto the span of the first k PCs

$$p'' = c + \sum_k C_k \langle C_k | \phi(z) - c\rangle_{\mathcal{H}} = \sum_k C_k \langle C_k | \phi(z)\rangle_{\mathcal{H}} + C_{off}$$

where $c = \frac{1}{m}\sum_{i=1}^{m}\phi(z_i)$ and $C_{off} = c - \sum_k C_k \langle C_k | c\rangle_{\mathcal{H}}$. Having computed p'' we then use the pre-imaging procedure as above to get a denoised version of $\hat{x}$ in the input space $\mathcal{X}$.

4.4 KPCA on the hyper-plane

When using the KPCA algorithm we are neglecting the information contained in the implicit surface f. One idea to include such a knowledge into a KPCA-like algorithm is to choose the PCs such that they are all embedded in the hyper-plane.

To do so, we first project the $\phi(z_i)$ orthogonally onto the hyper-plane representing the object's surface using equation (4.1). Then, we use KPCA as above. The only difference is that now the p_i are taken as the input points for a PCA, not the $\phi(z_i)$. Therefore, K is now given as $K_{ij} = \langle p_i | p_j \rangle$.

4.5 Projection in input space $\mathcal{X}$

This method uses the implicit surface function $f : \mathcal{X} \to \mathbb{R}$ but neglects the knowledge about $\mathcal{H}$ and the hyper-plane interpretation. We estimate a smooth implicit surface (see Chapter 3) and then try to project noisy points z onto the implied surface directly in $\mathcal{X}$.

To do so, we search starting from a noisy point z along the gradient direction $\nabla f(z)$ until we find a point with zero function value – a surface point. We assume that z is close enough to the implicit surface, such that the distance field is still valid. The gradient $\nabla f(z)$ then points into the direction that yields an orthogonal projection. Finding the root t^* of $g(t) = f(z + t\nabla f(z)),\ t \in \mathbb{R}$ which has the smallest absolute value $|t^*|$, i.e. is closest to z, is done using the Newton root-finding algorithm. The final denoised point $\hat{x}$ then is $z + t^* \nabla f(z)$.

4.5.1 Connection to the hyper-plane projection methods

There is a close connection between the projection in $\mathcal{H}$ and the one in input space $\mathcal{X}$ (at least for radial basis kernels). In the former, we move along the direction w in $\mathcal{H}$. In the latter, we search in the gradient direction $\nabla f(z)$ in $\mathcal{X}$. We will show that both directions are the same – at least for small step sizes.

The implicit surface function $f : \mathcal{X} \to \mathbb{R},\ f(x) = \langle w | \phi(x) \rangle_{\mathcal{H}} + b = \sum_i \alpha_i k(x, z_i) + b$ implies

$$\begin{aligned} \text{gradient in } \mathcal{H}: \quad & \nabla_{\phi(x)} f(x) &&= w \\ \text{gradient in } \mathcal{X}: \quad & \nabla_x f(x) &&= \sum_i \alpha_i \nabla_x k(x, z_i) \end{aligned}$$

Let $I : \mathcal{H} \to \mathcal{X}$ be the *pre-image function*, i.e. for every vector $\mu \in \mathcal{H}$ it returns the point $I(\mu) \in \mathcal{X}$ for which $\|\phi(I(\mu)) - \mu\|^2_{\mathcal{H}}$ is minimal. Thus, it is the approximate inverse of $\phi : \mathcal{X} \to \mathcal{H}$. The closest pre-image $z \in \mathcal{X}$ of a vector $\mu \in \mathcal{H}$ is determined by $\|\phi(z) - \mu\|^2_{\mathcal{H}}$ being minimal. This leads to the condition

$$\nabla_z \langle \mu | \phi(z) \rangle_{\mathcal{H}} = 0. \tag{4.3}$$

Theorem 2. *For a differentiable radial basis kernel it is*

$$I(\phi(x) + \theta\, w) = x + c\, \theta\, \nabla_x f(x) + O(\theta^2). \tag{4.4}$$

That implies that the w direction in $\mathcal{H}$ to first order equals the gradient direction in $\mathcal{X}$. If we do not use too large steps, the two methods – projection onto a hyper-plane in $\mathcal{H}$ and a projection onto a non-linear manifold in $\mathcal{X}$ – are very similar.

Proof. Let us examine the special case of a Gaussian kernel $k(x,y) = e^{-\lambda\|x-y\|^2}$. We then have

$$\nabla_x f(x) = -2\lambda \sum_i \alpha_i e^{-\lambda\|x-z_i\|^2}(x - z_i).$$

The pre-image $r_\theta = I(\phi(x) + \theta w)$ fulfils condition (4.3):

$$\begin{aligned}
\nabla_{r_\theta} \langle \phi(x) + \theta w | \phi(r_\theta) \rangle_{\mathcal{H}} &= 0 \\
\nabla_{r_\theta} \left(\langle \phi(x) | \phi(r_\theta) \rangle_{\mathcal{H}} + \theta \langle w | \phi(r_\theta) \rangle_{\mathcal{H}} \right) &= 0 \\
\nabla_{r_\theta} k(x, r_\theta) + \theta \sum_i \alpha_i \nabla_{r_\theta} k(z_i, r_\theta) &= 0 \\
e^{-\lambda\|x-r_\theta\|^2}(x - r_\theta) + \theta \sum_i \alpha_i e^{-\lambda\|z_i-r_\theta\|^2}(z_i - r_\theta) &= 0 \\
\Rightarrow r_\theta &= x + \theta e^{\lambda\|x-r_\theta\|^2} \sum_i \alpha_i e^{-\lambda\|z_i-r\|^2}(z_i - r_\theta)
\end{aligned}$$

This is a recursive equation for r_θ. If we plug in one iteration we get:

$$\begin{aligned}
e^{\lambda\|x-r_\theta\|^2} &= e^{\lambda O(\theta)} = 1 + O(\theta) \\
e^{-\lambda\|z_i-r_\theta\|^2} &= e^{-\lambda\|z_i-x-O(\theta)\|} = e^{-\lambda\|z_i-x\|^2}(1 + O(\theta)) \\
(z_i - r_\theta) &= (z_i - x) + O(\theta) \\
\Rightarrow r_\theta &= x - \theta \sum_i \alpha_i e^{-\lambda\|x-z_i\|^2}(x - z_i) + O(\theta^2) \\
\Rightarrow r_\theta &= x + \theta \frac{1}{2\lambda} \nabla_x f(x) + O(\theta^2)
\end{aligned}$$

This proof would also work for any other differentiable radial basis kernel. □

4.6 Experiments

4.6.1 The test data and the noise model

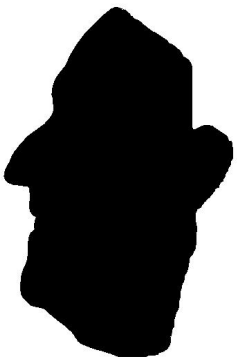


Figure 4.2: *The test data shape*

In this series of experiments we used a "surface" in 2D, i.e. the contour of a shape. This makes the visualisation much easier and also allows for critical speedups as the number of

points m can be kept small ($m \leq 1000$). Nevertheless, we are convinced that the results transfer to the 3D case.

We used the shape in Figure 4.2 to extract surface points as well as surface normals of unit length. An approximately equally spaced set of surface points was taken to be the x_i ($m = 1000$). To simulate acquisition noise we added spherical Gaussian noise to the points x_i as well as to the normals. This then yielded the z_i used for the denoising experiments. The Gaussian width σ_{noise} for noise in the position of the surface points was set to $\frac{1}{300}$th of the object diameter. This yields a point distribution through which the human user can still fit a smooth curve (see Figure 4.3, right). The unit length normals were disturbed with Gaussian noise of width $\sigma = 0.2$ which is comparable to an average angular error of order of $10°$.

The implicit surface function was constructed from the z_i as described in Chapter 3. Because the experiments were done very early in the work for this thesis, we did still use a Gaussian kernel and no multi-scale procedure. We also did not fix the offset b yet, but included it in the SVR optimisation. The width of the used Gaussian kernel σ was set to $\frac{1}{80}$th of the object size. The kernel width is a measure of the size of the smallest surface feature which we want to recover. Variations much smaller than this are flattened out. The regularization parameter was determined by cross validation. The resulting surface smoothly interpolates the noisy points (see Figure 4.3). We think that apart from tremendous speed-ups the results of this section will not change when one uses a more sophisticated SVR-like reconstruction algorithm.

The surface tends to lie a little bit outside the original contour in regions where the surface is bend inwards. The same holds vice versa for outward pointing bends. This is due to the fact that given spherically symmetric noise the centre of mass of the z_i is lying outside/inside the real surface. The optimal reconstruction will follow these points, and thus will show the observed deviations.

All experiments where conducted several times with independently drawn noise. The variability between separate runs gives some notion of how stable the results are. In all runs undertaken there was only little change from one instance to the other.

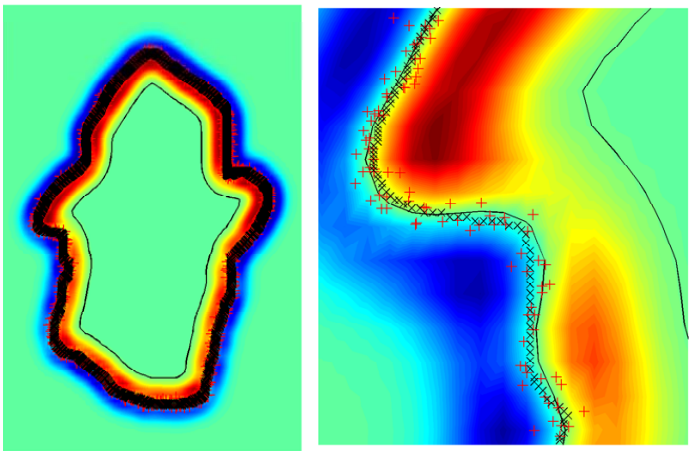


Figure 4.3: *A reconstructed surface. The red points are the noisy ones z_i, the black ones are the original ones x_i. The black line represents the surface and the colour values encode the function value.*

4.6.2 Evaluation measures

To evaluate how well the denoising routines work, we used three different measures:

- (D1) the function value of the implicit surface function at the new, denoised points $\hat{x}_i$. The implicit surface function approximately equals the the signed distance function. Thus this measure returns the distance to the implicit surface. The implicit surface itself is supposed to be noise free, thus, a good denoising routine would yield zero for this error measure.
- (D2) the mean distance of the denoised points $\hat{x}_i$ to the original points x_i. Optimally this error measure would reduce to zero for a denoising routine. However, there are some components of the noise that we cannot expect to be filtered out. Shifts parallel to the surface cannot be recovered. Below we compute expectations of how well a good denoising routine should be able to work given our Gaussian noise model.
- (D3) the mean distance of the denoised points $\hat{x}_i$ to the noisy points z_i. This measure just shows whether the denoising process has any effect at all.

For the measure D2, i.e. the mean distance $\|\hat{x}_i - x_i\|$, we compute two values. First, we compute the expectation of D2 for the given noisy points z_i:

$$a = \int \|v\| \, p_{noise}(v) dv = \sqrt{\frac{\pi}{2}} \sigma_{noise}$$

where $v = z - x$ is the difference of the noisy to the original point and p_{noise} the Gaussian noise model. After denoising we can hope that noise effects orthogonal to the hyper-plane are removed whereas parallel noise cannot be filtered out. Thus, the expectation of D2 after denoising is the expectation of the length of v with one dimension projected out. For any fixed vector n of unit length we have

$$b = \int \sqrt{\|v\|^2 - \langle v|n\rangle^2} p_{noise}(v) dv = 2 \int_{-\pi/2}^{\pi/2} d\phi \int_0^\infty dr r (r \cos\theta) p(r) = \sqrt{\frac{2}{\pi}} \sigma_{noise} = \frac{2}{\pi} a$$

The numerical values in our experiments are a = 78 and b = 50 (in appropriately chosen units that we will use throughout this chapter). These numbers give an upper and a lower bound on what to expect for a good denoising algorithm (The computations are dimension specific. According to our test dataset we give results for two dimensions).

4.6.3 Denoising by projection to hyper-plane in $\mathcal{H}$

For the hyper-plane projection algorithm described in Section 4.2 the following results were obtained using orthogonal projections (We always give the mean and the standard deviation of an error measure evaluated for all the points z_i):

	D1	±	D2	±	D3	±
denoised	47	36	78	41	0.72	0.50
(noisy)	48	36	79	41		

The numbers show that the denoised points $\hat{x}_i$ are almost identical to the given noisy ones z_i. This may be due to the following argument:

p is given as $c_1\phi(z_i) + c_2 w$. One observations is that $\frac{c_2}{c_1} = \frac{\langle \phi(z_i)|w\rangle}{\|w\|^2}$ is generally very small, i.e. on the order of 10^{-3}. The pre-imaging algorithm then faces the following situation: We are searching a pre-image (i.e. a Gaussian bump) that best approximates p (i.e. one big Gaussian bump centred at z_i and many very small ones w). Naturally, the results then is close to the "big bump" z_i.

Given an RBF kernel, all image points lie on the unit sphere in $\mathcal{H}$. But p generally lies within the sphere and is therefore not a got place to look for a pre-image. Therefore we tried spherical projections (see Section 4.2.3). That yielded the following results:

	D1	±	D2	±	D3	±
denoised	47	36	78	41	0.73	0.52
(noisy)	48	36	79	41		

These results are almost identical to the results of the first procedure. They show that even though we are now on the surface of the unit sphere in $\mathcal{H}$, there is still no good pre-image nearby. Again the coefficients of the $\phi(z_i)$ are much larger than the ones for the w.

4.6.4 Denoising by KPCA

Unlike hyper-plane projection methods which constrain just one degree of freedom in $\mathcal{H}$ we can force the projections p'' of a KPCA to lie in a finite dimensional subspace of $\mathcal{H}$. Thus we hope to see some non-trivial results.

The principal components (PCs) we obtained are depicted in Figure 4.4 and Figure 4.5. We plot them by colour-coding the functions $g_C(x) = \langle C|\phi(x)\rangle$. Note that the first PCs describe large scale variations in the data set (see Figure 4.4). Later PCs (see Figure 4.5) encode smaller, more localised variations than can be assumed to be noise.

The $\phi(z_i)$ were projected onto the span of the first k PCs and pre-imaging was applied. Results for different k are shown in the table below and are depicted in Figure 4.6.

k	D1	±	D2	±	D3	±
10	30	26	421	529	421	526
50	18	18	77	55	73	48
100	18	18	59	37	42	31
200	44	33	75	39	6	6
(noisy)	48	36	79	41		

Using very few PCs, there is not enough variability in the model, the points cluster in zones of highest density. The set of denoised points gets more and more distributed along the surface, the more PCs are used in the projection. If the number of PCs is too high, we are not effectively restricting our points any more, the denoised points $\hat{x}_i$ more and more equal the noisy ones z_i. The optimum number of PCs to use seems to be around 50-100 for our example which corresponds to about 5%-10% of the data points.

For k around 100 the error measure D2, i.e. the mean of $\|\hat{x}_i - x_i\|$, is coming close to its optimum value $b = 50$.

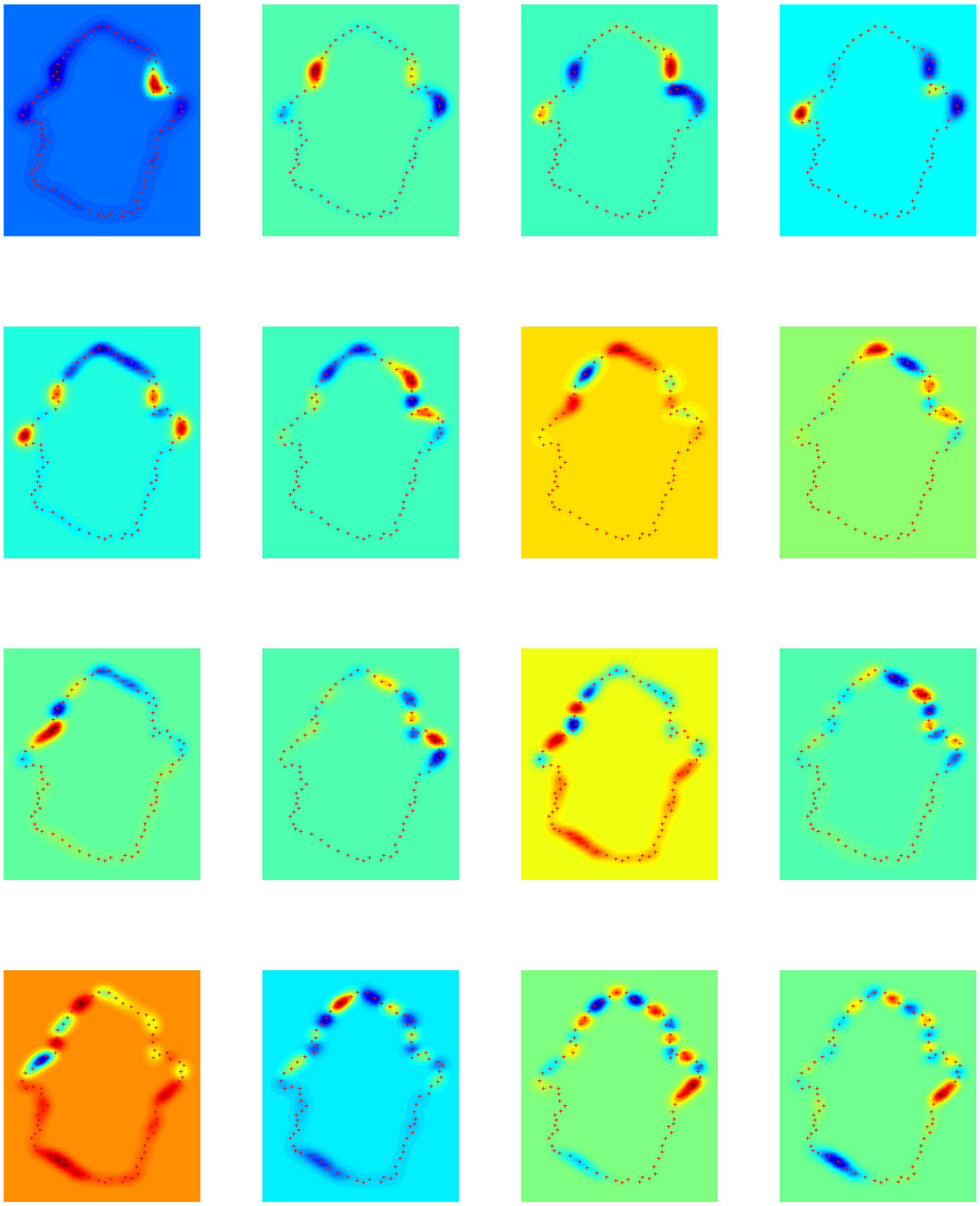


Figure 4.4: *The first 16 principal components we got from a KPCA. For clarity of presentation, the red points just show a subset of the* $m = 1000$ *points used in the experiment.*

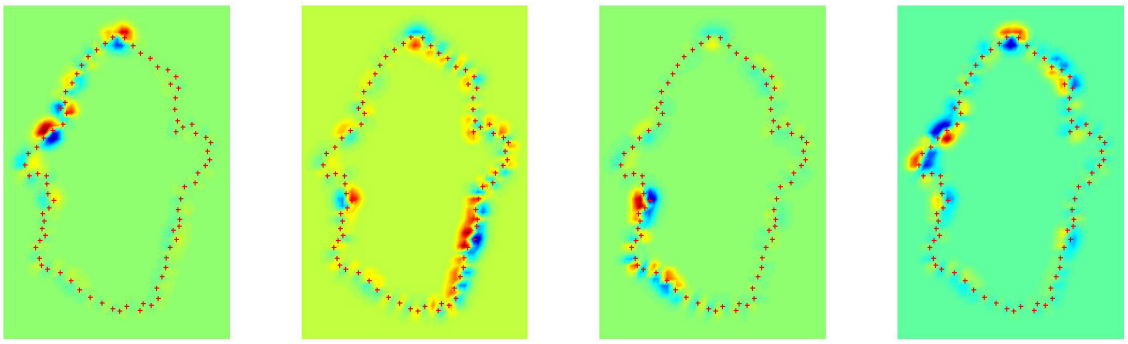


Figure 4.5: *The principal components number 100 to 104 derived from a KPCA. For clarity of presentation, the red points just show a subset of the* $m = 1000$ *points used in the experiment.*

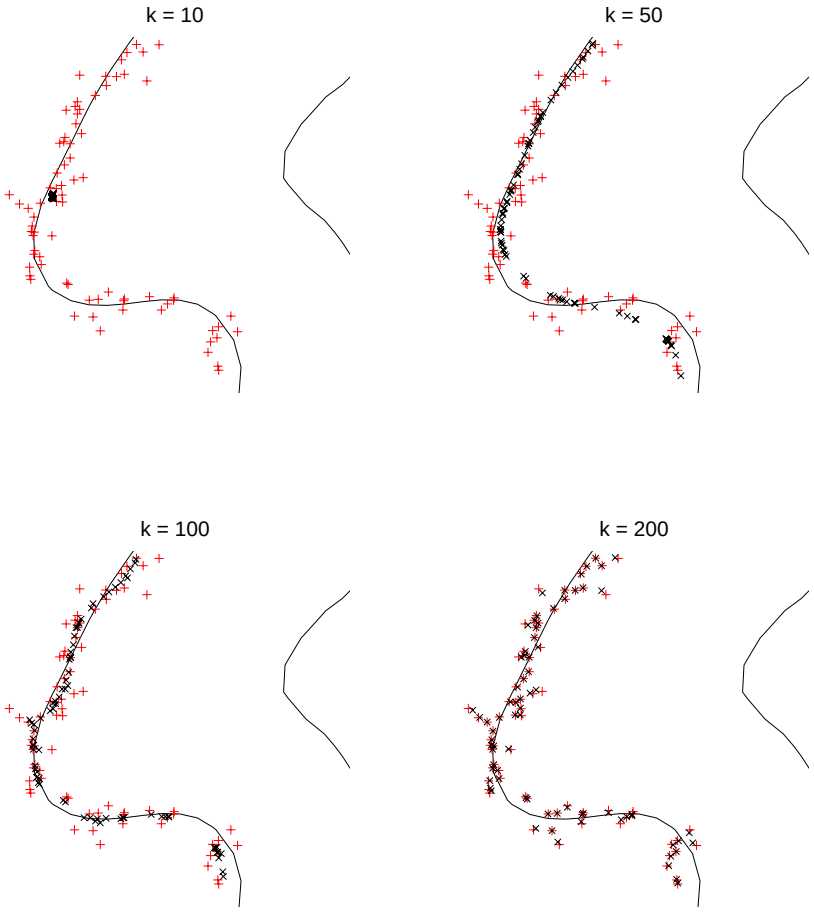


Figure 4.6: *Denoising with KPCA using differently many principal components of the KPCA: The red points are the noisy points* z_i*, the black ones the new, denoised points* $\hat{x}_i$*. The black line shows the estimated implicit surface.*

4.6.5 Denoising by KPCA on hyper-plane

We may further try to improve the results of the KPCA by selecting the PCs only from the hyper-plane representing a smooth surface through all the given points z_i. We achieved the following results:

k	D1	±	D2	±	D3	±
10	30	26	422	537	422	533
50	18	18	77	55	73	48
100	18	18	59	37	42	31
200	44	33	75	39	6	6
(noisy)	48	36	79	41		

The results are almost exactly the same as for the normal KPCA. Also the images of the PCs are more or less identical to Figure 4.4, or a figure of the denoising results is indistinguishable from Figure 4.6.

