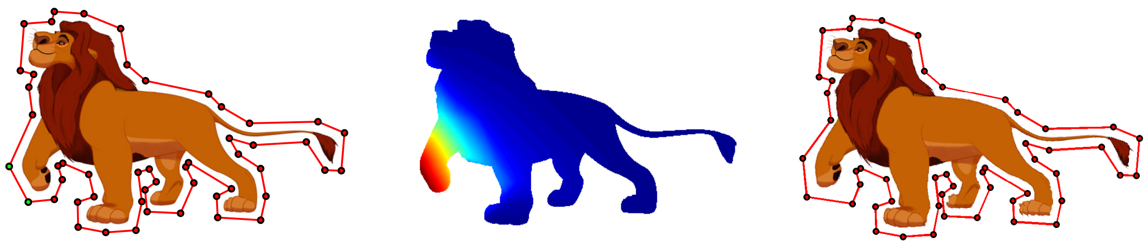
Kernel Coordinates

Kernel Coordinates



Christos Sakaridis

Master Thesis
April 2016

Prof. Dr. Markus Gross
Dr. Cengiz Öztireli

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

Abstract

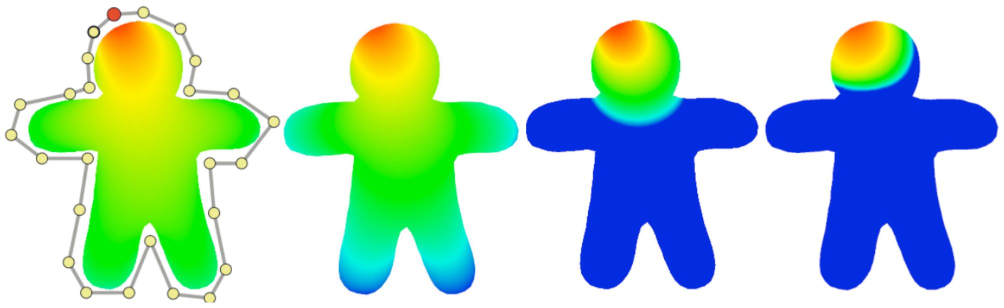
A pivotal task in shape deformation is calculation of weight functions, also named coordinates, to blend transformations defined at control handles, binding the shape to the handles. This has previously been attacked by meshing the shape and optimizing a global energy at bind time. In this thesis, we develop a novel, mesh-less method to compute coordinates by solving individual, decoupled problems at each point and using radial basis interpolation to interpolate coordinates smoothly across the domain. This shift reduces the cost of the binding step significantly and hence allows user-driven interactive modification of weights during pose time, which is infeasible with previous methods. Our method can also handle shapes comprising parts with different dimensionality, in contrast to mesh-based approaches. Preservation of salient details of the shape that are defined during interaction is possible in our framework by coupling weight optimization locally.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Master Thesis

Kernel Coordinates



Introduction

Geometry and image deformation is one of the fundamental operations in processing of visual content. One proven way of deforming a shape is via a reduced space with intuitive controls such as handles. In order to allow for intuitive interaction and robust deformations, the interpolating functions, called coordinates, need to satisfy certain properties. So far, these properties are enforced by formulating a constrained optimization problem, which is solved by using finite element methods. This requires costly meshing, leading to reduced smoothness and limiting the methods to mesh-able domains, and an expensive solution of a global system. Furthermore, it does not allow for user control by steering the weights. In this project, we will develop a mesh-less method for defining coordinates, which will solve all mentioned problems.

Task Description

- Reading and understanding the literature on deformations and kernels
- Implementing an application for 2D image plane deformation
- Experimenting with different kernels, sampling methods, and deformation controls
- Extending the implementation to 3D for surface and general non-manifold deformations

Skills

- Good programming skills
- Curiosity and creativity

Remarks

A written report and an oral presentation conclude the thesis. The thesis is will be overseen by Prof. Markus Gross and supervised by Dr. Cengiz Öztireli.

Contact

For further information, please contact the thesis coordinator (cgl-thesis@inf.ethz.ch)

Acknowledgments

Throughout the completion of this thesis, I have not stood alone. It has been a great pleasure for me to be supervised by Cengiz Öztireli, a brilliant scientist with innovative ideas, who always greeted me with a warm smile. I enjoyed working with him and having fruitful discussions that boosted the progress of the thesis, and I found great inspiration in his words: "Don't be afraid to go deep and don't be lazy to go broad".

I have also been fortunate enough to work in an ideal environment like the Computer Graphics Laboratory and have my thesis overseen by Markus Gross, an outstanding personality in the field of Computer Graphics.

I would like to cordially thank the General Michael Arnaoutis Foundation for its invaluable support that helped me complete my Master studies at ETH.

Finally, I want to thank my family and Ioanna, for the completion of this thesis would have been impossible without their encouragement and patience.

Contents

List of Figures

List of Tables

1 Introduction

Shape deformation is a central topic in shape and geometry processing, with applications such as character animation, motion synthesis and image warping. A lot of deformation metaphors have been used in order to model the way an object changes its shape, including free-form transforms, curve-based or RBF-based methods, and approaches using cages or skeletons to provide a simplified version of the object's structure. Most of these approaches involve a set of control structures bound on the object, each of which is associated with a specific affine space transformation. These transformations are then combined, or *blended*, at each point of the object to produce the final deformation. Irrespective of the particular blending scheme that is used (for instance linear blend skinning), the blending *weights* of the control handles' transformations are subject to certain requirements that guarantee intuitive deformation results.

More specifically, the weights should ideally be smooth, shape-aware and local, meaning that a control handle influences only nearby parts of the shape significantly. Additionally, the weights should respect any structure that is present in the object. Recent work on designing blending weights has gradually moved towards covering an ever growing subset of these abstract properties.

Further requirements for the weights (also named coordinates) are expressed in a more formal, mathematical setting for the linear blend skinning case and they also aid intuitiveness of the deformation. First, weights should partition the unity, so that when the same transformation is applied to all control handles, the whole object is also transformed in this way. Second, when no transformation is applied to the control handles, the rest-pose of the shape should be reproduced (reproduction property). Third, the weight of a control handle should be 1 at this control handle and 0 at all others, which implies that each control handle only influences itself (Lagrange property). Fourth, in cage-based deformation, where control handles constitute simple points and are only subject to pure translations, weights should be linear on all edges and faces of the cage, to ensure that the latter always preserve their linearity. Finally, several authors, such as Jacobson et al. [1], argue for non-negative weights, while others, such as Wang et al. [2], accept negative weights of small magnitude.

Most recent works on designing weights [1, 3, 2] use a mesh to represent the object. This type of approach restricts weight smoothness to the granularity of the mesh. Moreover, shapes comprising structures of different dimensionality, e.g. volumes combined with surfaces in 3D, cannot be handled in the mesh-based setting. Weights are usually computed via global optimization of some rigidity energy over the mesh. The high computational cost of this step leads almost invariably to a clear temporal separation between the binding or rigging stage (weight calculation) and the deformation stage (blending of control handle transformations using the calculated weights), which hinders interactivity. To the best of our knowledge, the only exception to this paradigm is the method in [2], where the user can interactively add or remove control handles but does not have direct control on the corresponding weights.

Kernel coordinates is a novel method for computing weights to blend control handle transformations, based on reproducing kernel bases. It involves meshless sampling of the object's domain and optimization of weights defined at individual point samples, which are subject to user control. Weights at sampling locations are constrained during optimization to satisfy the aforementioned properties. Eventually, dual kernel functions are used to smoothly interpolate these weights at any point of the object and get coordinate functions that automatically inherit all prescribed properties.

More formally, suppose that a real function f is sampled at n locations $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The value of the kernel for a pair of these sampling locations is denoted by $k(\mathbf{x}_i, \mathbf{x}_j)$ and, for an arbitrary point $\mathbf{x}$, we write the kernel function corresponding to the i-th sampling location as

$$k_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i) \tag{1.1}$$

and stack all of them in vector

$$\mathbf{k}(\mathbf{x}) = \begin{bmatrix} k_1(\mathbf{x}) & \cdots & k_n(\mathbf{x}) \end{bmatrix}^T . \tag{1.2}$$

Using this notation, the $n \times n$ kernel matrix is defined as

$$\mathbf{K} = \begin{bmatrix} \mathbf{k}(\mathbf{x}_1)^T \\ \vdots \\ \mathbf{k}(\mathbf{x}_n)^T \end{bmatrix} \tag{1.3}$$

and its (i, j)-th element is denoted by $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

The above defined kernel functions are used as a basis to interpolate f at sampling locations. Denote by

$$\mathbf{f} = \begin{bmatrix} f(\mathbf{x}_1) & \cdots & f(\mathbf{x}_n) \end{bmatrix}^T$$

the vector of values of f at sampling locations. Interpolation coefficients $\mathbf{a}$ are by definition the solution of the linear system

$$\mathbf{K}\mathbf{a} = \mathbf{f} \Rightarrow \mathbf{a} = \mathbf{K}^{-1}\mathbf{f}. \tag{1.4}$$

Thus, the interpolant $\bar{f}$ is expressed as

$$\bar{f}(\mathbf{x}) = \mathbf{a}^T\mathbf{k}(\mathbf{x}) = \mathbf{f}^T\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}), \tag{1.5}$$

where vector

$$\mathbf{a}(\mathbf{x}) = \begin{bmatrix} a_1(\mathbf{x}) & \cdots & a_n(\mathbf{x}) \end{bmatrix}^T = \mathbf{K}^{-1}\mathbf{k}(\mathbf{x}) \tag{1.6}$$

contains the dual kernel functions.

In the coordinates setting, there are m control handles, each one associated with a coordinate function. Dropping the bar notation for the sake of simplicity, the interpolant of the i-th coordinate function w_i at the above set of sampling locations is similarly given by

$$w_i(\mathbf{x}) = \mathbf{w}_i{}^T\mathbf{a}(\mathbf{x}). \tag{1.7}$$

We stack the samples of all coordinate functions in matrix

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1{}^T \\ \vdots \\ \mathbf{w}_m{}^T \end{bmatrix}, \tag{1.8}$$

the elements of which are subject to optimization that is based on user input and satisfaction of certain properties. The vector of *kernel coordinates* at point $\mathbf{x}$ can then be expressed as

$$\mathbf{w}(\mathbf{x}) = \mathbf{W}\mathbf{a}(\mathbf{x}). \tag{1.9}$$

The main idea behind our approach is the direct transfer of all desired properties that are possessed by weights of matrix $\mathbf{W}$ to kernel coordinates $\mathbf{w}(\mathbf{x})$. We defer the analysis for this result to Chapter 3, pointing out that the key related property is

$$\sum_{j=1}^{n} k_j(\mathbf{x}) = 1. \tag{1.10}$$

Sampling the shape with points instead of using a proper mesh enables handling of non-closed shapes with non-manifold junctions and parts of different dimensionality. Initialization of weights at sampling locations can be done in a way that respects the shape, e.g. via heat diffusion from control handles using geodesic distances, so that shape-awareness and locality are guaranteed.

Optimization is then performed *individually* for each point sample based on the initial values of the corresponding weights, which can drastically reduce running time, as parallelization is straightforward. This acceleration paves the way for a new level of interaction: the user can inspect deformations produced with current weights and provide feedback in the form of a local modification of weights, e.g. using a smooth brush, to reach a more intuitive deformation. The system then re-optimizes only for points with modified weights based on user input, enabling a fast response. This way, a design loop at a low level (direct manipulation of the weights) is introduced, which effectively integrates the binding with the deformation stage.

Another important direction we aim at is user-driven structure-aware deformation, which corresponds to design of weights at a higher level. This can be performed in two modes: first, the user initially defines structural features of the original object that should be respected through the entire interaction. These features can be symmetries, which induce symmetric weights, or salient lines (or sets of lines) of the shape that should be preserved as such (under translations) or deform in the same way. Second, the user paints regions that should deform as rigidly as possible with a soft rigidity brush, which is introduced in [1]. Rigidity terms in the objective induce a coupling between points, albeit only at a local level in the vicinity of the painted region, which relieves us of the need for global optimization and allows re-optimizing at interactive rates, similarly to direct weight manipulation.

2 Related Work

Reproducing kernels have been proposed in [4] as an alternative to standard pointwise sampling in image synthesis. In this work, Lessig et al. use dual kernel functions as continuous basis functions for representing signals, introducing a sampling approach which we transfer to the coordinate functions setting for blending transformations.

Shape deformation has received great interest from researchers during recent years. A popular approach is to compute generalized barycentric coordinates across the interior of a control structure called a *cage*, which is a polyhedron that encloses the shape and captures the basic structure of it. One of the early cage-based methods is mean value coordinates [5, 6], which accept a closed-form solution based on geometry of the cage but assume large negative values for non-convex shapes and are not shape-aware. Harmonic coordinates [7] address the negativity issue by employing a volumetric solution in the interior of the cage and constitute shape-aware, local weights with no interior extrema. Further advance towards locality has been made by Zhang et al. in their local barycentric coordinates work [3], where the shape is discretized using a triangular mesh (a common choice in most recent works) and optimization on this mesh involves an objective that induces sparsity to the weights. Bounded biharmonic weights [1] are differentiated from previous methods in that they are not essentially generalized barycentric coordinates, but they combine control handles of different types, such as points, skeletons and cages, which can be placed directly on the object. While optimization in the bounded biharmonic weights framework can incorporate energy terms that help to preserve certain parts of the shape more faithfully during deformation, it does not impose linear reproduction like other methods do, which brings about unintuitive deformations under linear transformation functions for cage handles.

A different paradigm for shape deformation originates in the as rigid as possible (ARAP) method [8], where the shape is represented as a triangular mesh, control handles coincide with certain vertices of the mesh and a two-step global optimization is run at pose time to limit triangle distortion. Similar elastic energies are utilized in [9] to produce skinning deformations in real time by optimizing a non-linear reduced model in the subspace of deformations. The linear subspace design proposed in [2] elaborates the use of subspaces in deformation and unifies linear blend skinning and generalized barycentric coordinates, minimizing a mesh-based quadratic ARAP energy that guarantees linear reproduction and enables interactive addition and removal of control handles at pose time, which is not feasible for existing cage-based methods.

Radial basis interpolation has been previously used in [10] to devise a 3D RBF-based deformation scheme which uses a set of spheres that model rigidity of the object's interior along with a set of points on its surface to guide interpolation. The authors choose a compact inverse multiquadric function as the basis function in contrast to our choice of Gaussian RBF, and use *interior* distance with respect to the shape as the argument of the RBF to gain shape-awareness. A meshless technique is presented in [11] to equip 2D deformation spaces of smooth basis functions with theoretical guarantees for injectivity and bounded distortion, by enforcing relevant constraints on a set of carefully selected points.

Structure-based shape editing relies on identifying meaningful parts of objects and establishing parameters for each individual part and relations between distinct parts, encapsulating structure as additional constraints on the deformation. State of the art methods for structure-aware deformation are summarized in [12]. A representative of these methods is iWires [13], which preserves geometric features of salient crease lines ("wires") of man-made objects and relations between these "wires", while trying to stay close to user-specified constraints. The deformation approach for images proposed by Schaefer et al. [14] also incorporates structural features of the image in the form of lines to control deformation at crucial parts of the image.

3 Theoretical Analysis of Kernel Coordinates

3.1 Derivation of Properties

We extend the analysis of Chapter 1 towards justification of our claim that kernel coordinates automatically inherit the various properties required for blending weights. We use the same notation as in Chapter 1 in the following.

3.1.1 Dual Kernel Functions

First, we derive partition of unity, Lagrange property and reproduction for dual kernel functions which are used to perform interpolation in the kernel coordinates framework.

Under certain assumptions for sampling and the parameters of the kernel (see Section 3.2 for a detailed study of the regular sampling and Gaussian kernel case), it holds for the values of kernel functions at an arbitrary point $\mathbf{x}$ that

$$\sum_{j=1}^{n} k_j(\mathbf{x}) = c,$$

where c is a positive constant. Simply scaling the kernel by $1/c$ yields

$$\sum_{j=1}^{n} k_j(\mathbf{x}) = 1 \Rightarrow \mathbf{k}(\mathbf{x})^T \mathbf{1} = 1. \tag{3.1}$$

Since (3.1) is true at all sampling points, we can collect the n corresponding individual equations into

$$\mathbf{K}\mathbf{1} = \mathbf{1} \Rightarrow \mathbf{K}^{-1}\mathbf{1} = \mathbf{1}. \tag{3.2}$$

Based on (3.1), (3.2) and symmetry of $\mathbf{K}$, we show that the dual kernel functions $a_j(\mathbf{x})$, $j = 1, \ldots, n$ satisfy the *partition of unity* property:

$$\sum_{j=1}^{n} a_j(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T\mathbf{1} = \mathbf{k}(\mathbf{x})^T\mathbf{K}^{-1}\mathbf{1} = \mathbf{k}(\mathbf{x})^T\mathbf{1} = 1. \tag{3.3}$$

Furthermore, the dual kernel functions obey the *Lagrange* property by definition. In particular, we denote the l-th column of $\mathbf{K}^{-1}$ by $\mathbf{k}_l^{-1}$ and, due to symmetry of $\mathbf{K}$, we obtain

$$\mathbf{a}(\mathbf{x}_j) = \mathbf{K}^{-1}\mathbf{k}(\mathbf{x}_j) = \sum_{l=1}^{n} k_{jl}\mathbf{k}_l^{-1} \Rightarrow a_i(\mathbf{x}_j) = \sum_{l=1}^{n} k_{jl}k_{il}^{-1} = \left(\mathbf{K}^{-1}\mathbf{K}\right)_{ij} = \delta_{ij}. \tag{3.4}$$

Last, we turn to the *reproduction* property and confine our derivation to the case of a Gaussian kernel. Let $G_\sigma(\mathbf{x})$ be the isotropic Gaussian with standard deviation σ that is centered at the origin, given by

$$G_\sigma(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right). \tag{3.5}$$

The scale of the Gaussian simply coincides with σ in this setting. This Gaussian will serve as the kernel function in the spatial domain. The Gaussian kernel is then expressed as

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = G_\sigma(\mathbf{x}_i - \mathbf{x}_j). \tag{3.6}$$

Point $\mathbf{x}$ is *reproduced* by the dual kernel functions $a_j(\mathbf{x})$ if and only if

$$\sum_{j=1}^{n} a_j(\mathbf{x})\mathbf{x}_j = \mathbf{x}. \tag{3.7}$$

The first step towards proving (3.7) is to differentiate (3.1) and exploit the form of the Gaussian gradient:

$$\begin{aligned}\nabla\left(\sum_{j=1}^{n} k_j(\mathbf{x})\right) &= -\frac{1}{\sigma^2}\sum_{j=1}^{n}(\mathbf{x}-\mathbf{x}_j)k_j(\mathbf{x}) = \mathbf{0} \Rightarrow \\ \sum_{j=1}^{n}\mathbf{x}k_j(\mathbf{x}) &= \sum_{j=1}^{n}\mathbf{x}_j k_j(\mathbf{x}) \Rightarrow \\ \mathbf{x} &= \sum_{j=1}^{n}\mathbf{x}_j k_j(\mathbf{x}).\end{aligned} \tag{3.8}$$

Denoting

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \ldots & \mathbf{x}_n \end{bmatrix}, \tag{3.9}$$

(3.8) can be rewritten in matrix form as

$$\mathbf{x} = \mathbf{X}\mathbf{k}(\mathbf{x}). \tag{3.10}$$

In addition, collecting all n equations of the form (3.10) at sampling locations, we arrive at

$$\mathbf{X} = \mathbf{X}\mathbf{K}^T \Rightarrow \mathbf{X}\mathbf{K}^{-1} = \mathbf{X}. \tag{3.11}$$

We use (3.10) and (3.11) to derive (3.7) as follows:

$$\sum_{j=1}^{n} a_j(\mathbf{x})\mathbf{x}_j = \mathbf{X}\mathbf{a}(\mathbf{x}) = \mathbf{X}\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}) = \mathbf{X}\mathbf{k}(\mathbf{x}) = \mathbf{x}. \tag{3.12}$$

3.1.2 Kernel Coordinates

Based on the properties which have been derived for dual kernel functions, we show that kernel coordinates defined through (1.9) also satisfy the same properties, if we enforce appropriate constraints on weights at sampling locations which constitute matrix $\mathbf{W}$.

More specifically, we require that weights at every sampling point $\mathbf{x}_j$ partition the unity, which is expressed as

$$\sum_{i=1}^{m} w_{ij} = 1,\ \forall j \in \{1, \ldots, n\} \tag{3.13}$$

or equivalently

$$\mathbf{1}^T\mathbf{W} = \mathbf{1}^T. \tag{3.14}$$

Additionally, we demand that weights at every sampling point reproduce the latter as a combination of the respective control points. If we stack control points as columns of a single matrix $\mathbf{C}$, i.e.

$$\mathbf{C} = \left[\begin{array}{ccc} \mathbf{c}_1 & \ldots & \mathbf{c}_m \end{array}\right], \tag{3.15}$$

then the above requirement for point $\mathbf{x}_j$ reads:

$$\mathbf{C}\mathbf{w}_j = \mathbf{x}_j. \tag{3.16}$$

Collecting for all sample points yields

$$\mathbf{C}\mathbf{W} = \mathbf{X}. \tag{3.17}$$

We also require that weights on sample points which coincide with control points obey the Lagrange property. Sampling is performed in such a way that every control point is also a member of the set of sampling locations. However, since in general there are more sampling points than control handles, i.e. $n > m$, we need to establish a convenient correspondence between indices of the two sets. Without loss of generality, we set $\mathbf{x}_i = \mathbf{c}_i,\ i = 1, \ldots, m$, so that control points correspond to sample points with the smallest m indices. Then, Lagrange property is expressed as

$$w_{ij} = \delta_{ij},\ \forall i, j \in \{1, \ldots, m\}. \tag{3.18}$$

The above constraints on weights are enforced in an optimization step which is presented in Chapter 4. These constraints suffice to show that the prescribed properties are satisfied everywhere in space by kernel coordinates.

We begin with *partition of unity*, which is shown using (3.3) and (3.14):

$$\sum_{i=1}^{m} w_i(\mathbf{x}) = \mathbf{1}^T\mathbf{w}(\mathbf{x}) = \mathbf{1}^T\mathbf{W}\mathbf{a}(\mathbf{x}) = \mathbf{1}^T\mathbf{a}(\mathbf{x}) = 1. \tag{3.19}$$

It is also straightforward to prove *reproduction* for kernel coordinates by combining (3.12) with (3.17):

$$\sum_{i=1}^{m} w_i(\mathbf{x})\mathbf{c}_i = \mathbf{C}\mathbf{w}(\mathbf{x}) = \mathbf{C}\mathbf{W}\mathbf{a}(\mathbf{x}) = \mathbf{X}\mathbf{a}(\mathbf{x}) = \mathbf{x}. \tag{3.20}$$

Finally, we show that Lagrange property holds for kernel coordinates due to (3.4) and (3.18):

$$\mathbf{w}(\mathbf{c}_i) = \mathbf{w}(\mathbf{x}_i) = \mathbf{W}\mathbf{a}(\mathbf{x}_i) = \mathbf{W}\mathbf{e}_i = \mathbf{w}_i = \mathbf{e}_i, \tag{3.21}$$

where $\mathbf{e}_i$ is the i-th standard basis vector of the respective Euclidean space.

3.2 Anti-aliasing Analysis for Regular Sampling in 2D

To determine anti-aliasing conditions when sampling the Gaussian kernel, we utilize its Fourier transform (we use *angular frequency* as the independent variable in the Fourier domain). In 2D, this is given by

$$\mathrm{FT}\left\{G_\sigma(x,y)\right\} = 2\pi\sigma^2 \exp\left(-\frac{{\omega_1}^2 + {\omega_2}^2}{2\left(\frac{1}{\sigma}\right)^2}\right). \tag{3.22}$$

In the two-dimensional setting, the set of sampling locations forms a regular 2D grid in the simplest case. Let T be the common sampling period in both dimensions, so that consecutive samples along a horizontal or vertical line of the grid are T units of length away from each other. This corresponds to sampling the kernel function with a 2D periodic impulse train which can be expressed through a double infinite sum of Dirac deltas:

$$s(x,y) = \sum_{n_1=-\infty}^{+\infty}\sum_{n_2=-\infty}^{+\infty} \delta(x - n_1T)\delta(y - n_2T). \tag{3.23}$$

The 2D Fourier transform of the impulse train of (3.23) is then

$$\mathrm{FT}\{s(x,y)\} = S(\omega_1,\omega_2) = \frac{4\pi^2}{T^2}\sum_{k_1=-\infty}^{+\infty}\sum_{k_2=-\infty}^{+\infty} \delta\left(\omega_1 - k_1\frac{2\pi}{T}\right)\delta\left(\omega_2 - k_2\frac{2\pi}{T}\right). \tag{3.24}$$

We aim at deriving the optimal scale for the Gaussian kernel under the above sampling configuration. This involves a trade-off between:

- avoiding aliasing effects, and
- ensuring good conditioning of the resulting kernel matrix.