We do not exactly know why that is. Presumably it is again due to the pre-imaging step. The constructed points on the hyper-plane in $\mathcal{H}$ do not have close, meaningful pre-images in $\mathcal{X}$.

4.6.6 Denoising by projection in input space $\mathcal{X}$

As described in Section 4.5 we used a line search directly in the input space $\mathcal{X} = \mathbb{R}^2$ to determine points which lie on the constructed implicit surface. We followed the gradient direction $\nabla f(z)$ to obtain the results below (see also Figure 4.7):

	D1	±	D2	±	D3	±
denoised	0.015	0.028	57	36	50	38
(noisy)	48	36	79	41		

The procedure works fine. D1 is negligible and D2 close to the optimum of $b = 50$.

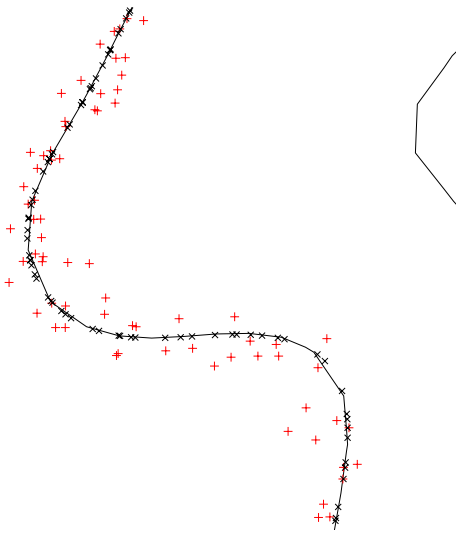


Figure 4.7: *The red points are the noisy ones z_i, the black the denoised ones $\hat{x}_i$ using the projection approach in $\mathcal{X}$. The black line shows the estimated implicit surface.*

4.6.7 Denoising by the moving least squares method

We implemented an additional method for yet another comparison, namely the moving least squares (MLS) method as proposed by [Alexa et al., 2003]: The key idea here is not to compute a global representation of the surface, but to do a locally weighted least squares fit of a plane for each query point separately. Then the query point is projected onto this plane to yield the denoised version $\hat{x}$.

We obtained the following results (see also Figure 4.8):

	D1	±	D2	±	D3	±
denoised	20	18	59	37	44	34
(noisy)	48	36	79	41		

The described algorithm includes a non convex optimisation problem to determine the best fitting local plane. In about 1% of the times the determination of the best local plane approximation did not converge.

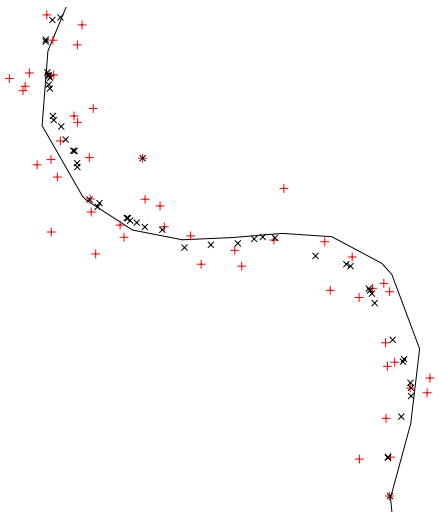


Figure 4.8: *An image denoised by MLS. The red points are the noisy ones z_i, the black the denoised ones $\hat{x}_i$.*

4.7 Conclusions

The following table summarises the above findings and includes a visual rating score which ranges from - - (very bad) to ++ (very good):

Method	D1	±	D2	±	Visual	Computing time
(noisy)	48	36	79	41		
Orthogonal Proj. in $\mathcal{H}$	47	36	78	41	- -	3 min
Sperical Proj. in $\mathcal{H}$	47	36	78	41	- -	3 min
KPCA	18	18	59	37	+	5.5 h
KPCA on Hyp.	18	18	59	37	+	43 h
Proj. in $\mathcal{X}$	0.015	0.028	57	36	++	20 min
MLS	20	18	59	37	+	2.8 h

One note concerning computing times: Our routines were (hopefully) correct, straight forward implementations of the algorithms. They were in no way efficient. Most of the time was consumed for evaluating large kernel expansions or dealing with non sparse kernel matrices of size $m \times m$ (We used a Gaussian kernel and no multi-scale!). The computations could be dramatically sped up using a compactly supported kernel and some multi-scale procedure as was used in Chapter 3. However, the given timings bear some relative information of the necessary amount of work.

As a conclusion one could say that even though the projection onto the implicit surface is conceptually simpler to perform in $\mathcal{H}$ than in the input space $\mathcal{X}$, the necessary pre-imaging routine destroys this complexity and speed advantage. The fact, that only very few points in $\mathcal{H}$ have a good pre-image, renders the results of the hyper-plane projection approaches somewhat unclear. KPCA may work quite well if one chooses the right number of principal components. However, constructing smooth implicit surfaces from noisy point data is very efficient (see Chapter 3). With a few rounds of a Newton root-search directly in $\mathcal{X}$, one can get good denoising results – reliably and fast.

Chapter 5

Including Prior Knowledge

5.1 Introduction

So far, we have constructed implicit surfaces from a number of given, potentially noisy surface points (see Chapter 3). The surfaces can describe any possible object. But what if we knew how the object approximately looks like? For example we may reconstruct a whole sequence of one 3D object that is slowly deforming. Can we use the prior estimated surfaces in the next step to get a more accurate reconstruction? Is it possible to achieve a high quality model with less input data (that can be acquired faster)? These are the questions we try to answer in this chapter.

Like in Chapter 4, the work for this chapter was done at an early stage of this thesis' work. At that time, we still used standard ϵ-insensitive SVR on one scale for the reconstruction of implicit surfaces. If we included the new sparse multi-scale representation, some small changes would have to be made. How much that could influence the findings in the experimental section is discussed in the conclusions.

5.2 Template SVR (T-SVR)

Suppose we have already constructed an implicit surface, the so called *template*, that is described through the function $\bar{f}(x) = \langle \bar{w} | \phi(x) \rangle_{\mathcal{H}} + \bar{b}$. Now we are given another set of surface points from a similar surface, e.g. from a different, but similar object or from the same object slightly transformed.

One idea to use the prior knowledge$\bar{f}(x)$ in the construction of an implicit surface function $f(x) = \langle w | \phi(x) \rangle_{\mathcal{H}} + b$ for the new points is the following: The data fitting term remains as before but the regularization term is changed in order to just penalise deviations from the template $\bar{f}$. Using the hyper-plane interpretation of $\bar{f}$, the information about it is encoded in $\bar{w} \in \mathcal{H}$. The idea is therefore not to use $\|w\|^2$ as a regularizer but $\|w - \bar{w}\|^2$. This approach shares a lot of similarity to the semi-parametric regression method described in [Smola et al., 1999].

Some of the information contained in $\bar{f}$ is also encoded in the offset $\bar{b}$. However, we assume that the offsets are the same (or at least very similar) for all the implicit functions and therefore do not contribute to relative changes.

Using the above approach, the trivial reconstruction problem with no data points given would yield the template $\bar{f}$. Newly given surface points may give rise to small deformations thereof.

If we have training points $\{(x_i, y_i)\}_{i=1,\dots,m}$, which are constructed from the given surface points as described in Section 3.3.2, we can write the primal problem as follows (compare to Chapter 2):

$$\begin{aligned} \min_{w,b,\xi^{(*)}} \quad & 0.5\,\|w-\bar{w}\|^2_{\mathcal{H}} + C\sum \xi^{(*)} \\ \text{s.t.} \quad & \langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i \le \epsilon + \xi_i \quad \forall i = 1,\dots,m \\ & -(\langle w|\phi(x_i)\rangle_{\mathcal{H}} + b - y_i) \le \epsilon + \xi_i^* \quad \forall i = 1,\dots,m \\ & \xi^{(*)} \ge 0 \end{aligned}$$

Following the standard dualization procedure described in [Schölkopf and Smola, 2002], the Lagrange function with the dual variables $\alpha^{(*)}, \beta^{(*)}$ is built and the problem is rewritten as

$$\begin{aligned} \max_{\alpha^{(*)},\beta^{(*)}} \min_{w,b,\xi^{(*)}} \quad & L(w,b,\xi^{(*)},\alpha^{(*)},\beta^{(*)}) \\ \text{s.t.} \quad & \alpha^{(*)} \ge 0 \quad \beta^{(*)} \ge 0 \end{aligned}$$

This new problem that is unconstrained in the primal variables $w, b, \xi^{(*)}$ has the optimality conditions

$$\begin{aligned} 0 = \tfrac{\partial L}{\partial w} \quad & = w - \bar{w} + \sum_{i=1}^m \alpha_i \phi(x_i) - \sum_{i=1}^m \alpha_i^* \phi(x_i) \\ 0 = \tfrac{\partial L}{\partial b} \quad & = \sum_{i=1}^m (\alpha_i - \alpha_i^*) \\ \forall i = 1,\dots,m \quad 0 = \tfrac{\partial L}{\partial \xi_i^{(*)}} \quad & = C - \alpha_i^{(*)} - \beta_i^{(*)} \end{aligned}$$

Inserting these results we get the final dual problem as

$$\begin{aligned} \min_{\alpha^{(*)}} \quad & 0.5 \left\| \sum_{i=1}^m (\alpha_i - \alpha_i^*)\phi(x_i) \right\|^2_{\mathcal{H}} + \sum_{i=1}^m (\alpha_i - \alpha_i^*)(-\bar{f}(x_i) + \bar{b} + y_i) + \epsilon \sum \alpha^{(*)} \\ \text{s.t.} \quad & \sum_{i=1}^m (\alpha_i - \alpha_i^*) = 0, \quad 0 \le \alpha_i^{(*)} \le C \quad \forall i = 1,\dots,m \end{aligned}$$

with $f(x) = \langle w|\phi(x)\rangle_{\mathcal{H}} + b = -\sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x) + b + \bar{f}(x) - \bar{b}$.

The function offset b can be computed exploiting the Karush-Kuhn-Tucker (KKT) optimality conditions [Schölkopf and Smola, 2002]. For all $j = 1,\dots,m$ we have:

$$0 < \alpha_j < C \quad \Rightarrow \quad b = y_j + \epsilon - \langle w|\phi(x_j)\rangle_{\mathcal{H}} = y_j + \epsilon + \sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x_j) - \bar{f}(x_j) + \bar{b}$$

$$0 < \alpha_j^* < C \quad \Rightarrow \quad b = y_j - \epsilon - \langle w|\phi(x_j)\rangle_{\mathcal{H}} = y_j - \epsilon + \sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x_j) - \bar{f}(x_j) + \bar{b}$$

(given a fixed j, either $\alpha_j > 0$ or $\alpha_j^* > 0$, but not both).

The resulting dual problem is very similar to the one for standard SVR (see Chapter 2): The only difference is that as training input we do not use y_i but the residual $y_i - (\bar{f}(x) - \bar{b})$, and at the end we add the template function $\bar{f}(x) - \bar{b}$ to the result.

5.3 Multiple Templates SVR (MT-SVR)

Suppose now, that we do not just have one template $\bar{f}$ given, but a whole set of them $\left\{\bar{f}_i\right\}_{i=1,\dots,n}$. These could for example be given from reconstructions of a 3D video sequence or from a database of objects of one class, e.g. the MPI face database [Blanz and Vetter, 1999].

Given new surface points, we want to reconstruct a surface using the domain knowledge included in the templates. Alternatively, we would like to know which of the templates fits the newly given data points best. This could be used for example to recognise a face scan given a head database.

As above the information about the $\bar{f}_i(x) = \langle \bar{w}_i | \phi(x) \rangle_{\mathcal{H}} + \bar{b}_i$ is included in the vectors $\bar{w}_i \in \mathcal{H}$. We will just regularise functions which leave the span of those vectors, e.g. we set the regularizer to be $\|w - \sum_{i=1}^n d_i \bar{w}_i\|^2$ with the convexity constraints $\sum_{i=1}^n d_i = 1$ and $d_i \geq 0$ for all $i = 1, \dots, n$.

The primal optimisation problem for $f(x) = \langle w | \phi(x) \rangle_{\mathcal{H}} + b$ then reads

$$\begin{aligned}
\min_{w,b,\xi^{(*)},d} \quad & 0.5 \left\| w - \sum_{i=1}^n d_i \bar{w}_i \right\|_{\mathcal{H}}^2 + C \sum \xi^{(*)} \\
\text{s.t.} \quad & \langle w | \phi(x_i) \rangle_{\mathcal{H}} + b - y_i \leq \epsilon + \xi_i \quad \forall i = 1, \dots, m \\
& -(\langle w | \phi(x_i) \rangle_{\mathcal{H}} + b - y_i) \leq \epsilon + \xi_i^* \quad \forall i = 1, \dots, m \\
& \xi^{(*)} \geq 0, \quad \sum_{i=1}^n d_i = 1, \quad d_i \geq 0 \quad \forall i = 1, \dots, n
\end{aligned}$$

We apply the same dualization method as above, yielding the dual problem

$$\begin{aligned}
\min_{\alpha^{(*)},\delta \in \mathbb{R}} \quad & 0.5 \left\| \sum (\alpha_i - \alpha_i^*) \phi(x_i) \right\|_{\mathcal{H}}^2 + \sum_i (\alpha_i - \alpha_i^*) y_i + \epsilon \sum \alpha^{(*)} + \delta \qquad (5.1) \\
\text{s.t.} \quad & \sum_{j=1}^m (\alpha_j - \alpha_j^*) \langle \phi(x_j) | \bar{w}_i \rangle_{\mathcal{H}} \geq -\delta \quad \forall i = 1, \dots, n \\
& \sum_{i=1}^m (\alpha_i - \alpha_i^*) = 0, \quad 0 \leq \alpha_i^{(*)} \leq C \quad \forall i = 1, \dots, m
\end{aligned}$$

The KKT optimality conditions yield a system of linear equations which can be inverted to get the offset b and the coefficients d_i. For all $j = 1, \dots, m$ we have:

$$\begin{aligned}
0 < \alpha_j < C \quad & \Rightarrow \quad y_j + \epsilon + \sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x_j) = b + \sum_{i=1}^n \langle \bar{w}_i | \phi(x_j) \rangle_{\mathcal{H}} d_i \\
0 < \alpha_j^* < C \quad & \Rightarrow \quad y_j - \epsilon + \sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x_j) = b + \sum_{i=1}^n \langle \bar{w}_i | \phi(x_j) \rangle_{\mathcal{H}} d_i
\end{aligned}$$

The solution function f is given as

$$f(x) = \langle w | \phi(x) \rangle_{\mathcal{H}} + b = -\sum_{i=1}^m (\alpha_i - \alpha_i^*) k(x_i, x) + b + \sum_{i=1}^n d_i (\bar{f}_i(x) - \bar{b}_i).$$

The dual problem (5.1) is convex in the dual variables $\alpha^{(*)}, \delta$. It can therefore be solved efficiently.

The use of the convex combination of templates only makes sense, if the linear combination of two templates is again a sensible object itself. E.g. for two templates and a mixing coefficient 0.5, the resulting linear combination should represent an object that is in between the two templates.

For functions $f : \mathcal{X} \to \mathbb{R}$ this assumption is true (see Section 5.4). However, when we deal with implicit surfaces, the focus is lying on the level-sets of functions $f : \mathcal{X} \to \mathbb{R}$. The zero level-set of two mixed template functions is not necessarily a level-set that is intermediate to the two templates. More precisely, the intermediate level-set will only make sense, if the two templates are rather close to each other, such that their signed distance fields overlap (see Figure 5.1).

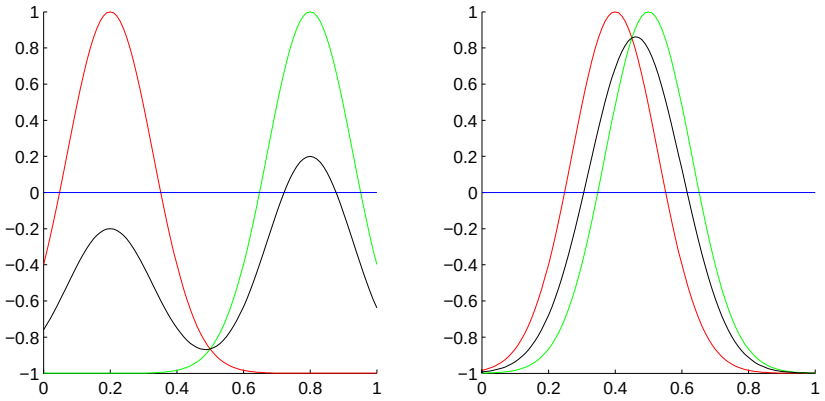


Figure 5.1: *The function value for several implicit surface functions is plotted versus one input dimension $\mathcal{X}$. The offset b is negative for all functions. Points where the functions cross the zero level (blue) are surface points. (left) Two implicit surface functions (red and green) are shown that are rather dissimilar. The convex combination of those two (black, 40% to 60%) also creates some surface points, but these are not in between the surface points of the templates (intuitively, we would expect the surface points to be shifted in parallel from one template to the other). (right) Two initial surfaces (red and green) that are more similar than on the left, i.e. closer together, such that the distance fields still overlap. In this case, the convex combination (black) defines new surface points that can be seen as intermediate.*

The fact, that we need a linear object space and that level-sets do not necessarily form such a linear space, is a fundamental problem of the proposed method when applied to implicit surfaces. More details will be given in the experimental Section 5.5, and in Chapter 6 we will examine more deeply how to produce good intermediate surfaces given two templates.

For detecting to which template a newly given point cloud fits best and for using this information to get the best possible reconstruction, one would like to include a term into the optimisation objective that favours results where only one of the d_i equals one and the others are negligible. However, this turns out to be an intrinsically non-convex problem. We do not know how to solve it much more efficiently than just testing each template using the T-SVR (see Section 5.2).

5.4 Experimental results with functional data

In this section we will examine how well the Template SVR (T-SVR) and the Multiple Templates SVR (MT-SVR) work in the case where the template objects form a linear space. We will deal with one dimensional functions $f : \mathbb{R} \to \mathbb{R}$ for which a convex combination is surely some intermediate object given two template functions. Only in the next section we will again look at implicit surfaces, i.e. the zero level-sets of such functions, for which the linearity assumption is in general not true.

5.4.1 Data approximation and smoothness

One necessary property of the T-SVR and the MT-SVR to be considered in approximation applications is the ability to nicely smooth and fit newly given data regardless of what the templates are.

We used the template functions $\{\sin(x), \cos(x), 1 - 2x\}$ for the MT-SVR and just $\{\sin(x)\}$ for the T-SVR. These templates were point sampled and reconstructed with a normal SVR algorithm to yield the template vectors $\bar{w}_i \in \mathcal{H}$. The kernel used throughout this series of experiments was a Gaussian of width $\sigma = 0.1$.

The test data points ($m = 20$) then were drawn equally spaced from the function $f(x) = x^2$ disturbed by Gaussian noise of width $\sigma = 0.01$. The regularization constant C was chosen by inspection.

Given these templates how well can the two algorithms interpolate/approximate the data? We evaluated the results again by visual inspection (see Figure 5.2):

method	smooth	approximation error	comment
P-SVR	Ok	Ok	looks nicely quadratic
MP-SVR	Ok	Ok	looks nicely quadratic

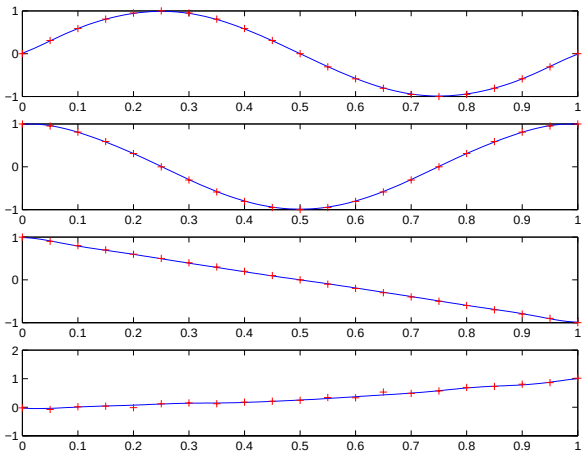


Figure 5.2: *The top three plots show the template functions reconstructed from the red points using standard SVR. The bottom figure shows the test data and the reconstruction of an MT-SVR using the templates. The bottom function looks smooth and approximates the data points nicely.*

5.4.2 Discrimination

For the MT-SVR another property to test is how well the best fitting template is selected. Therefore, we again trained a standard SVR for the template functions $\{\sin(x), \cos(x), 1-2x, x, x^2, \sinh(x), 1-\sinh(x)\}$ and then run the MT-SVR on a test dataset ($m = 200$) drawn from the function $f(x) = 1-2x$. We additionally distorted the test data with Gaussian noise of width $\sigma = 0.05$.

Even with the relative high noise level, the correct template was nicely recovered, i.e. the correct coefficient d_i was close to one. The same holds true for other examples. If there are several very similar templates (e.g. add $1-2\sinh(x)$ to the template list) then the coefficients of both templates will have high values.

5.4.3 Smooth, meaningful hole filling

In this section, we will test how well the T-SVR of the MT-SVR are able to combine the information given by a new point data set with the information contained in a template database.

Suppose we are selecting one of the templates, e.g. $\sin(x)$, and draw samples from it with a large hole in the middle. When reconstructing the function with a normal SVR, that function is not really well defined in the middle of the hole. The function just follows the implicit smoothness assumption that is given through the kernel. I.e. for a Gaussian the function will eventually drop to zero. The idea behind the (Multiple) Template(s) SVR is that this hole could be filled with the prior information given by the template function(s). In Figure 5.3 one can see how this prediction nicely works.

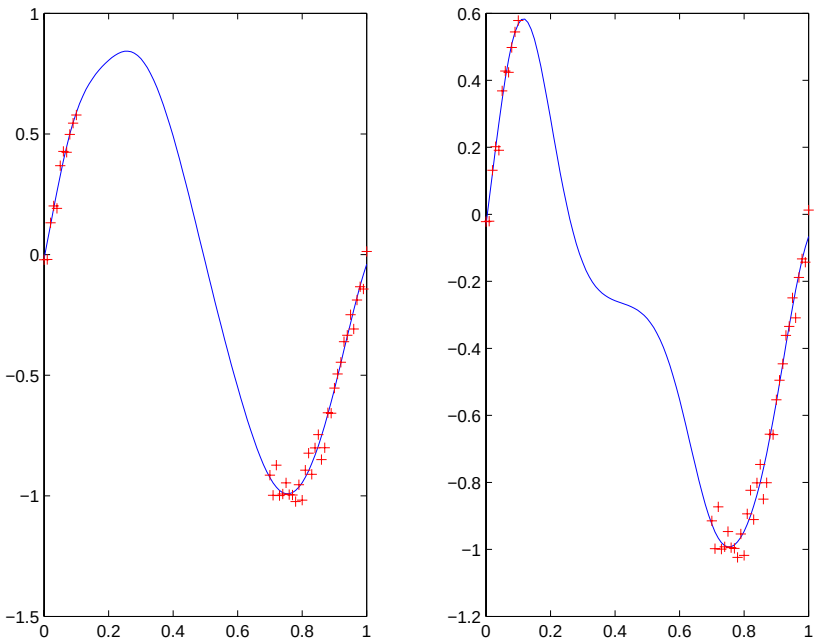


Figure 5.3: *The red points are drawn from* $\sin(x)$ *and some Gaussian noise was added. (left) a reconstruction (blue line) using the MT-SVR with templates that included* $\sin(x)$*, (right) a reconstruction using standard SVR*

A slightly harder and more realistic test is whether the algorithms are able to fill a hole also

for a dataset that is not drawn from one of the templates. Most often we will not want to reconstruct the same function again, but something similar.

We therefore sampled points from a non-template function, e.g. $f(x) = 1.3\sin(x)$. Again the dataset contained a rather large hole and we evaluate our methods by judging at how well this hole is filled. Like before, the MT-SVR (T-SVR) nicely fills the whole using the smoothness information given by the templates (see Figure 5.4). It also approximates the new data points well, i.e. it not just uses the prior but adapts it to the given situation. Remember that no template can have a weight d_i greater than 1, so the deviations of $1.3\sin(x)$ from $\sin(x)$ have to be modelled with a new kernel expansion.

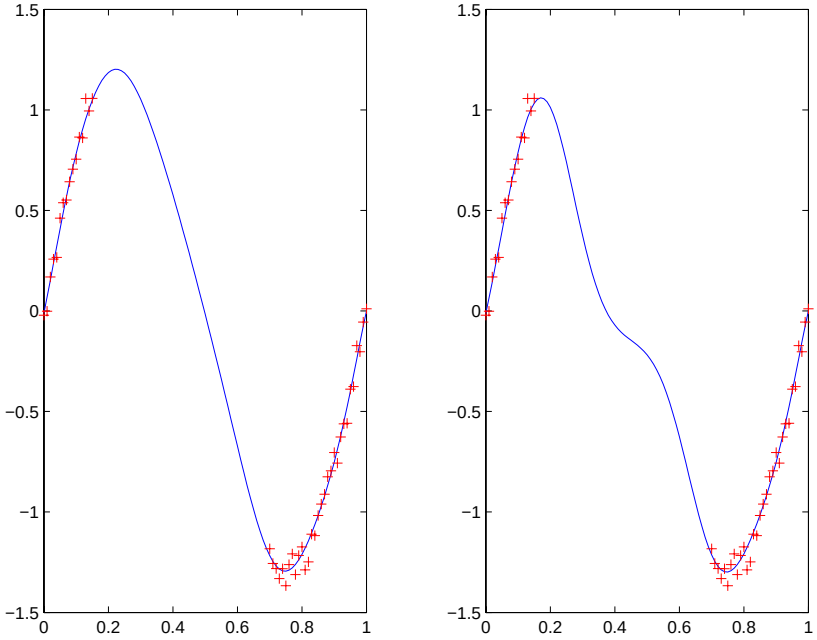


Figure 5.4: *The red points are drawn from* $1.3\sin(x)$ *and some Gaussian noise was added. (left) a reconstruction of the function using the MT-SVR with templates that did not include* $1.3\sin(x)$. *However, the set of templates contained a similar function* $\sin(x)$ *that helps to meaningfully fill the gap in the middle. (right) a reconstruction using standard SVR*

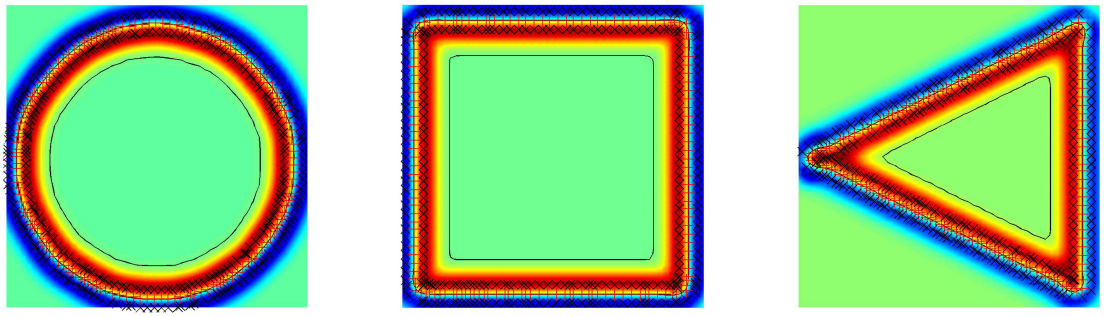


Figure 5.5: *The templates used for the 2D level-set experiments. Here (as in the following figures), we constructed implicit surface functions (function value colour-coded) from the given surface points (red). The training points that are generated from the surface points (see Section 3.3.2) are depicted as black crosses. The resulting implicit surface is shown as a black line.*

5.5 Experimental results with level-set data

In this section we will try to do the same experiments as in the section above. However, we will try to apply them to implicit surfaces, i.e. zero level-sets of functions. For this data-type it is not clear that the linear combination of two objects makes sense (it does so only when the two objects are very close to each other, see Figure 5.1).

5.5.1 Data approximation and smoothness

We used the templates depicted in Figure 5.5. They were represented as implicit surface functions $\bar{f}_i$ that were constructed using an early version of the reconstruction algorithm described in Section 3. We generated the off-surface points as described in Section 3.3.2, but we used standard ϵ-insensitive SVR to compute the implicit surface functions. We used just one scale with a Gaussian kernel. The offsets b were always close to zero as the numbers of inside and outside training points were approximately equal. How the more powerful sparse multi-scale representation described in Chapter 3 changes the results, will be discussed in the conclusions.

We tried to reconstruct the guitar (Figure 5.6), an object that is not in the database. It should be possible to reconstruct arbitrary objects with the T-SVR or the MT-SVR algorithm.

The result can be seen in Figure 5.7. The image shows that both algorithms construct a surface in the neighbourhood of the given test data points. However, the overall reconstruction process fails, as both methods have to use one of the template functions (or a number thereof whose weights d_i sum to 1). None of the templates fits the newly given object. That is why these templates imply spurious surfaces in regions far away from the now correct surface.

On the one hand this is exactly the effect we want to see (using the T/MT-SVR objectives): Far away from the given data points the function just behaves like the best fitting template. On the other hand this is also greatly disturbing. It implies that, whenever our new object is not very similar to one of the templates, the method fails.

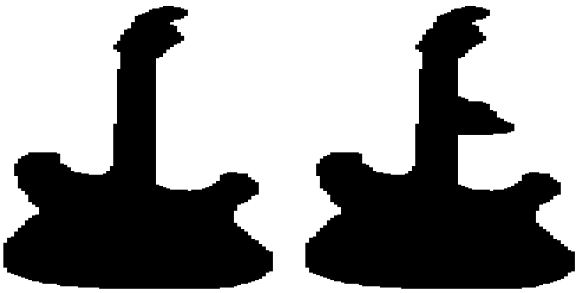


Figure 5.6: *The guitar shape used for reconstruction experiments (left). On the right, the guitar has got a small bump on the handle, a shape used in the last reconstruction experiments (Section 5.5.3).*

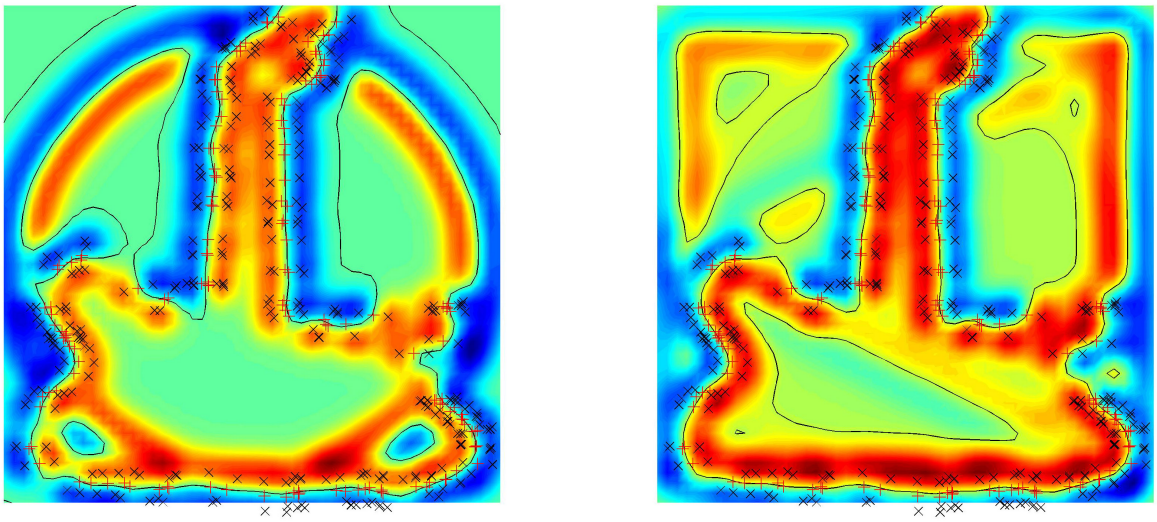


Figure 5.7: *Reconstruction of the guitar (Figure 5.6, left) the T-SVR (left) and the MT-SVR (right). The templates depicted in Figure 5.5 were used.*

5.5.2 Discrimination

Here, we ask again the following question: Given some templates and new data points drawn from one of the templates with added acquisition noise, does the MT-SVR find the right template?

We sampled 150 points from the guitar template, added some noise (Gaussian, $\sigma = 0.01$) and tried to reconstruct it using MT-SVR. We used the templates depicted in Figure 5.5 as well as the guitar shape (Figure 5.6, left). The resulting implicit surface function is depicted in Figure 5.8. For relatively few data points ($m \approx 50$) we could already identify the guitar, i.e. the corresponding d_i was close to one.

Nevertheless, even small components d_i of a wrong template produce surfaces in regions where we do not have data points. Thus, the reconstructed function is not a good reconstruction – even though the guitar was included as a template.

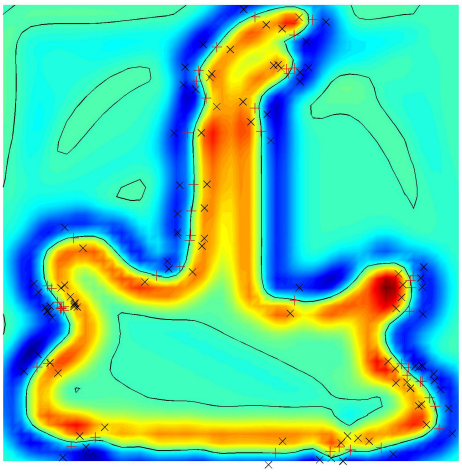


Figure 5.8: *A reconstruction of one of the templates using the MT-SVR and the data points (red) marked in the image. As templates we took the objects from Figure 5.5 as well as the guitar.*

5.5.3 Smooth, meaningful hole filling

Suppose we are given just partial data and want to reconstruct a complete surface by applying our prior knowledge encoded in the templates.

We used the same templates as above (triangle, circle, square and guitar) and tried to reconstruct a dataset which was split in half. In Figure 5.9 (upper row) we used data points sampled from one of the template functions (the guitar), and in Figure 5.9 (bottom row) we used a dataset drawn from a shape that was not in the template database, i.e. the guitar with the bump (see Figure 5.6, right), but similar to one of the templates.

The method works nicely in both cases. Using the prior knowledge contained in the template(s) the (M)T-SVR is able to reconstruct the complete guitar quite nicely. Standard SVR does not know about the left half of the guitar and therefore just drops to zero there.

Again, small components of the wrong template already imply extra surfaces that are not justified by the given data points. Thus, the overall reconstructed function is not acceptable.

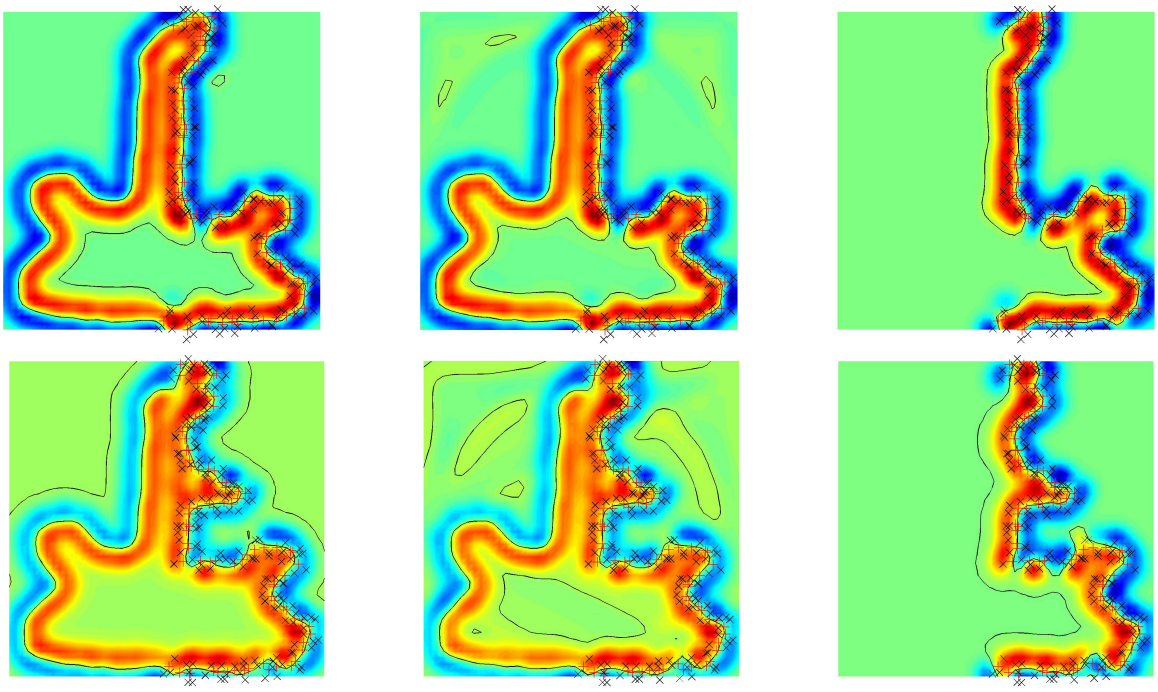


Figure 5.9: *Reconstructions of a point could (black crosses) sampled from the guitar dataset (upper row), or from the guitar with a bump (bottom row). Different algorithms were used: (left) T-SVR, (middle) MT-SVR, (right) standard SVR. As templates the shapes in Figure 5.5 as well as the guitar were used.*

5.6 Conclusions

The experiments in this section were conducted at an early stage of this thesis' work. How would the results for the level-set experiments change if we used the new powerful representation for implicit surfaces that is described in Chapter 3?

The new representation has a fixed offset b that is far from zero. Thus, small components of the wrong template would not produce the spurious surfaces that can be seen in Figure 5.8. The added small function terms would not lift the implicit surface function above the surface level in an area with prior function value close to b.

The new description also extrapolates the signed distance fields further into the space away from the surface – while at the same time still being able to model small surface features. The linear combination of implicit surface functions makes only sense, if the signed distance fields overlap. With the new structure many more objects would fulfil this property and could therefore form a valid linear space.

However, the linear combination of implicit surface functions is in principle not a good way to transform surfaces into each other. Most surface structures are lost during the transition. This effect is explained in much more detail in Chapter 6.

To conclude, one could probably improve the results of the level-set experiments if one used the new representation described in Chapter 3. However, implicit surface functions just form a sensible linear space if they are very similar to each other. It is therefore hard to see, how the presented approaches could effectively be used to handle general implicit surfaces. For objects that really form linear object spaces, however, the described methods seem much more promising.

Chapter 6

Dense Correspondence Fields

6.1 Introduction

Computing correspondences between two objects is a long studied, important problem. They are often computed for objects like 2D images or 3D volume data sets. In this work, we will focus on correspondences between surfaces embedded in $\mathcal{X} \subseteq \mathbb{R}^D$.

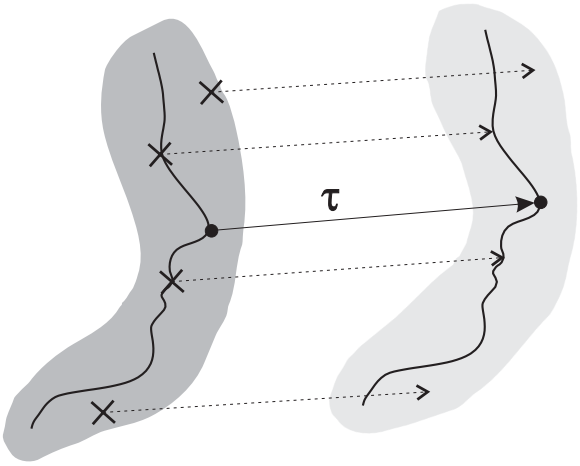


Figure 6.1: *The mapping $\tau : \mathcal{X} \to \mathcal{X}$ connecting the two given surfaces. The regions $\mathcal{X}$ are shaded grey.*

The goal is to match two objects in the sense that for each point on object 1 we find another point on object 2 that "belongs" - corresponds - to the first. E.g. suppose we mark the tip of the nose on one head then the task is to find the tip of the nose on the other head. One can either compute these correspondences just for a few points - this is often easier - or find a partner for each point of the objects. In the latter case we speak of a *dense correspondence*.

A dense correspondence can be defined as an one-to-one mapping $\tau : \mathcal{X} \to \mathcal{X}$ from object 1 to object 2 – the *transfer function*. Corresponding points are set to be x and $\tau(x)$ (see Figure 6.1). We also speak of this mapping as a *warp*, *transformation* or *deformation field* because one can also imagine that the mapping τ implies some deformation of the input space $\mathcal{X}$ that maps the surface of object 1 onto the surface of object 2. It does so such that the relevant features of the objects, e.g. nose, eyes, etc., are aligned with each other.

What defines two points to "belong" together? One could say [Alexa, 2002] that two points

are matching each other if they are similar in

1. function/semantics,
2. appearance/shape/geometry,
3. or relation to other matching points.

The first criterion is hard to control computationally as it would presuppose some form of high-level recognition of the object. Yet, many recognition methods first compute the correspondences of the given object to a model class and only then try to recognise it [Blanz and Vetter, 2003].

The appearance of a surface point can be described through its texture and local surface geometry. The geometric properties should be independent of the surface representation, therefore the means of differential geometry seem a good way to describe them [Alexa, 2002]. As shown above (Section 3.7) one can extract normals, mean or Gaussian curvature trustfully from an implicit representation.