To avoid aliasing, it is essential to pick a large enough scale that leads to an adequately concentrated Gaussian at the origin of the frequency domain, with negligible overlap with every impulse except the one at the origin. On the other hand, the scale needs to be small enough, in order for the spatial Gaussian not to flatten excessively and thus render the kernel matrix ill-conditioned. A quantification of this argument follows.

When we sample the Gaussian kernel using (3.23), the sampled signal $l(\mathbf{x}) = \sum_i k(\mathbf{x}, \mathbf{x}_i)$ is given by the convolution of the kernel with the impulse train and its Fourier transform is directly related to the product of their respective Fourier transforms:

$$l(x,y) = G_\sigma(x,y) * s(x,y) \xrightarrow{\text{FT}}$$
$$L(\omega_1,\omega_2) = \frac{8\pi^3\sigma^2}{T^2}\exp\left(-\frac{{\omega_1}^2+{\omega_2}^2}{2\left(\frac{1}{\sigma}\right)^2}\right)\sum_{k_1=-\infty}^{+\infty}\sum_{k_2=-\infty}^{+\infty}\delta\left(\omega_1 - k_1\frac{2\pi}{T}\right)\delta\left(\omega_2 - k_2\frac{2\pi}{T}\right). \tag{3.25}$$

The Gaussian in the frequency domain has standard deviation $\sigma_F = 1/\sigma$. As previously mentioned, we want this Gaussian not to overlap with any impulse except the one at the origin. This is effectively true if the Gaussian has decayed enough at the impulses that are closest to the origin, i.e. $2\pi/T$ units away from its mode. Its value at these locations is

$$2\pi\sigma^2\exp\left(-\frac{2\pi^2\sigma^2}{T^2}\right). \tag{3.26}$$

The ratio of the above value over the zero-frequency magnitude of the Gaussian is

$$\exp\left(-\frac{2\pi^2\sigma^2}{T^2}\right) \tag{3.27}$$

and depends on the ratio σ/T. We pick σ such that (3.27) is as small as possible. Putting the closest impulses $3\sigma_F$ away from the Gaussian's mode means that $\sigma = 3T/2\pi$ and results in (3.27) being approximately $1.1 \cdot 10^{-2}$, which is small but not completely negligible. Instead, we set $2\pi/T$ equal to $6\sigma_F$ to obtain:

$$\hat{\sigma} = \frac{3T}{\pi}. \tag{3.28}$$

For the last choice, (3.27) becomes approximately $1.5 \cdot 10^{-8}$, which is negligible. Consequently, we use (3.28) to choose the scale of the kernel based on the sampling period.

Substituting in (3.25), we obtain:

$$L(\omega_1,\omega_2) = 72\pi\exp\left(-\frac{{\omega_1}^2+{\omega_2}^2}{2\left(\frac{1}{\hat{\sigma}}\right)^2}\right)\sum_{k_1=-\infty}^{+\infty}\sum_{k_2=-\infty}^{+\infty}\delta\left(\omega_1 - k_1\frac{2\pi}{T}\right)\delta\left(\omega_2 - k_2\frac{2\pi}{T}\right). \tag{3.29}$$

Demanding that $l(x,y) = c$ and equating the initial value of the resulting Fourier transform to the one derived from (3.29), we arrive at

$$l(x,y) = \frac{18}{\pi}. \tag{3.30}$$

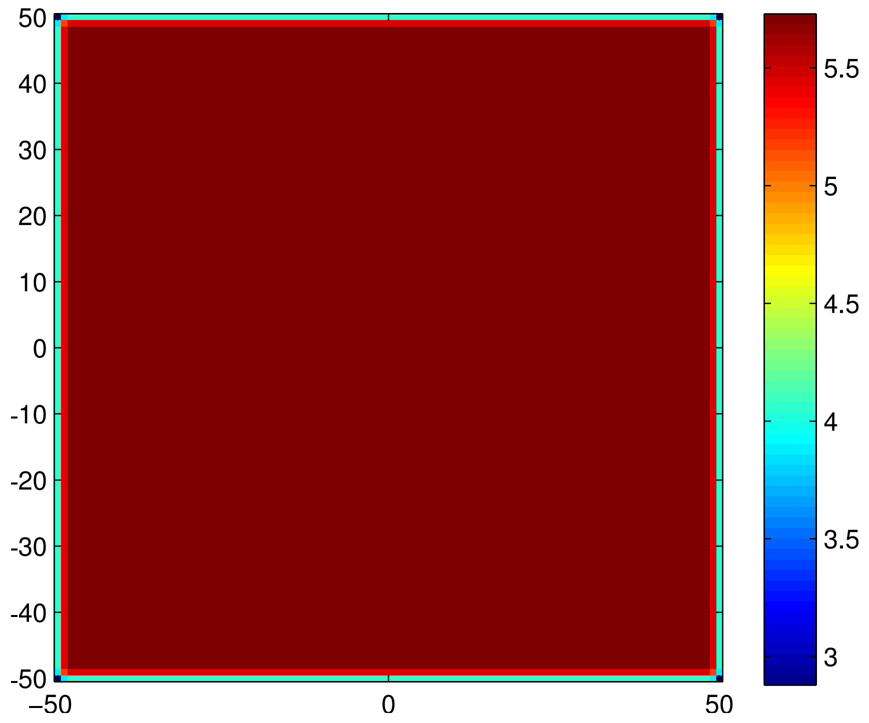


Figure 3.1: Sums of rows of kernel matrix $\mathbf{K}$ for points forming a 2D grid with spacing $T = 1$.

3.3 Experimental Verification of Derived Properties

3.3.1 Regular Sampling

Using the result of (3.28), we evaluate $\mathbf{K}$ for a 2D grid of 101×101 points, spaced at intervals of length $T = 1$. We then compute the sums of rows of $\mathbf{K}$ (equal to $\sum_{j=1}^{101^2} k(\mathbf{x}_i, \mathbf{x}_j)$ for the i-th row), which should ideally be constant, according to the relevant property. Since each row of $\mathbf{K}$ corresponds to a particular point of the grid, we plot these sums as a color image in Figure 3.1. One can observe that near the grid boundary, the sums become significantly smaller, due to the finiteness of the grid (it would be constant everywhere only for an infinite grid). Other than that, they are effectively constant in the whole interior of the grid's region. For different spacings of the grid's samples, i.e. $T \neq 1$, we also observe an identical behavior of the examined sums.

The numerical inversion of $\mathbf{K}$, which is constructed based on the analysis in Section 3.2, is performed without any warnings in MATLAB. Its condition number is in the order of magnitude of 10^3 for all experiments. This confirms that the matrix is not ill-conditioned.

At this point, one is able to evaluate the dual kernel functions for a point $\mathbf{x}$, which assume the form of an n-vector:

$$\mathbf{a}(\mathbf{x}) = \mathbf{K}^{-1}\mathbf{k}(\mathbf{x}), \tag{3.31}$$

where $\mathbf{k}(\mathbf{x})$ is defined as

$$\mathbf{k}(\mathbf{x}) = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) & \ldots & k(\mathbf{x}, \mathbf{x}_n) \end{bmatrix}^T . \tag{3.32}$$

We evaluate the dual kernel functions that correspond to a 2D grid of 41×41 points on a denser "background" grid of 201×201 points. The evaluation points are placed in the central quarter of the sampling grid. We experiment with $T = 2$ and $T = 0.5$ for the spacing of points and get practically identical results in both cases. The dual kernel functions for 3 selected points of the grid are shown in Figure 3.2. As expected, the functions vary with the same frequency as the sampling frequency of the grid. The profile of the functions is smooth, with no abrupt changes. A significant observation is that at certain locations, the functions assume strictly *negative* values.

We also examine the rest of the desired properties of the functions for the above grid. Firstly, we check whether the partition of unity holds, by computing the maximum absolute deviation of the dual kernel functions' sum from 1, i.e. $\max_{\mathbf{x}} |\mathbf{a}(\mathbf{x})^T\mathbf{1} - 1|$. This quantity is $2.2 \cdot 10^{-4}$, which indicates a good compliance with the property at every point.

Next, we focus on the Lagrange property. We isolate the grid points that lie in the evaluation area and compare the corresponding functions to their ideal values, summing the absolute differences of the respective elements over all points:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} |a_i(\mathbf{x}_j) - \delta_{ij}| \, .$$

This "deviation" is equal to $5.1 \cdot 10^{-11}$, which means that the Lagrange property is satisfied with very high accuracy.

Last, we measure how precisely the reproduction property is satisfied. We utilize the l_∞ norm of the displacement of a reproduced point from its initial position, normalized by the spacing of sample points:

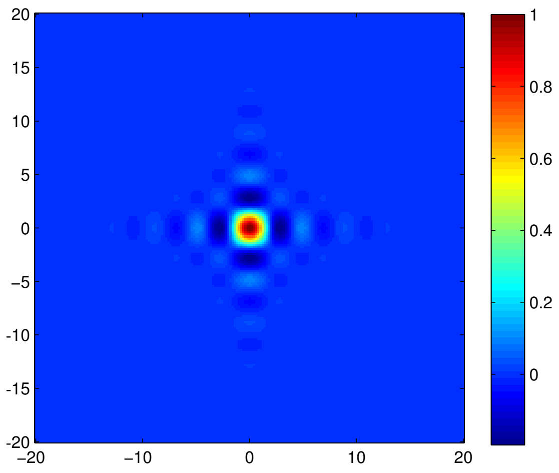
$$D(\mathbf{x}) = \frac{\|\sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{x}_i - \mathbf{x}\|_\infty}{T}.$$

We compute certain statistics for D over all background points: its maximum value is $3.4 \cdot 10^{-3}$, its mean value $0.60 \cdot 10^{-3}$ and its median value $0.33 \cdot 10^{-3}$. Based on these figures, the reproduction property is *approximately* satisfied, since there is a non-negligible displacement relative to the spacing of the evaluation points.

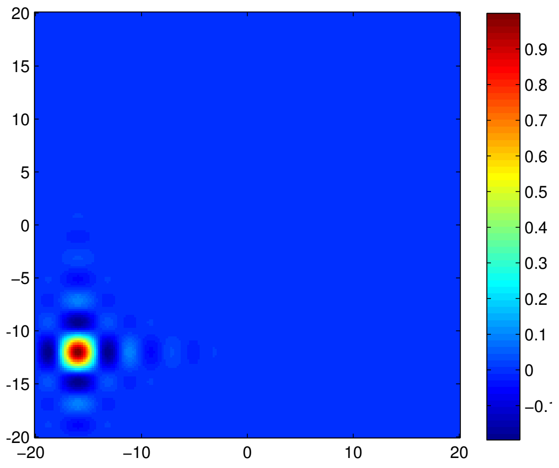
3.3.2 Blue Noise Sampling

We further test dual kernel functions using blue noise sampling. More specifically, we construct a point distribution, based on the code provided by Balzer et al. [15], which consists of 1024 points scattered over the region $[-30,\, 30]^2$. For this experiment, we set $\sigma = 2.5$ and compute $\mathbf{K}$, plotting the sums of its rows in Figure 3.3. The result is similar to the one obtained for a grid (Figure 3.1). However, in the blue noise case, there are small fluctuations of the value of the sums even in the inner part of the region. The condition number of $\mathbf{K}$ is approximately $2.3 \cdot 10^5$. We observed that for larger values of σ, the aforementioned fluctuations attenuate a little, albeit at the cost of a condition number that is some orders of magnitude larger. On the contrary, smaller values of σ result in a better-conditioned kernel matrix, but also in larger fluctuations of the sums, which render kernel coordinates imprecise.

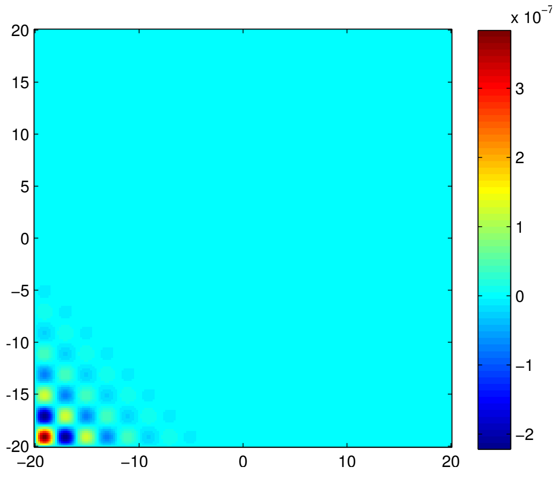
The next step of this experiment is to evaluate dual kernel functions on a grid of 160×160 points. The evaluation points are placed in the central quarter of $[-30,\, 30]^2$. The dual kernel



(a)



(b)



(c)

Figure 3.2: Dual kernel functions of 3 selected points of a 41×41 grid, evaluated over the inner quarter of the grid's region. The functions correspond to (a) the central point of the grid, (b) a point in the lower left part of the evaluation area and (c) the point at the lower left corner of the grid which does not lie inside the evaluation area.

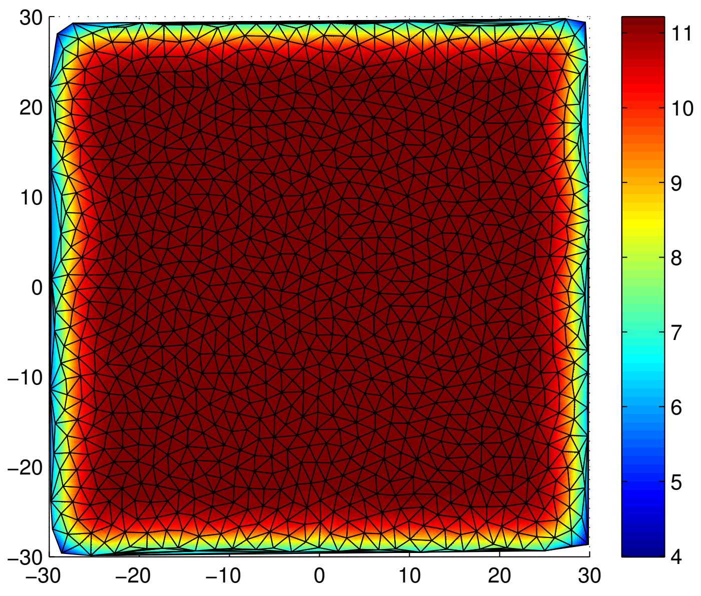


Figure 3.3: Sums of rows of kernel matrix $\mathbf{K}$ for 1024 points distributed as blue noise, with $\sigma = 2.5$.

functions for 3 selected points of the distribution are shown in Figure 3.4. The functions are smooth and vanish as distance from their corresponding point increases. Their profile is not identical for distinct points (as was the case for regular sampling), since the local configuration of points varies spatially. Still, the dominant frequency of the profiles is roughly the same for different coordinates. Again, the functions do assume negative values.

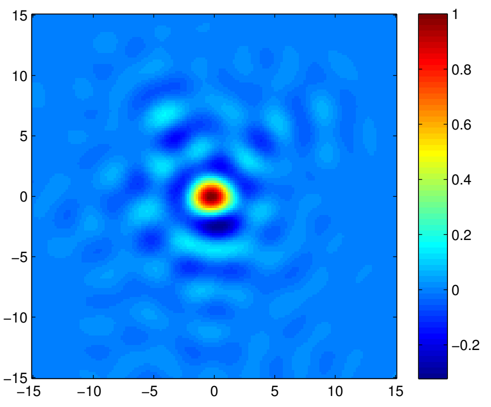
We test the desired properties of dual kernel functions, using the same measures as in Section 3.3.1. The maximum absolute deviation of the functions' sum from 1 is $3.4 \cdot 10^{-4}$, which indicates a good compliance with partition of unity at every point.

The Lagrange property is slightly more difficult to test, as in general there are no evaluation points coinciding exactly with any point of the blue noise distribution. In view of this fact, we compute dual kernel functions *at* the points of the distribution and do not use the background grid at all. The deviation we defined in Section 3.3.1 is $5.7 \cdot 10^{-9}$, which reflects that the Lagrange property is quite accurately satisfied.

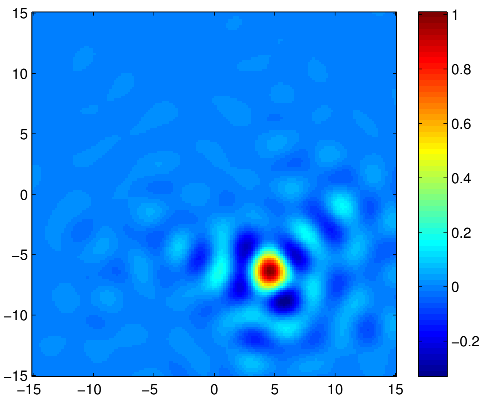
Finally, we examine the reproduction property, measuring the l_∞ norm of the displacement of a reproduced point from its initial position:

$$\left\| \sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{x}_i - \mathbf{x} \right\|_\infty .$$

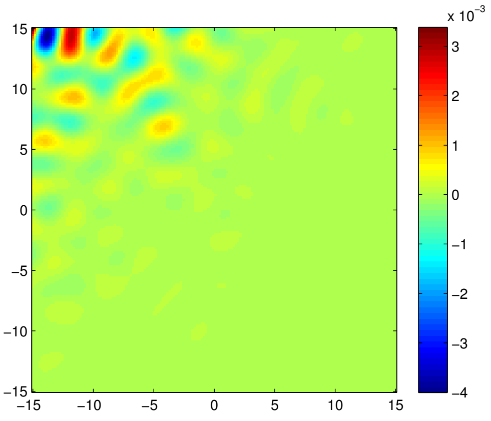
We compute certain statistics for this measure over all evaluation points: its maximum value is $1.1 \cdot 10^{-2}$, its mean value $0.15 \cdot 10^{-2}$ and its median value $0.10 \cdot 10^{-2}$. Based on these figures, the reproduction property is *approximately* satisfied, since there is a non-negligible displacement relative to the spacing of the points of the distribution, which is in the order of magnitude of 1.



(a)



(b)



(c)

Figure 3.4: Dual kernel functions of 3 selected points from a blue noise distribution, evaluated over the inner quarter of the distribution's region. The scale of the Gaussian kernel is $\sigma = 2.5$. The functions correspond to (a) a point near the center of the region, (b) a point in the lower right part of the evaluation area and (c) a point which does not lie inside the evaluation area.

3.4 Local Extrema of Kernel Coordinates

Several previous works that introduce weights to blend transformations, such as [7, 1, 3], have stressed the importance of absence of local extrema in the interior of the shape and verified it for their weights either theoretically or experimentally. A notable exception to this collection of works is [2], where the authors justify the small, attenuating oscillations of their weights around zero with fairness that is gained in interpolation. We use a 1D setting similar to the one in [2] to investigate how kernel coordinates behave with respect to local extrema.

In particular, let us consider the case of regular sampling in 1D. Our focus is on the general case of interpolating a function f (not necessarily one corresponding to a weight) sampled at regularly spaced locations, using our kernel interpolation approach. We use the Gaussian kernel for our analysis. Let $j \in \mathbb{Z}$ index sample points x_j and suppose that for some $l \in \mathbb{Z}$ it holds that

$$f(x_j) = 0,\ \forall j > l \text{ and} \tag{3.33}$$

$$f(x_j) > 0,\ \forall j \leq l. \tag{3.34}$$

As we have shown in our experimental examination of the regular sampling case in 2D in Figure 3.2, dual kernel functions assume bounded negative values at certain regions of the space. It can be shown that in 1D, function $a_l(x)$ that corresponds to sample point x_l assumes its minimum, which has negative value, in the interval $(x_{l+1},\, x_{l+2})$. Denote the corresponding minimum point by x^*. We will show that under (3.33), (3.34) and one more assumption on f, its interpolant $\bar{f}$ has a strict local minimum in $(x_{l+1},\, x_{l+2})$.

At x^*, the value of the interpolant is given by

$$\begin{aligned}
\bar{f}(x^*) &= \sum_j a_j(x^*) f(x_j) \\
&\overset{(3.33)}{=} \sum_{j \leq l} a_j(x^*) f(x_j) \\
&= \sum_{k=0}^{+\infty} a_{l-k}(x^*) f(x_{l-k}) \\
&= \sum_{k=0}^{+\infty} a_{l-2k}(x^*) f(x_{l-2k}) + a_{l-2k-1}(x^*) f(x_{l-2k-1}).
\end{aligned}$$

It can be shown that $a_{l-2k}(x^*) < 0$ and $a_{l-2k-1}(x^*) > 0,\ \forall k \in \mathbb{N}$. Moreover, $|a_{l-2k}(x^*)| > |a_{l-2k-1}(x^*)|,\ \forall k \in \mathbb{N}$. If we further assume for f that

$$\frac{f(x_{l-2k})}{f(x_{l-2k-1})} > \frac{|a_{l-2k-1}(x^*)|}{|a_{l-2k}(x^*)|},\ \forall k \in \mathbb{N}, \tag{3.35}$$

then $a_{l-2k}(x^*) f(x_{l-2k}) + a_{l-2k-1}(x^*) f(x_{l-2k-1}) < 0,\ \forall k \in \mathbb{N}$ and summing over all k leads to $\bar{f}(x^*) < 0$. Combining this with $\bar{f}(x_{l+1}) = \bar{f}(x_{l+2}) = 0$, which is due to the Lagrange property, shows that $\bar{f}$ has a strict local minimum in $(x_{l+1},\, x_{l+2})$.

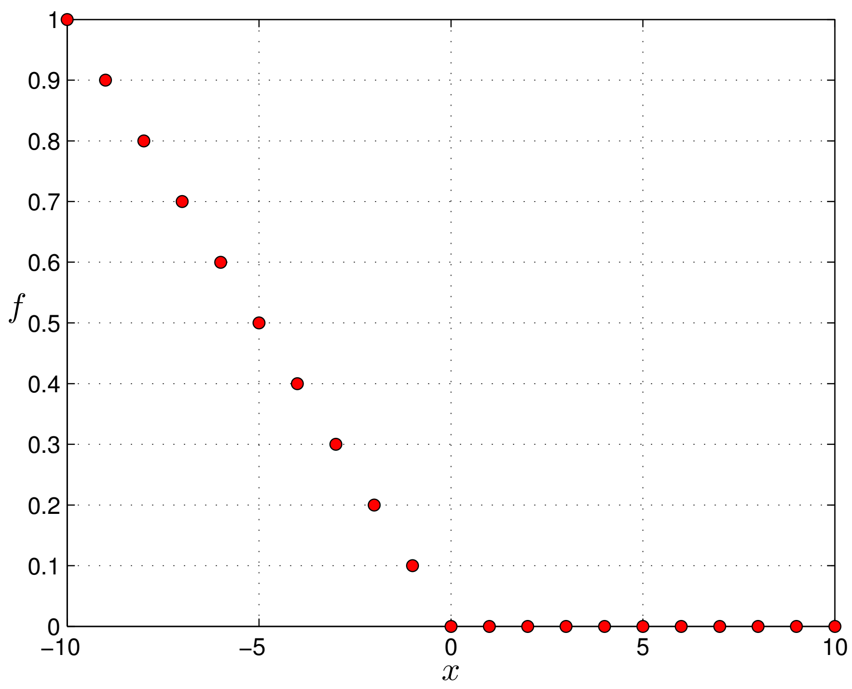


Figure 3.5: Function f defined in (3.36), sampled regularly at 21 points.

We provide an example for the above analysis, using 21 samples $x_1, \ldots, x_{21}$ spaced regularly with period $T = 1$ in the interval $[-10, 10]$. The interpolated function is defined as

$$f(x) = \left\{ \begin{array}{rr} -x/10, & x < 0 \\ 0, & x \geq 0 \end{array} \right\} \tag{3.36}$$

and its samples are illustrated in Figure 3.5. In this case, $l = 10$. We use the optimal scale parameter for the Gaussian kernel based on (3.28) (the analysis is almost the same in 1D as in 2D). We show the dual kernel function for $x_{11} = 0$ in Figure 3.6. Finally, Figure 3.7 demonstrates the existence of a local minimum of the interpolant $\bar{f}$ in $(x_{11}, x_{12}) = (0, 1)$. Oscillation of the interpolant also extends to the next interval, although at a smaller magnitude.