The plausibility of points' relative characteristics, e.g. that the nose is always lying in between the mouth and the eyes, is hard to evaluate automatically. For each object class one could define a set of rules for the relations among some specific feature points. Yet, these rules have to be found in the first place and it is not easy to include such hard constraints into an optimisation problem. Another way is to express one feature point as the weighted sum of its neighbours. One could learn these weights from a database and then try to retain them while fitting the feature points onto a newly given object. However, this method does not yield a dense correspondence. A simple (not necessarily sufficient) condition is that the transfer function τ is smooth. Points which are close on object 1 will then be somewhat close also on object 2. The local neighbourhood relations thus will be preserved.

Because a perfect computational control of these similarity measures is hard to achieve, it is critical for algorithms that compute correspondence fields to accept a variable amount of user input as well. This user input is most often given through matched point pairs. E.g. the user may tell the positions of the nose tips by clicking on them.

One counterintuitive phenomenon is that correct correspondences are the more important, the more similar the two objects are. For very distinct objects the correspondence is not really well defined and also humans cannot say whether a specific realization of τ is correct or not. On the other hand for similar objects, humans can draw very exact connections between points and a correspondence field violating this judgement can easily be detected [Alexa, 2002].

In this work only correspondence fields between topologically identical objects will be considered. If the topology changes, no smooth deformation field τ from one object to the other exists. It is in general very hard to deal with non-continuous deformation fields computationally. Also humans mostly do not know how to transform two topologically different objects into each other.

The applications for dense correspondences are many - too many to give a complete reference here. Corresponding point pairs in stereo images allow 3D reconstruction, an optical flow field allows image segmentation and velocity estimates. It can also be used for tracking purposes.

With the help of correspondence fields one can also compare the whole or parts of an object to a model database. Correspondences can thus be used in object recognition.

In this work, we will mostly test our correspondences through the implied morph between the two objects. This is done by transferring corresponding points onto each other in a linear way. One can also see this as *warping* the object space 1 to fit the object space 2 while aligning all relevant features. The idea is that the medium objects will look visually plausible if and only if relevant feature points are correctly mapped onto each other (see Figure 6.2).

A B C D

Figure 6.2: *The two head models (**A**) and (**D**) are mixed once without any correspondence information (**B**) and once including it (**C**).*

This idea lies at the heart of linear models in computer graphics and vision [Blanz and Vetter, 1999; Shelton, 2000] which are some of the most powerful tools in computer vision. Given a (mostly small) database of prototype objects which are in correspondence, we can create arbitrary convex combinations of these. The mixtures will also belong to the class of admissible objects and therefore we can characterise a very flexible model with just a few mixture coefficients. [Blanz and Vetter, 1999] have used this to build a morphable face model which can e.g. be applied to reconstruct a 3D face geometry from a single 2D image, an otherwise under-constrained problem [Blanz and Vetter, 2003].

6.2 Related work

In this section we will review some related methods of how to compute correspondence fields, leaving aside the applications, many of which are obvious. One can distinguish two groups of algorithms in the literature: The first group works based on feature points, i.e. they compute the correspondence fields only on some distinct points. The other group - to which our algorithm belongs - computes a dense correspondence defined for each point of the input object.

A good review of different techniques for matching 2D shapes is given in [Veltkamp and Hagedoorn, 1999]. However, many methods, e.g. line based methods, are limited to the matching of curves and do not apply to 3D surfaces.

A nice introduction into methods used to compute optical flow on 2D images is given in [Bruhn et al., 2002]: They compare local methods with the global techniques. The local methods, such as the Lucas-Kanade method [Lucas and Kanade, 1981], compute correspondences for those points where it can be done reliably, e.g. feature points with a significantly characteristic neighbourhood structure. The Horn-Schunck approach [Horn and Schunck, 1981] representing the class of global methods minimises some kind of energy functional depended on the transfer

function. It includes one term for data fitting and one term enforcing smoothness, the so-called regularization term. In optical flow computations similar pixel intensities are the similarity measure of choice. [Bruhn et al., 2002] state that local methods are in general more robust to occlusions and image noise, because an occlusion for example might imply a non smooth deformation field which is highly penalised in the global context. However, many applications require a dense correspondence field to be given.

These 2D image methods also extend to optical flow computations on 3D volume datasets. One application in medical imaging is given by [Rueckert and Frangi, 2003]. Here, local similarity is measured based on a mutual information criterion.

A completely different way of establishing correspondences between a set of objects is described in [Davies et al., 2001]: Here the final application of the correspondence information is already guiding the matching process. It is assumed that the correspondences are used to build a linear object model. A high quality model should be as compact as possible. One can derive criteria from that on where to place corresponding points. However, the method seems computationally expensive and only yields a sparse correspondence for some feature points.

Optical flow methods are typically applied to objects that are defined on the whole space $\mathbb{R}^D$. Things are different if one looks at surfaces embedded in $\mathbb{R}^D$. They can be seen as sub-manifolds. The domain and the image of the transfer function τ are thus restricted to a non-trivial set. This introduces several problems on how to construct and how to parameterise such a function.

An extensive review of techniques to morph 3D meshes – the most common surface representation – is given in [Alexa, 2002]: The standard idea is to parameterise the surfaces onto 2D domains and then find a mapping between these. By studying mappings between the parameterisation domains one guarantees that a point lying on the surface of object 1 will be mapped to a point lying on the surface of object 2. However, it is in general very difficult to parameterise a complex surface onto a 2D domain. The mappings between the parameterisation domains are often guided through user input, given as corresponding point pairs, smoothness constraints or measures of similarity comparing the implied 3D geometry. Smoothness, e.g. continuity, often takes the geodesic distance on the surface as its base metric. This provides a better control of closeness of two points on a surface than the Euclidean distance, as the latter might be misleading near bended parts of the surface.

[Blanz and Vetter, 1999] also use a parameterisation approach to build the MPI face database: The head surfaces are projected onto a cylinder. Thereby the 3D points are parameterised onto a 2D domain and they retrieve a range image on which they then compute a dense optical flow field.

[Schreiner et al., 2004] propose a mesh morphing technique which avoids the parameterisation. They extract coarse approximations of two given objects such that these share the same mesh connectivity. The process has to be initialised manually, i.e. the user has to select some starting points which will form the vertices of the coarse approximation. Afterwards they split each face of the coarse mesh recursively while adjusting the relative positions of new found vertices to fit each other. The similarity measure includes a smoothness constraint but no geometry information is used. The method seems highly dependent on how the initial coarse mesh is chosen. It is also quite complicated and hard to implement.

In order to avoid parameterisation problems one can extend the domain of the transfer function from the surface itself to the space surrounding it $\mathcal{X} \subseteq \mathbb{R}^D$ - an idea proposed by [Huang et al., 2003] which is also used in our approach. However, the transfer function τ then

has to be selected in a way such that it maps surface points to surface points. [Huang et al., 2003] include a penalty term into their optimisation problem which penalises mappings which do not have this property. This approach is very similar to ours. However, they use B-splines to model the transfer function and the presented examples are of relatively low complexity. They also do not include differential properties into their similarity measure.

[Ham et al., 2005] propose an algorithm that computes correspondences between point sampled sub-manifolds given some pairs of point correspondences. Their algorithm is designed to work in high dimensional spaces $\mathbb{R}^D$. The regularization they use is based on a graph Laplacian for the nearest neighbour graphs of the point sampled manifolds. They do not use any similarity of geometric properties of the manifolds in their matching criterion.

[Cohen-Or et al., 1998] compute a warp field from a regression based on a few given point pairs and resolve the resulting mismatches of the deformed surface with the target surface by blending implicit descriptions into each other. This way, they can achieve plausible morphs, but do not solve the correspondence problem itself.

In general, the correspondence problem can be split into two parts: First, one might align the two objects through a rigid transformation such as rotation, translation or scaling. The second, elastic part is more difficult and is the central piece of the methods mentioned above. Given some landmark points that can be matched there are numerous algorithms to estimate the optimal rigid transformation [Haralick and Shapiro, 1992]. Another approach to find a rigid alignment is to use iterative closest pairs methods [Rusinkiewicz and Levoy, 2001]: In a cycle one first determines corresponding points by just searching for the closest possible point. Then one aligns the two objects and repeats until the process converges.

Once the correspondence information is given for two surfaces, there may still arise some problems of how to morph one object into the other. For the most common surface representation, i.e. meshes, one will intuitively just linearly move the vertices from the initial to the target position and keep the mesh connectivity constant. In order to really end up with the second mesh differences in the mesh structure have to be resolved. [Alexa, 2002] summarises methods how to solve this problem, e.g. by the construction of super meshes. One also needs to take care that no self-intersections of the mesh occur during the transformation and that the mesh still connects nearest neighbours [Alexa, 2002]. A smooth object that is heavily deformed may become very non-smooth if the triangle mesh describing it is not re-sampled during the process. Subdivision surfaces are proposed to partly solve this problem [Zorin and Schröder, 2000].

If no model knowledge is available the simplest way to interpolate from the initial to the target position is linear. Apart from problems with mesh intersections, etc. it would not be reasonable to do anything else. If, however, such model knowledge is available, one might try to keep the intermediate steps as close to the model as possible. One such approach for a Gaussian mixture model is described in [Saul and Jordan, 1996].

6.3 Algorithmic idea

Our approach to establish a dense correspondence between two surfaces is to optimise a cost functional dependent on the deformation field τ over an appropriate function space $\mathcal{H}$. We do not parameterise the surfaces onto a lower dimensional domain but we extend the domain and the image of the transfer function τ from the surfaces to the whole space $\mathcal{X} \subseteq \mathbb{R}^D$ in which

the surfaces are embedded. To enforce a valid correspondence field, i.e. one which maps surface-to-surface, we include a cost term which penalises functions violating that constraint. Our method is able to process a variable amount of user input given through matching point pairs. It selects the transfer function τ based on smoothness assumptions but also based on differential geometric properties or texture information.

More formally: Suppose we are given two surfaces of the objects O_1 and O_2, both being elements of an object class $\mathcal{O}$. We assume the surfaces are embedded in a domain $\mathcal{X} \subseteq \mathbb{R}^D$. The goal is to "learn" a transfer function $\tau : \mathcal{X} \to \mathcal{X} \subseteq \mathbb{R}^D$.

In [Schölkopf et al., 2005] we described our method in a way that is able to handle also full volume datasets, e.g. 2D images and 3D volumes. However, in this work we will restrict ourselves to surfaces embedded in $\mathcal{X} \subseteq \mathbb{R}^D$ in favour of clearer presentation.

6.3.1 Locational cost

The main idea of our approach is that intuitively, a good warp from the surface of O_1 to the surface of O_2 has the general property that it will map $x \in \mathcal{X}$ to a point $\tau(x)$ such that *relative to* x, O_1 *looks like* O_2 *relative to* $\tau(x)$. For instance, it could be that O_1 in the vicinity of x is similar to O_2 in the vicinity of $\tau(x)$, with respect to a specific discrepancy measure.

We formalise this intuition in a cost function

$$c_{loc}(O_1, x, O_2, \tau(x)). \tag{6.1}$$

Due to its conceptual similarity to the *locational kernels* defined by [Bartlett and Schölkopf, 2001], we refer to c_{loc} as a *locational cost function*.

Locational cost functions can readily be constructed if, for instance, we are given *feature functions* $f_1, f_2 : \mathcal{X} \to \mathbb{R}$ capturing relevant properties of the surfaces of O_1, O_2. Note that these feature functions are not just defined on the surfaces but in the whole volume $\mathcal{X}$ surrounding them.

For surface warping applications, a good choice of such a feature function is the signed distance function, that we constructed in Chapter 3. In the following the symbols f_1, f_2 will always confer to these. One may also think of a colour/texture function $f_C : \mathcal{X} \to \mathbb{R}$ which appropriately extends the colour/texture information given on the surface into the whole volume $\mathcal{X}$. For different methods to construct these *surface feature functions* see Section 6.5.

We now list some examples of locational cost functions $c_{loc}(O_1, x, O_2, \tau(x))$ and the incentives behind them. For computational reasons to be explained below, we focus on differentiable examples.

1. Preserving signed distances: (This measures whether surface points get mapped to surface points. It is therefore a necessary condition and will be included into all computations.)
$$|f_1(x) - f_2(\tau(x))|^2 \tag{6.2}$$

2. Preserving normal directions:
$$\|\nabla f_1(x) - \nabla f_2(\tau(x))\|^2 \tag{6.3}$$

3. Preserving mean or Gaussian curvature:

$$\left|c_{m/G}(f_1, x) - c_{m/G}(f_2, \tau(x))\right|^2 \tag{6.4}$$

where $c_{m/G}(f, x)$ computes the iso-surface mean/Gaussian curvature using the signed distance function $f : \mathcal{X} \to \mathbb{R}$ [Thirion and Gourdon, 1995]

4. Preserving higher order differential characteristics:

$$\sum_{i=0}^{\infty} \alpha_i d\left(\nabla^i f_1(x), \nabla^i f_2(\tau(x))\right) \tag{6.5}$$

where the α_i are weighting coefficients determining the contribution of the higher order terms and d some differentiable metric

5. Preserving the surface texture:

$$\left|f_{C1}(x) - f_{C2}(\tau(x))\right|^2 \tag{6.6}$$

6. Preserving multiple approximations of the signed distance function

$$\sum_i \left|f_1^i(x) - f_2^i(\tau(x))\right|^2 \tag{6.7}$$

where the f^i are the SVR functions computed at each scale i of the multi-scale procedure that is used to construct the implicit surface functions f_1, f_2 (see Chapter 3).

7. Mapping points with similar local shape complexity

The implicit surface functions we construct in Chapter 3 adapt their local kernel centre density to the local shape complexity. Therefore the density of kernel centres can be used as a similarity measure.

8. Preserving some feature averaged over a neighbourhood: E.g.

$$\sum_{\Delta x \in \mathcal{N}} \left|f_1(x + \Delta x) - f_2(\tau(x) + \Delta x)\right|^2 \tag{6.8}$$

where $\mathcal{N}$ is some "neighbourhood" set such as the vertices of a regular simplex centred at 0.

9. Preserving multiple feature functions: E.g.

$$\lambda_{dist} \left|f_1(x) - f_2(\tau(x))\right|^2 + \lambda_{norm} \left\|\nabla f_1(x) - \nabla f_2(\tau(x))\right\|^2 \tag{6.9}$$

6.3.2 Landmark point training cost

Often one can easily extract some special landmark points from an object and match those. One can also just click onto two corresponding points on the two objects yielding pairs of landmark points $(x_1, z_1), \ldots, (x_m, z_m)$ (where x_i belongs to O_1 and z_i to O_2). These can be incorporated into the objective function using a term $c_p(\tau(x_1), \ldots, \tau(x_m), z_1, \ldots, z_m)$.

The above joint dependence of the cost on all point pairs allows the incorporation of constraints on the relative position of points. However, these are hard to retrieve in general. We therefore will only use an additive cost

$$c_p(\tau(x_1), \ldots, \tau(x_m), z_1, \ldots, z_m) = \sum_{i=1}^{m} \left\|\tau(x_i) - z_i\right\|^2. \tag{6.10}$$

6.3.3 Avoiding self-intersections

When optimising over the transfer functions τ we should take care that they do not imply non-physical deformations of the embedded surfaces. E.g. a surface may not bend around and intersect with itself while turning the inside surface to the outside. Our approach so far does not include any mechanism preventing such behaviour. First experiments also showed that often the original surface is deformed to fit the target surface and is folded to do so. Below we propose a method to avoid such unwanted situations.

Following the argumentation of [Gain and Dodgson, 2001] a necessary condition for self-intersections to occur is that the determinant of the Jacobian of the transfer function $|J_\tau(x)|$ is smaller than zero for some points $x \in \mathcal{X}$. Clearly, the identity warp has $|J_\tau(x)| = 1$ everywhere. It is well-known that $|J_\tau(x)| > 0$ ensures at least local invertibility.

We propose to include a penalty term into the objective function which favours $|J_\tau(x)| > 0$. One possibility is

$$(\epsilon - |J_\tau(x)|)_+^2 \,, \tag{6.11}$$

where $(x)_+ = \max\{x, 0\}$. This error function is differentiable in τ, a property needed for efficient optimisation.

This penalisation approach does not guarantee $|J_\tau(x)| > 0$. We relax this constraint similar to the surface-to-surface mapping condition and assume that well chosen weights will enforce valid correspondence fields.

6.3.4 Objective function

Locational costs, pairs of feature points, and the self-intersection penalty may not sufficiently constrain the problem to lead to a satisfying overall solution. Especially in regions of low contrast of the feature functions, there are many ways to chose a valid transfer function τ if we select it from a general function class $\mathcal{H}$. We thus need to include a regularization term $R(\tau)$ into the objective function to enforce a unique solution (cf. also Chapter 2): Among all possible solutions for τ we give a bias towards a maximally smooth one where smoothness is measure through $R(\tau)$.

Putting together the pieces, we end up with an objective function of the form

$$\begin{aligned} J(\tau) \;\; = \;\; & R(\tau) + \lambda_p \sum_{i=1}^{m} \|\tau(x_i) - z_i\|^2 \\ & + \lambda_{loc} \int_{\mathcal{X}} c_{loc}(O_1, x, O_2, \tau(x)) \; d\mu(x) \\ & + \lambda_{si} \int_{\mathcal{X}} (\epsilon - |J_\tau(x)|)_+^2 \; d\mu(x), \end{aligned} \tag{6.12}$$

to be minimised over all warps $\tau : \mathcal{X} \to \mathcal{X} \subseteq \mathbb{R}^D$. The space of admissible transfer functions $\mathcal{H} \subseteq L_2(\mathcal{X})$ has to guarantee that $R(\tau)$ can be computed. The parameters $\lambda_p, \lambda_{loc}, \lambda_{si} \geq 0$ determine the relative influence of the error terms, and μ is a measure whose support covers the area in which we want to estimate the deformation field, e.g., the vicinity of the surface of O_1.

6.4 Optimization approaches

There are two distinct ways to proceed with the minimisation problem (6.12): Extracting Euler-Lagrange equations or using a sampling-SVR approach. We will briefly explain both methods below but focus in our experiments onto the latter.

6.4.1 Euler-Lagrange equations

We could chose a simple regularizer, e.g.

$$R(\tau) = \sum_{d=1}^{D} \|\nabla \tau_d\|^2_{L_2(\mathcal{X})} .$$

We thus would regularise the component functions τ_d of τ separately. Better performance may be possible using more sophisticated regression schemes [Weston et al., 2003; Bakir et al., 2004; Micchelli and Pontil, 2005], but we do not pursue this in the present work.

Then we can derive Euler-Lagrange equations from the condition

$$\frac{\partial}{\partial \lambda} J(\tau + \lambda g)|_{\lambda=0} = 0, \ \forall g \in \mathcal{H}.$$

This leads to the following system of partial differential equations that can be discretised on a grid:

$$\begin{aligned} 0 \quad \equiv \quad & -2 \sum_{d=1}^{D} \Delta \tau_d(x) + 2\lambda_p \sum_{i=1}^{m} (\tau(x_i) - z_i)\delta(x - x_i) \\ & + \lambda_{loc} \, \partial_4 c_{loc}(O_1, x, O_2, \tau(x)) \\ & - 2\lambda_{si} \ (\epsilon - |J_\tau(x)|)_+ \frac{\partial \, |J_\tau(x)|}{\tau(x)} \end{aligned}$$

This method is simple and the regularizer has an easily understandable meaning. The given feature point correspondences could be included as fix constraints into the discretised equation system. The coupling matrix resulting from this regularizer given a simple approximation scheme like $\Delta f(x_i) = \frac{1}{h^2}(f(x_{i-1} - 2f(x_i) + f(x_{i+1}))$ would be sparse. However, the final problem is a system of non linear equations which is hard to solve.

6.4.2 Sampling - SVR style

Another method is to sample points from the domain $\mathcal{X}$ and replace the integral in (6.12) by a finite sum. This sampling may not be the mathematically most exact approximation of the integral, however, the intuition behind the continuous and the sampled version of (6.12) is the same if the sampling is uniform and dense enough.

As typical for SVR, we chose τ from the class of functions $\mathcal{H}$ that can be interpreted as hyper-planes in a RKHS. We thus set the components τ_d of τ to be

$$\tau_d = x + \langle w | \phi(x) \rangle_{\mathcal{H}} \tag{6.13}$$

and the regularization term to be the standard large margin regularizer

$$R(\tau) = \frac{1}{2}\sum_{i=1}^{D} \|\mathbf{w}_d\|_{\mathcal{H}}^2. \tag{6.14}$$

Suppose we are given the landmark points pairs $\{(x_i, z_i)\}_{i=1,\ldots,m}$ and let the sampled points be $\{x_i\}_{i=m+1,\ldots,m+n}$. The generalised representer theorem (see Chapter 2 or [Schölkopf and Smola, 2002]) then tells us that we can write τ_d as an expansion of the x_i, i.e. $\tau_d = \sum_{j=1}^{m+n} \alpha_{j,d} k(x_j, x)$. The finite dimensional version of the objective functional (6.12) then reads

$$\begin{aligned} J(\alpha) \quad = \quad & \frac{1}{2}\sum_{d=1}^{D}\sum_{i,j=1}^{m+n} K_{i,j}\alpha_{i,d}\alpha_{j,d} + \lambda_p \sum_{d=1}^{D}\sum_{i=1}^{m}(\tau_d(x_i) - z_{i,d})^2 \\ & + \lambda_{loc} \sum_{i=1}^{m+n} c_{loc}(O_1, x_i, O_2, \tau(x_i)) \\ & + \lambda_{si} \sum_{i=1}^{m+n} (\epsilon - |J_\tau(x)|)_+^2 \end{aligned} \tag{6.15}$$

As for the implicit surface reconstruction (see Section 3.5), we use the Wu kernel (3.1). It will again lead to a sparse kernel matrix K. This allows for evaluation of the functional in linear time $O(m+n)$.