We turn to a function that is more relevant for our method, namely a weight function obtained with heat diffusion starting from a certain point (we provide details for heat diffusion in Section 4.7). In particular, we define 101 regularly spaced samples with sampling period $T = 1$ in the interval $[-50, 50]$. We run heat diffusion from the point $x_{21} = -30$ and evaluate the interpolant of the resulting weights in the interval $[-30, 30]$. Results for 5 iterations of heat diffusion are shown in Figure 3.8 and for 20 iterations in Figure 3.9. In both cases, there are obvious oscillations of the interpolant, which are slightly less intense for 20 iterations. Figures 3.8(c) and 3.9(c) clearly show that even when samples have strictly positive values, the interpolant may assume negative values in between, due to the ripple effect. Figures 3.8(d) and 3.9(d) depict the continuation of oscillations between positive and negative values in the far right part of the evaluation interval, albeit with a rapidly attenuating magnitude.

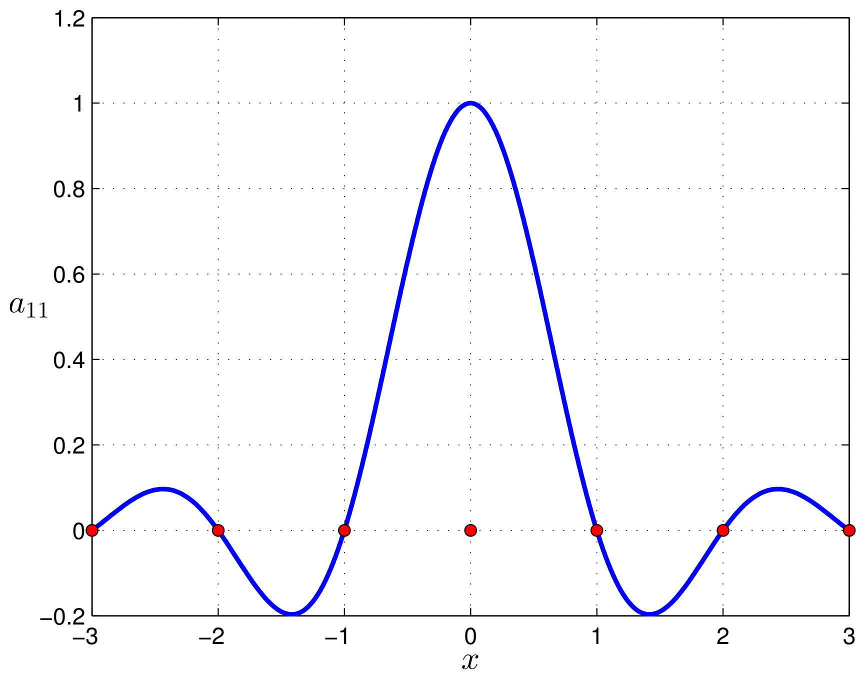


Figure 3.6: Dual kernel function $a_{11}(x)$ pertaining to sample point $x_{11} = 0$, refined to interval $[-3,\, 3]$.

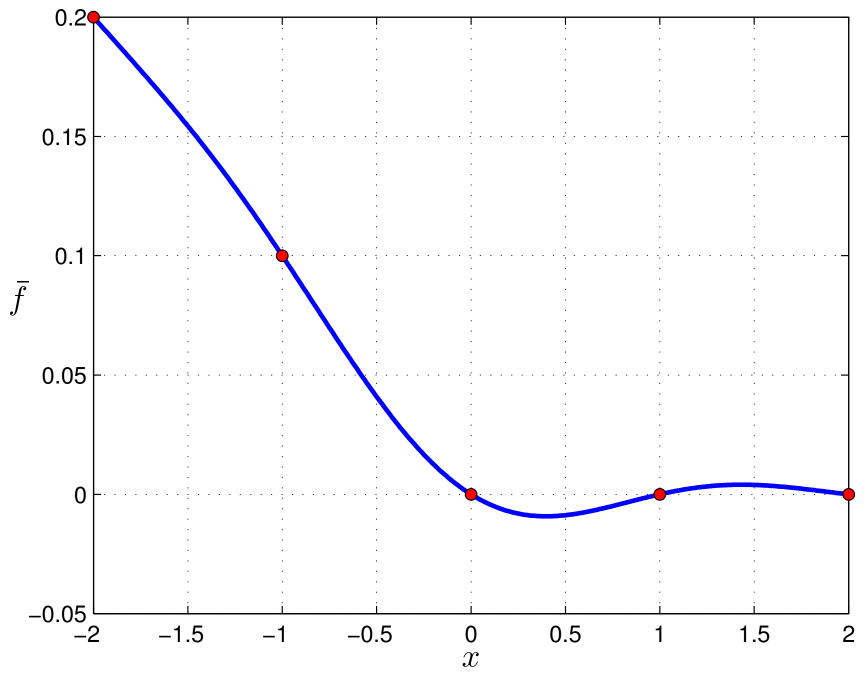


Figure 3.7: Interpolant $\bar{f}$ for samples of f, refined to interval $[-2,\, 2]$.

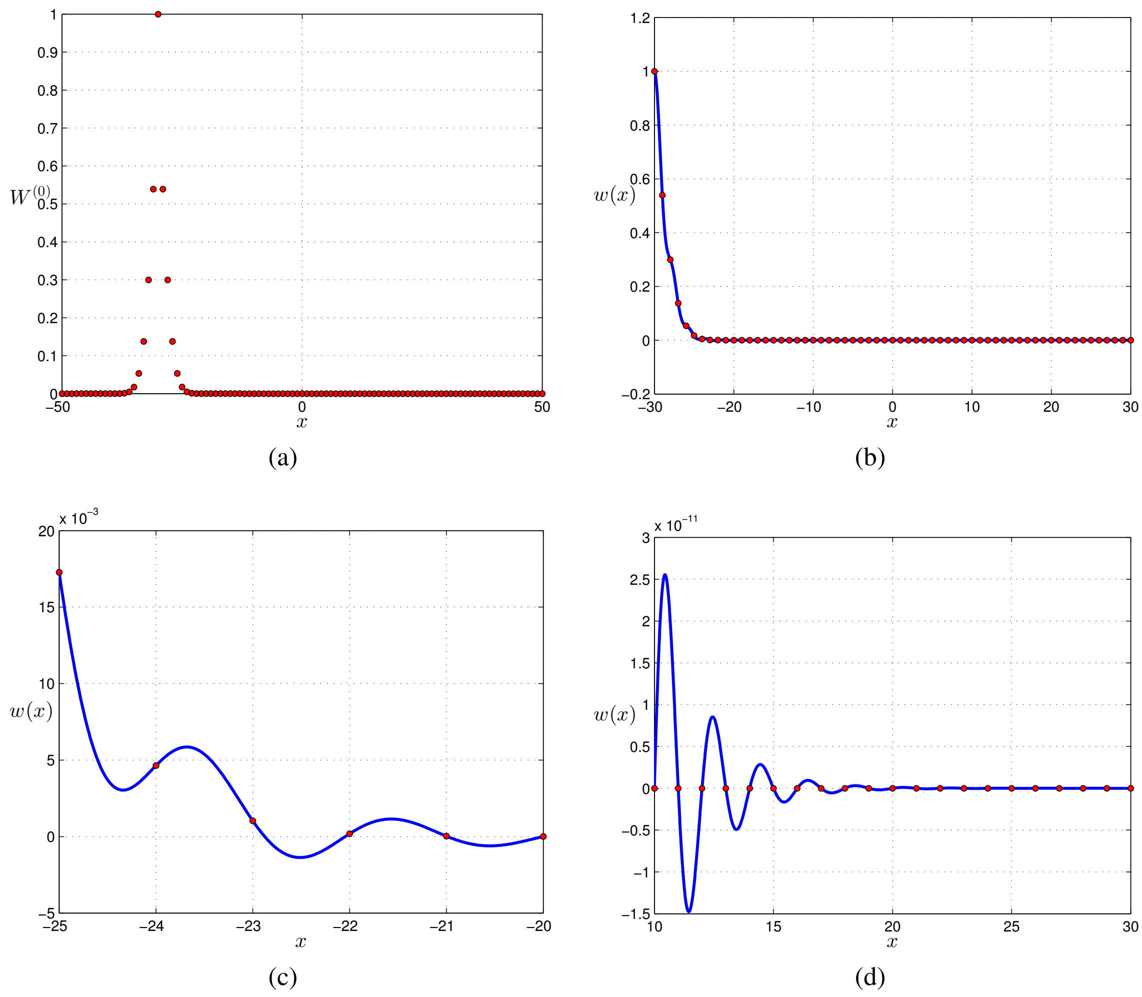


Figure 3.8: Example of interpolation of weights obtained with heat diffusion. (a) Weight samples after 5 heat diffusion iterations, (b) Interpolant refined to the interval $[-30,\ 30]$, (c) Zoom of (b) in the interval $[-25,\ -20]$, (d) Zoom of (b) in the interval $[10,\ 30]$.

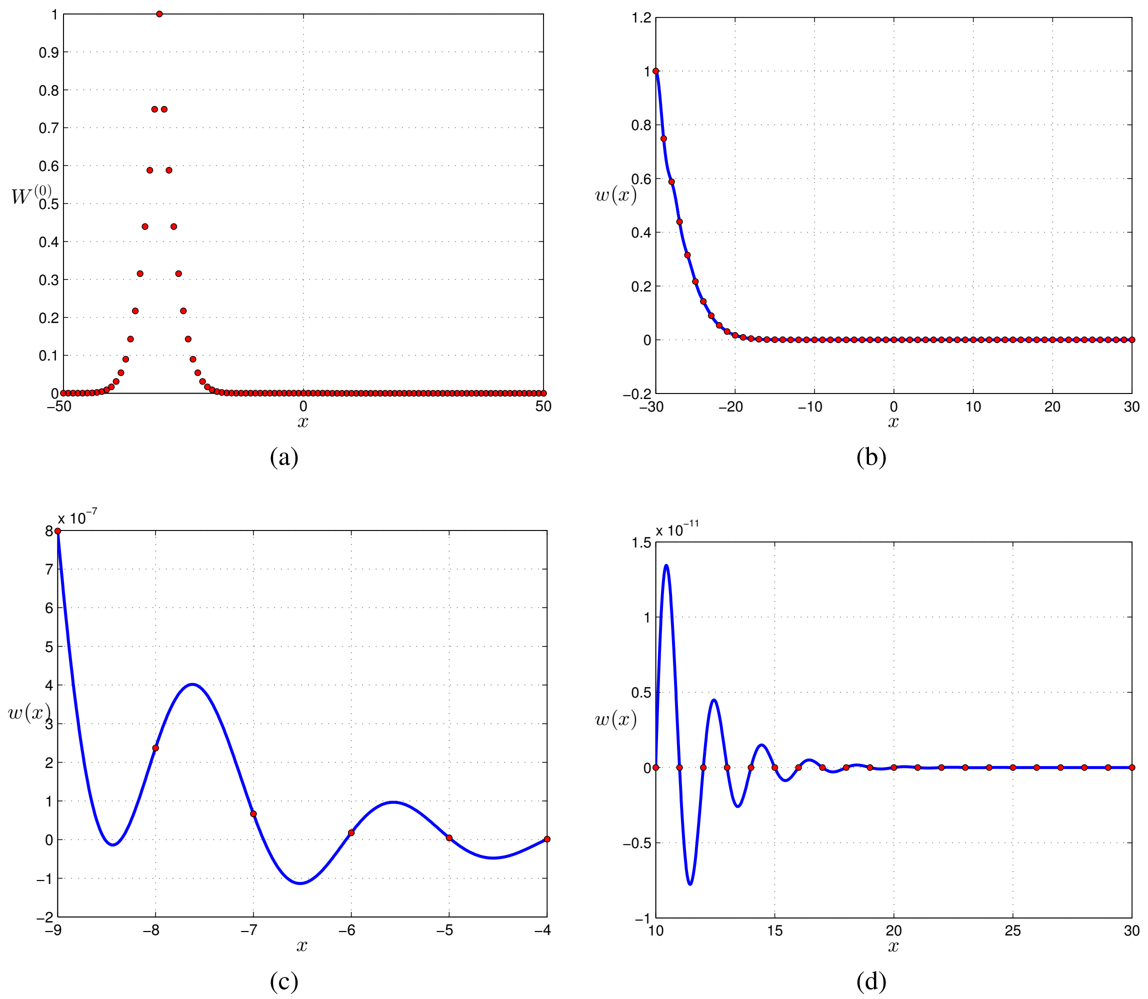


Figure 3.9: Example of interpolation of weights obtained with heat diffusion. (a) Weight samples after 20 heat diffusion iterations, (b) Interpolant refined to the interval $[-30,\,30]$, (c) Zoom of (b) in the interval $[-9,\,-4]$, (d) Zoom of (b) in the interval $[10,\,30]$.

3.5 Jacobian Matrix for 2D Deformation with Kernel Coordinates

Poranne and Lipman [11] connect the distortion induced by a continuously differentiable planar (2D) mapping to the singular values of its Jacobian matrix. In the case of kernel coordinates, this Jacobian can be derived straightforwardly when linear blend skinning is the underlying model for applying the deformation to the shape.

More specifically, let $\mathbf{f}(\mathbf{x})$ denote the image of an arbitrary point $\mathbf{x}$ (both in homogeneous representation) under the examined mapping and $\mathbf{T}_i$ denote the transformation related to $\mathbf{c}_i$, $i = 1, \ldots, m$. Linear blend skinning implies that given the weights $\mathbf{w}(\mathbf{x})$ for this point, its image is simply given by

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \mathbf{T}_1\mathbf{x} & \cdots & \mathbf{T}_m\mathbf{x} \end{bmatrix} \mathbf{w}(\mathbf{x}). \tag{3.37}$$

(3.37) can be re-expressed as

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \mathbf{T}_1 & \cdots & \mathbf{T}_m \end{bmatrix} \begin{bmatrix} w_1(\mathbf{x})\mathbf{I}_{d+1} \\ \vdots \\ w_m(\mathbf{x})\mathbf{I}_{d+1} \end{bmatrix} \mathbf{x}, \tag{3.38}$$

so that standard properties of matrix differentiation can be used to compute the Jacobian matrix. In particular, if we denote the final linear blend skinning transformation applied to $\mathbf{x}$ by $\mathbf{T}(\mathbf{x})$, it follows that

$$\begin{aligned} J\mathbf{f}(\mathbf{x}) &= \begin{bmatrix} \mathbf{T}_1 & \cdots & \mathbf{T}_m \end{bmatrix} \left(\begin{bmatrix} w_1(\mathbf{x})\mathbf{I}_{d+1} \\ \vdots \\ w_m(\mathbf{x})\mathbf{I}_{d+1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}\nabla w_1(\mathbf{x})^T \\ \vdots \\ \mathbf{x}\nabla w_m(\mathbf{x})^T \end{bmatrix} \right) \\ &= \mathbf{T}(\mathbf{x}) + \begin{bmatrix} \mathbf{T}_1\mathbf{x} & \cdots & \mathbf{T}_m\mathbf{x} \end{bmatrix} J\mathbf{w}(\mathbf{x}). \end{aligned} \tag{3.39}$$

In our kernel coordinates framework, the weights are computed as an interpolation of user-defined weights $\mathbf{W}$ with dual kernel functions $\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$, leading to the following form:

$$\mathbf{w}(\mathbf{x}) = \mathbf{W}\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}). \tag{3.40}$$

Due to linearity of the differential operator, the Jacobian matrix for (3.40) is

$$J\mathbf{w}(\mathbf{x}) = \mathbf{W}\mathbf{K}^{-1}J\mathbf{k}(\mathbf{x}). \tag{3.41}$$

Focusing on the Gaussian kernel, the gradient of the i-th element of $\mathbf{k}(\mathbf{x})$, as defined in (3.32), is

$$\nabla k(\mathbf{x}, \mathbf{x}_i) = -\frac{1}{\sigma^2} k(\mathbf{x}, \mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i). \tag{3.42}$$

Using the notation of (3.9), we obtain

$$J\mathbf{k}(\mathbf{x}) = -\frac{1}{\sigma^2} \operatorname{diag}\left(\mathbf{k}(\mathbf{x})\right) \left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right). \tag{3.43}$$

Finally, we combine (3.39) with (3.41) and (3.43) into

$$J\mathbf{f}(\mathbf{x}) = \mathbf{T}(\mathbf{x}) - \frac{1}{\sigma^2}\left[\begin{array}{ccc} \mathbf{T}_1\mathbf{x} & \cdots & \mathbf{T}_m\mathbf{x} \end{array}\right]\mathbf{W}\mathbf{K}^{-1}\,\mathrm{diag}\left(\mathbf{k}(\mathbf{x})\right)\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right). \tag{3.44}$$

4 Weight Optimization

In our framework, we let the user interact with the application and input her preferred weights for various parts of the object. However, user-specified weights (or weights used to initialize the interface) do not possess all the desired properties in general. Ideally, we would like weights to be as "close" to what the user has indicated as possible and at the same time satisfy all properties. We construct various objectives that measure this proximity and use different sets of constraints to model the desired properties, which lead to distinct optimization problems.

4.1 Quadratic Programming

First, we employ a quadratic cost for the deviation from the input weights, which leads to a well-defined quadratic program for each point. For the j-th input point $\mathbf{x}_j$, let ${\mathbf{w}_j}^{(0)}$ be the vector of input weights for this point and $\mathbf{w}_j$ the final weight vector, which is subject to optimization.

The quadratic objective is defined as

$$\min_{\mathbf{w}_j} \left\{ \left\| \mathbf{w}_j - {\mathbf{w}_j}^{(0)} \right\|_2^2 \right\}$$

and it can be reformulated by dropping the constant term as

$$\min_{\mathbf{w}_j} \left\{ {\mathbf{w}_j}^T \mathbf{w}_j - 2 \left({\mathbf{w}_j}^{(0)} \right)^T \mathbf{w}_j \right\} .$$

The properties of the weights—reproduction, partition of unity and non-negativity—can be expressed in matrix notation as linear equalities or inequalities, hence resulting in a quadratic program. Consequently, n quadratic programs are to be solved in total, the j-th of which is

$$\min_{\mathbf{w}_j} \left\{ {\mathbf{w}_j}^T \mathbf{w}_j - 2 \left({\mathbf{w}_j}^{(0)}\right)^T \mathbf{w}_j \right\} \tag{4.1a}$$

$$\text{s.t. } \mathbf{C}\mathbf{w}_j = \mathbf{x}_j, \tag{4.1b}$$

$$\mathbf{1}^T \mathbf{w}_j = 1, \tag{4.1c}$$

$$\mathbf{w}_j \geq \mathbf{0}. \tag{4.1d}$$

4.2 Lifting Non-negativity

Solving a quadratic program like (4.1) can be costly, especially if the number of control weights, i.e. the dimensionality of the problem, is relatively large. Therefore, it is possible to omit the non-negativity constraint (4.1d) and recast the optimization in a form that admits an closed-form solution:

$$\min_{\mathbf{w}_j} \left\{ {\mathbf{w}_j}^T \mathbf{w}_j - 2 \left({\mathbf{w}_j}^{(0)}\right)^T \mathbf{w}_j \right\} \tag{4.2a}$$

$$\text{s.t. } \mathbf{C}\mathbf{w}_j = \mathbf{x}_j, \tag{4.2b}$$

$$\mathbf{1}^T \mathbf{w}_j = 1. \tag{4.2c}$$

We form the Lagrangian for (4.2) as

$$L(\mathbf{w}_j,\, \boldsymbol{\lambda},\, \mu) = {\mathbf{w}_j}^T \mathbf{w}_j - 2 \left({\mathbf{w}_j}^{(0)}\right)^T \mathbf{w}_j + \boldsymbol{\lambda}^T \left(\mathbf{C}\mathbf{w}_j - \mathbf{x}_j\right) + \mu \left(\mathbf{1}^T \mathbf{w}_j - 1\right) \tag{4.3}$$

and set its partial derivatives to 0 to obtain the expressions for the Lagrange multipliers and the optimal weight vector:

$$\boldsymbol{\lambda} = 2 \left(\mathbf{C}\mathbf{C}^T - \frac{1}{m} \mathbf{C}\mathbf{1} \left(\mathbf{C}\mathbf{1}\right)^T \right)^{-1} \left(\mathbf{C}{\mathbf{w}_j}^{(0)} + \frac{1 - \mathbf{1}^T {\mathbf{w}_j}^{(0)}}{m} \mathbf{C}\mathbf{1} - \mathbf{x}_j \right), \tag{4.4}$$

$$\frac{\mu}{2} = \frac{\mathbf{1}^T {\mathbf{w}_j}^{(0)} - 1 - \frac{1}{2} \mathbf{1}^T \mathbf{C}^T \boldsymbol{\lambda}}{m}, \tag{4.5}$$

$$\mathbf{w}_j = {\mathbf{w}_j}^{(0)} - \frac{1}{2} \mathbf{C}^T \boldsymbol{\lambda} - \frac{\mu}{2} \mathbf{1}. \tag{4.6}$$

Based on (4.4)–(4.6) which only involve linear operations, we can compute the output weights very fast, compared to the original quadratic program that included the non-negativity constraint. Of course, this comes at the expense of weights assuming negative values at certain areas of the object.

4.3 Linear Precision on Edges

Unfortunately, omitting the non-negativity constraint leads to loss of linear precision on the edges (or faces) of the cage, since reproduction alone cannot force the weights to be linear on the edges when they can assume negative values. In order to guarantee linear precision on the edges, we must explicitly set all weights to zero for sample points that lie on some edge, except the ones corresponding to the control points at the ends of this edge.

For such a 2D point $\mathbf{x}$, let $k,\, l \in \{1,\, \dots,\, m\}$ be the distinct indices of the control points at the ends of the edge. Partition of unity and reproduction at this point mean that

$$\begin{aligned} w_k + w_l &= 1, \\ w_k \mathbf{c}_k + w_l \mathbf{c}_l &= \mathbf{x}. \end{aligned}$$

We solve this linear system to obtain expressions for the two non-zero weights:

$$w_k = \frac{(\mathbf{x} - \mathbf{c}_l)^T (\mathbf{c}_k - \mathbf{c}_l)}{\|\mathbf{c}_k - \mathbf{c}_l\|^2}, \tag{4.7a}$$

$$w_l = \frac{(\mathbf{x} - \mathbf{c}_k)^T (\mathbf{c}_l - \mathbf{c}_k)}{\|\mathbf{c}_k - \mathbf{c}_l\|^2}. \tag{4.7b}$$

4.4 ℓ_1 Optimization

Instead of a quadratic cost, it is also valid to penalize the ℓ_1 distance of the output weights from the desired weights. This way, the difference of the two vectors becomes sparse. We denote this difference by $\mathbf{v}_j = \mathbf{w}_j - \mathbf{w}_j{}^{(0)}$ and, omitting the non-negativity constraint again, the problem is formulated as

$$\min_{\mathbf{v}_j} \left\{ \|\mathbf{v}_j\|_1 \right\} \tag{4.8a}$$

$$\text{s.t. } \mathbf{C}\mathbf{v}_j = \mathbf{x}_j - \mathbf{C}\mathbf{w}_j{}^{(0)}, \tag{4.8b}$$

$$\mathbf{1}^T \mathbf{v}_j = 1 - \mathbf{1}^T \mathbf{w}_j{}^{(0)}. \tag{4.8c}$$

This minimization of ℓ_1 norm under linear equality constraints is studied in [16], where the problem is recast to a linear program and solved via a primal-dual algorithm. Candès and Romberg provide in the supplementary material of [16] a MATLAB function, `l1eq_pd`, that performs the relevant optimization. We use this function to solve (4.8).

4.5 ℓ_1-norm Regularization

While the problem of (4.8) guarantees a sparse *difference* vector between the desired weights and the optimized ones, we would rather have sparse optimized weights, which implies locality of the weights and aids interpretation of how each control handle influences the shape. The least absolute shrinkage and selection operator (LASSO) [17] regularizes a quadratic cost with an ℓ_1 norm term to induce sparsity in the resulting vector. However, the LASSO pertains to an unconstrained setting, whereas our framework (e.g. (4.1)) essentially involves constraints that guarantee reproduction, partition of unity and non-negativity. Therefore, adding an ℓ_1-norm regularizer results in the following modification of (4.1), where $\lambda > 0$:

$$\min_{\mathbf{w}_j} \left\{ \left\| \mathbf{w}_j - {\mathbf{w}_j}^{(0)} \right\|_2^2 + \lambda \left\| \mathbf{w}_j \right\|_1 \right\} \tag{4.9a}$$

$$\text{s.t. } \mathbf{C}\mathbf{w}_j = \mathbf{x}_j, \tag{4.9b}$$

$$\mathbf{1}^T \mathbf{w}_j = 1, \tag{4.9c}$$

$$\mathbf{w}_j \geq \mathbf{0}. \tag{4.9d}$$

The combination of constraints (4.9c) and (4.9d) implies that for any feasible point of (4.9), $\left\| \mathbf{w}_j \right\|_1 = 1$. This means that the ℓ_1-norm regularizer is constant for all feasible weights and hence can be omitted from the objective function, reducing problem (4.9) to our original quadratic programming formulation (4.1). Consequently, ℓ_1-norm regularization does not provide us with a different (potentially sparse) result than the one obtained with quadratic programming.

4.6 Weight Initialization with Heat Diffusion

The initial weights ${\mathbf{w}_j}^{(0)}$ of the optimization affect the obtained solution greatly, no matter which of the above formulations is used. For our first, simple experiments, initialization can be performed synthetically, using the inverse squared Euclidean distance between the sample point and the control point:

$${w_{ij}}^{(0)} = \frac{1}{1 + \left(\frac{\left\| \mathbf{x}_j - \mathbf{c}_i \right\|}{p} \right)^2}. \tag{4.10}$$

However, using Euclidean distances of the ambient space to compute weights is known to disrespect the particularities of the processed shape. To ensure shape-awareness of the initial values of the weights which are fed to optimization, we employ *heat diffusion* on the shape. More specifically, each coordinate is initialized to a Dirac delta at the corresponding control point:

$${w_{ij}}^{(0)} \leftarrow \delta_{ij},\; i = 1, \ldots, m,\; j = 1, \ldots, n. \tag{4.11}$$

In (4.11), we use the same convention for indexing sample points as in (3.18) (the first indices of sample points correspond to the control points) to simplify the expression. Subsequently, we iteratively update weights for sample points other than those coinciding with control points by diffusing heat from the latter. This operation involves the same Gaussian kernel matrix $\mathbf{K}$ that we use to interpolate weights. We note that weights at control points are kept fixed during diffusion to their initial values which obey the Lagrange property. Each iteration of the diffusion can be expressed in matrix form as

$$\mathbf{W}^{(0)}_{:,\mathcal{D}} \leftarrow \mathbf{W}^{(0)} \mathbf{K}_{:,\mathcal{D}}, \tag{4.12}$$

where $\mathcal{D} = \{m + 1, \ldots, n\}$ indexes all sample points apart from the control points. We refer to the above variant as heat diffusion with impulse initialization.

In practice, we also want sample points along the edges of a cage to have linear initial weights, so that subsequent optimization is trivially guided to these linear values that satisfy all constraints. For this reason, we use (4.7) to initialize weights for such points to linear values and keep them fixed across heat diffusion iterations. As a result, the update equation for heat diffusion is modified to

$$\mathbf{W}^{(0)}_{:,\mathcal{I}} \leftarrow \mathbf{W}^{(0)}\mathbf{K}_{:,\mathcal{I}}, \tag{4.13}$$

where $\mathcal{I}$ indexes sample points that do not lie on an edge. We term this variant heat diffusion with fixed linear weights on edges.

4.7 Experimental Results

We experiment on a 2D grid consisting of 101×101 points, spaced at intervals of length $T = 0.1$. We use four control points at the corners of the grid and define synthetic initial weights based on (4.10), using $p = 1$. These synthetic weights are shown in Figures 4.1(a)–4.1(d). We perform optimization for the weights with all above-mentioned methods and present the results in Figures 4.1(e)–4.1(t). Partition of unity has forced the weights to grow in the middle area of the grid and become less local, however the smoothness of the initial weights has been preserved in all cases. The solutions obtained with ℓ_1 optimization and quadratic cost minimization appear smoother, whereas quadratic programming produces weights that are not differentiable along certain lines of the grid's region. It is noteworthy that for quadratic cost minimization, the weights become *negative* near the corner opposite the respective control point. On the contrary, despite the fact that non-negativity is not explicitly included in the formulation of ℓ_1 minimization, the resulting weights assume positive values almost in the entire grid and the few negative weights have very small magnitude.

We also test weight initialization with heat diffusion on the above grid, using both variants described in Section 4.6. For the processed grid, we ran 5 iterations of heat diffusion in both cases. Initial weights acquired with heat diffusion with impulse initialization are shown in Figures 4.2(a)–4.2(d). We perform optimization for these weights with all methods (except quadratic cost minimization with no linear precision on edges) and present the results in Figures 4.2(e)–4.2(p). Results for quadratic programming and ℓ_1 optimization are almost identical with initialization through inverse squared distance weights. On the other hand, quadratic cost minimization without non-negativity constraints yields linearly decaying weights in the interior of the grid, while weights on the edges deviate from this pattern because of enforcement of linear precision.

Heat diffusion with fixed linear weights on edges results in initial weights shown in Figures 4.3(a)–4.3(d). We run optimization on these weights with all methods (apart from quadratic cost minimization with explicit linear precision on edges, since the input weights to optimization already satisfy all properties, including linear precision) and demonstrate the optimization output in Figures 4.3(e)–4.3(p). Results for quadratic programming and ℓ_1 optimization are almost identical with the ones obtained using heat diffusion with impulse initialization. Furthermore, the weights obtained with quadratic cost minimization without non-negativity constraints only differ with the previous result near the boundary of the grid's region, where we observe a smoother transition from linear values to the distinct linear pattern of the interior that also occurs with the previous heat diffusion variant.

To further verify that the optimized weights are intuitive, we isolate the points on the diagonals of the grid and on its edges. We visualize the results for the control point at the lower

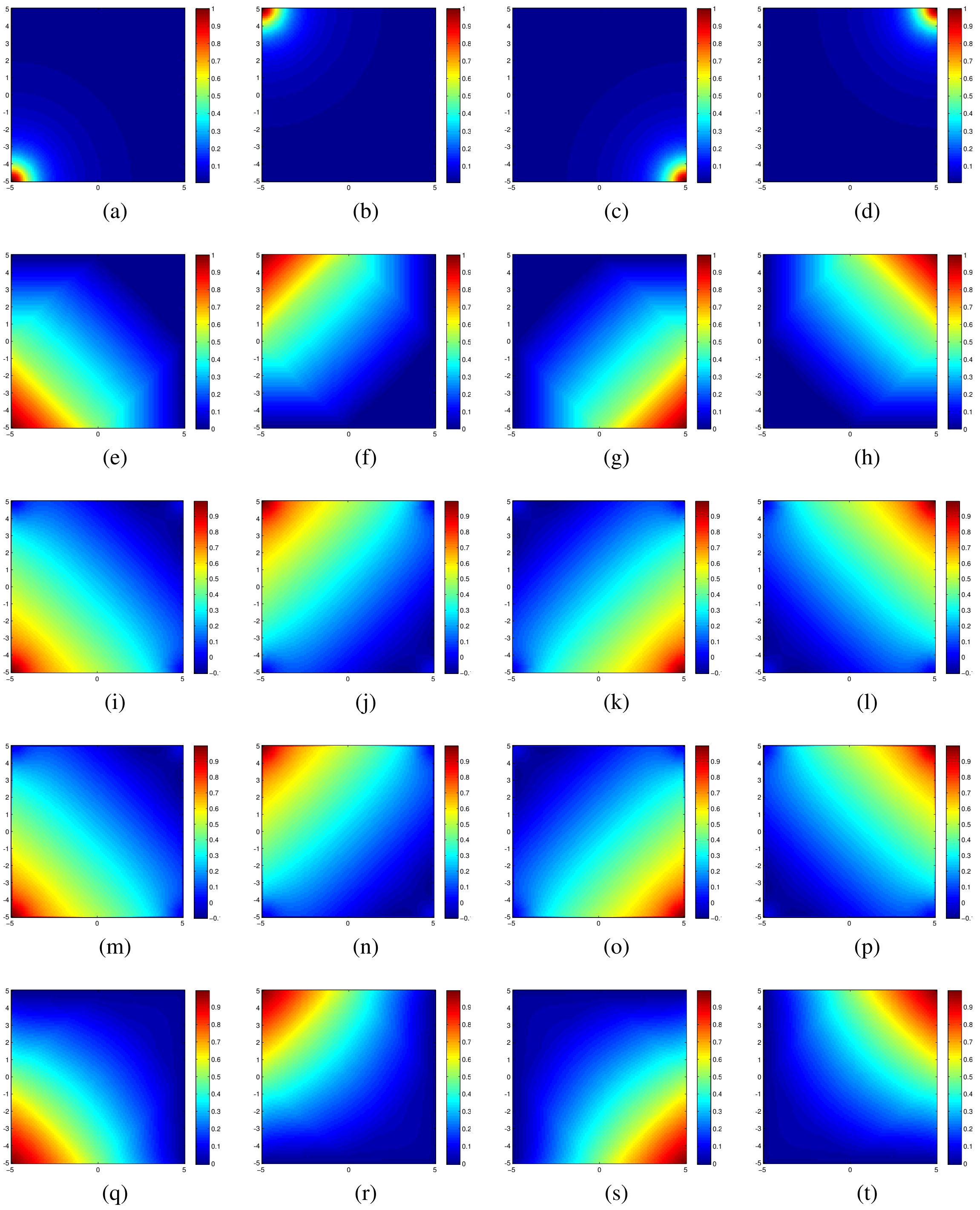


Figure 4.1: Comparison of approaches to weight selection on a 101×101 grid. Four control points are used, each one corresponding to a column of the figure. The input weights, based on the reciprocal quadratic distance from the control points, are presented in the first row. The weights are computed with quadratic programming (second row), quadratic cost minimization without non-negativity constraints (third row), quadratic cost minimization without non-negativity constraints and with linear precision on edges (fourth row) and ℓ_1 norm minimization (fifth row).

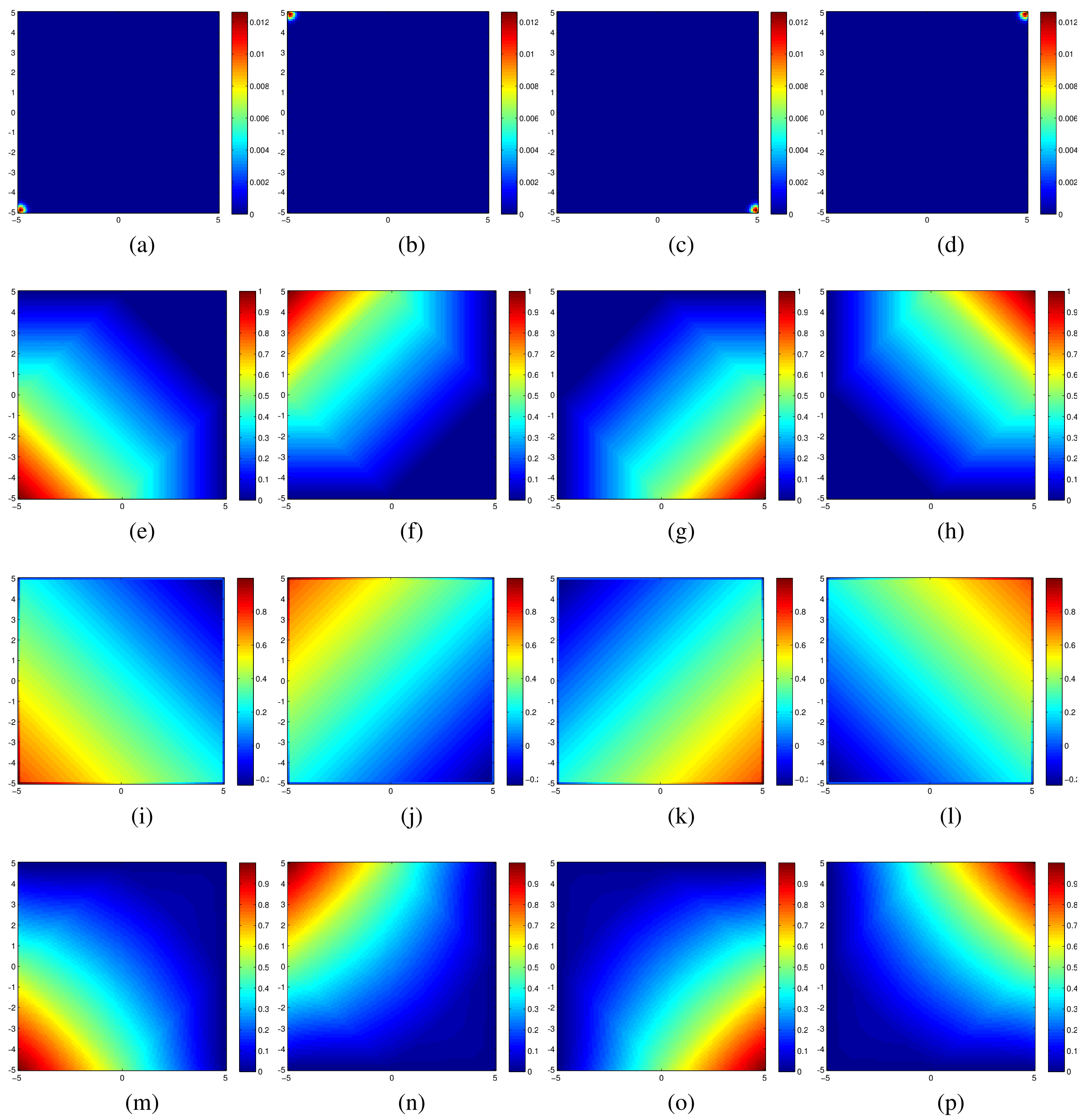


Figure 4.2: Comparison of approaches to weight selection on a 101×101 grid. Four control points are used, each one corresponding to a column of the figure. The input weights, based on heat diffusion with impulse initialization, are presented in the first row. The weights are computed with quadratic programming (second row), quadratic cost minimization without non-negativity constraints and with linear precision on edges (third row) and ℓ_1 norm minimization (fourth row).

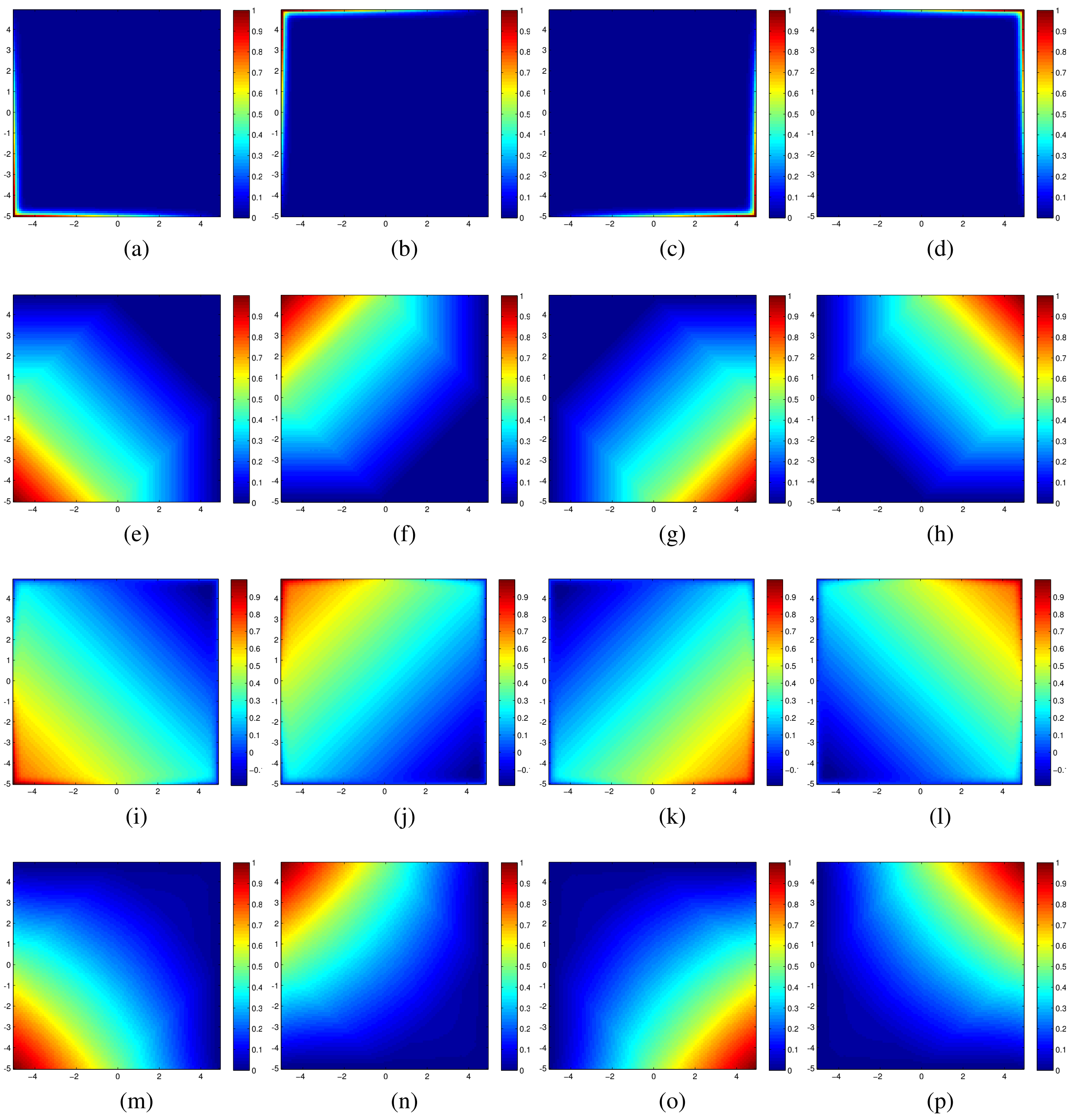


Figure 4.3: Comparison of approaches to weight selection on a 101×101 grid. Four control points are used, each one corresponding to a column of the figure. The input weights, based on heat diffusion with fixed linear weights on edges, are presented in the first row. The weights are computed with quadratic programming (second row), quadratic cost minimization without non-negativity constraints (third row) and ℓ_1 norm minimization (fourth row).

Method	**Time** (s)	**Time per point** (ms)
QP	17.1	1.7
QEC	0.109	0.011
ℓ_1	7.61	0.75

Table 4.1: Running times of different optimization approaches for weight selection. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

left corner of the grid in Figures 4.4, 4.5 and 4.6, which correspond to initialization via inverse squared distance, heat diffusion with impulse initialization and heat diffusion with fixed linear weights on edges respectively. The results were identical for the rest of the control points. We found that for quadratic programming and ℓ_1 optimization, weights are effectively monotonic along the neighboring edges of the control point and the relevant diagonal, irrespective of the initialization method (Figures 4.4(a)–4.4(c), 4.4(m)–4.4(o), 4.5(a)–4.5(c), 4.5(i)–4.5(k), 4.6(a)–4.6(c), 4.6(i)–4.6(k)). On the contrary, weights resulting from quadratic cost minimization without non-negativity are not monotonic on the diagonal in any case. Furthermore, these weights do not satisfy linear precision on the edges when it is not enforced explicitly or ensured through proper initialization, as we can deduce from Figures 4.4(f)–(g). This is due to weights of control points away from the edges assuming non-zero values, which is illustrated in Figure 4.4(h). Once linear precision is enforced by calculating the relevant weights directly from the constraints (Figures 4.4(j)–(k) and 4.5(f)–(g)) or ensured via providing linear initial weights to the optimization (Figures 4.6(f)–(g)), weights do become linear along the edges. As regards ℓ_1 optimization and quadratic programming, linear precision on edges is effectively satisfied even *without* any explicit constraints, since weights from control points away from an edge are practically zero (Figures 4.4(d), 4.4(p), 4.5(d) and 4.5(l)). We should also note that the profile of weights for quadratic programming along the diagonal of the grid (Figures 4.4(a), 4.5(a) and 4.6(a)) bears a clear resemblance to the example we have illustrated in Section 3.4 and more specifically in Figures 3.5–3.7. Therefore, when interpolating these weights to evaluate kernel coordinates, we should expect oscillations around zero near the point where weights exhibit the characteristic “knee”.

We measured the running time for weight optimization with each method. In Table 4.1, we provide the total running time for each method, as well as the average time spent for each point of the grid (in order to have a result that is independent of the grid size). As expected, the closed-form solution of quadratic cost without non-negativity constraints takes by far the least time to compute. Between the two remaining methods, ℓ_1 optimization is at least twice as fast as quadratic programming, needing only few iterations of the primal-dual algorithm to reach the optimum of the corresponding linear program.

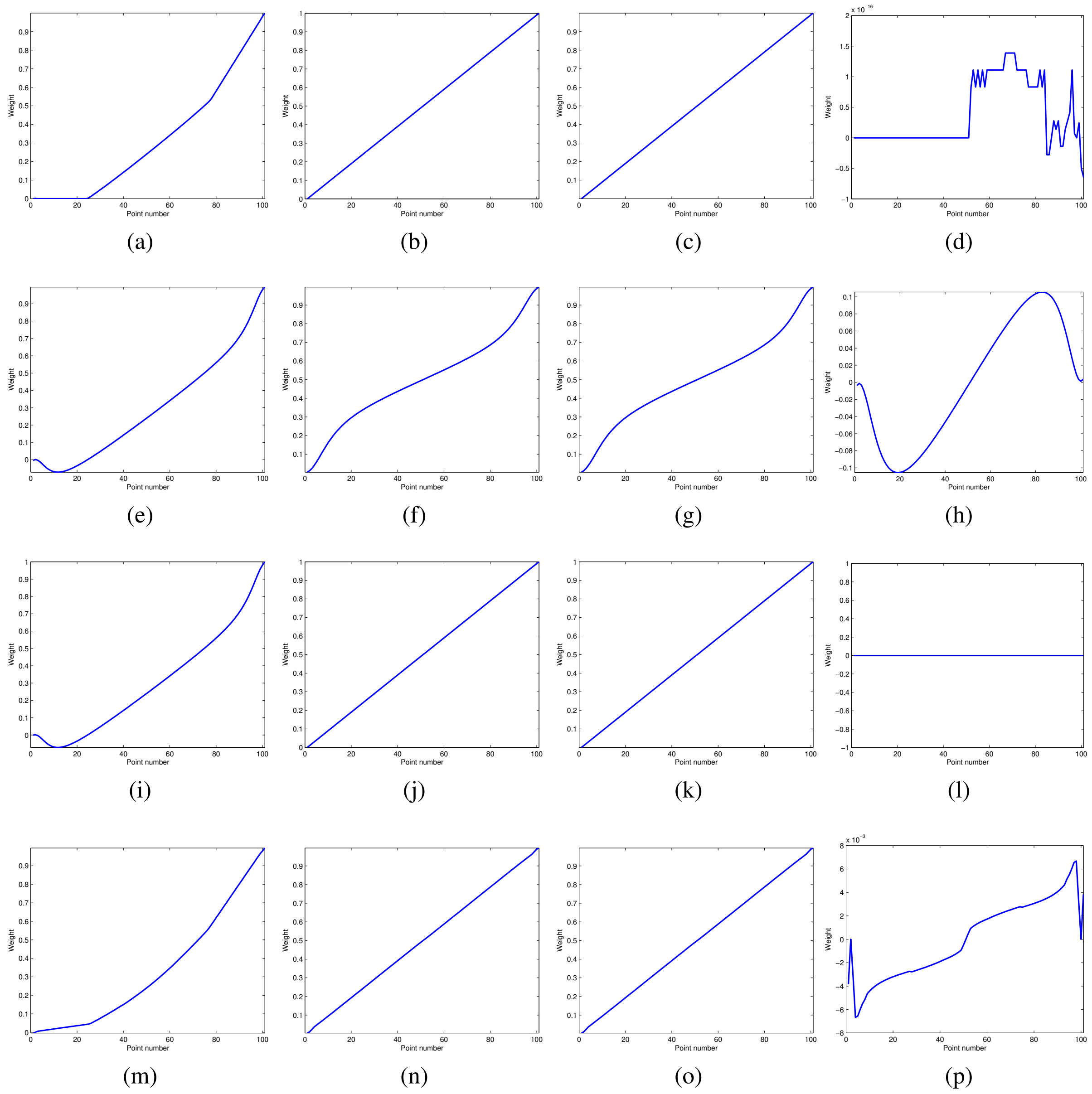


Figure 4.4: Variation of optimized weights on the diagonal and edges of a 101×101 grid with four control points. Weights were initialized with (4.10). The control point at the lower left corner of the grid was chosen to create the plots. The horizontal axis of the plots shows the point number on the diagonal or edge, starting from the point farthest from the examined control point. We compare quadratic programming (first row), quadratic cost without non-negativity constraints (second row), quadratic cost without non-negativity constraints and with linear precision on edges (third row) and ℓ_1 cost (fourth row). The first column corresponds to the weights on the diagonal, the second column to the weights on the edge to the left of the control point, the third column to those on the edge to the right of the control point and the fourth column to those on an edge that lies away from the control point.