However, the optimisation problem (6.15) remains a difficult non-convex problem with spurious local minima. In such problems, it is helpful to construct a good starting point. If we have landmark points given (i.e. $m > 0$), one way to proceed is to set λ_{loc} and λ_{si} to zero initially. In that case, the problem can be decomposed into D one dimensional problems of the form

$$\underset{\mathbf{w}_d \in \mathcal{H}}{\text{minimize}} \frac{1}{2}\|\mathbf{w}_d\|^2 + \lambda_p \sum_{i=1}^{m} |\tau_d(x_i) - z_{i,d}|^2 . \tag{6.16}$$

This is a convex quadratic program corresponding to an SVR with a squared loss function. Taking this as an initial solution, we then optimise the complete objective using a memory limited Newton like optimisation method [Liu and Nocedal, 1989].

In general it is difficult to judge the quality of a morph automatically as explained in more detail in section Section 6.8. Therefore, it is not trivial to find a good stopping criterion for this descent method and we used just a fixed number of steps, typically in the order of 50 steps.

To further stabilise the optimisation process, we apply a multi-scale scheme. We use wide kernels in order to make sure that the sparse feature point correspondences lead to a good initial guess in a larger vicinity. For matching detail structures on the surface of the second object, we need enough flexibility in the model as provided by smaller kernels. We iterate the optimisation procedure from coarse to fine and approximate the residual errors on the next finer scale. We stop the refinement process once the kernel width reaches the size of the smallest surface features. As kernel widths, a cascade of $\sigma_0, \frac{\sigma_0}{2}, \frac{\sigma_0}{4}, \ldots$ is used where σ_0 is typically chosen on the order of $\frac{1}{2}$ of the object's diameter.

The volume $\mathcal{X}$ is sampled at each scale at a resolution necessary to yield on optimisation problem as small as possible. We extract approximately equally spaced points by recursively

subdividing an initial bounding box until the boxes reach a resolution smaller than a chosen fraction the kernel width (in our case $\frac{1}{2}$). We then take the centre point of those boxes to be a sampling point in our optimisation problem. Boxes are just subdivided if they lie within the volume $\mathcal{X}$. This way we efficiently construct a uniform sampling yielding a sparse kernel matrix in a runtime that scales proportional to the sampling volume.

Many steps of the above proposed algorithm resemble the one for implicit surface reconstruction. Hence, we could reuse most of our code when implementing this algorithm.

6.5 Feature functions from surface data

Some non-geometric features are just defined on the surface of an object, think of colour and texture. If we want to include this information into a similarity measure as given above we need a way construct a feature function which is defined on the whole domain $\mathcal{X}$ where we want to compute the transformation τ, not just on the surface.

We assume that the feature is given as scalar values $\{c_i\}_{i=1,\dots,m}$ at specific surface points $\{x_i\}_{i=1,\dots,m}$. E.g. the colour of a 3D scanned object is determined through a pixel image. These pixels can be projected onto the surface.

The feature function to construct should fulfil the following criteria:

1. It should be at least once differentiable in order to allow for efficient optimisation over τ.
2. For query points far away from the object's surface it should have a constant value, e.g. the mean of all feature values given.
3. A query close to one of the data points x_i should yield the corresponding feature value c_i.
4. The feature function should be fast to compute even when the number of data points given is large ($m \approx 10^5$).

6.5.1 Inverse distance weighting

One way to extend the information given at the surface points $\{x_i\}_{i=1,\dots,m}$ is to set

$$f_C(x) \equiv \frac{\sum \frac{c_i}{\|x-x_i\|}}{\sum \frac{1}{\|x-x_i\|}}.$$

This method does not require a scaling parameter. It fulfils the conditions that far away for the given data it takes on an average value, and close to one point x_i is equals this point's value c_i very closely. One can also compute it efficiently using a Fast Multipole Method (FMM) [Beatson and Greengard, 1997] in a time requirement scaling as $O(m \log m)$.

We experimented with the FMM and got fast and exact results when using a Gaussian weighting function instead of the inverse distance. With the inverse distance weighting function we could not achieve good results as the higher order derivatives of the inverse distance function are more complicated to compute than for the Gaussian which factors into individual dimensions.

However, even if the implementation difficulties could be resolved, there were still some problems: First of all the function f_C leads to numerical instabilities if one evaluates it close to one of the data points x_i. The evaluation at one of the x_i is not even well-defined. A cut-off when x approaches x_i leads to an non differentiable function f_C. Close to a point x_i the function value is almost completely determined by the feature information given there. This is in principle desirable but does not yield a valid gradient to improve on a possible miss-match. The derivatives are also highly fluctuating around points x_i which leads again to numerical instabilities.

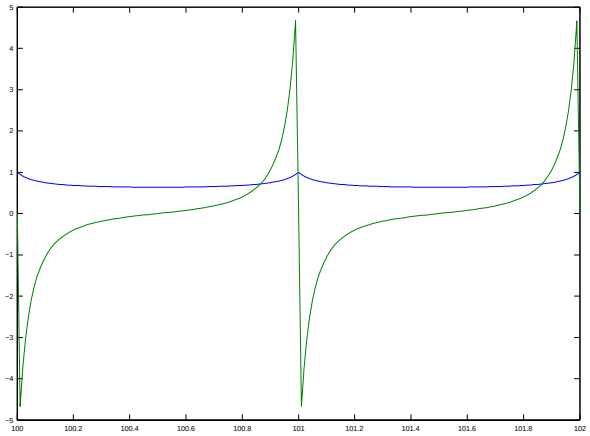


Figure 6.3: *Given a chain of centres located at positions $x_i = 0, \ldots, 200$ with feature values $c_i = \sin \frac{x_i}{100}\pi$ the feature function f_C looks like the above blue line (only a small magnified piece is plotted). The derivatives (green) along that line oscillate heavily. A shift of the* sin *along the line would be hard to detect with our transformation algorithm given this a feature function.*

6.5.2 k-nearest neighbour averaging

Another idea is to use the average feature value c_i of the k nearest neighbours (k-NN) of the query point x, i.e.

$$f_C(x) \equiv \frac{1}{k} \sum_{i \in \text{k-NN}(x)} c_i.$$

This feature function can be computed efficiently using a fast nearest neighbour searching technique [Merkwirth et al., 2000]. However, it is not differentiable, nor does it yield a uniform value far away from the surface.

6.5.3 Adaptive kernel smoothing

The third possibility is to smear out the given surface data points with a linear "molifier", e.g. the Wu kernel function used above. Thus,

$$f_C(x) \equiv \sum c_i k_\sigma(x, x_i).$$

If one wants exact values close to a data point x_i the smoothing kernel has to be well localised. In this case the feature function does not effectively extend its information away from the

surface and cannot be used for the optimisation of τ in regions further away from the surface than the supposedly small kernel support σ.

If on the other hand we choose kernel smoothers with larger support, then many data points potentially have to be summed to evaluate the feature function. This will significantly slow down the computations. It also does not yield the exact values c_i close to the points x_i.

Effectively, we are having multiple different smoothness assumptions on our function depending on the distance to the surface. But with just one kernel at hand we are implicitly enforcing a single measure of smoothness (see Chapter 2).

One solution to this problem is to use a multi-scale regression procedure similar to the one used for implicit surface reconstruction (see Chapter 3):

We set $f_C(x) = \sum_\sigma \sum_t \alpha_t k_\sigma(x, x_t) + b$. The offset b is fixed to the mean of the given feature values $\{c_i\}_{i=1,\dots,m}$. For each scale in decreasing kernel size order, we extract an equally spaced training set x_t from the given feature points $\{x_i\}_{i=1,\dots,m}$. The training values are computed as the kernel weighted averages of the neighbourhood, i.e.

$$y_t = \frac{\sum_i c_i k_\sigma(x_i, x_t)}{\sum_i k_\sigma(x_i, x_t)}.$$

The thus constructed training points give rise to a sparse kernel matrix and we use the same simplified ϵ-insensitive SVR as in Section 3.5 to compute a smooth regression function which extends from the surface into space. Again as above, on subsequent finer scales we just fit the remainder and set the final feature function to be the sum of the contributions of each individual scale.

This way the smoothness assumptions are less strong close to the surface where the feature function may have larger deviations. Further away from the given surface points the function will gradually become smoother and end up with the mean feature value.

Using the techniques explained here and in Chapter 3 we can then efficiently compute the feature function f_C and use it as a similarity measure on the whole domain $\mathcal{X}$.

6.6 Experiments

6.6.1 Effect of the signed distance cost function

Without the signed distance cost function (6.2) the transfer function τ is just matching the given landmark correspondence pairs and yields a minimal deformation due to the regularization term. This method is called a Free-Form-Deformation (FFD). In this section, we focus on how the signed distance cost function (6.2) influences the results.

Figure 6.4 demonstrates our deformation field algorithm in a simple toy environment. We fit implicit surface functions to two shapes, and construct a warp field using three given landmark pairs. We show its effect on some test points. If we neglect the distance to surface error term (6.2) in the objective (6.12), then the best warp is one which maps the corresponding points onto each other and leaves the geometry of the object unchanged as much as possible. The deformation field including (6.2), on the other hand, guarantees that surface points of object one are mapped onto surface points of object two. Also for off surface points, the implicit surface function value and thereby the distance to the surface is nicely preserved.

We next move to 3D data, including head scans from the MPI face database [Blanz and Vetter, 1999], two chess pieces taken from http://www.buckrogers.demon.co.uk and the head

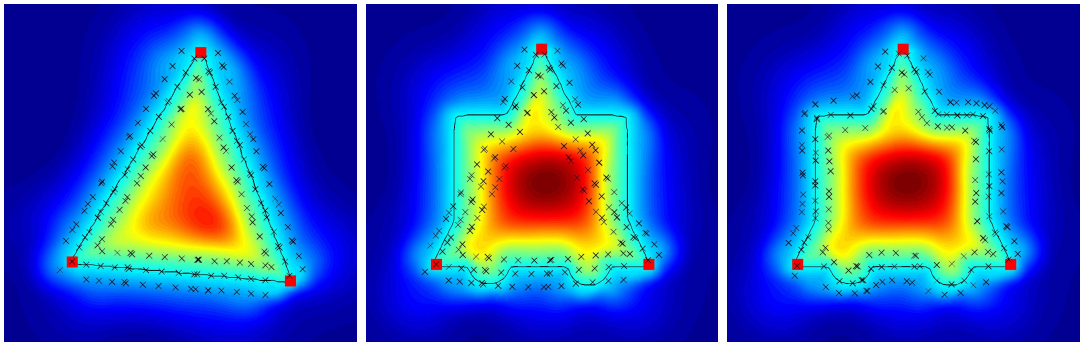


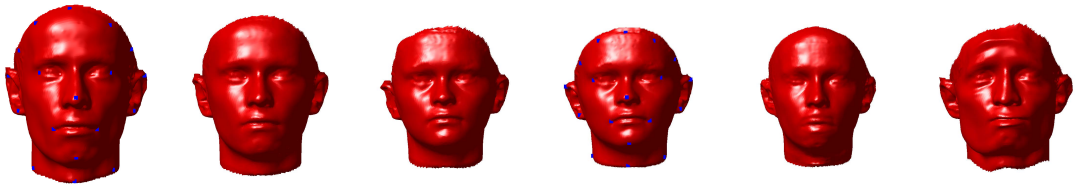
Figure 6.4: *A warp between the left triangle and the middle/right figure (black lines). The function values of the implicit surface functions are colour coded. We mark 3 correspondences (red squares) on both objects and visualise the warp on a number of test points (black crosses). In the centre image, the warp is estimated just from the given correspondences, i.e., without our signed distance cost function (6.2). Points between the triangles' corners do not fall on the target shape. With the signed distance term (right), the warp maps surface points to surface points. The distance to the surface is also preserved for off-surface points.*

of the Stanford bunny which can be downloaded from the Stanford 3D Scanning Repository (http://graphics.stanford.edu/data/3Dscanrep/).

In Figure 6.5, we took two heads and first defined 18 correspondences on distinct feature points as well as on the borders of the shape. Then, we fitted two implicit representations to the models and calculated the deformation field in the neighbourhood using our the proposed algorithm with the signed distance cost function (6.2). After having computed the deformation field τ we applied it to the vertices of a polygonal mesh describing the starting surface. We linearly interpolate between the initial and final positions of the vertices to get the intermediate steps.

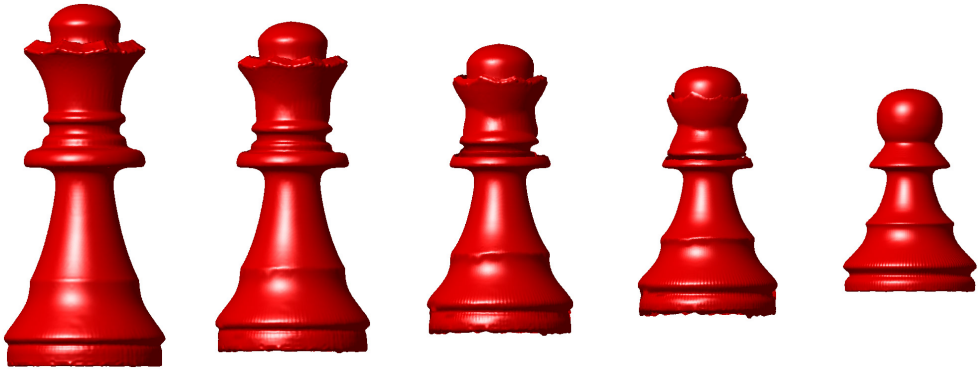
The resulting transformation looks visually plausible. It preserves the local geometry during the morph and produces valid and realistic shapes at intermediate steps. The transformed mesh approximates the geometry of the target object, and differences in the implicit surface function values between initial and target points are reduced to values on the order of 10^{-3} of the object diameter while keeping the correspondences within a distance of 10^{-4}. The same process was applied to yield Figure 6.7 and Figure 6.6.

Videos of the transitions presented above as well as other similar material can be found on http://www.kyb.tuebingen.mpg.de/bs/people/steinke/videos/.



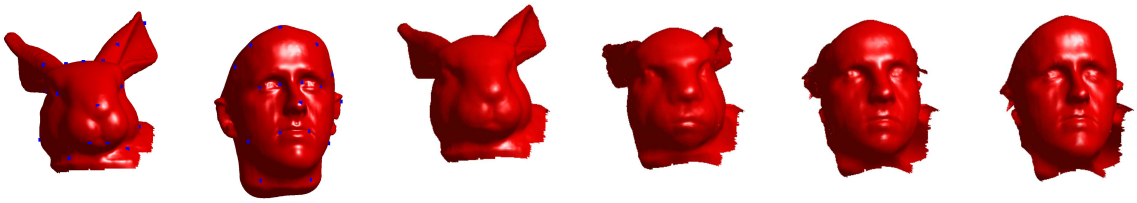
A Start | B 50% | C 100% | D Target | E No Corr. | F Features only

Figure 6.5: *A morph between two faces (A and D, with markers). Figures B and C show intermediate morphs of the mesh vertices. Note that the 100% image equals the target image D precisely. Figure E shows a 50% morph without correspondences where the surface is just projected from face A to D, producing ghost structures (e.g., a double mouth). Figure F shows an elastic deformation just using the point correspondences without the signed distance cost function (6.2): The corresponding points are mapped accurately, but the overall face structure is not aligned with the target D.*



Start | 25% | 50% | 75% | Target

Figure 6.6: *We used the same technique as in Figure 6.5 to generate a morph between a queen and a pawn. However, no pairs of landmark correspondences were given to the algorithm. Note, that due to rotational symmetry of the objects one could restrict τ to be a radially symmetric. Nevertheless we applied the general framework here giving reasonable results.*



Start | Target | 25% | 50% | 75% | 100%

Figure 6.7: *The Stanford bunny's head morphed into a human head. We used the same technique as in Figure 6.5.*

6.6.2 Feature functions constructed from surface data

Next, we show some examples how a feature function $f_C : \mathcal{X} \to \mathbb{R}$ may look like that is constructed only from surface feature information. One may think of colour or other texture properties that are just defined on the surface. We assume that the feature can be expressed as a real value and build a feature function $f_C : \mathcal{X} \to \mathbb{R}$ from it. More complex features can be encoded with several feature functions.

In Figure 6.8 and Figure 6.9 we used the algorithm proposed in Section 6.5.3 to extend scalar surface values to the whole volume $\mathcal{X}$ which was taken to be the whole space $\mathbb{R}^D$.

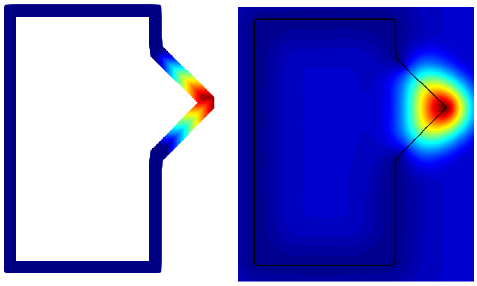


Figure 6.8: *(left) the surface feature values colour-coded, (right) the constructed feature function* $f_C : \mathbb{R}^2 \to \mathbb{R}$ *colour-coded*

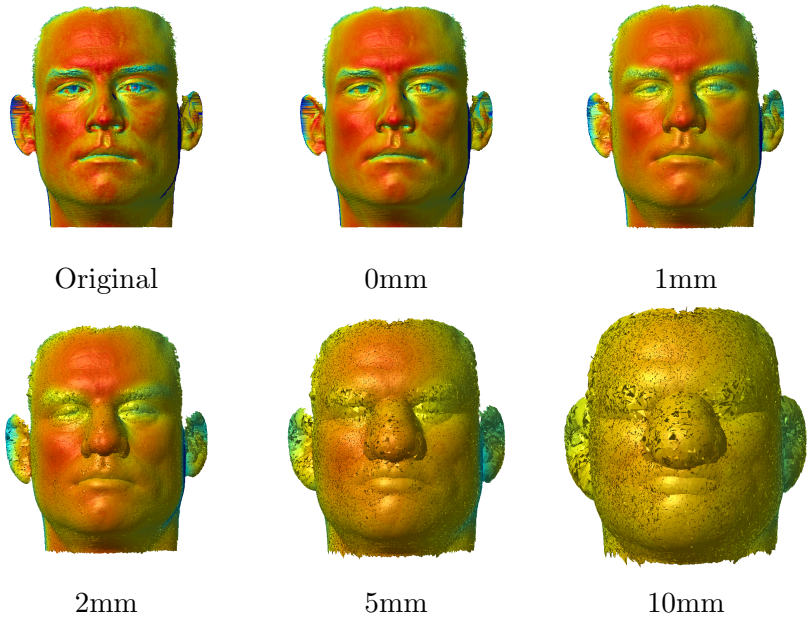


Figure 6.9: *The color intensity of a 3D head (top left) is used to build a feature function* $f_C : \mathbb{R}^3 \to \mathbb{R}$. *The intensity values are colour-coded in order to increase the visual contrast. The following images show the feature function* f_C *on surfaces that are at a fixed distance in front of the true surface. See, how surfaces close to the original accurately show the original texturing. The further we move off the surface, the more the texture properties are blurred towards the mean value.*

6.6.3 Effect of different locational cost functions

Next, we show experiments illustrating the effect of different locational cost functions.

We created two toy shapes in 2D (see Figure 6.10), that have an obvious correspondence, i.e. the bump on the right side is moved downwards.

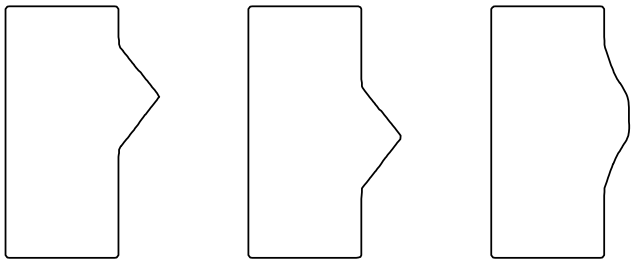


Figure 6.10: *(A,B) the 2D toy shapes, (C) a shape derived from a 50% implicit blending of (A) and (B). See how the surface feature (the sharp edge of the bump) is lost.*

First, we constructed the transfer function τ just using the signed distance cost function (6.2). The result in Figure 6.11 (left) shows that the upper part is moved inwards, the lower one outwards. This is the minimal transformation given our simple similarity criterion. If we now switch on the normal cost function (6.3), then the implied similarity measure is able to capture the idea that the bump is just moved downwards. See how the correspondence field adjusts correctly and how this implies that the 50% morph is visually plausible. Similarly, we can include color information on the surface that also implies a shifting the bump (see Figure 6.12). Again the error term based on the color feature function leads to the correct deformation field.

Compare these approaches to simple implicit blending (Figure 6.10,C) where the 50% morph, i.e. the zero level-set of $f_{50\%}(x) = 0.5f_1(x) + 0.5f_2(x)$, does not yield a satisfactory morph under the obvious assumption that just the bump moves and does not deform itself.