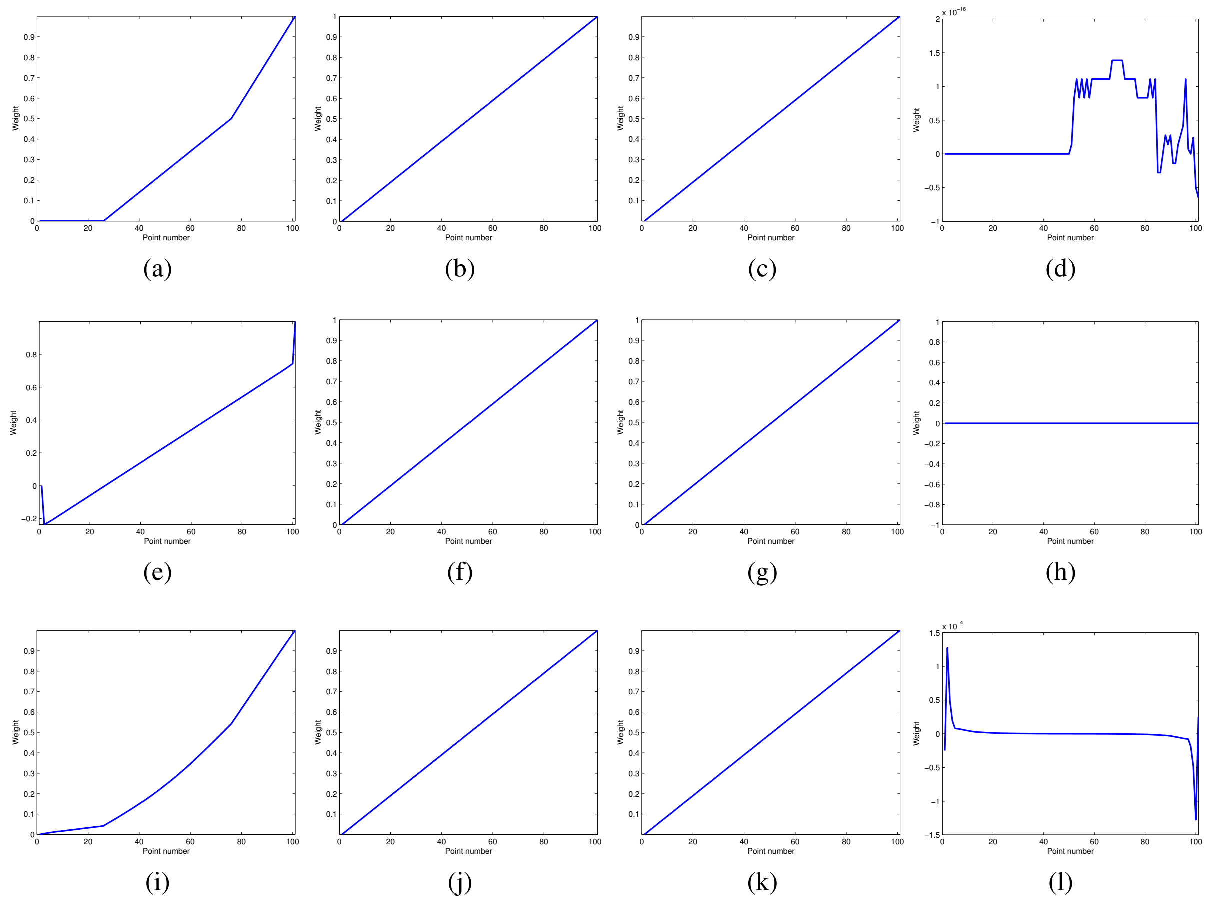
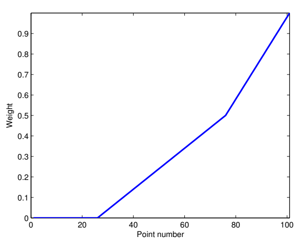
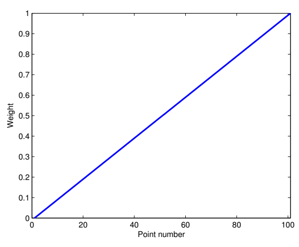


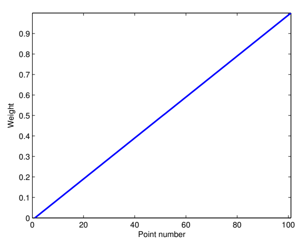
Figure 4.5: Variation of optimized weights on the diagonal and edges of a 101×101 grid with four control points. Weights were initialized with heat diffusion with impulse initialization. The control point at the lower left corner of the grid was chosen to create the plots. The horizontal axis of the plots shows the point number on the diagonal or edge, starting from the point farthest from the examined control point. We compare quadratic programming (first row), quadratic cost without non-negativity constraints and with linear precision on edges (second row) and ℓ_1 cost (third row). The first column corresponds to the weights on the diagonal, the second column to the weights on the edge to the left of the control point, the third column to those on the edge to the right of the control point and the fourth column to those on an edge that lies away from the control point.



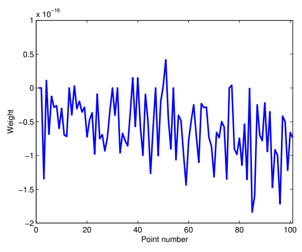
(a)



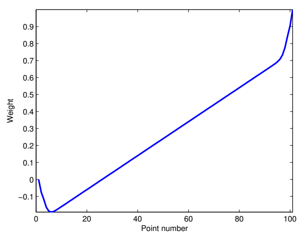
(b)



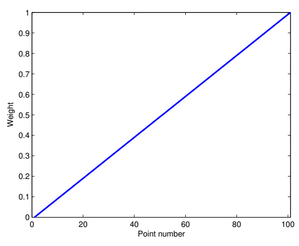
(c)



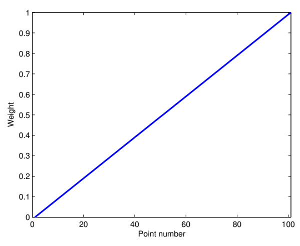
(d)



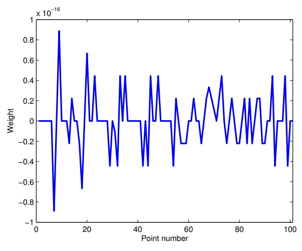
(e)



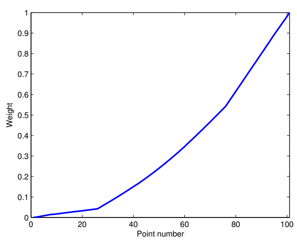
(f)



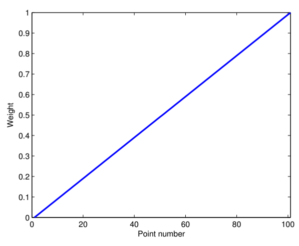
(g)



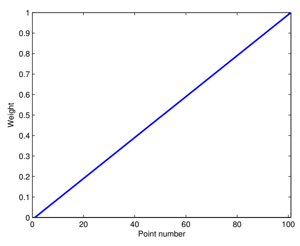
(h)



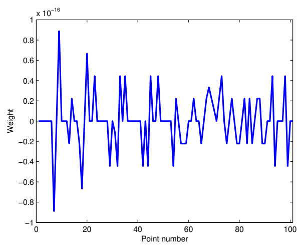
(i)



(j)



(k)



(l)

Figure 4.6: Variation of optimized weights on the diagonal and edges of a 101×101 grid with four control points. Weights were initialized with heat diffusion with fixed linear weights on edges. The control point at the lower left corner of the grid was chosen to create the plots. The horizontal axis of the plots shows the point number on the diagonal or edge, starting from the point farthest from the examined control point. We compare quadratic programming (first row), quadratic cost without non-negativity constraints (second row) and ℓ_1 cost (third row). The first column corresponds to the weights on the diagonal, the second column to the weights on the edge to the left of the control point, the third column to those on the edge to the right of the control point and the fourth column to those on an edge that lies away from the control point.

5 Structure-aware Weights

To simplify the interface for weight design, we examine the possibility of exploiting inherent *structure* of the processed object. Our motivation comes from the well-justified principles of structure-aware shape processing that are consolidated in [12]. Structure-based models pave the way for intuitiveness and plausibility of the edited shape, since they incorporate non-local relations and preserve structure, in contrast to local differential operator methods. More specifically, the authors of [12] argue that if a user is only allowed to process the object at a very low level related to geometric primitives, she is significantly deprived of design freedom. They propose processing at a higher, structural level as a solution to this "content creation bottleneck". In our framework, the user applies local modifications to control weights, which are at a rather low level with respect to the final deformation. Consequently, it is crucial that we provide her with some high-level handles as well, to make interaction more efficient and intuitive.

5.1 Generic Framework

Mitra et al. point out in [12] that most of the approaches to structure-aware shape editing include two main steps:

1. an analysis step, in which structure in the input data is discovered.
2. a processing step, in which the original shape is edited (deformed) to produce new versions.

The analysis step is essential, as the input data are usually not annotated with structure. While identifying structure might be cast to the user manually indicating several parts of the object which are related, it is preferable to automatically recognize structural properties and relieve the user from this burden. In particular, deformation models like cage-based methods focus on single input shapes and use an a priori model to represent relations between parts of the object.

This model encodes constraints on differential properties of deformation functions (e.g. related to smoothness). Therefore, structure can be extracted by fitting this fixed model to the input shape. A key source of structure that drives this kind of analysis is *symmetry*.

The resulting structure is encoded as a constraint which restricts the ways in which the object can deform during the processing step, leading to fewer degrees of freedom in editing and making it easier to achieve satisfactory edits. This is the essence of *structure-aware shape editing*. We refer to structure-aware shape editing that employs deformation models as *structure-aware deformation*.

5.2 Parts, Parameters and Relations

Mitra et al. [12] provide a unifying definition of structure for a given shape. A shape is a collection of distinct parts, each one equipped with a set of parameters, which are coupled by certain relations regarding their arrangement. Structure-aware deformation aims at preservation of abstract relations, like salient features of the shape or specific types of symmetry. To achieve these goals, researchers have introduced adaptivity of the deformation to the content of the shape and established global relations between parts of the object.

For instance, let us consider the iWires approach of [13] to deform man-made objects. The key concept of “wires”, which are crease lines detected on the object’s surface, serves as the metaphor for the parts of object. Salient properties of these wires and relations between them, like co-planarity, are respected during deformation, creating additional constraints that are combined with user constraints to compute the final deformed version of the object.

Therefore, a pivotal step towards integrating structure-aware deformation into the rest of our kernel coordinates pipeline is identifying parts, parameters and relations in the cage-based deformation context. For example, one could argue that each handle of the cage corresponds to a part of the object.

5.3 Lines

In the linear subspace design framework of [2], regions with fully specified affine transformations serve as alternative controls to simple control points that undergo only translations. In that framework, it is relatively easy to model some user-defined structural relations between these regions, which serve as parts. For instance, let us consider the case where the user wants two lines which are parallel in the original shape to remain parallel in all subsequent deformations of the interaction session. After defining the lines, the user can simply indicate that their affine transformations should always be the same, so that when she deforms one of them, the same deformation is automatically applied to the other. This way, the lines will always remain parallel. In fact, we can argue that the structural relations are “hard-coded” in this setting.

A similar approach is not possible in the kernel coordinates framework, since our method does not model hard constraints. Instead, a plausible path to preserving high-level features such as parallelism is via constraining the weights of the deformation. This means effectively that the

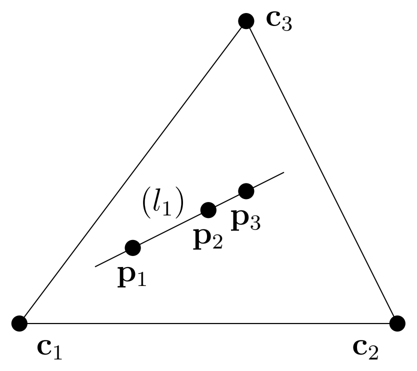


Figure 5.1: A simple cage with three control points and a line (l_1) inside it, with three distinct points.

weights of all points lying on parallel lines become coupled.

At this point, we make some important clarifications regarding the following analysis.

- We examine the final kernel coordinates along lines, not weights at sampling locations. However, the relevant processing is ultimately performed on the latter and the derived relations are "propagated" to the former.
- We assume that control handles undergo pure translations without any rotation, as is usually the case in standard cage-based methods. In case the user adds a rotation to a control handle, then the defined lines become curves as they lack rigidity, and parallelism is no longer relevant.

5.3.1 Preservation of Lines

A necessary condition for lines to remain parallel is that both are deformed into a line. Thus, we begin the analysis of conditions on the weights by demanding exactly the above statement.

Let us consider a cage comprising m control points $\mathbf{c}_i$, $i \in \{1, \ldots, m\}$ and a line (l_1) inside this cage which contains three distinct points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. A two-dimensional instance of this setting is shown in Figure 5.1.

Since all three points lie on the same line, it holds for some $a,\ b \neq 0$ that

$$a\,(\mathbf{p}_2 - \mathbf{p}_1) + b\,(\mathbf{p}_3 - \mathbf{p}_1) = 0. \tag{5.1}$$

The control points undergo arbitrary translations $\mathbf{c}_i{}' = \mathbf{c}_i + \mathbf{t}_i$, which we stack in matrix $\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \ldots & \mathbf{t}_m \end{bmatrix}$. We assume that the weights of all three points satisfy the reproduction property and that the ones for the first two points are given. Our goal is to identify the weights of the third point so that the deformed points $\mathbf{p}_1{}',\ \mathbf{p}_2{}',\ \mathbf{p}_3{}'$ still lie on a single line. This means that there exist some $c,\ d \neq 0$ such that for all $\mathbf{T}$

$$c\,(\mathbf{p}_2{}' - \mathbf{p}_1{}') + d\,(\mathbf{p}_3{}' - \mathbf{p}_1{}') = 0. \tag{5.2}$$

From reproduction, it follows that

$$\mathbf{p}_j{}' = \mathbf{p}_j + \sum_{i=1}^{m} w_i(\mathbf{p}_j)\mathbf{t}_i,\ j = 1,\ 2,\ 3.$$

Using matrix notation, we get

$$\mathbf{p}_2{}' - \mathbf{p}_1{}' = \mathbf{p}_2 - \mathbf{p}_1 + \mathbf{T}(\mathbf{w}(\mathbf{p}_2) - \mathbf{w}(\mathbf{p}_1)) \text{ and} \tag{5.3}$$

$$\mathbf{p}_3{}' - \mathbf{p}_1{}' = \mathbf{p}_3 - \mathbf{p}_1 + \mathbf{T}(\mathbf{w}(\mathbf{p}_3) - \mathbf{w}(\mathbf{p}_1)). \tag{5.4}$$

We choose $c = a$ and $d = b$ to obtain

$$\mathbf{T}(a(\mathbf{w}(\mathbf{p}_2) - \mathbf{w}(\mathbf{p}_1)) + b(\mathbf{w}(\mathbf{p}_3) - \mathbf{w}(\mathbf{p}_1))) = 0,\ \forall \mathbf{T}.$$

Since this equality must hold for all $\mathbf{T}$, the vector multiplying it must be zero, which leads to

$$\mathbf{w}(\mathbf{p}_3) = \mathbf{w}(\mathbf{p}_1) - \frac{a}{b}(\mathbf{w}(\mathbf{p}_2) - \mathbf{w}(\mathbf{p}_1)). \tag{5.5}$$

Reproduction and partition of unity can be easily verified for the derived $\mathbf{w}(\mathbf{p}_3)$, assuming that they hold for $\mathbf{w}(\mathbf{p}_1)$ and $\mathbf{w}(\mathbf{p}_2)$. Equation (5.5) simply indicates that it is sufficient that weights vary linearly across the line in order for it to be preserved under arbitrary translations. Given the weights just at two distinct points on the line, we can compute the weights for every other point based on this linear variation. Let $\mathbf{p}_1,\ \mathbf{p}_2$ be the points where the weights are given; we will call these the *base points* from now on. Then, elaborating on (5.5), we can express the weight of an arbitrary point $\mathbf{p}$ on (l_1) as

$$\mathbf{w}(\mathbf{p}) = \mathbf{w}(\mathbf{p}_1) + \operatorname{sgn}\left((\mathbf{p} - \mathbf{p}_1)^T(\mathbf{p}_2 - \mathbf{p}_1)\right) \|\mathbf{p} - \mathbf{p}_1\| \frac{\mathbf{w}(\mathbf{p}_2) - \mathbf{w}(\mathbf{p}_1)}{\|\mathbf{p}_2 - \mathbf{p}_1\|}. \tag{5.6}$$

We will denote $\mathbf{d}_1 = \frac{\mathbf{w}(\mathbf{p}_2) - \mathbf{w}(\mathbf{p}_1)}{\|\mathbf{p}_2 - \mathbf{p}_1\|}$, which is the vector that determines the linear variation of weights along (l_1). The sufficient condition (5.6) for line preservation ensures that when the line is deformed, a *uniform* scaling is applied to the original spacing of a set of points on it.

5.3.2 Preservation of Parallelism

We use the result of Section 5.3.1 to examine under which conditions two lines that are parallel in the original shape remain parallel under arbitrary translations. An indicative sketch is given in Figure 5.2.

More specifically, let $\mathbf{p}_1,\ \mathbf{p}_2,\ \mathbf{p}_3$ be three distinct points on (l_1) and $\mathbf{p}_4,\ \mathbf{p}_5,\ \mathbf{p}_6$ be three distinct points on (l_2). Suppose that the base points are $\mathbf{p}_1$ and $\mathbf{p}_2$ for (l_1) and $\mathbf{p}_4$ and $\mathbf{p}_5$ for (l_2). Consequently, the expressions for the weights of the remaining points, $\mathbf{p}_3$ for (l_1) and $\mathbf{p}_6$ for (l_2), are

$$\mathbf{w}(\mathbf{p}_3) = \mathbf{w}(\mathbf{p}_1) + \operatorname{sgn}\left((\mathbf{p}_3 - \mathbf{p}_1)^T(\mathbf{p}_2 - \mathbf{p}_1)\right) \|\mathbf{p}_3 - \mathbf{p}_1\|\, \mathbf{d}_1 \text{ and} \tag{5.7}$$

$$\mathbf{w}(\mathbf{p}_6) = \mathbf{w}(\mathbf{p}_4) + \operatorname{sgn}\left((\mathbf{p}_6 - \mathbf{p}_4)^T(\mathbf{p}_5 - \mathbf{p}_4)\right) \|\mathbf{p}_6 - \mathbf{p}_4\|\, \mathbf{d}_2. \tag{5.8}$$

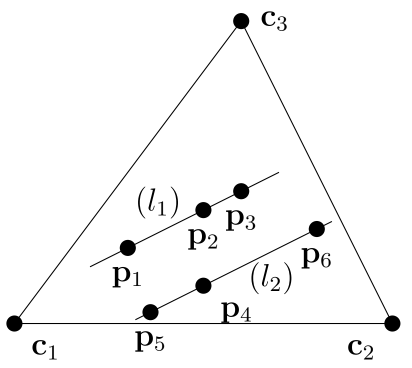


Figure 5.2: A simple cage with three control points and two parallel lines (l_1) and (l_2) inside it.

Since the pairs $(\mathbf{p}_1,\ \mathbf{p}_3)$ and $(\mathbf{p}_4,\ \mathbf{p}_6)$ lie in parallel lines, it holds for some $a,\ b \neq 0$ that

$$a\,(\mathbf{p}_3 - \mathbf{p}_1) + b\,(\mathbf{p}_6 - \mathbf{p}_4) = 0. \tag{5.9}$$

Following the same steps as in the analysis for a single line, we obtain the following sufficient condition for the parallelism to hold for the deformed points as well:

$$a(\mathbf{w}(\mathbf{p}_3) - \mathbf{w}(\mathbf{p}_1)) + b(\mathbf{w}(\mathbf{p}_6) - \mathbf{w}(\mathbf{p}_4)) = 0.$$

Substituting (5.7) and (5.8) and using the fact that

$$\frac{a}{b} = -\operatorname{sgn}\big((\mathbf{p}_3 - \mathbf{p}_1)^T(\mathbf{p}_6 - \mathbf{p}_4)\big)\,\frac{\|\mathbf{p}_6 - \mathbf{p}_4\|}{\|\mathbf{p}_3 - \mathbf{p}_1\|},$$

we arrive at the following relation between $\mathbf{d}_1$ and $\mathbf{d}_2$:

$$\mathbf{d}_2 = \frac{\operatorname{sgn}\big((\mathbf{p}_3 - \mathbf{p}_1)^T(\mathbf{p}_6 - \mathbf{p}_4)\big)\operatorname{sgn}\big((\mathbf{p}_3 - \mathbf{p}_1)^T(\mathbf{p}_2 - \mathbf{p}_1)\big)}{\operatorname{sgn}\left((\mathbf{p}_6 - \mathbf{p}_4)^T(\mathbf{p}_5 - \mathbf{p}_4)\right)}\mathbf{d}_1. \tag{5.10}$$

A careful analysis of the sign functions occurring in (5.10) yields the final result:

$$\mathbf{d}_2 = \operatorname{sgn}\big((\mathbf{p}_2 - \mathbf{p}_1)^T(\mathbf{p}_5 - \mathbf{p}_4)\big)\,\mathbf{d}_1. \tag{5.11}$$

The sign function merely accounts for situations such as the one in Figure 5.2, where the configurations of the base points are not “consistent” between the lines. In words, it suffices that the linear variation of weights along the two lines follows exactly the same pattern in order for the lines to remain parallel under any translation. Moreover, (5.11) implies that the same amount of uniform scaling is applied to both lines, which corresponds intuitively to an *affine* transformation.

5.3.3 Implementation

At interaction time, the user can define line segments on the shape by indicating their start and end point. Regular sampling is then performed on these lines, followed by inverse mean shift sampling of the shape, where samples on edges and lines are kept fixed. The user can also choose to load a sampling distribution for the processed shape from a previous session where she had already defined certain lines.

Having determined the set of sampling points, we run heat diffusion similarly to the scenario where no lines are defined. Next, we use the start and end point of each line segment as its base points and run optimization only for these two samples of each line to get their weights. Let us denote the base points of some line with $\mathbf{x}_s$ and $\mathbf{x}_e$ and their optimized weights with $\mathbf{w}_s$ and $\mathbf{w}_e$ respectively. The weights of all intermediate samples along this line can be calculated directly from (5.6) using linear interpolation independently for each coordinate. This way, kernel coordinates will also assume a linear profile along the line (since they interpolate weights at sampling points), provided that weights of neighboring points near the line do not deviate excessively from those of points on the line. This last condition is enforced by post-processing the optimized weights of these neighboring points (which have been computed agnostically to the presence of the line) so that their values approximate the weights on the line as distance from it decreases.

5.4 Rigidity

Another powerful structural feature whose preservation is very often desirable is rigidity. In the framework of bounded biharmonic weights developed in [1], the notion of rigidity is directly incorporated into optimization as an additional energy term. This term penalizes the squared magnitude of variation (gradient) of the weights over a region Π that should behave as rigid. Moreover, the user can control how strict this rigidity constraint is at each part of Π by painting with a rigidity brush to define a positive weighting function ρ, yielding a term of the form

$$R = \sum_{j=1}^{m} \frac{1}{2} \int_{\Pi} \rho \left\|\nabla w_j\right\|^2 dV \tag{5.12}$$

to be included in the objective. While in [1] the above integral is discretized on a mesh, our method is meshless and hence we can directly compute the gradient appearing in (5.12). Taking into account (3.41) and (3.43), the Jacobian matrix of the weight vector at $\mathbf{x}$ is

$$J\mathbf{w}(\mathbf{x}) = -\frac{1}{\sigma^2}\mathbf{W}\mathbf{K}^{-1} \operatorname{diag}\left(\mathbf{k}(\mathbf{x})\right)\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right). \tag{5.13}$$

In terms of this Jacobian, the rigidity term R is just the integral of its squared Frobenius norm over Π weighted by ρ. Denoting the j-th row of $\mathbf{W}$ by $\mathbf{w}_j{}^T$ and making use of the Hadamard product, this can be expressed as

$$\begin{aligned} R &= \frac{1}{2\sigma^4} \sum_{j=1}^{m} \int_{\Pi} \rho \left\| \left(\mathbf{w}_j{}^T \mathbf{K}^{-1} \operatorname{diag}\left(\mathbf{k}(\mathbf{x})\right)\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right)\right)^T \right\|^2 d\mathbf{x} \\ &= \frac{1}{2\sigma^4} \sum_{j=1}^{m} \int_{\Pi} \rho\, \mathbf{w}_j{}^T \mathbf{K}^{-1} \left[\left(\mathbf{k}(\mathbf{x})\mathbf{k}(\mathbf{x})^T\right) \odot \left(\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right)\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T\right)^T \right) \right] \mathbf{K}^{-1}\mathbf{w}_j \, d\mathbf{x}. \end{aligned} \tag{5.14}$$

The form of (5.14) expresses the rigidity energy for kernel coordinates in the general case. Assuming a soft rigidity brush, the user may have created a mask of arbitrary form, so we cannot directly express R in terms of the user-controlled weights $\mathbf{W}$. A simplification is applicable if we assume a constant mask over the whole region. In this particular case, due to linearity of matrix multiplication and the integral operator, we get

$$R = \frac{\rho}{2\sigma^4} \sum_{j=1}^{m} \mathbf{w}_j{}^T \mathbf{K}^{-1} \left(\int_{\Pi} \left(\mathbf{k}(\mathbf{x})\mathbf{k}(\mathbf{x})^T \right) \odot \left(\left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T \right) \left(\mathbf{1}\mathbf{x}^T - \mathbf{X}^T \right)^T \right) d\mathbf{x} \right) \mathbf{K}^{-1} \mathbf{w}_j. \tag{5.15}$$

We denote the integral in (5.15), which constitutes an $n \times n$ matrix, by $\mathbf{P}$. In general, the value of this integral depends on the particular shape of region Π, which is controlled by the user. We estimate it using the same sampling as the one for kernel coordinates computation, by calculating the sum of the integrand values over all the points of the sampling distribution that lie inside Π. For regular sampling, this can be expressed as

$$\mathbf{P} \approx T^d \sum_{s:\, \mathbf{x}_s \in \Pi} \left(\mathbf{k}(\mathbf{x}_s)\mathbf{k}(\mathbf{x}_s)^T \right) \odot \left(\left(\mathbf{1}\mathbf{x}_s{}^T - \mathbf{X}^T \right) \left(\mathbf{1}\mathbf{x}_s{}^T - \mathbf{X}^T \right)^T \right). \tag{5.16}$$

For blue noise sampling, factor T^d in (5.16) is replaced by $|\Pi|/q$, where q is the total number of points that lie in Π. This implies that we need an estimate of the volume of Π in order to calculate $\mathbf{P}$. However, when the sampling distribution is regular enough like in our pipeline, the above factor is approximately the same for any chosen Π, which allows us to use only the sum in (5.16) as an estimate of $\mathbf{P}$ and adjust parameter ρ accordingly, so that the weight of the rigidity term remains the same. Once $\mathbf{P}$ has been computed, we can also compute $\mathbf{L} = \mathbf{K}^{-1}\mathbf{P}\mathbf{K}^{-1}$ and the rigidity energy in terms of $\mathbf{L}$ is simply

$$R = \frac{\rho}{2\sigma^4} \sum_{j=1}^{m} \mathbf{w}_j{}^T \mathbf{L} \mathbf{w}_j. \tag{5.17}$$

We can write the sum of quadratic forms of $\mathbf{L}$ in (5.17) as a single quadratic form, using vector

$$\mathbf{w}_{\text{full}} = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{bmatrix},$$

which stacks all weight vectors of individual points into a single vector of nm elements. In particular, rigidity energy becomes

$$R = \frac{\rho}{2\sigma^4} \mathbf{w}_{\text{full}}{}^T \mathbf{L}_{\text{full}} \mathbf{w}_{\text{full}}, \tag{5.18}$$

where $\mathbf{L}_{\text{full}}$ is a sparse $(nm) \times (nm)$ matrix (with non-zero elements originating from $\mathbf{L}$) of the form

$$\mathbf{L}_{\text{full}} = \begin{bmatrix} l_{11}\mathbf{I}_m & \cdots & l_{1n}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ l_{n1}\mathbf{I}_m & \cdots & l_{nn}\mathbf{I}_m \end{bmatrix}. \tag{5.19}$$

The sparsity pattern of $\mathbf{L}_{\text{full}}$ indicates that optimization of user-driven weights is globally coupled and cannot be performed individually for the points lying in Π.

Nevertheless, we can exploit the sparsity of $\mathbf{P}$ to decouple optimization by performing a suitable change of variables. More specifically, the form of the left factor of the Hadamard product in (5.16) indicates that p_{kl} is practically zero if either $\mathbf{x}_k$ or $\mathbf{x}_l$ have larger distance from Π than a certain threshold θ. Let $\mathcal{N} = \{i : d(\mathbf{x}_i, \Pi) \leq \theta\}$ denote the index set corresponding to points that are closer to Π than this threshold, with cardinality r. We use this index set to select the corresponding rows and columns of $\mathbf{P}$ and obtain the $r \times r$ submatrix

$$\tilde{\mathbf{P}} = \mathbf{P}_{\mathcal{N}\mathcal{N}} \tag{5.20}$$

which contains all non-zero elements of $\mathbf{P}$. We will generally use the tilde accent to indicate the part of a matrix that pertains to sample points close enough to Π. The remaining factors of the quadratic forms in (5.17) can be grouped by setting

$$\mathbf{u}_j = \mathbf{K}^{-1}\mathbf{w}_j,\ j = 1, \ldots, m. \tag{5.21}$$

This can also be expressed in matrix form as

$$\mathbf{U} = \mathbf{W}\mathbf{K}^{-1} \Rightarrow \mathbf{W} = \mathbf{U}\mathbf{K}. \tag{5.22}$$

Using $\mathcal{N}$ to select columns from $\mathbf{U}$, we obtain the $m \times r$ matrix

$$\tilde{\mathbf{U}} = \mathbf{U}_{:,\mathcal{N}}. \tag{5.23}$$

We stack all columns of $\tilde{\mathbf{U}}$ into a vector with rm elements:

$$\tilde{\mathbf{u}} = \operatorname{vec}(\tilde{\mathbf{U}}). \tag{5.24}$$

Exploiting the fact that the only non-zero elements of $\mathbf{P}$ are in $\tilde{\mathbf{P}}$, we write the rigidity energy only in terms of $\tilde{\mathbf{u}}$ as

$$R(\tilde{\mathbf{u}}) = \frac{\rho}{2\sigma^4}\sum_{j=1}^{m} \mathbf{u}_j{}^T\mathbf{P}\mathbf{u}_j = \frac{\rho}{2\sigma^4}\tilde{\mathbf{u}}^T \begin{bmatrix} \tilde{p}_{11}\mathbf{I}_m & \cdots & \tilde{p}_{1r}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{p}_{r1}\mathbf{I}_m & \cdots & \tilde{p}_{rr}\mathbf{I}_m \end{bmatrix} \tilde{\mathbf{u}}. \tag{5.25}$$

For brevity, from now on we will write

$$\mathbf{B} = \frac{\rho}{2\sigma^4} \begin{bmatrix} \tilde{p}_{11}\mathbf{I}_m & \cdots & \tilde{p}_{1r}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{p}_{r1}\mathbf{I}_m & \cdots & \tilde{p}_{rr}\mathbf{I}_m \end{bmatrix}. \tag{5.26}$$

The above form reveals the decoupling of weights of points farther from Π than θ from the rigidity energy, so that we have to optimize jointly only for r points, where r is generally much smaller than n.

In practice, in order for weights in a rigid part of the object to be in congruence with weights at surrounding points, we follow a slightly modified approach. Let us define two extra index sets,

$$\mathcal{I} = \{i : \mathbf{x}_i \in \Pi\} \tag{5.27}$$

and

$$\mathcal{O} = \mathcal{N} \setminus \mathcal{I}. \tag{5.28}$$

These sets correspond to points inside Π and outside but near Π respectively, and it holds that $\mathcal{N} = \mathcal{I} \cup \mathcal{O}$. Without loss of generality, we assume that every element of $\mathcal{I}$ is smaller than every element of $\mathcal{O}$, so that it is valid to write

$$\tilde{\mathbf{P}} = \begin{bmatrix} \mathbf{P}_{\mathcal{I}\mathcal{I}} & \mathbf{P}_{\mathcal{I}\mathcal{O}} \\ \mathbf{P}_{\mathcal{O}\mathcal{I}} & \mathbf{P}_{\mathcal{O}\mathcal{O}} \end{bmatrix}, \tag{5.29}$$

$$\tilde{\mathbf{u}} = \begin{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} \\ \tilde{\mathbf{u}}_{\mathcal{O}} \end{bmatrix}, \tag{5.30}$$

and, with a slight abuse of notation,

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{\mathcal{I}\mathcal{I}} & \mathbf{B}_{\mathcal{I}\mathcal{O}} \\ \mathbf{B}_{\mathcal{O}\mathcal{I}} & \mathbf{B}_{\mathcal{O}\mathcal{O}} \end{bmatrix}. \tag{5.31}$$

The method we apply to modify weights towards preservation of rigidity is summarized in the following steps:

1. We optimize individually for all points of the shape with quadratic programming to obtain matrix $\mathbf{W}$ and compute the "decoupled" weights $\tilde{\mathbf{u}}$ for points inside and near Π using (5.22), (5.23) and (5.24).
2. We optimize the rigidity term (5.25) *only* for weights of points *inside* Π, $\tilde{\mathbf{u}}_{\mathcal{I}}$, while keeping weights $\tilde{\mathbf{u}}_{\mathcal{O}}$ of points outside Π fixed. This way, we force weights inside Π to stay close to the surrounding pattern.
3. The original representation of weights with matrix $\mathbf{W}$ is recovered through (5.22), using the updated values of $\mathbf{U}$ for points in Π.

Regarding the constraints for optimization of the rigidity term, the reproduction constraints (4.1b) for each point $\mathbf{x}_j,\ j \in \mathcal{I}$ can be reformulated using (5.22) and exploiting the sparsity pattern of $\mathbf{K}$. Let us denote $\tilde{\mathbf{X}} = \mathbf{X}_{:,\mathcal{N}}$, $\tilde{\mathbf{x}} = \mathrm{vec}(\tilde{\mathbf{X}}) = \begin{bmatrix} \tilde{\mathbf{x}}_{\mathcal{I}}^T & \tilde{\mathbf{x}}_{\mathcal{O}}^T \end{bmatrix}^T$ and $\tilde{\mathbf{K}} = \mathbf{K}_{\mathcal{N}\mathcal{N}}$. Then, stacking constraints for all points gives

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{C} & \cdots & \tilde{k}_{1r}\mathbf{C} \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{C} & \cdots & \tilde{k}_{qr}\mathbf{C} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} \\ \tilde{\mathbf{u}}_{\mathcal{O}} \end{bmatrix} = \tilde{\mathbf{x}}_{\mathcal{I}} \Rightarrow$$

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{C} & \cdots & \tilde{k}_{1q}\mathbf{C} \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{C} & \cdots & \tilde{k}_{qq}\mathbf{C} \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} = \tilde{\mathbf{x}}_{\mathcal{I}} - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{C} & \cdots & \tilde{k}_{1r}\mathbf{C} \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{C} & \cdots & \tilde{k}_{qr}\mathbf{C} \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}. \tag{5.32}$$

Similarly, the partition of unity constraints (4.1c) can be expressed collectively as

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{1}^T & \cdots & \tilde{k}_{1q}\mathbf{1}^T \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{1}^T & \cdots & \tilde{k}_{qq}\mathbf{1}^T \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} = \mathbf{1} - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{1}^T & \cdots & \tilde{k}_{1r}\mathbf{1}^T \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{1}^T & \cdots & \tilde{k}_{qr}\mathbf{1}^T \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}. \tag{5.33}$$

Finally, the non-negativity constraints (4.1d) for each point can be brought into the following, joint format:

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{I}_m & \cdots & \tilde{k}_{1q}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{I}_m & \cdots & \tilde{k}_{qq}\mathbf{I}_m \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} \geq - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{I}_m & \cdots & \tilde{k}_{1r}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{I}_m & \cdots & \tilde{k}_{qr}\mathbf{I}_m \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}. \tag{5.34}$$

To sum up, a single quadratic program is solved to obtain the modified weights for all points affecting Π:

$$\min_{\tilde{\mathbf{u}}_{\mathcal{I}}} \left\{ \tilde{\mathbf{u}}_{\mathcal{I}}^T \mathbf{B}_{\mathcal{I}\mathcal{I}} \tilde{\mathbf{u}}_{\mathcal{I}} - 2\tilde{\mathbf{u}}_{\mathcal{O}}^T \mathbf{B}_{\mathcal{O}\mathcal{I}} \tilde{\mathbf{u}}_{\mathcal{I}} \right\} \tag{5.35a}$$

$$\text{s.t.} \quad \begin{bmatrix} \tilde{k}_{11}\mathbf{C} & \cdots & \tilde{k}_{1q}\mathbf{C} \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{C} & \cdots & \tilde{k}_{qq}\mathbf{C} \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} = \tilde{\mathbf{x}}_{\mathcal{I}} - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{C} & \cdots & \tilde{k}_{1r}\mathbf{C} \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{C} & \cdots & \tilde{k}_{qr}\mathbf{C} \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}, \tag{5.35b}$$

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{1}^T & \cdots & \tilde{k}_{1q}\mathbf{1}^T \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{1}^T & \cdots & \tilde{k}_{qq}\mathbf{1}^T \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} = \mathbf{1} - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{1}^T & \cdots & \tilde{k}_{1r}\mathbf{1}^T \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{1}^T & \cdots & \tilde{k}_{qr}\mathbf{1}^T \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}, \tag{5.35c}$$

$$\begin{bmatrix} \tilde{k}_{11}\mathbf{I}_m & \cdots & \tilde{k}_{1q}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q1}\mathbf{I}_m & \cdots & \tilde{k}_{qq}\mathbf{I}_m \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{I}} \geq - \begin{bmatrix} \tilde{k}_{1(q+1)}\mathbf{I}_m & \cdots & \tilde{k}_{1r}\mathbf{I}_m \\ \vdots & \ddots & \vdots \\ \tilde{k}_{q(q+1)}\mathbf{I}_m & \cdots & \tilde{k}_{qr}\mathbf{I}_m \end{bmatrix} \tilde{\mathbf{u}}_{\mathcal{O}}. \tag{5.35d}$$

6 Results

6.1 Implementation Details

After optimizing for weights at sample points that are contained in the $m \times n$ matrix $\mathbf{W}$, kernel coordinates are evaluated at a set of query points $\mathbf{x}$, according to (3.40). These query points can be, for instance, all pixels of an image that is deformed, and their number is in general much greater than the number of sample points, n. Our goal is to determine a fast method to implement (3.40), avoiding direct inversion of $n \times n$ matrix $\mathbf{K}$, which is prohibitive when n grows large for detailed shapes.

We begin with the observation that $\mathbf{K}$ is inherently sparse, with few elements of non-negligible magnitude, corresponding to pairs of sample points that are close to each other. To implement (3.40), we set all elements k_{ij} for which $d(\mathbf{x}_i, \mathbf{x}_j) > \theta = 5\sigma$ to zero, so that $\mathbf{K}$ admits a sparse representation. We perform a Cholesky decomposition of this "sparsified" version of $\mathbf{K}$, exploiting its symmetry and positive definiteness:

$$\mathbf{K} = \mathbf{R}^T\mathbf{R}, \tag{6.1}$$

where $\mathbf{R}$ is upper triangular. Computing the Cholesky decomposition is fast due to sparsity of $\mathbf{K}$. We use $\mathbf{R}$ to solve m linear systems

$$\mathbf{U}\mathbf{K} = \mathbf{W} \tag{6.2}$$

with respect to the unknowns in the $m \times n$ matrix $\mathbf{U}$. This is done fast in two steps that include only one forward and one back substitution, thanks to the triangular structure of $\mathbf{R}$. Regarding kernel functions $\mathbf{k}(\mathbf{x})$, we use the same sparse representation as for $\mathbf{K}$ to reduce memory requirements and speed up the final multiplication

$$\mathbf{w}(\mathbf{x}) = \mathbf{U}\mathbf{k}(\mathbf{x}), \tag{6.3}$$

enabling inexpensive evaluation of kernel coordinates.

6.2 Evaluation of Kernel Coordinates on 2D Shapes

We combine the results of Chapters 3 and 4 to compute the blending weights for simple 2D shapes and compare our method to previous approaches.

6.2.1 Star Shape

In [3], Zhang et al. use a five-pointed star with ten control points to manifest the behavior of their local barycentric coordinate scheme. We sample this shape with a regular grid that extends a little beyond the star in every direction, so that computation of kernel coordinates is precise inside the shape. Note that we did not construct the grid in a way that the edges of the star are sampled by it. We run heat diffusion with impulse initialization and then perform optimization to get $\mathbf{W}$. Finally, we compute $\mathbf{w}(\mathbf{x})$ following the pipeline described in Section 6.1.

We present the blending weights for two representative control points, which are the same as in [3]. The first one lies at a convex vertex of the star, while the second lies at a concave vertex, which is in the convex hull of the rest of the control points. The weights are shown in linear scale for all optimization methods in Figure 6.1 and in logarithmic scale for quadratic programming and ℓ_1 optimization in Figure 6.2. With respect to Figure 6.2, we must point out that all weights of value smaller or equal to 10^{-5} (therefore negative weights as well) are mapped to the lower end of the colorscale.

Table 6.1 summarizes the properties that are satisfied (up to a certain precision) with each optimization method on the star, using grid sampling. We note that for QP and ℓ_1 optimization, the Lagrange property does hold for convex vertices. Furthermore, blending weights for QP assume negative values in areas that are far from the respective control point. However, the magnitude of these negative weights remains quite small (in the order of 10^{-3}). ℓ_1 weights are practically non-negative throughout the entire region of the star, since only 9 evaluation points have at least one negative weight. All these points lie at some convex vertex of the star and their negative weights correspond to control points at either of the "opposite" convex vertices. As far as QEC is concerned, the fact that grid sampling is used hinders linear precision, because there are no sample points exactly on the edges which could be explicitly forced to have linear weights.

In the QEC case, we observe that the weight of the concave vertex (Figure 6.1(d)) becomes larger at the close convex vertices than it is at the control point itself, which means that locality is lost. Comparing QP and ℓ_1 optimization, the former produces slightly more local weights, taking into account the logarithmic plots of Figure 6.2. Finally, QEC is the fastest method to optimize $\mathbf{W}$ on the star, followed by ℓ_1 optimization.

A different approach to sampling the star is via blue noise. This way, it is possible to sample the edges of the shape regularly and then use an inverse mean shift approach with a Gaussian kernel to distribute points uniformly across the interior. In particular, edges of the shape are sampled so that the sampling period is approximately equal for distinct edges. Sample points in the interior are initialized randomly and their positions are iteratively updated by taking gradient descent steps, while points on edges are kept fixed across all iterations. We use superscript (t) to denote the position of a sample point at the t-th iteration of this algorithm. The inverse mean

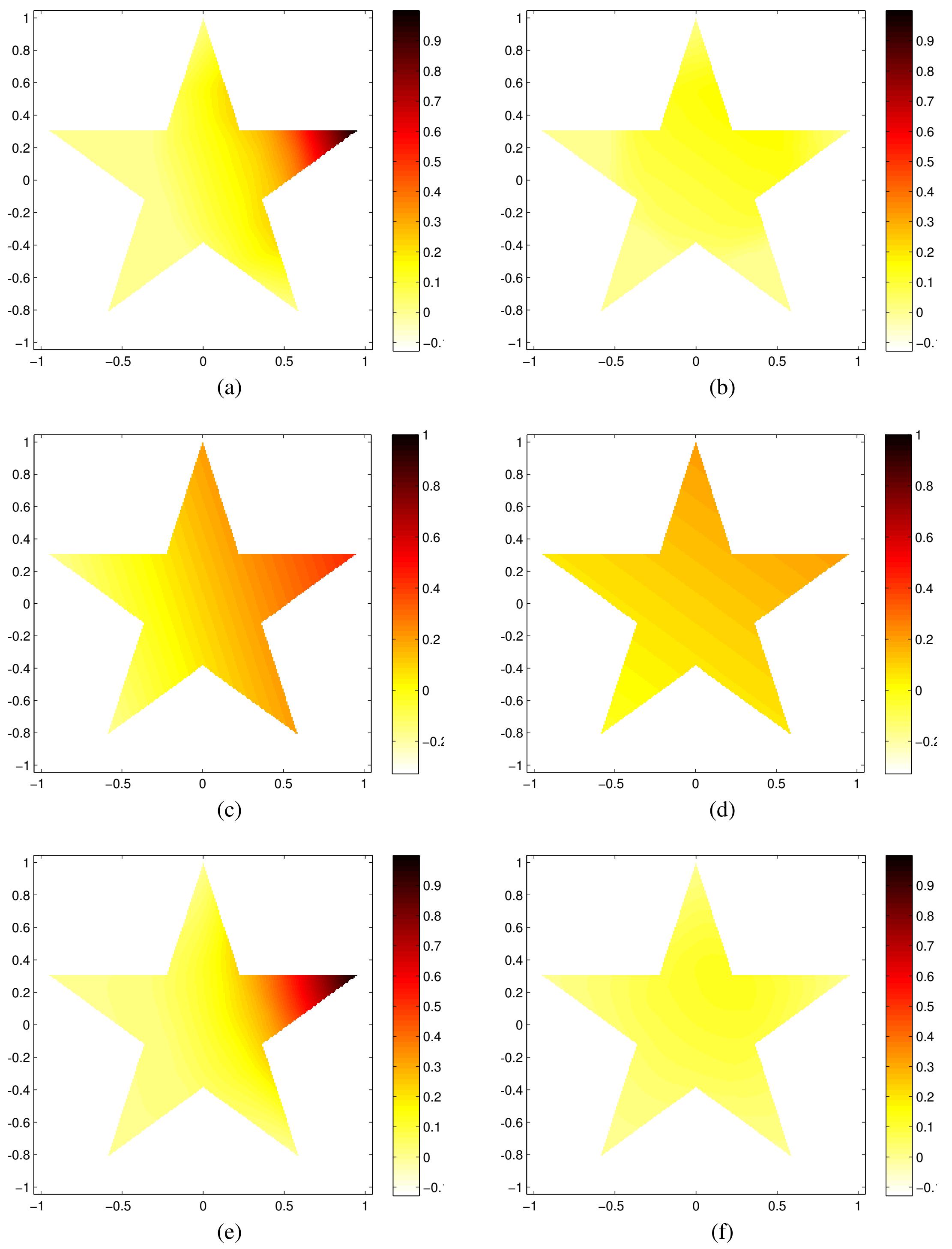
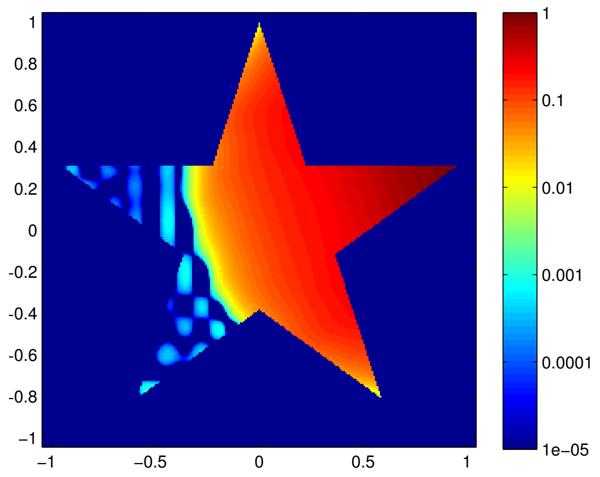
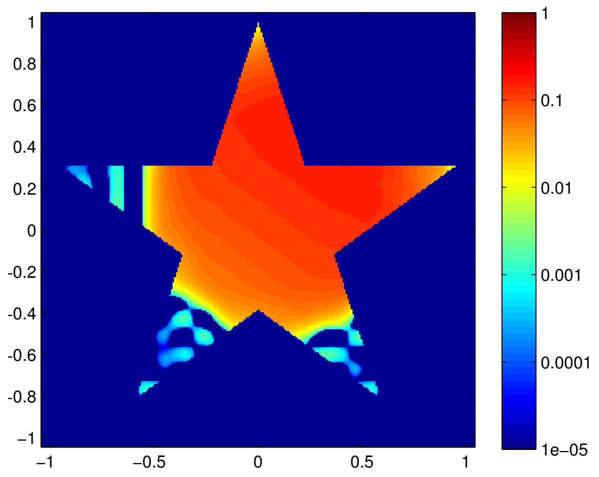


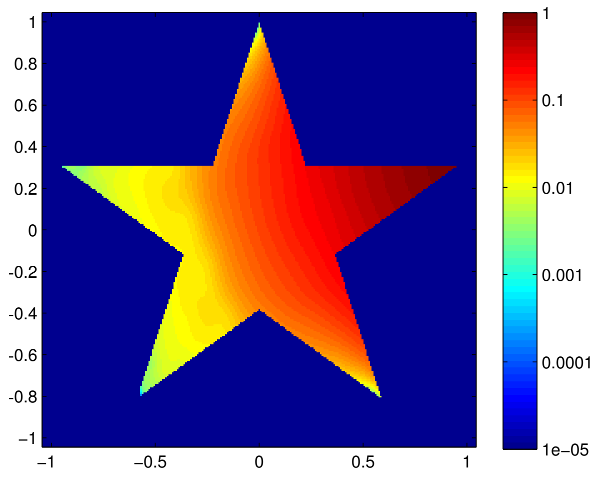
Figure 6.1: Blending weights for a five-pointed star with ten control points sampled with a regular grid. Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with impulse initialization is used for weight initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).