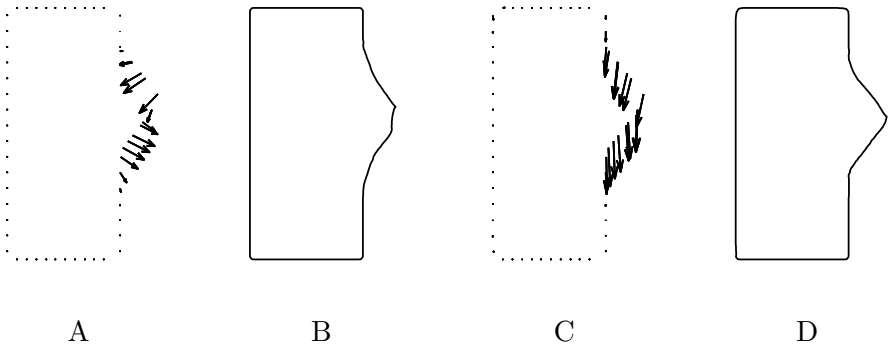


Figure 6.11: *(A) the transformation field τ based on distance to surface only, (B) the implied 50% morph, (C) the transformation field τ based on distance to surface and normal preservation, (D) the implied 50% morph*

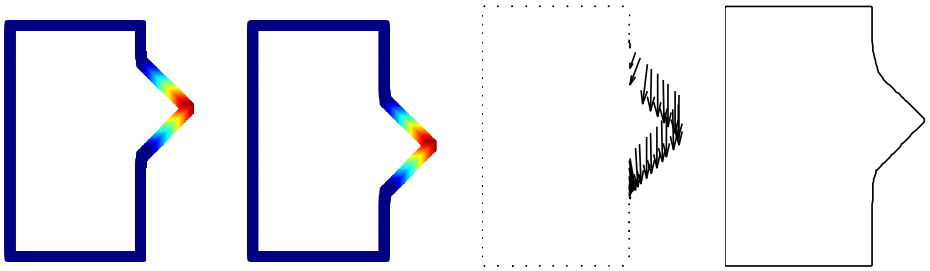


Figure 6.12: *(A,B) the two shapes of Figure 6.10 with color values given on the surface, (C) transformation field τ based on signed distance preservation (6.2) and color preservation (6.6), (D) the implied 50% morph*

In order to test the similarity measures on 3D surfaces we took the same heads as where used in Figure 6.5. If we combine the normal cost function (6.3) with the signed distance cost function (6.2), we can derive a visually plausible morph without any manually selected landmark point pairs (see Figure 6.13). The only manual interaction is a rigid alignment of the two faces. The two heads are of significantly different size (see Figure 6.2) adding to the difficulty of the correspondence problem.

Figure 6.13: *A morph between a male and a female head from the MPI face database (see also Figure 6.5). The deformation field was computed using just the signed distance cost function (6.2) and the normal criterion (6.3). No manually positioned markers are needed. The 3D morph – of which only the frontal view is given – was then computed by linear interpolation of the vertex positions of a mesh describing the initial head. The color was linearly interpolated in RGB space.*

6.6.4 Effect of preventing self-intersections

Our experiments show that in general penalising negative determinants $|J_\tau(x)|$ is very helpful: it stabilises the morphs and prevents foldings of the surface onto itself.

A 2D toy experiment is presented in Figure 6.14 and described in the figure caption. Note that the penalty term for self-intersections (6.11) effectively reduces the number of points where $|J_\tau(x)| < 0$. Thus, the transformation is locally invertible at all sampling points.

The determinant $|J_\tau(x)|$ is Lipschitz-continuous if the derivatives of the transfer function τ have this property. That is fulfilled, if one uses the proposed Wu kernel. If the values of $|J_\tau(x)|$ on the sampling/training points are bounded away from zero by a positive constant, one could construct trust regions around these points such that $|J_\tau(x)| > 0$ for all $x \in \mathcal{X}$ that fall into that region. For a dense enough sampling these trust regions would cover the whole of $\mathcal{X}$. Thereby, the morph would be injective on the whole domain and would strictly out-rule all self-intersections.

In our calculations we could not easily derive a tight bound for the Lipschitz constant of $|J_\tau(x)|$ and therefore could not define reasonably large trust regions. We assumed that our standard sampling scheme already provided invertibility to a high degree. Our experiments in section Section 6.7 build on the invertibility of τ and they did never fail within $\mathcal{X}$.

A 3D experiment is conducted in Figure 6.15. See how the proposed penalty term helps to stabilise the transformation and avoid wrinkles in the surface.

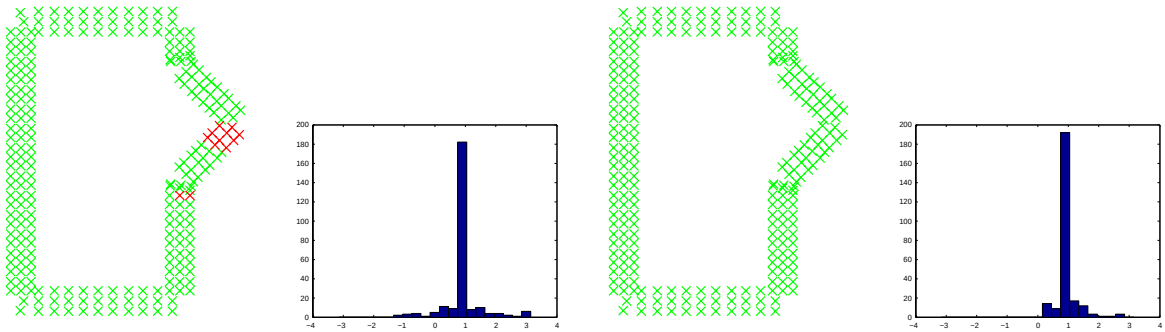
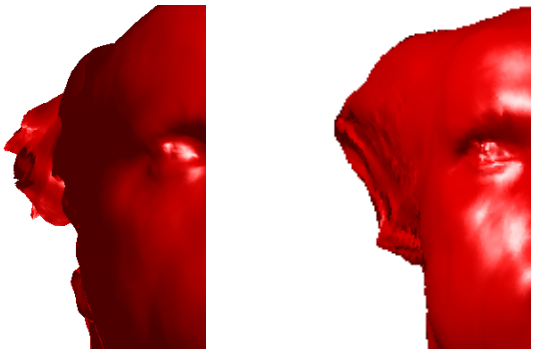


Figure 6.14: *Using shapes in Figure 6.10, we sampled some points around the initial shape and evaluated $|J_\tau(x)|$ there. (A,B) show values from a correspondence field τ computed without using the self-intersection error term (6.11). In (A), the values of the determinant $|J_\tau(x)|$ are color coded at the positions of the sampled points (green if greater than zero, red else); next to it in (B) a histogram of the determinant values. In (C,D) the same plots are shown for a correspondence field τ where the self-intersection error term (6.11) was included into the optimisation. See how on the left the are non-injective regions, especially in those areas where the transfer function takes non-trivial values. Using the designed error term, the local non-invertibilities are avoided and the determinant values $|J_\tau(x)|$ even have a positive distance away from the zero (see histogram in (D)).*

6.6.5 Effect of the deformation for off-surface points

Here, we show how the deformation field is not just defined on the surface but extends into space in a sensible way.

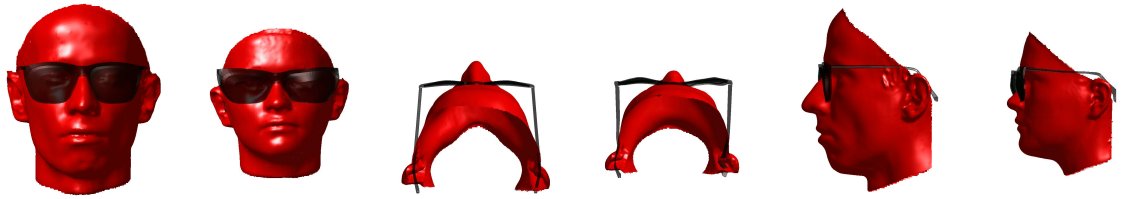


without (6.11) including (6.11)

Figure 6.15: *The two figures show the ear of the Stanford bunny morphed into the human face (see also Figure 6.7). We show 50% morphs. The left figure shows the effect of a correspondence field computed without penalising self-intersections (6.11), the right one uses this term. See how the object is crumbled and self-intersecting on the left, whereas it is rather smoothly deformed on the right.*

In the following experiment we fitted a pair of sunglasses to the initial head (we again took the MPI head dataset from Figure 6.5). We transformed the mesh of the glasses with the same warp τ that was used in Figure 6.5. That means that the deformation was computed without any knowledge about the glasses. The glasses are automatically deformed to fit the smaller head, and to preserve the distance to the surface (see Figure 6.16).

The same effect can be seen in Figure 6.4. The distance to the surface for off-surface points is nicely preserved by the warp τ.



A B C D E F

Figure 6.16: *To demonstrate how our deformation field extends from the surface into space, we put sunglasses on the start head (Figures A, C and E). The mesh vertices of the glasses were transformed with our deformation field to yield Figures B, D and F. Note how the glasses are smoothly deformed to fit the new head and to keep the distance to the facial surface approximately constant.*

6.6.6 Speed

The computational effort to construct the warps τ depends significantly on how densely the desired warping volume $\mathcal{X}$ is sampled and how many scales are used. It also depends on the number of features in the locational cost function.

Using just one scale we examined the runtime for increasing sample sizes m and different locational cost functions. We got the following results for an optimisation with 100 rounds

on a 2 GHz PC with 1 GB memory: The underlying test objects were the heads depicted in Figure 6.13. All times are given in seconds.

m	44	158	828	4288
d2s	1.3	3.3	13.2	63.4
d2s + si	1.5	4.9	24.2	126.8
d2s + si + norm	2.9	9.4	43.6	221.1
d2s + si + norm + col	3.6	11.4	52.4	256.8

d2s: using the distance to surface error term (6.2), si: using the self-intersection error term (6.11), norm: using the normal error term (6.3), col: using the color error term (6.6)

The different settings all scale linearly in the number of sample points m.

For a complete morph of 4 scales which seemed to be reasonable for the head datasets the overall runtime was on the order of 15 min using 22k kernel centres. The number of kernel centres seems quite high for these comparably easy shapes. An adaptive sampling scheme could potentially save a lot of computing time.

6.7 Partial correspondence - transplanting objects

Often we want to find a correspondence field between an object O_1 and just a part of a second one O_2. If we choose O_1 to be our source and O_2 to be the target object, our algorithm is able to handle that without any modifications.

This possibility opens up a nice application: The estimated deformation field τ maps points on O_1 onto O_2. In regions far off O_1 – outside $\mathcal{X}$ – τ is equal to the identity map. Additionally, we enforce invertibility of the deformation using the penalisation term (6.11). Given that O_1 just maps to a small part of O_2, we could transform the whole of O_2 with the inverse deformation field τ^{-1}. Firstly, this would yield the original object O_1. Secondly, in areas further away from $\tau(O_1)$ we would get back the second object, as τ^{-1} approaches the identity map there (similar to τ). In between, we would get a smooth transition.

For an example see Figure 6.17: we took a human head and matched it with a part of a cylinder. Then we employed τ^{-1} onto the whole cylinder. This yielded another cylinder with the human head smoothly transplanted into it.

How can we effectively compute the inverse mapping τ^{-1}? The obvious idea is to optimise the following objective for each query point y:

$$\tau^{-1}(y) = \operatorname*{argmin}_{x \in \mathcal{X}} \|\tau(x) - y\|$$

However, this is a non-trivial non-convex optimisation problem. In order to guarantee a smooth inverse mapping and to enable effective evaluations of the inverse field, another idea is needed.

We propose to take points describing O_1, get their transformed locations $\tau(O_1)$ and train a deformation μ with the proposed deformation algorithm. If the point pairs $\{O_1, \tau(O_1)\}$ in reverse order are used as input, i.e. as landmark correspondence pairs, and no other locational cost functions are employed, then μ will closely resemble the actual τ^{-1}. It can be computed very efficiently. Having constructed μ we can quickly evaluate it in order to transplant the shapes as presented in Figure 6.17.

Figure 6.17: *The head O_1 (left) transplanted onto the cylinder O_2 (middle) to yield the mixed object (right).*

6.8 Evaluating the correspondence field

For computing the transfer functions τ, a crucial point is to choose the right weighting parameters $\lambda_{loc}, \lambda_p, \lambda_{si}$. These have to be determined on each scale separately to give the best results. Determining the parameters is hence a high dimensional optimisation problem itself.

The key question is how one can asses the quality of a correspondence field automatically. A human can easily validate a correspondence field by judging the visual plausibility of a transformation video. It is important that parts with the same meaning are mapped onto each other and no blending of non-matching pieces occurs. However, this evaluation requires high-level cognitive understanding of the objects, which (so far) cannot be performed on a computer.

In order to effectively optimise the parameters we nevertheless need some automatic quality measurement. Below we will give some necessary conditions that a good morph would have to fulfil. However, these are not sufficient to uniquely determine whether a proposed correspondence field is visually correct.

1. The mapping τ should be surface-to-surface, i.e. the signed distance should be closely preserved. That property should not only be the case for the sampled points that were included in the optimisation procedure but for all points in $\mathcal{X}$. Thus, we also have to test how well our function generalises.

2. If the locational cost function terms ((6.3), (6.6), ...) can effectively be minimised, this is a strong hint that the correspondence field makes sense. Because then we have discovered an inherent similarity in the dataset. However, the converse is not true. If

we cannot minimize the locational costs very well, we may just have chosen the wrong features, or the two objects are similar just on a semantic level (think of a photographed head vs. a sketch of a face).

3. If a sufficient number of matched landmark point pairs on the two objects is available, they can be used to evaluate the quality. One could optimise the parameters in an leave-one-out scheme or try to compute the morph without the landmark points and then only use them for assessing the quality.

4. One more advanced test is to compute the correspondence field from object 1 to objects 2 and afterwards from object 2 to to object 1. If our algorithm gave perfect correspondences the concatenation of the two transfer functions should yield the identity map on $\mathcal{X}$. To measure the degree to which this criterion is fulfilled one could again sample some points from $\mathcal{X}$ and determine $\sum_i \|\tau_{2\to1}(\tau_{1\to2}(x_i)) - x_i\|$.

Optimising the parameters for every morph separately is very time consuming. Yet, we may hope that the parameters can be transferred within an object class without loosing much quality. Our experiments showed that this is the case for 3D head models.

Another issue is the multiplicity of scales. Optimally, one would propose a set of parameters for all scales at once, then compute the morph and finally judge the quality of it as described above. However, the number of parameters that increases linearly with the number of scales renders it impractical to test a suitable dense set of parameters. The number of trials would explode exponentially with the number of scales.

Instead, we are optimising the parameters on the coarsest scale first. Then we pick the optimal parameter set for the first scale and use it when selecting parameters in the second scale, and so on. This does not necessarily yield the globally optimal parameter set but it seems a reasonable assumption that a good morph will behave nicely after each scale. Using this hierarchical parameter optimisation method, we can limit the number of trials to be linear in the number of scales.

6.8.1 Parameter selection experiments

Initially, we tested our hierarchical method for the 2D toy problem depicted in Figure 6.10. For selecting the parameters λ, we used a quality function depended on the distance to the surface (1) and on the invertibility criterion (4) , i.e. for some points $\{x_i : i = 1, \dots, m\} \subseteq \mathcal{X}$ we computed

$$E(\lambda) = \frac{3}{m} \sum_i |f_1(x_i) - f_2(\tau_\lambda(x_i))| + \frac{1}{m} \sum_i \left\| x_i - \tau_\lambda^{-1}(\tau_\lambda(x_i)) \right\| . \tag{6.17}$$

First, we did not use any special feature functions, but just optimised the weights λ_{d2s} of the signed distance cost function (6.2). The resulting values of $E(\lambda_{d2s})$ for different λ_{d2s} are shown in Figure 6.18. We can then select the λ_{d2s} yielding maximal quality – minimal $E(\lambda_{d2s})$. This yielded already a modest result for the correspondence field τ (see Figure 6.19, left).

Then we fixed the parameters of the signed distance cost function λ_{d2s} to the optimal ones found, and searched for the best weightings of the other locational cost functions. We included the normal (6.3) and the color (6.6) locational cost function into our search. A non-zero

weighting of the normal feature function (6.3) ensures better invertibility and is thus preferred. Using these features we are able to detect the shift of the bump automatically without any manual interaction – not even in the choice of the feature function (see Figure 6.19, right).

The found optimal weighting parameters then were

Scale	1	2	3
λ_{d2s}	10^3	10^3	10^4
λ_{normal}	10^3	0	10^4
λ_{color}	10^5	10^5	10^6

Throughout all experiments we manually set the weight for the self-intersection term λ_{si} to a high value, i.e. 10^6. This term only has an effect, if violations are detected, and does not influence the optimisation as long as there are no self-intersections.

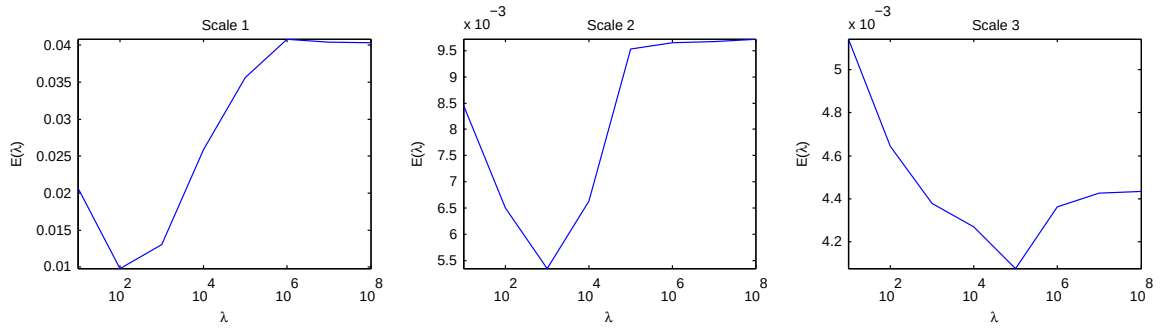


Figure 6.18: *The validation error (6.17) for different weights $\lambda = \lambda_{d2s}$ of the signed distance cost function (6.2). On each scale, we can find a unique minimum whose parameter value is then used for validating the parameters on the refined scales.*

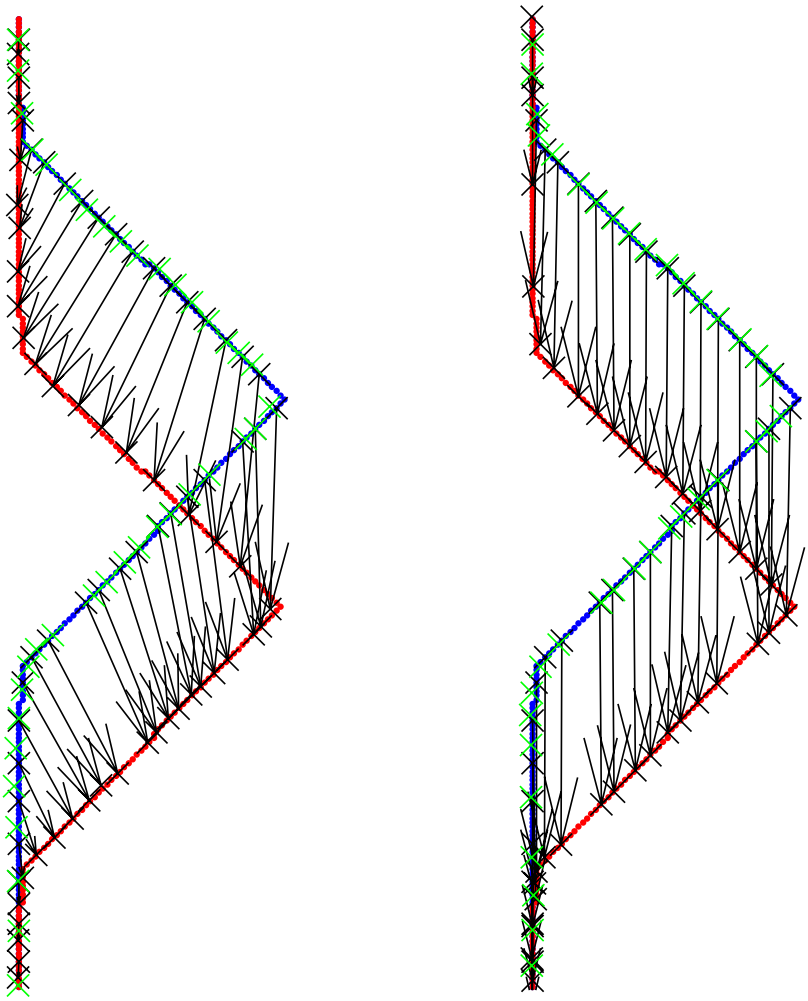


Figure 6.19: *Magnified parts of morphs between the two test objects in Figure 6.10: (blue) the initial shape, (red) the target shape, (black, with arrows) corresponding surface points, (green) the black points mapped back with the inverse transformation that was computed independently (with the same parameters). The parameters were selected automatically minimizing the quality functional $E(\lambda)$ (6.17). (left) only signed distance cost function used, (right) also normal and color features used.*

A B C D

Figure 6.20: *The head (A) morphed into head (B): in (C) and (D) the deformed head (A) is shown. An optimal morph should match the shape of (B) exactly while preserving the colouring of (A). The parameters were selected automatically to minimise the quality functional $E(\lambda)$ (6.17). In (C) no feature functions were used, in (D) color and normal features were used. See that (D) matches the head (B) much better than (C), but is far from being perfect.*

We also tested the parameter selection method as explained above on 3D head models (see Figure 6.20). The obtained parameter set is plausible and the resulting morphs are of moderate quality. However, we could achieve better results when selecting the parameters manually (see Figure 6.13).

Another observations is that the minimal values of $E(\lambda)$ still ranged around 2% of the object diameter. For comparison, the identity warp would yield a value of 5%. Thus, the optimisation does not significantly decrease the error measure.

The are three possible reasons to explain these observations:

- We did not find the global minimum for the individual optimisation problems (6.12) that compute the morphs τ_λ.

  Because we cannot effectively judge the morph quality automatically, it is hard to know, when to stop the optimisation. We typically used a fixed number of steps.

  Additionally, the optimisation problem (6.15) is highly non-linear. We do not know whether we got stuck in one of the many local minima.

- We did not test the optimal weighting parameter set λ. It is lying somewhere in between our (rather coarse) test grid.

- There is no weighting parameter set λ that reduces $E(\lambda)$ to really small values. We need other locational cost functions to guide the correspondence computation.