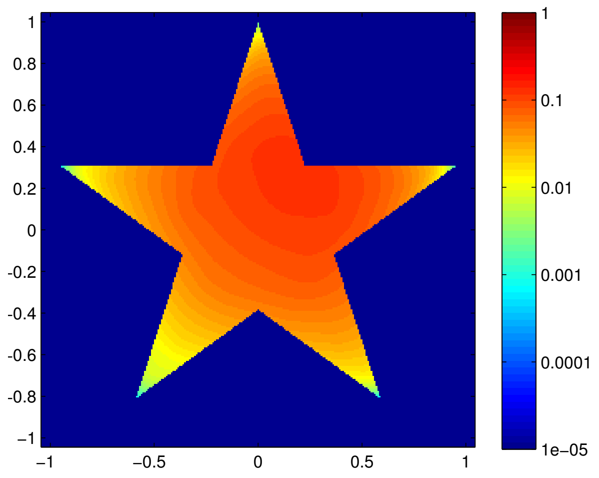
(a)



(b)



(c)



(d)

Figure 6.2: Blending weights in logarithmic scale for a five-pointed star with ten control points sampled with a regular grid. Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with impulse initialization is used for weight initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row) and ℓ_1 optimization (second row).

Properties	**QP**	**QEC**	ℓ_1
Reproduction	Yes	Yes	Yes
Partition of unity	Yes	Yes	Yes
Lagrange property	No	No	No
Linear precision on edges	No	No	No
Non-negativity	No	No	Yes

Table 6.1: Properties of weights derived with different optimization methods for $\mathbf{W}$ and grid sampling of the star. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

shift iteration for an individual point in the interior of the shape is

$$\mathbf{x}_k^{(t)} = \mathbf{x}_k^{(t-1)} + \lambda \sum_{j=1}^{n} \frac{\mathbf{x}_k^{(t-1)} - \mathbf{x}_j^{(t-1)}}{s^2} G_s(\mathbf{x}_k^{(t-1)} - \mathbf{x}_j^{(t-1)}), \tag{6.4}$$

where λ is a constant step size that is chosen empirically. We also underline that in general, the scale s of the Gaussian which is used in (6.4) is different than the scale σ of the Gaussian kernel used to compute the coordinate functions in (3.6).

We used 80 points to sample the edges of the star and afterwards initialized 130 points near its center, as shown in Figure 6.3(a). We then ran inverse mean shift according to (6.4), using a step size of $\lambda = 4 \cdot 10^{-4}$ for the updates and $s = 0.03$ for the Gaussian kernel. After 4000 iterations, the interior points were stabilized at the configuration of Figure 6.3(b).

Using the distribution of Figure 6.3(b), we repeat the same procedure as in the previous grid sampling setting to calculate the blending weights $\mathbf{w}(\mathbf{x})$. We set $\sigma = 0.13$ for the Gaussian kernel. The weights for the two representative control points are presented in Figure 6.4 (linear scale) and Figure 6.5 (logarithmic scale).

Table 6.2 summarizes the properties that are satisfied (up to a certain precision) with each optimization method on the star, using blue noise sampling. Reproduction and partition of unity are satisfied with all methods, but the respective precision is rather low. We note that for QP and ℓ_1 optimization, the Lagrange property does hold for convex vertices. Furthermore, blending weights for QP assume negative values in areas that are far from the respective control point. However, the magnitude of these negative weights remains quite small (in the order of 10^{-2}). ℓ_1 weights are practically non-negative throughout the entire region of the star.

Weights for QEC exhibit artifacts (Figures 6.4(c)–(d)), apparently due to enforcement of linear precision on edges without appropriate modification of $\mathbf{W}$ for points close to an edge. Comparing QP and ℓ_1 optimization, the former produces slightly more local weights, taking into account the logarithmic plots of Figure 6.5. With respect to speed, the ranking of the 3 methods is the same as for grid sampling.

The inadequate accuracy in reproduction and partition of unity with the previous blue noise sampling method is related to the fact that we only sample the interior of the star, which results

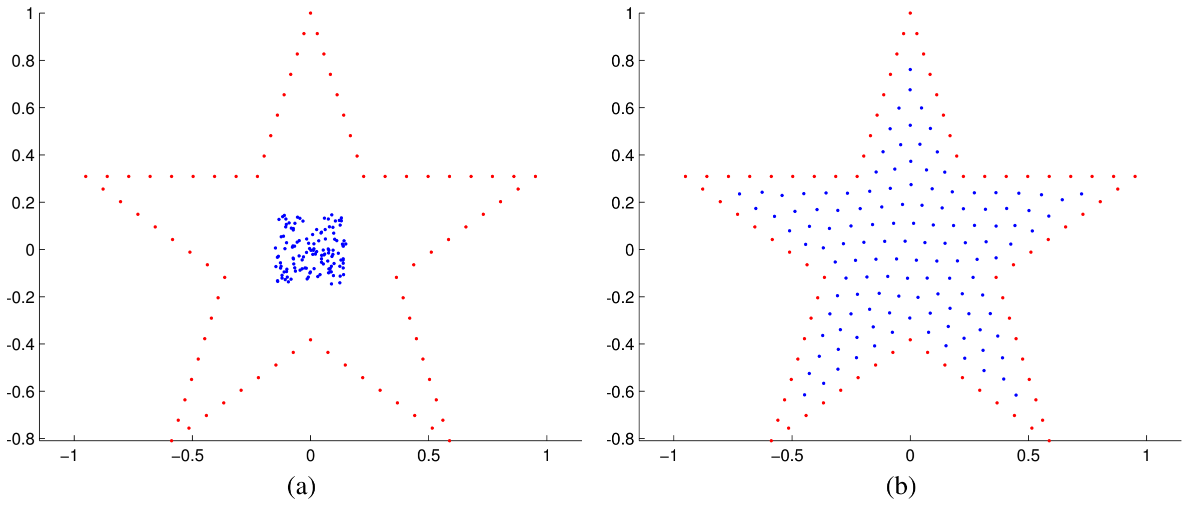


Figure 6.3: Blue noise sampling of the star shape. Sample points on edges are shown in red and those in the interior of the star are shown in blue. (a) depicts the initial configuration of sample points. (b) shows the final distribution of points after 4000 inverse mean shift iterations.

Properties	**QP**	**QEC**	ℓ_1
Reproduction	Yes	Yes	Yes
Partition of unity	Yes	Yes	Yes
Lagrange property	No	Yes	No
Linear precision on edges	No	Yes	No
Non-negativity	No	No	Yes

Table 6.2: Properties of weights derived with different optimization methods for $\mathbf{W}$ and blue noise sampling of the star. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

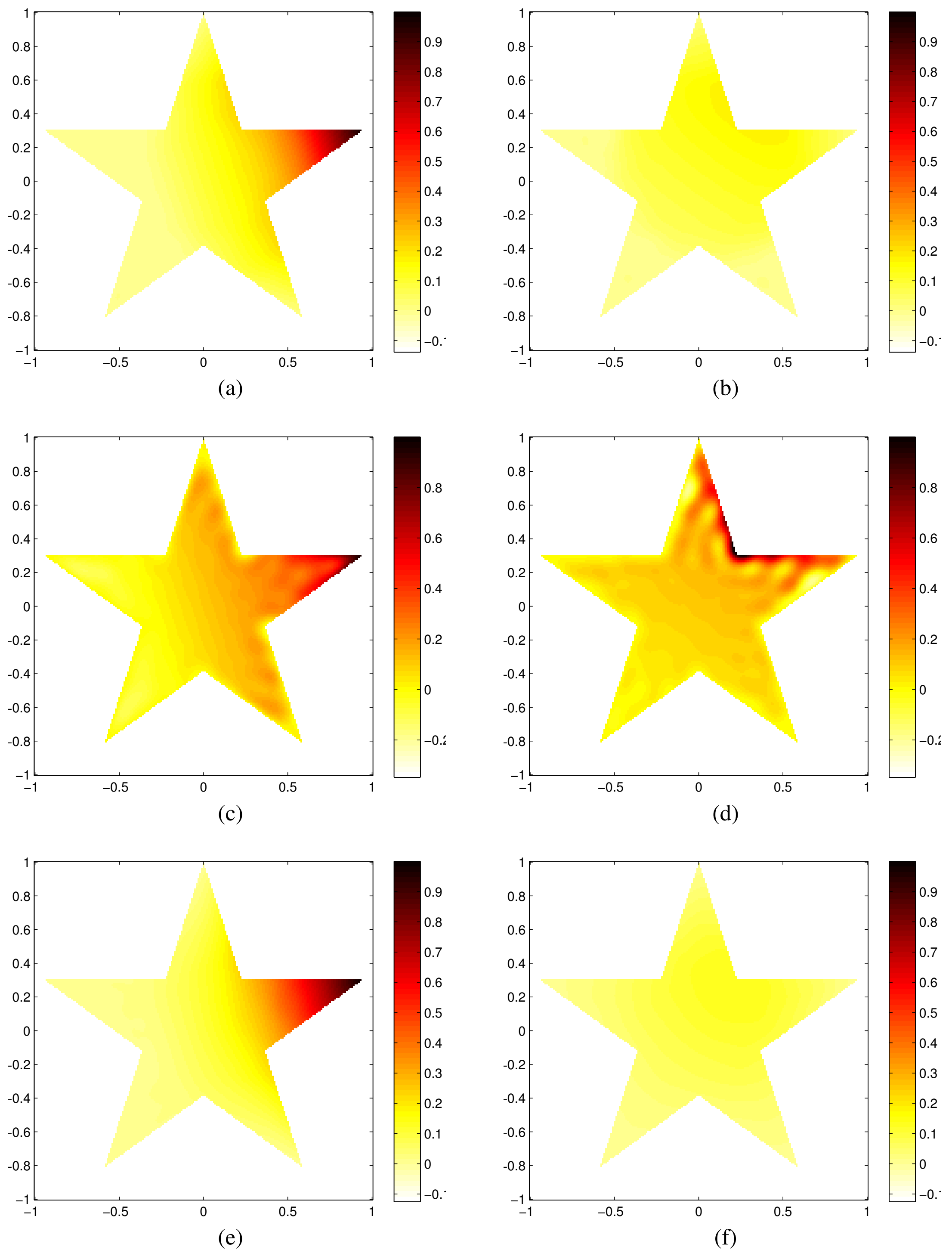


Figure 6.4: Blending weights for a five-pointed star with ten control points sampled with blue noise. Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with impulse initialization is used for weight initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).

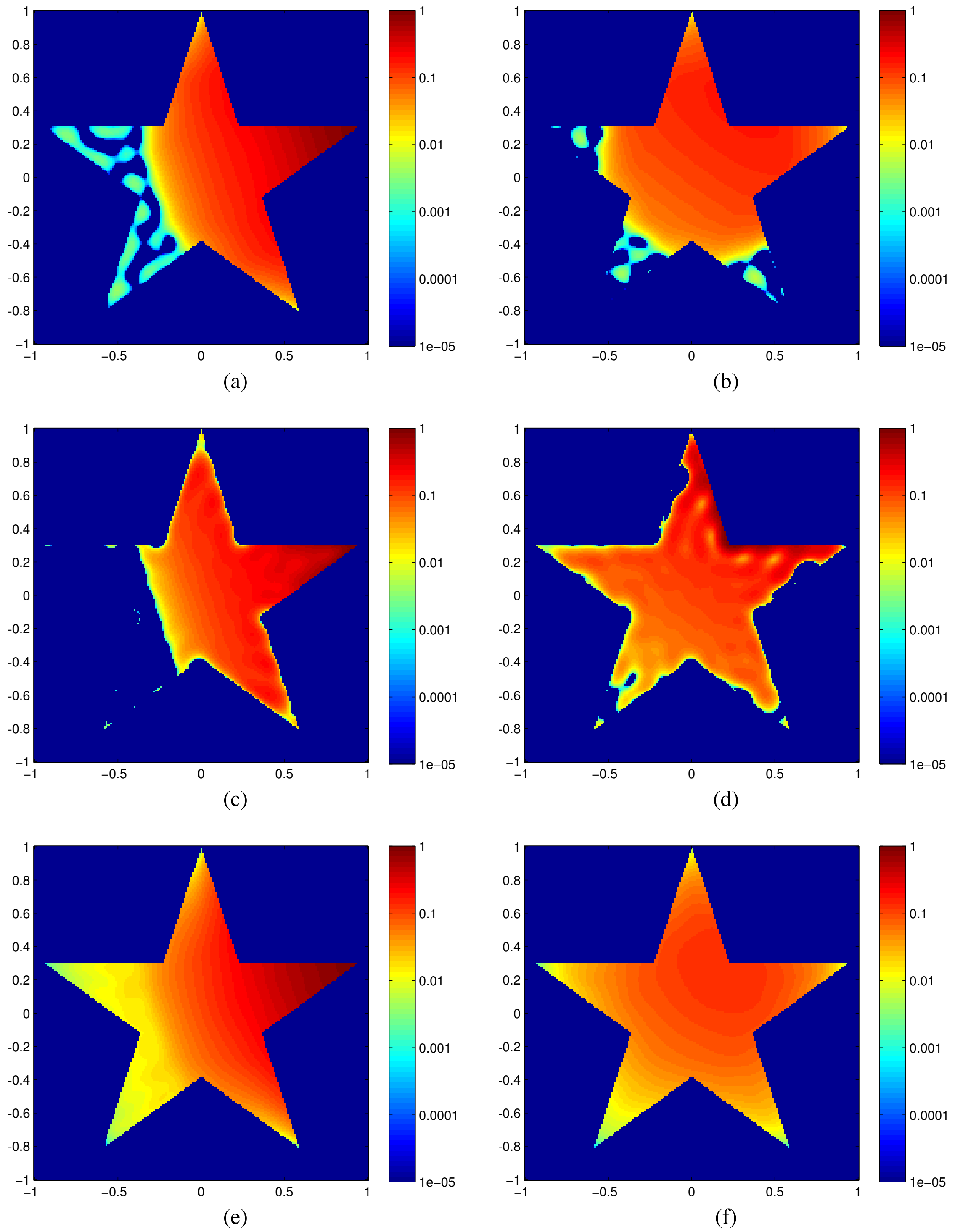


Figure 6.5: Blending weights in logarithmic scale for a five-pointed star with ten control points sampled with blue noise. Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with impulse initialization is used for weight initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).

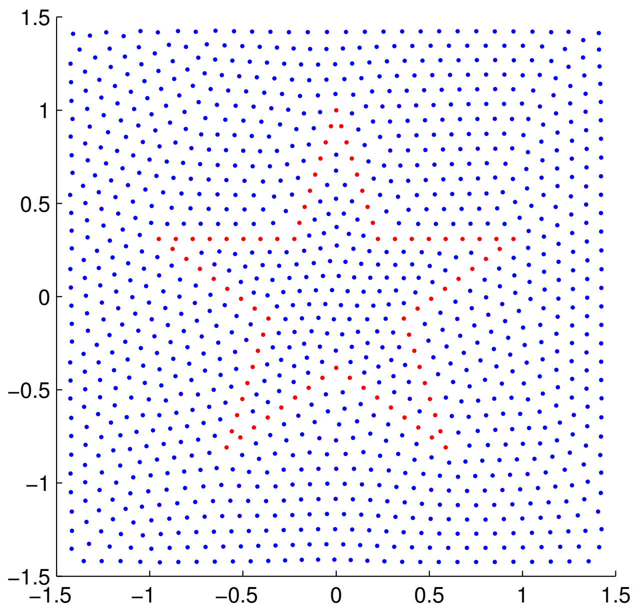


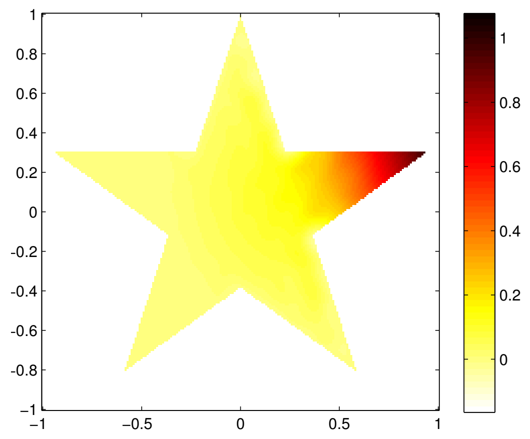
Figure 6.6: Blue noise sampling of the star shape. Sample points on edges are shown in red and those in the interior or the exterior of the star are shown in blue. We show the final distribution of points after 4000 inverse mean shift iterations for interior points and 30000 iterations for exterior points.

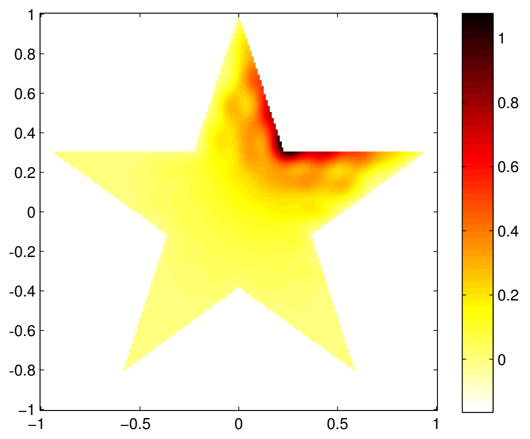
in deviations, mainly at points near the boundary. To solve this issue, we augment our sampling distribution with points from the region that surrounds the star. In particular, we ran inverse mean shift on 915 points which were bounded from inside by the sample points on the edges of the star and from outside by some fixed points regularly distributed on the edges of a square. We used $\lambda = 3 \cdot 10^{-4}$ and $s = 0.025$ for these exterior points and, after 30000 iterations, the algorithm converged to the distribution that is shown in Figure 6.6.

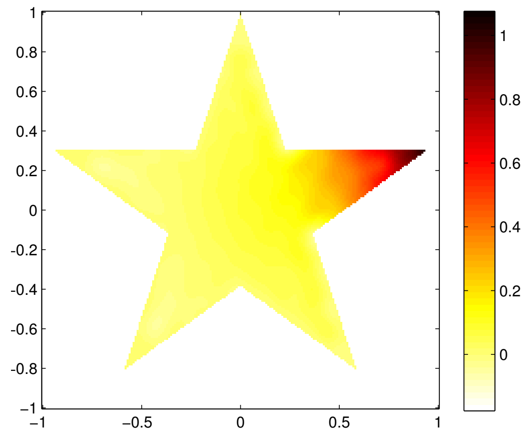
Furthermore, blending weights obtained with the previous two methods generally lack linear precision on edges and do not satisfy the Lagrange property, which is due to the way we compute $\mathbf{W}$. In Chapter 4, we showed that initialization via heat diffusion with fixed linear weights along the edges results in linear $\mathbf{W}$ on the edges. It is therefore reasonable to choose this method in the setting of the star too, in hope of getting linearly precise blending weights. We calculate the blending weights $\mathbf{w}(\mathbf{x})$ using the distribution of Figure 6.6. The weights for the two representative control points are presented in Figure 6.7 (linear scale) and Figure 6.8 (logarithmic scale).

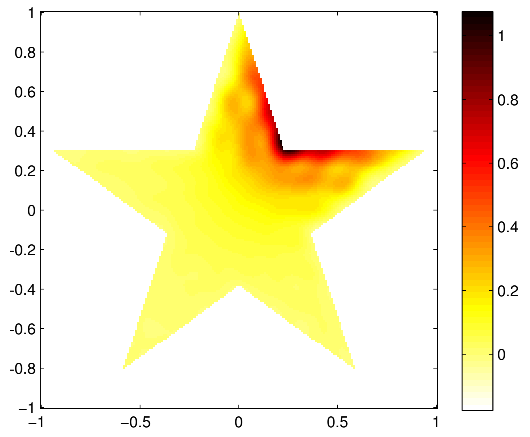
Table 6.3 summarizes the properties that are satisfied (up to a certain precision) with each optimization method on the star when we combine blue noise sampling of the interior *and* exterior of the star with heat diffusion with fixed linear weights on edges. Reproduction, partition of unity and Lagrange property are very accurately satisfied with all methods. While non-negativity is not satisfied with any method, we note that the magnitude of all negative weights is rather small (in the order of 10^{-2}). Notably, no method achieves linear precision on edges, however, we observed that the respective weights are generally close to the ideal values.