However, so far we cannot efficiently test which of these three hypothesis is the problem. If we can guess a better parameter set λ than what the automatic selection procedure yields, we can use it. But we cannot know in advance, whether it really exists.

We also tested a quality criterion that judged the quality of our correspondence field relative to the correspondence information given from the MPI head database [Blanz and Vetter, 1999]. These correspondences seem quite plausible – especially in distinct feature regions like mouth, eyes, etc. –, however, they cannot be regarded as complete ground truth. The automatic parameter selection scheme was not able to reproduce the given correspondence information. Again, the objective value could not be reduced significantly and the same problems as above occur.

6.9 Discussion

Our algorithm for computing dense correspondence fields is a comparatively easy method and achieves state-of-the-art results.

It is relatively easy, because it avoids a highly non-trivial parameterisation step. At the same time our method is able to deal with arbitrary objects and we do not pose any restrictions onto topology or the like.

The transformations that we are computing are differentiable and thereby stable to small deviations of the input. This is not the case if one would just project the first surface onto the second – apart from the fact that simple projections do not align the relevant features.

Off-surface points can be matched in a sensible way that is just determined by the information of the surfaces. This may turn out to be a useful property if one for example wants to transfer some bone structure within a head from one person to another. The off-surface training points also increase the stability of the transformation.

Nevertheless, there are some problems to this approach. The crucial point is that we do not have a highly reliable, non-interactive way of evaluating the quality of a morph. Without the high-level recognition that is so far just achieved by human observers a reasonable discrimination between good and bad morphs is difficult. This makes it hard to select the select the correct parameters automatically. It is also hard to know when to stop the correspondence optimisation routine.

These problems are rather severe, and in order to use the presented algorithm in real world applications one would have to come up with some new approaches to solve them. However, the difficulty to evaluate the morph quality automatically is a problem that also occurs for many other correspondence algorithms.

One algorithmic modification to enhance the results could be to use another optimisation algorithm: The objective (6.15) is highly non-linear. Stochastic gradient methods show some robustness not to get stuck in local minima. One could try algorithms like SMD [Schraudolph, 1999].

Another way to improve the method is to combine our dense correspondence algorithm with an initial step where characteristic features located on sparse interest points are matched in a pairwise manner. These matched point pairs could be included as landmark correspondences to jump-start our algorithm. We may be able to combine the invariance that can be achieved with point based approaches with the advantages given through a full correspondence field that is computed in our approach.

Sometimes two corresponding points may not look very similar on two given objects, because their correspondence is just defined in a semantic way. If not only two distinct objects are given but a complete series where one object is slowly transformed into the other, one might capture some sense of that high-level similarity by tracking a given point. E.g. the corner of a mouth does not necessarily look similar for different facial expressions. Yet, in a video the corner moves smoothly and we may be able to detect a valid correspondence field in each small step, thereby constructing a correspondence between two not so similar objects.

The presented morphing algorithm is not dependent on the dimension of the input space $\mathcal{X}$. One might also try to solve higher dimensional problems. One example application could be the following: The implicit surfaces described in Chapter 3 can also model 4D objects, e.g. 3D objects plus a time dimension, in which these objects are deforming. With the help of

the correspondence algorithm described here, we could try to match two of those 4D objects in order to get a smooth morph between two videos – video morphing. Related work in this area – although using different techniques – is [Sand and Teller, 2004].

There is an intrinsic dead-lock between correspondence computations and recognition: The latter is typically based on the former and vice versa. In the future we could therefore try to explore methods that iteratively solve the two problems in a single approach.

Chapter 7

Conclusions

Implicit surfaces are a valuable tool for many applications in computer graphics. Current research is often directed towards animating objects that are solely represented as point clouds. In order to render these points, techniques similar to our implicit surface reconstruction approach are employed. We hope to have given a suitable tool that allows easy working with these point based representations.

The problem of computing correspondence fields between objects – especially surfaces in 3D – has long been studied. We have presented an algorithm that is structurally simple but still allows for state-of-the-art performance.

There is on-going work to add an easily usable interface to our morphing algorithm and to produce a plug-in for a standard 3D modelling framework that incorporates our algorithms.

An easy to use correspondence algorithm that works mostly automatically would allow to build object models that consist of hundreds or even thousands of templates of one class. This will finally yield enough flexibility in these models to trustfully recognise more and more complex situations. The key problem so far has been a high quality, automatic correspondence computation.

Overall, we have shown some interesting approaches of how to use machine learning tools in the computer graphics / computer vision field. These methods build a powerful, yet simple framework for doing inference or – more loosely speaking – estimating functions. Many real world problems of computer graphics can be casted in such a way and we expect many more applications to be seen in the future.

Appendix A

Coefficient Magics

In this chapter we will describe some ideas that so far did not lead to successful experiments. We have not yet explored all possibilities to improve the results, and it remains to be seen whether the proposed ideas will in the future lead to successful applications. We will argue, that this will most likely be difficult.

A.1 Introduction

So far, we have constructed implicit surface functions which take the form

$$f(x) = \sum_s \sum_i \alpha_{i,s} k_s(x, x_i) + b \tag{A.1}$$

where s lists the different scales. In this chapter we examine whether and how the α_i are characteristic for the local shape of the described surface.

If one could correlate the α_i and the local shape of the described surface, the idea is that one could match the kernel centres x_i from two different surfaces based on the characteristics of the α_i. One could then morph between the two surfaces by shifting the α_i from one of the implicit representations to the other one. Note, that with this matching algorithm we do not compute a dense correspondence (like in Chapter 6) but match the kernel centres instead. This method would be some new form of implicit blending which may have the possibility to preserve surface features during the morph and map parts with the same meaning onto each other. As seen in Section 6, this is a vital property for an acceptable morph.

The research in this direction was motivated by the following observation (see Figure A.1): The figure depicts four different 3D scans of the same person, two of them are almost but not completely identical. We computed implicit surface representations from the given point clouds and then coloured a surface mesh with a quantity depended on the local coefficient characteristic. Thus for every mesh vertex $v \in \mathbb{R}^3$ we computed

$$\sum_{\{i : x_{i,s} \text{inside the surface}\}} \alpha_{i,s} k_s(v, x_{i,s}). \tag{A.2}$$

The resulting images show that corresponding parts have similar colours. Convex shaped parts are colour more red, concave ones more blue. This similarity may be exploited in a matching algorithm even though it is far from being perfect.

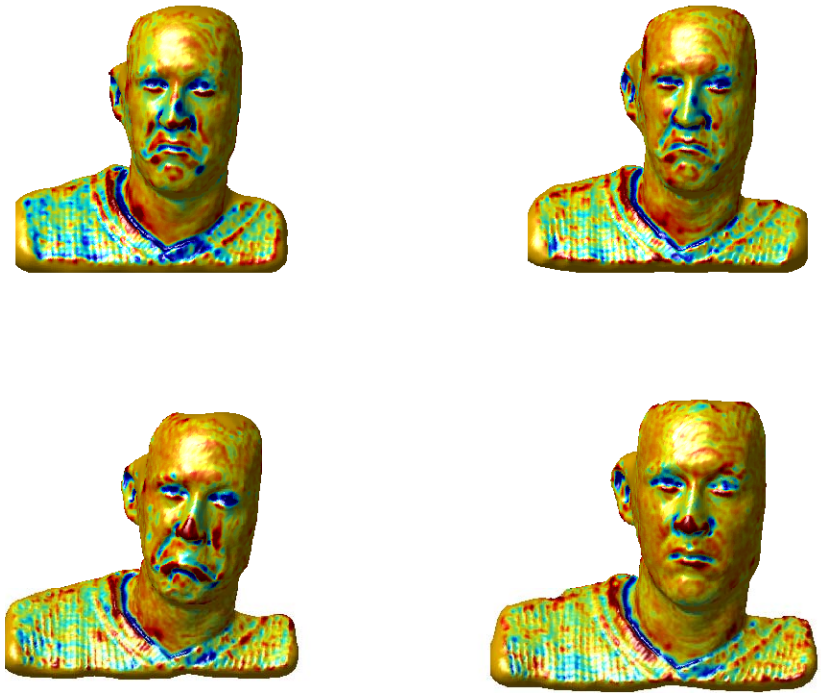


Figure A.1: *The coefficient code (A.2) for scale $s = 5$. Kernels in this scale have a width $\sigma = 1/32$ of the head diameter. The heads are taken from a time sequence of 3D scans. The first two (upper row), were recorded at subsequent time steps. Thus, they describe more or less the same geometry, but are constructed from structurally different point sets.*

In Section A.2, we show how one could match kernel centres depending on the local coefficient characteristics using the earth-mover distance (EMD). In Section A.3, we apply the theory onto two 2D toy datasets. We also will test the hypothesis of locally characteristic behaviour of the kernel coefficients α_i more closely. Unfortunately, we conclude that this assumption seems not to hold in general.

A.2 The Earth-Mover-Distance (EMD)

The Earth-Mover-Distance (EMD) is a well known distance function mainly applied to histograms or weighted point sets in $\mathbb{R}^D$ [Rubner et al., 2000]. It is used in many fields of research, often in content based image retrieval [Rubner et al., 2000] or in contour matching [Grauman and Darrell, 2004].

The idea behind EMD is that we have to move the some mass - i.e. the "earth" - which is initially located at some piles to some holes at different positions. We want to do so while performing as little work as possible. Work in this case is measured as mass times the distance it has to be transported.

Suppose we have two sets of piles of earth. The similarity between the two sets could be measured by taking one set as the initial mass sources and count the second set as holes. If the two configurations are identical, the total work would be zero because all points are matched with themselves and thus all the transport distances are zero. For non-identical point sets the amount of work to move the mass seems to be a good intuitive distance measure. If the sum of mass on both sides is the same the EMD is known to be a true metric [Rubner

et al., 2000].

Of course, one can easily interpret this problem as a graph problem, a property which we will later-on use to actually compute the EMD (see Section A.2.2). The algorithm is very similar to a minimum-cost matching but we do not restrict ourselves to one-to-one mappings, one pile might also be transported to two distinct holes. Computing the EMD not only provides us with a similarity score but also yields a mapping of how to transport the mass.

In our application, we imagine the mass/earth to be the coefficients α_i located at the kernel centres x_i. If two implicit surface functions are identical we do not have to move the coefficients at all, thus, their EMD is zero. For distinct implicit surfaces we would move the coefficients onto each other in a way that minimises the work. This could yield a sensible distance measure. If we morph by moving the coefficients from the piles to the holes (see Section A.2.3), we may retain the local surface structure during the transformation if the coefficients are locally characteristic.

A.2.1 Formal definition of the EMD

Let us imagine two weighted point sets $P = \{(x_i, \alpha_i)\}_{i=1,\dots,m}$, $Q = \{(x_j, \alpha_j)\}_{j=1,\dots,n}$ with all weights $\alpha_i, \alpha_j \geq 0$. Let f_{ij} be the amount of weight α_i moved from point $x_i \in P$ to $x_j \in Q$ and $d_{ij} = \|x_i - x_j\|$ be the distance between the two points x_i and x_j. Then

$$\begin{aligned}
\text{Work}(P,Q) &\equiv \min_{f_{ij}} \sum_i \sum_j d_{ij} f_{ij} \\
\text{subject to} \quad f_{ij} &\geq 0 \quad \forall i = 1,\dots,m,\ \forall j = 1,\dots,n \\
\sum_j f_{ij} &\leq \alpha_i \quad \forall i = 1,\dots,m \\
\sum_i f_{ij} &\leq \alpha_j \quad \forall j = 1,\dots,n \\
\sum_i \sum_j f_{ij} & \quad \text{maximal}
\end{aligned}$$

The EMD distance then is the normalised work, i.e.

$$\text{EMD}(P,Q) \equiv \frac{\text{Work}(P,Q)}{\sum_i \sum_j f_{ij}}.$$

For our purposes we have to deal with negative coefficients α_i as well. One could either allow for $f_{ij} \in \mathbb{R}$ or divide up the point sets P, Q into two parts each, one with the positive coefficients and one with negative coefficients. One could then just match the sets of corresponding sign and in the end sum the results.

The first method would lead to certain problems: Imagine one zero coefficient in P and two coefficients of large absolute value but of opposite sign on the target Q. Then, the EMD with $f_{ij} \in \mathbb{R}$ could match the one point with the two target points without any cost. This does not make sense in our context as we want to preserve the coefficients as much as possible. One could alternatively set the cost to be minimised to $\sum_{i,j} |f_{ij}|\, d_{ij}$. However, this yields a non-convex, non-linear problem that is hard to solve.

The second method divides up the problem into two parts and then sums up the flows and costs to get the final EMD. This seems more reasonable and also easier to implement in our case.

The used implicit surface functions (A.1) contain several scales. We always moved coefficients just within one scale. Otherwise one would have to think of a system how to account for the different kernel sizes σ.

A.2.2 Computing the EMD

In order to efficiently compute the EMD it can be reduced to a standard graph problem - namely min-cost-max-flow - which can be solved efficiently with standard algorithms.

Imagine the fully-connected bipartite graph of all kernel centres in P pointing towards all centres in Q. The EMD problem is then very similar to the min-cost-max-flow problem, which reads

$$\begin{aligned} &\min_{f_{ij}} \sum_i \sum_j d_{ij} f_{ij} \\ &\text{subject to} \quad f_{ij} \geq 0 \quad \forall i = 1, \ldots, m, \ \forall j = 1, \ldots, n \\ &\qquad \sum_j f_{ij} = \alpha_i \quad \forall i = 1, \ldots, m \\ &\qquad \sum_i f_{ij} = \alpha_j \quad \forall j = 1, \ldots, n \end{aligned}$$

The difference is that for the EMD we do not know how much mass is going to be moved. We just want it to be maximal. On the other hand in the min-cost-max-flow problem, all mass is being moved. We can resolve this difference by introducing a dummy node x_0 that is connected to all other nodes on both sides. The costs d_{i0}, d_{0j} of these new edges are set to twice the maximum of the distances d_{ij}. We also set the coefficient of the new node to be $-\sum_i \alpha_i + \sum_j \alpha_j$. Then the total sum of mass in the constructed network is zero, and the min-cost-max-flow problem has a feasible solution that is equivalent to the EMD problem. Because the cost for moving mass trough the dummy node is higher than moving it through any edge of the original network, the amount of mass being moved on the original edges will be maximal.

We apply a second simplification to the graph problem: The fully connected bipartite graph would have $O(N^2)$ edges, where $N = \max(m, n)$. Thus, big problems ($N > 10^4$) are not solvable due to memory and time constraints. To sparsify the graph we delete all edges longer than some distance d. This reduces the number of edges to $O(N)$ as the kernel centres are approximately equally spaced. d is chosen in relation to the kernel width σ that is used on the scale we are operating in. We typically set $d = 2\sigma$. The intuition behind this is as follows: If the two implicit surface functions are roughly similar it will normally not be necessary to transport mass very far away and the simplification will give a similar result to the original problem.

For multiple scales we might have introduced a severe limitation: The mass movement is restricted to smaller and smaller neighbourhoods as the kernel size σ shrinks. On finer scales we therefore do not delete the edges with $d_{ij} > 2\sigma$ but for each kernel centre x_i we search the closest kernel centre c_i of one scale above. We then pick the centre c_i' to which c_i moves most of its mass. We accept all edges (i, j) which have $\|x_i - c_i\| + \|x_j - c_i'\| < 2\sigma$.

The above min-cost-max-flow problem can be implemented very efficiently using CPLEX. It yields run-times of just a few seconds for problems with up to 10^6 nodes. For numerical stability we additionally set $f_{ij} \geq \epsilon > 0$ and neglect coefficients in the range $(-\epsilon, \epsilon)$. These would not contribute much to the EMD anyways.

In experiments it also turned out to help restricting flows from kernel centres inside the first shape to kernel centres inside the second shape - vice versa for outside lying points (see Section A.3).

A.2.3 Morphing with the EMD

Suppose the EMD problem has been solved for two implicit surface functions yielding the flows f_{ij} for all kernel centres x_i and x_j. One could then create a morph between the two shapes by moving the expansion coefficients α_i along the edges which posses non-vanishing flow f_{ij}. Parts of coefficients that cannot be moved through the network would be linearly switched on or off respectively. Thus, intermediate implicit surface functions would look like

$$\begin{aligned} f_\lambda(x) \equiv & \sum_i \sum_j f_{ij} k(x, \lambda x_i + (1-\lambda) x_j) \\ & + \lambda \sum_i (\alpha_i - \sum_j f_{ij}) k(x, x_i) \\ & + (1-\lambda) \sum_j (\alpha_j - \sum_i f_{ij}) k(x, x_j) \end{aligned}$$

for $\lambda \in [0, 1]$. This morph would minimize the EMD distance during all the steps and therefore would preserve the local surface features as much as possible - under the assumption that the local coefficient structure determines the local surface properties, and that the EMD flow computation has matched corresponding surface parts.

A.3 Experiments

A.3.1 A killer toy experiment

When applying the above presented theory to a tiny, well-constructed toy problem, we achieved very good results (see Figure A.2):

We constructed an implicit surface function with only 3 kernel centres. As a second object we took the same function but one kernel centre was split into two points that were lying closely to each other. The coefficient was evenly split among the two centres. This situation would for example occur if we reconstructed an implicit surface function several times based on the same geometry but different surface samplings.

Even though the second function is additionally rotated and translated, the EMD finds the correct correspondence structure. Thus, the intermediate morphing steps look plausible and preserve the local surface structure.

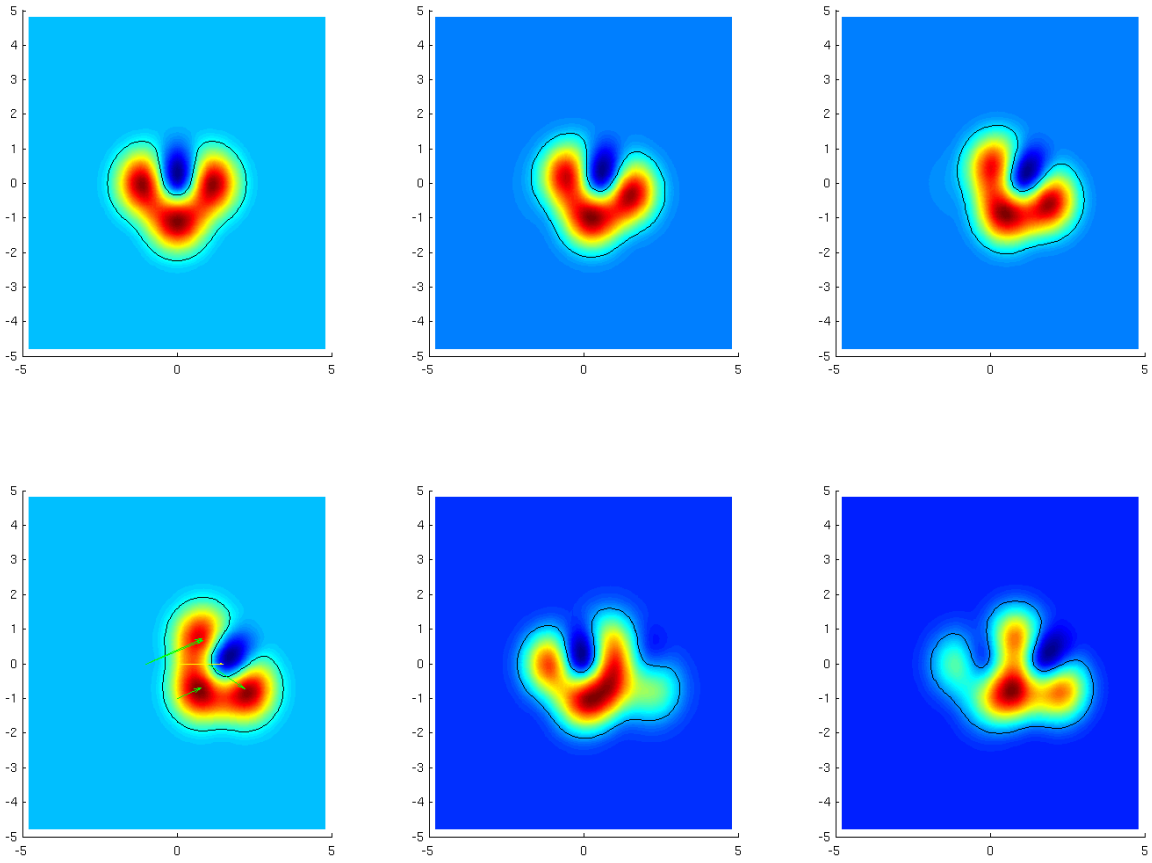


Figure A.2: *A simple test function in 2D whose values are colour coded and which defines an implicit surface (the black line). The upper left image shows the original function. The lower left image shows the same function but rotated, shifted and one kernel centre split into two. The arrows in the lower left image show the correspondences found by the EMD algorithm applied to the two depicted functions. Using these correspondences we can produce nice intermediate shapes (upper right figures, 30% and 60%). Compare them to figures obtained from simple convex combinations of the two functions (lower right figures, again 30% and 60%).*

A.3.2 The (almost) complete failure

However, when we tried to use our method also on bigger problems it generally did not perform well.

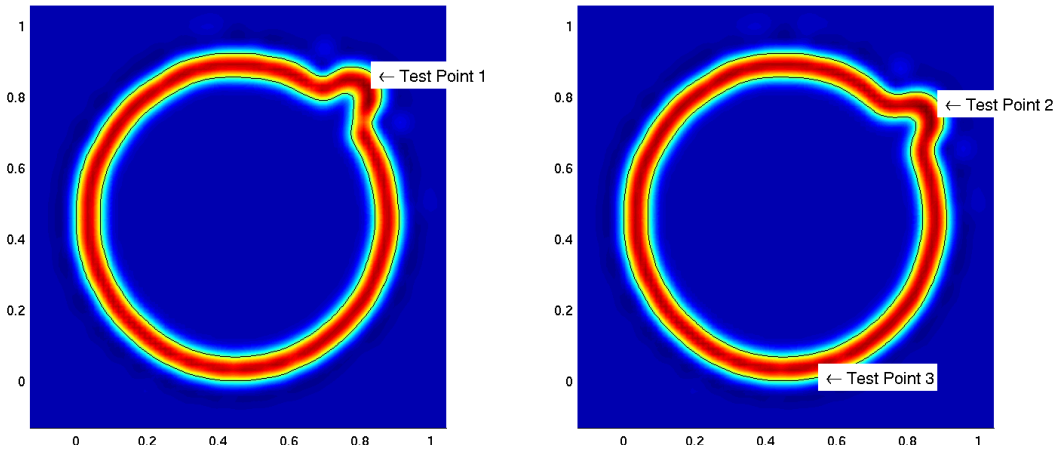


Figure A.3: *Two identical shapes (black) that are just rotated by 10°. The implicit surface functions are colour coded. For simplicity reasons we just used one scale for reconstruction.*

Figure A.3 shows two shapes and corresponding implicit surface functions. The circular shapes with one bump are almost identical, just rotated by 10°. The implicit surface functions are computed from different samplings of the surface, but describe the same geometry.

If we morphed based on a simple convex combination of the two implicit surface functions, one bump would disappear and a new would grow. With the EMD method the hope is that the local shape properties are preserved maximally, i.e. the bump is moving.

In Figure A.4 the results of the EMD computation are shown. The correspondences are not found correctly. Some mass is not transported locally as it should be, but far away. The correspondence links depicted by the arrows also do not show a consistent orientation. Using this information, the resulting morphs deform the surface in a chaotic way.

Similarly chaotic results for a 3D example are shown in Figure A.5.

A.3.3 Possible explanations

Why did the described approach fail so badly? We examine two possible reasons:

Firstly, we check whether the approximations we applied in order to compute the EMD efficiently are disturbing the results. Secondly, we recheck the basic assumption - namely that the local coefficients are characteristic of the local surface properties.

We have simplified the original formulation of the EMD quite a bit in order to compute the EMD more efficiently. Maybe some of the simplifications were not justified. Especially it could be that the sparsification of the EMD graph introduces some new constraints that do not allow for a sensible correspondence. That is why we computed how much mass was moved on average. If the reduced linkage did not allow the mass to be assigned to a corresponding partner we would have to include more links. However, on average about 90%-100% of the coefficient mass was transported through the constructed sparse network. Thus, deleting some edges of the dense graph seems to be a valid approximation.

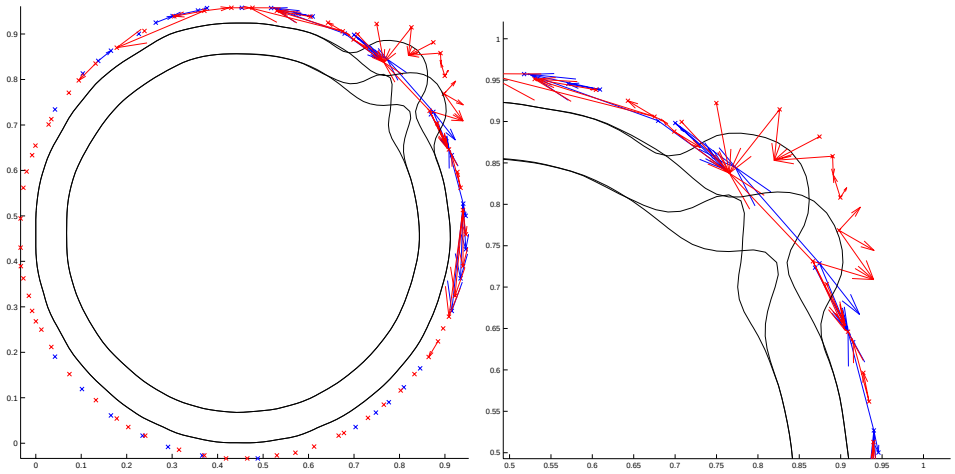


Figure A.4: *The flows of a EMD computed for the shapes in Figure A.3. The EMD flows shown here were computed just between all kernel centres lying outside the shape. This further simplifies the correspondence problem that the EMD has to solve. Red arrows symbolise a negative flow (red crosses denote kernel centres with a negative coefficient) and blue arrows stand for positive flows (blue crosses respectively). Despite the simple problem statement, the algorithm fails – the correspondence arrows do not consistently point in one direction.*

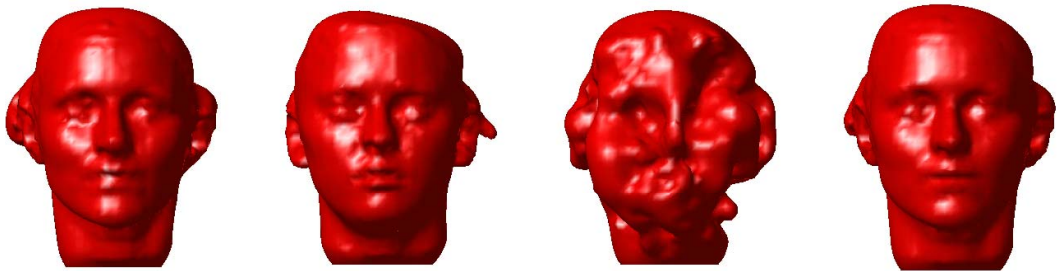


Figure A.5: *The two heads on the left are morphed using the EMD (middle right). In the right image a simple blending of the implicit surface functions is shown.*

The other potential explanation would be that the basic underlying assumption – namely that the local coefficient structure is characteristic for the local shape of the implied surface – is not true in general.

To test this more precisely, we took the 2D test objects depicted in Figure A.3 and compared local coefficient histograms. At certain test points (marked in Figure A.3) we extracted histograms of the expansion coefficients α_i of all kernel centres x_i within a radius of 2σ. The results are shown in Figure A.6. There are some similarities between test point 1 and test point 2 – optimally they should be identical –, however, there is no strong distinction in comparison to the histogram taken at test point 3 which ought to be significantly different.

A second test related to this question was to select the histogram at test point 1 and compared it with histograms taken at all other possible locations (see Figure A.7). To compare the histograms we normalised them and then used the EMD as a distance measure. The EMD is know to be a good metric for histograms [Rubner et al., 2000]. However, we got the same results as described below when using another metric for comparing the histograms, namely the L_1 norm.

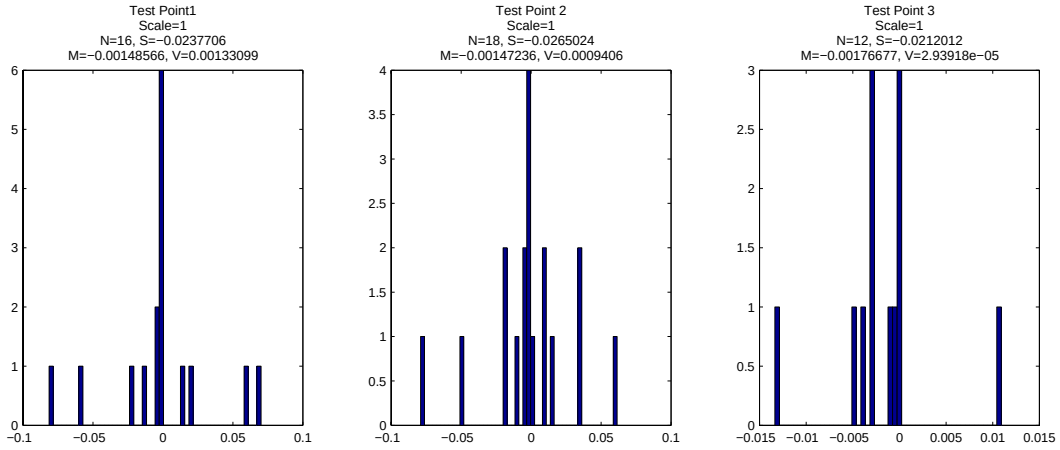


Figure A.6: *Local coefficient histograms extracted at the test points marked in Figure A.3. N is the number of kernel centres within a circle of radius 2σ around the test point, S is the sum of all coefficients, M the mean and V the variance.*

When we compared the histogram of test point 1 to the locations within the same image, we could typically identify the test point 1 from the similarity ratings. When we compared it to the locations in the second shape, we were able to detect some similarity at the position that corresponds to the test point 1 (i.e. test point 2). However, this only worked if we took kernel centres lying inside the shape into account. When looking at outside lying centres, no specific similarity between the corresponding points could be detected.

This renders the results somewhat unclear. Combined with the generally bad matching results that we obtained, our conclusion is that the basic assumption – i.e. that the local coefficient structure is characteristic for the local surface features – does not hold in general. If this is the case, then the proposed morphing based on the EMD does not make sense.

A.4 Another interpretation

We have

$$f(x) = \sum_i \alpha_i k(x, x_i) = \sum_i \alpha_i \int \delta_{x_i}(y) k(x, y) dy.$$

This can also be written as

$$f = \left(\sum_i \alpha_i \delta_{x_i} \right) * k(x, .).$$

Thus, if we consider similarities between the coefficients α_i, we are effectively comparing "inverse kernel transforms" – similar to Fourier transforms. Does it make sense to assume that the kernel transformations of two objects with locally similar surface structure are equal? That remains an open question.

A.5 Conclusions

The basic underlying assumption – i.e. that the local coefficient structure is characteristic for the local surface features – seems to be violated for most examples even though first

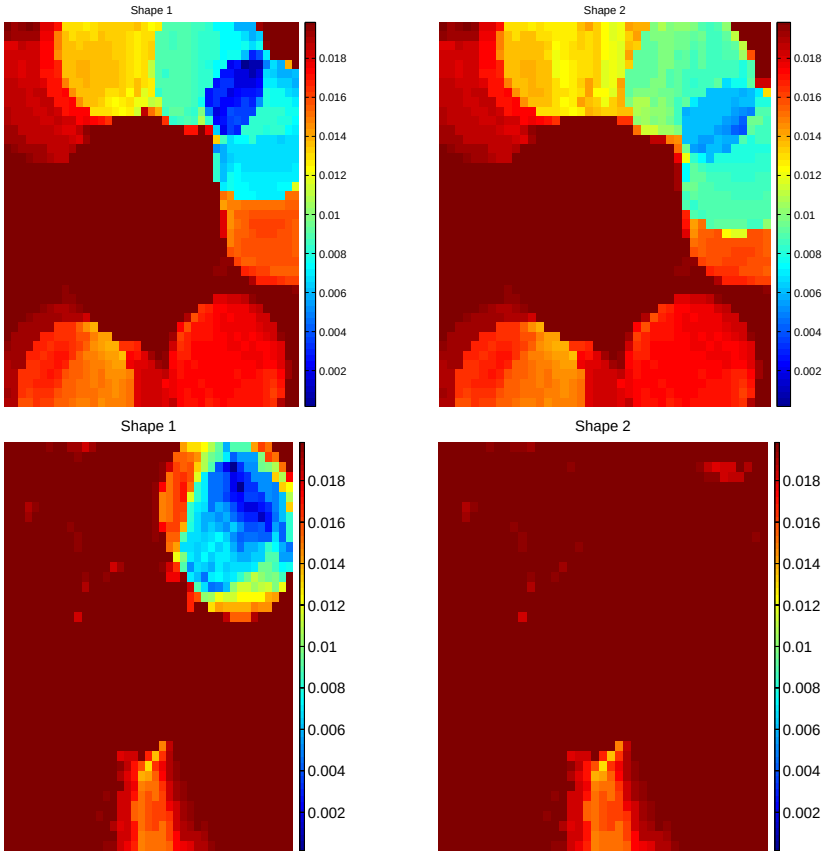


Figure A.7: *We took the shapes in Figure A.3 and compared a local histogram of coefficients taken at test point 1 to histograms taken at all other location in both shapes. The images colour code the similarity rating. (left column) the first shape that test point 1 is taken from, (right) the other shape. (upper row) only kernels lying inside the shapes are considered, (bottom row) only kernels lying outside the shapes are considered*

experiments showed a proper correspondence (see Figure A.1). Therefore, methods based on the EMD do not seem to be promising.

We also experimented with other methods of shifting kernel centres for morphing. We could for example select some landmark points on the surface, match them using some heuristic, and then expand this point information into a smooth space deformation $\tau : \mathcal{X} \to \mathcal{X}$. We could shift the kernels using the transformation $\tau_\lambda(x) \equiv \lambda x + (1-\lambda)\tau(x),\ 0 \leq \lambda \leq 1$ and define intermediate surfaces to be the zero sets of the following functions

$$f_\lambda(x) \quad \equiv \quad (1-\lambda) f_1(\tau_\lambda(x)) + f_2(\tau_\lambda^{-1}(x)),\ 0 \leq \lambda \leq 1$$

Some test experiments with that approach did not yield motivating results. Moving the kernel centres creates a lot of effects that are hard to control. For example it changes the density of kernels with unknown consequences for the implied surface.

Thus, it seems more promising to restrict oneself to the more classical approaches such as the one described in Chapter 6. The correct correspondence information remains the key feature for all morphing applications.

Bibliography

Alexa, M. (2002). Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–196.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. (2003). Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15.

Amenta, N. and Kil, Y. J. (2004). Defining point-set surfaces. *ACM Trans. Graph.*, 23(3):264–270.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404.

Bakir, G. H., Gretton, A., Franz, M., and Schölkopf, B. (2004). Sequential multivariate regression via stiefel manifold constraints. In *Proceedings of the DAGM-Symposium 2004*.

Bartlett, P. L. and Schölkopf, B. (2001). Some kernels for structured data. Technical report, Biowulf Technologies.

Beatson, R. and Greengard, L. (1997). A short course on fast multipole methods. In Ainsworth, M., Levesley, J., Light, W., and Marletta, M., editors, *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37. Oxford University Press.

Beatson, R. K. and Newsam, G. N. (1998). Fast evaluation of radial basis functions: Moment-based methods. *SIAM J. Sci. Comput.*, 19(5):1428–1449.

Blanz, V. and Vetter, T. (1999). A morphable model for the synthesis of 3d faces. In *SIGGRAPH'99 Conference Proceedings*, pages 187–194, Los Angeles. ACM Press.

Blanz, V. and Vetter, T. (2003). Face recognition based on fitting a 3D morphable model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(9):1063–1074.

Bruhn, A., Weickert, J., and Schnörr, C. (2002). Combining the advantages of local and global optic flow methods. In *DAGM-Symposium*, pages 454–462.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3D objects with radial basis functions. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 67–76. ACM Press / ACM SIGGRAPH.

Carr, J. C., Beatson, R. K., McCallum, B. C., Fright, W. R., McLennan, T. J., and Mitchell, T. J. (2003). Smooth surface reconstruction from noisy range data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–ff, New York, NY, USA. ACM Press.

Chang, C.-C. and Lin, C.-J. (2005). Libsvm: a library for support vector machines (version 2.31).

Cohen-Or, D., Solomovic, A., and Levin, D. (1998). Three-dimensional distance field metamorphosis. *ACM Trans. Graph.*, 17(2):116–141.

Cooke, T., Steinke, F., Wallraven, C., and Bülthoff, H. (2005). A similarity-based approach to perceptual feature validation. In *Proceedings of the 2nd Symposium on Applied Perception in Graphics and Visualization*, pages 59 – 66, New York, NY, USA. ACM Press.

Davies, R. H., Cootes, T. F., and Taylor, C. J. (2001). A minimum description length approach to statistical shape modelling. In *Information Processing in Medical Imaging*, pages 50–63.

Gain, J. E. and Dodgson, N. A. (2001). Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):289–298.

Girosi, F., Jones, M., and Poggio, T. (1993). Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. A.I. Memo No. 1430, MIT.

Grauman, K. and Darrell, T. (2004). Fast contour matching using approximate earth mover's distance. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, 2004 (CVPR 2004)*, pages 220–227.

Ham, J. H., Lee, D. D., and Saul, L. K. (2005). Semisupervised alignment of manifolds. In Ghahramani, Z. and Cowell, R., editors, *(to appear in) Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*.

Haralick, R. and Shapiro, L. (1992). *Computer and robot vision*, volume 2. Addison-Wesley, Reading, MA.

Horn, B. K. P. and Schunck, B. G. (1981). Determining optical flow. *Artif. Intell.*, 17(1-3):185–203.

Huang, X., Paragios, N., and Metaxas, D. N. (2003). Establishing local correspondences towards compact representations of anatomical structures. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003, 6th International Conference, Montréal, Canada, November 15-18, 2003, Proceedings, Part II*, pages 926–934.

Karkanis, T. and Stewart, A. J. (2001). Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications*, 22(2):60–69.

Klein, J. and Zachmann, G. (2004a). Point cloud collision detection. In *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2004)*, pages 567–576.

Klein, J. and Zachmann, G. (2004b). Proximity graphs for defining surfaces over point clouds. In *Eurographics Symposium on Point-Based Grahics (SPBG'04)*, pages 131–138.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.

Lucas, B. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679.

Luo, Z.-Q. and Tseng, P. (1992). On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35.

Merkwirth, C., Parlitz, U., and Lauterborn, W. (2000). Fast nearest-neighbor searching for nonlinear signal processing. *Physical Review E*, 62:2089–97.

Micchelli, C. and Pontil, M. (2005). On learning vector-valued functions. *Neural Computation*, 17:177–204.

Morse, B. S., Yoo, T. S., Chen, D. T., Rheingans, P., and Subramanian, K. R. (2001). Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 89, Washington, DC, USA. IEEE Computer Society.

Museth, K., Breen, D. E., Whitaker, R. T., and Barr, A. H. (2002). Level set surface editing operators. *ACM Trans. Graph. SIGGRAPH*, 21(3):330–338.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. (2003a). Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470.

Ohtake, Y., Belyaev, A., and Seidel, H.-P. (2003b). A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 292. IEEE Computer Society.

Ohtake, Y., Belyaev, A., and Seidel, H.-P. (2004). Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3):609–612.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research.

Roach, G. F. (1982). *Green's Functions - Second Edition*. Cambridge University Press.

Rubner, Y., Tomasi, C., and Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121.

Rueckert, D. and Frangi, A. F. (2003). Automatic construction of 3-d statistical deformation models of the brain using nonrigid registration. *IEEE Trans. on Medical Imaging*, 22(8):1014–1025.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. In *Proc. Conf. on 3D Digital Imaging and Modeling*, pages 145–152.

Sand, P. and Teller, S. (2004). Video matching. *ACM Trans. Graph.*, 23(3):592–599.

Saul, L. K. and Jordan, M. I. (1996). A variational principle for model-based morphing. In *NIPS*, pages 267–273.

Schaback, R. (1995). Creating surfaces from scattered data using radial basis functions. In Daehlen, M., Lyche, T., and Schumaker, L., editors, *Mathematical Methods for Curves and Surfaces*, pages 477–496. Vanderbilt University Press, Nashville.

Schaback, R. (2000). A unified theory of radial basis functions. *J. of Comp. and Appl. Math.*, 121:165–177.

Schölkopf, B., Giesen, J., and Spalinger, S. (2004). Kernel methods for implicit surface modeling. Technical Report 125, Max Planck Institute for Biological Cybernetics.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels*. MIT Press, Cambridge, MA.

Schölkopf, B., Steinke, F., and Blanz, V. (2005). Object correspondence as a machine learning problem. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 05)*.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. Technical Report IDSIA-09-99, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

Schreiner, J., Asirvatham, A., Praun, E., and Hoppe, H. (2004). Inter-surface mapping. *ACM Trans. Graph. SIGGRAPH*, 23(3):870–877.

Shelton, C. (2000). Morphable surface models. *Int. J. of Computer Vision*, 38(1):75–91.

Shen, C., O'Brien, J. F., and Shewchuk, J. R. (2004). Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press.

Smola, A., Friess, T., and Schölkopf, B. (1999). Semiparametric support vector and linear programming machines. *Advances in Neural Information Processing Systems*, 11:585 – 591.

Steinke, F. and Schölkopf, B. (2006). Machine learning methods for estimating operator equations. In *Proc. of SYSID06*. submitted.

Steinke, F., Schölkopf, B., and Blanz, V. (2005). Support vector machines for 3D shape processing. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2005)*, 24:285–294.

STL (1994). *Standard Template Library Programmer's Guide*. http://www.sgi.com/tech/stl/.

Thirion, J.-P. and Gourdon, A. (1995). Computing the differential characteristics of isointensity surfaces. *Journal of Computer Vision and Image Understanding*, 61(2):190–202.

Turk, G., Dinh, H. Q., O'Brien, J. F., and Yngve, G. (2001). Implicit surfaces that interpolate. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 62, Washington, DC, USA. IEEE Computer Society.

Turk, G. and O'Brien, J. F. (1999). Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 335–342, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer Verlag, New York.

Veltkamp, R. C. and Hagedoorn, M. (1999). State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University.

Wahba, G. (1977). Practical approximate solutions to linear operator equations when the data are noisy. *SIAM Journal on Numerical Analysis*, 14:651–667.

Wahba, G. (1990). *Spline models for observational data*. SIAM.

Walder, C., Chapelle, O., and Schölkopf, B. (2005). Implicit surface modelling as an eigenvalue problem. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 05)*.

Walder, C. J. and Lovell, B. C. (2003). Kernel based algebraic curve fitting. In *Proc. ICAPR'2003*.

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., and Vapnik, V. (2003). Kernel dependency estimation. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems*, volume 15, Cambridge, MA, USA. MIT Press.

Yang, C., Duraiswami, R., Gumerov, N. A., and Davis, L. (2003). Improved fast gauss transform and efficient kernel density estimation. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 464. IEEE Computer Society.

Zorin, D. and Schröder, P. (2000). Subdivision for modeling and animation. course notes for SIGGRAPH 2000.