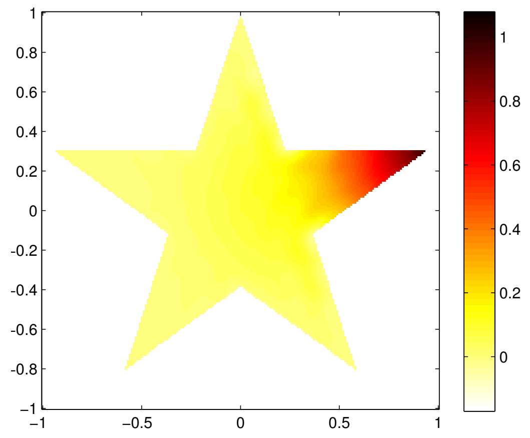
The ripples occurring in the weights with the last method imply local extrema in the interior of the star, which in turn result in unintuitive weights, as pointed out in [3]. This behavior is


(a)


(b)


(c)


(d)


(e)

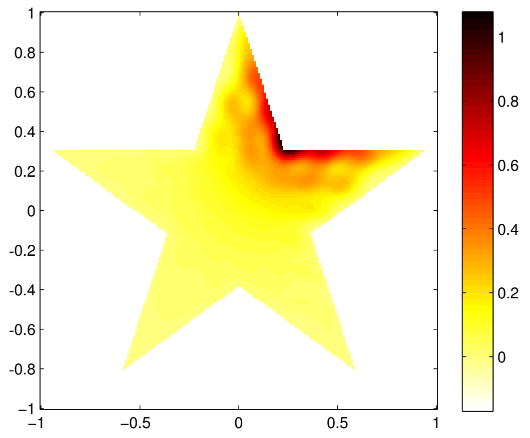
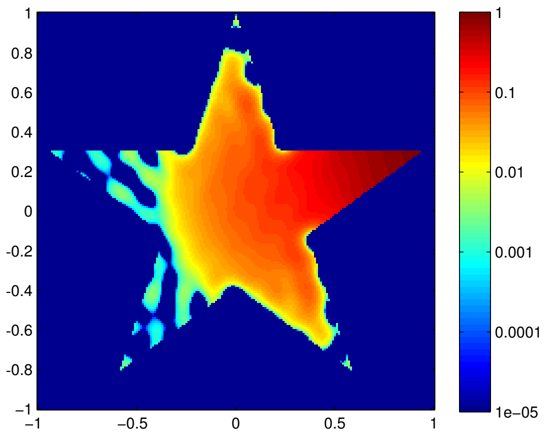
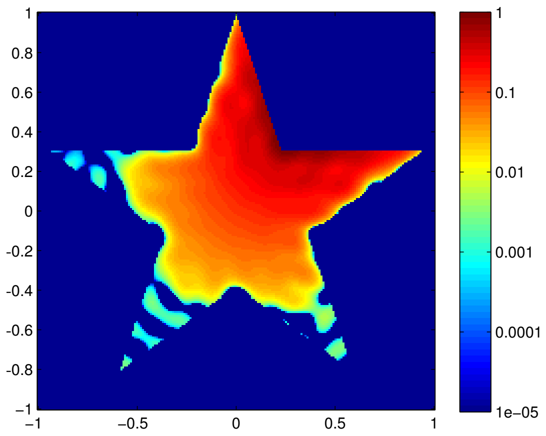

(f)

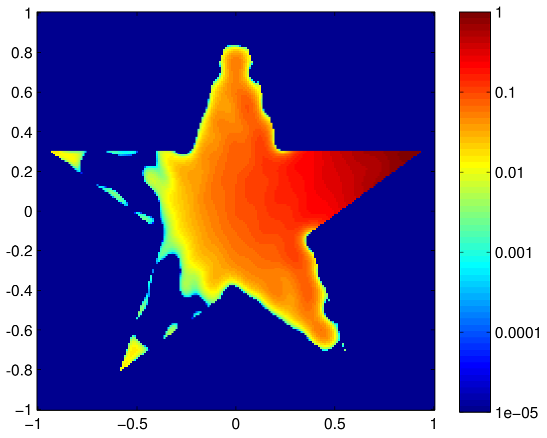
Figure 6.7: Blending weights for a five-pointed star with ten control points and blue noise sampling of its interior and exterior (Figure 6.6). Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).



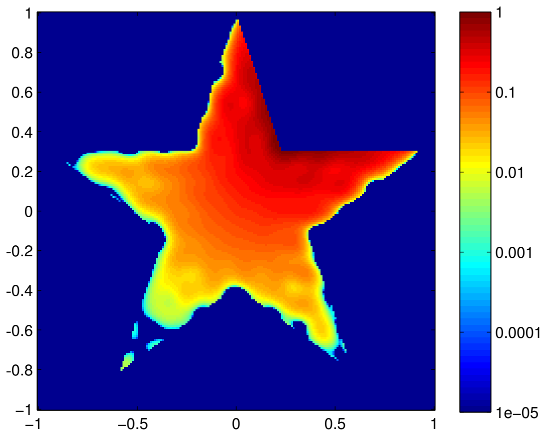
(a)



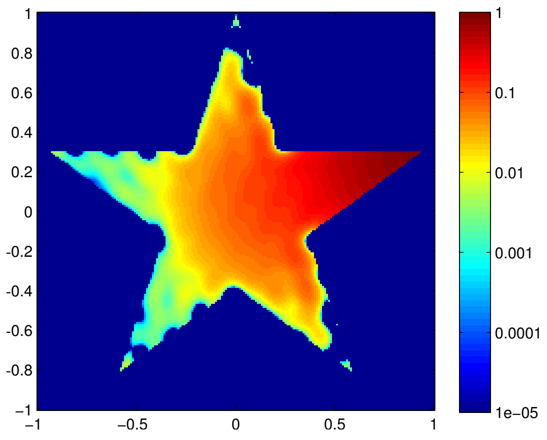
(b)



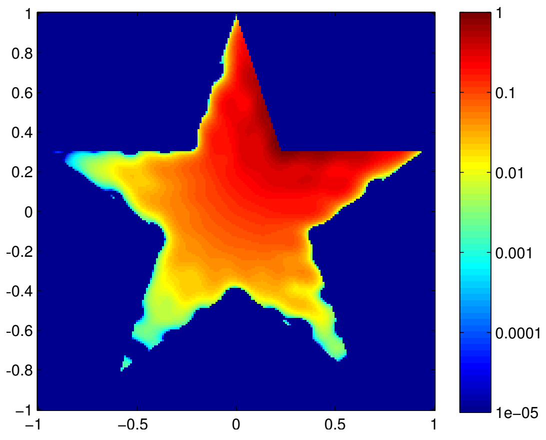
(c)



(d)



(e)



(f)

Figure 6.8: Blending weights in logarithmic scale for a five-pointed star with ten control points and blue noise sampling of its interior and exterior (Figure 6.6). Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).

Properties	**QP**	**QEC**	ℓ_1
Reproduction	Yes	Yes	Yes
Partition of unity	Yes	Yes	Yes
Lagrange property	Yes	Yes	Yes
Linear precision on edges	No	No	No
Non-negativity	No	No	No

Table 6.3: Properties of weights derived with different optimization methods for $\mathbf{W}$, blue noise sampling of the star (Figure 6.6) and heat diffusion with fixed linear weights on edges. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

apparently caused by the relatively sparse distribution of Figure 6.6 that we used to sample the domain. To mitigate this effect, we sample the star and its surrounding region more densely, using 300 points for the edges, 1500 points for the interior and 7500 points for an even narrower region surrounding the star bounded by a square. We run inverse mean shift separately for the interior points and the exterior points, performing 4000 iterations with $s = 0.01$ and $\lambda = 10^{-4}$ to distribute the former, and 28000 iterations with $s = 7.5 \cdot 10^{-3}$ and $\lambda = 5 \cdot 10^{-5}$ for the latter. The resulting distribution, which is shown in Figure 6.9, is then used to compute skinning weights, using heat diffusion with fixed linear weights on edges as initialization for the optimization.

Moreover, the standard deviation of the Gaussian kernel is tuned according to the trade-off analyzed in Section 3.2. This is done by converting the distribution of Figure 6.9 into a 1000×1000 grayscale image, where all pixels are black except those that contain a point. The Fourier spectrum of this image (using DFT) includes an impulse at the origin and a region with near-zero values around it (a property of blue noise). We select σ so that the Fourier-domain Gaussian is concentrated in this region. A visualization of this selection is provided in Figure 6.10, where we plot the magnitude of the Fourier transforms of the aforementioned image and the (discretized) Gaussian kernel for $\sigma = 0.02$ along the bisector of the second and fourth quadrant, which provides sufficient information because of isotropy of both functions. For this particular value of σ, the Gaussian fits nicely into the zero-content region of the distribution's transform and thus we use $\sigma = 0.02$ to instantiate the kernel.

The configuration outlined above leads to the weights depicted in Figure 6.11 (linear scale) and Figure 6.12 (logarithmic scale). With this configuration, the decay as the distance from the control point increases is much smoother and visually pleasing, without intense ripples. We note that in the QEC case, weights are far less local, exhibiting the same linear decay pattern in the interior of the star as in Figure 6.1.

Table 6.4 summarizes the properties that are satisfied (up to a certain precision) with each optimization method on the star when we combine dense blue noise sampling of the interior and exterior of the star with heat diffusion with fixed linear weights on edges. Reproduction, partition of unity and Lagrange property are very accurately satisfied with all methods. While non-negativity is not satisfied with any method, we note that the magnitude of all negative

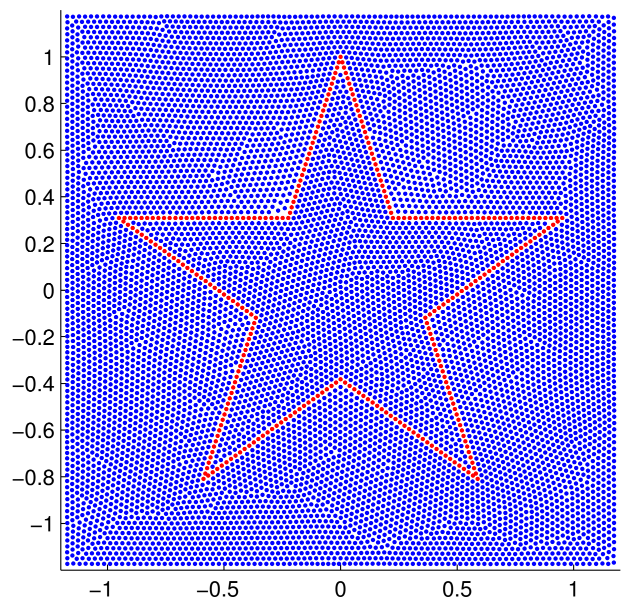


Figure 6.9: Blue noise sampling of the star shape with 1800 points in its interior and 7500 points outside its boundary. Sample points on edges (300 in total) are shown in red and those in the interior or the exterior of the star are shown in blue. We show the final distribution of points after 4000 inverse mean shift iterations for interior points and 28000 iterations for exterior points.

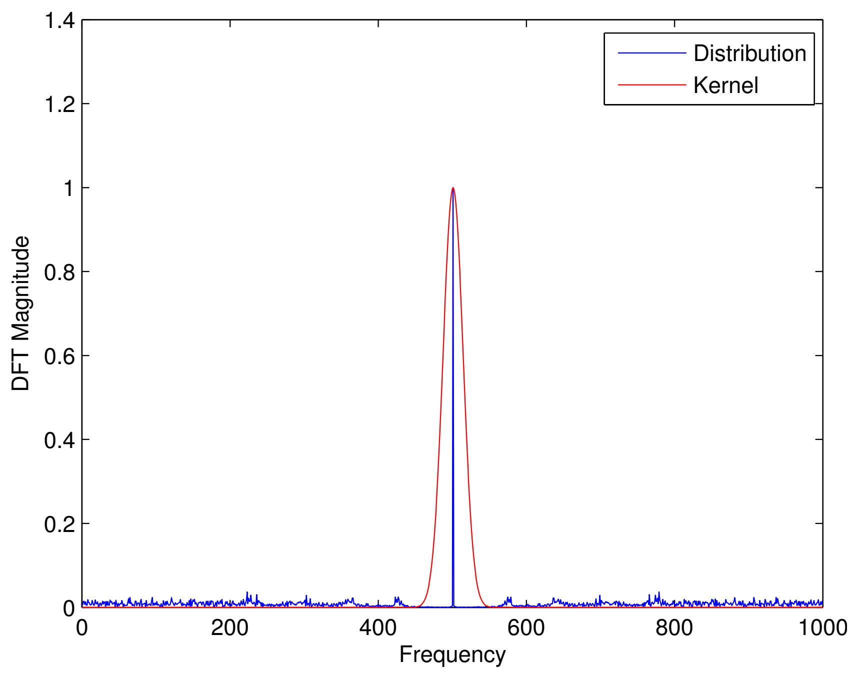
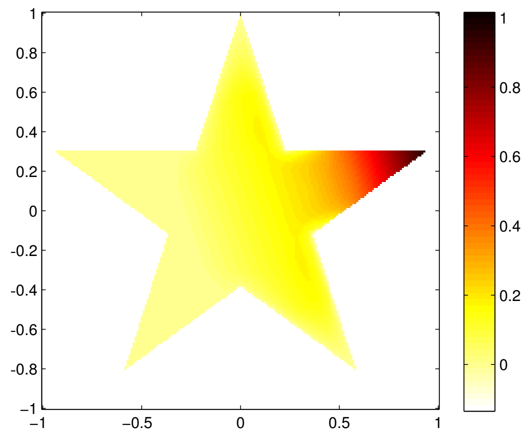
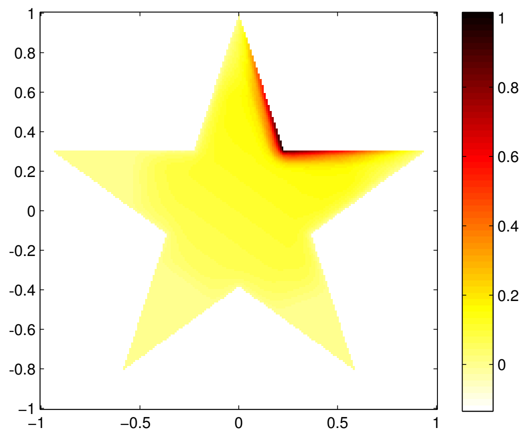


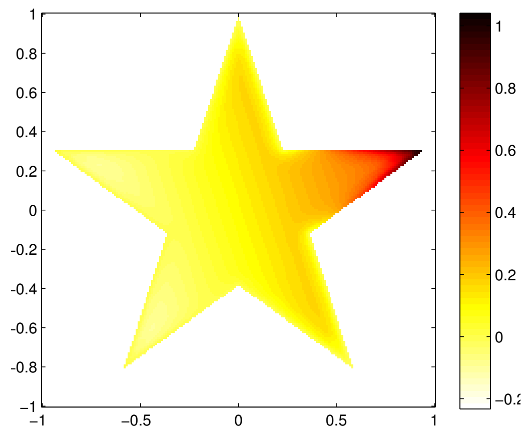
Figure 6.10: Fourier spectra of the distribution in Figure 6.9 (blue) and the Gaussian kernel for $\sigma = 0.02$ (red).



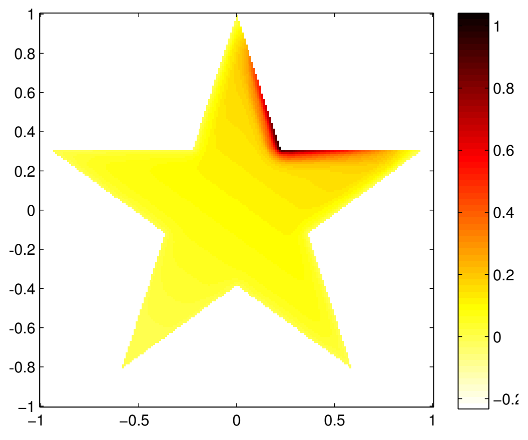
(a)



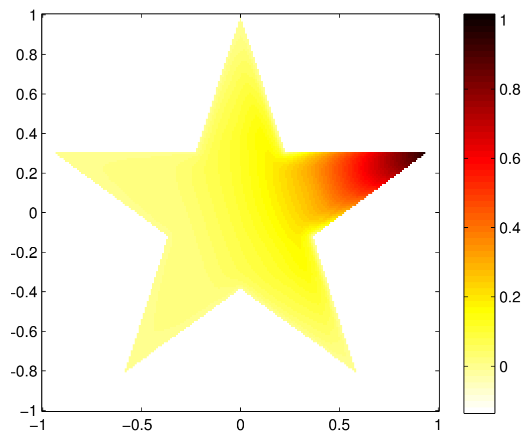
(b)



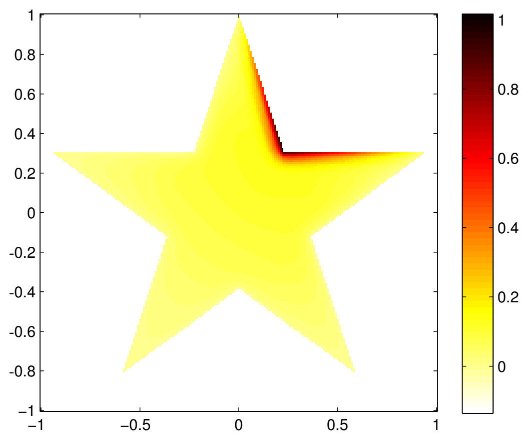
(c)



(d)



(e)



(f)

Figure 6.11: Blending weights for a five-pointed star with ten control points and dense blue noise sampling of its interior and exterior (Figure 6.9). Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).

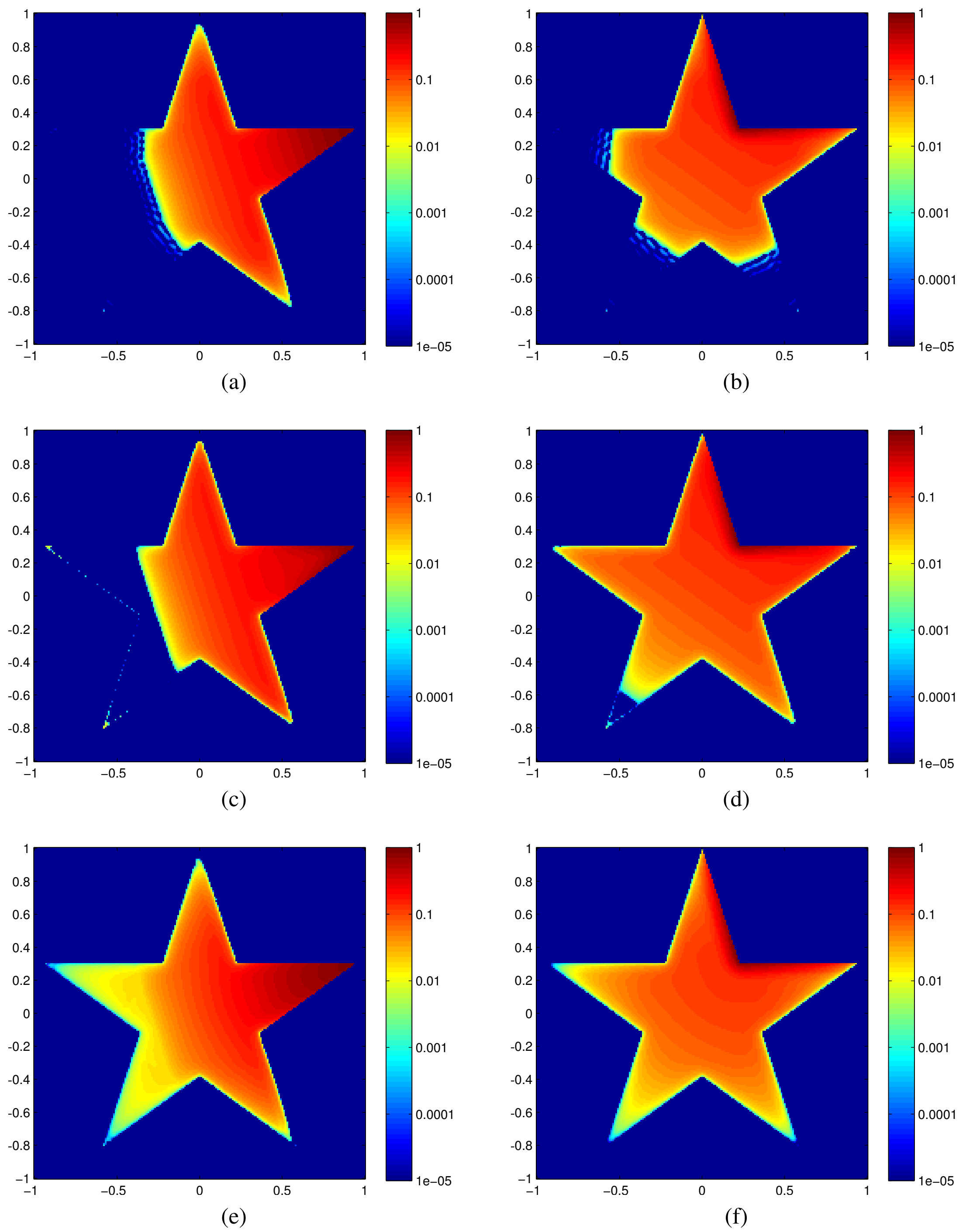
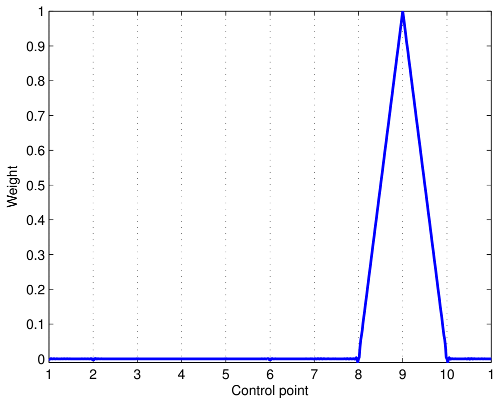
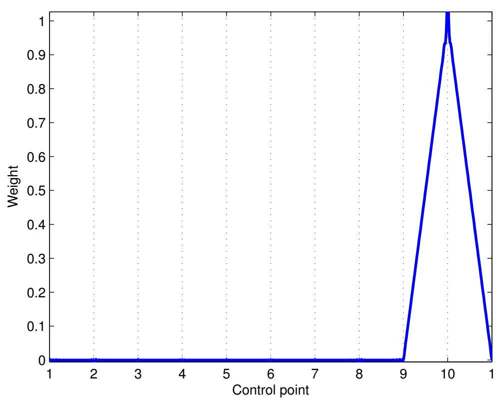


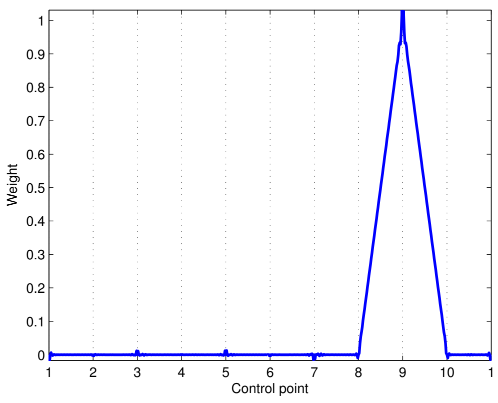
Figure 6.12: Blending weights in logarithmic scale for a five-pointed star with ten control points and dense blue noise sampling of its interior and exterior (Figure 6.9). Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).



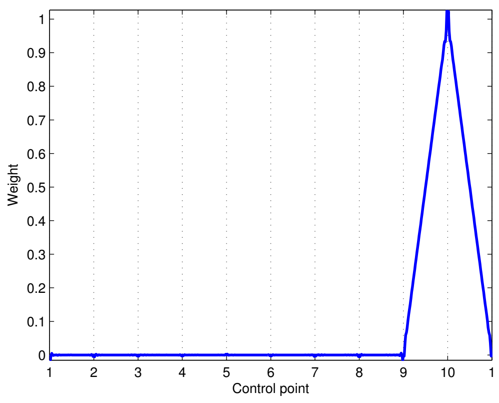
(a)



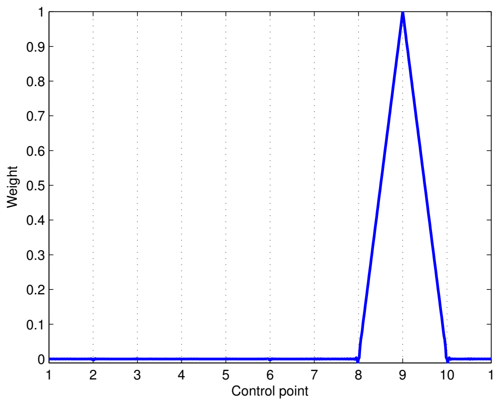
(b)



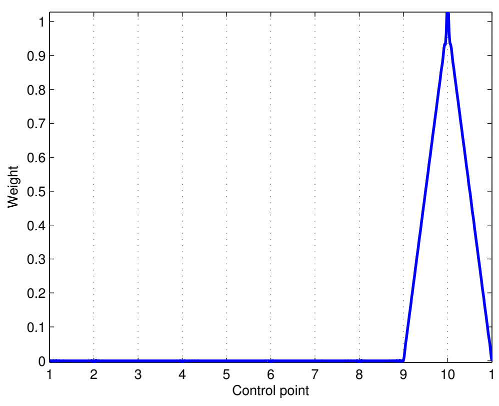
(c)



(d)



(e)



(f)

Figure 6.13: Blending weights along the edges of a five-pointed star with ten control points and dense blue noise sampling of its interior and exterior. We “scan” adjacent edges consecutively, starting from a control point and circling around the star to produce the plots. Results for two representative control points are shown. The first column pertains to a control point at a convex vertex, while the second column to one at a concave vertex. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of $\mathbf{W}$ is done with quadratic programming (first row), quadratic cost minimization without non-negativity constraints (second row) and ℓ_1 optimization (third row).

Properties	**QP**	**QEC**	ℓ_1
Reproduction	Yes	Yes	Yes
Partition of unity	Yes	Yes	Yes
Lagrange property	Yes	Yes	Yes
Linear precision on edges	No	No	No
Non-negativity	No	No	No

Table 6.4: Properties of weights derived with different optimization methods for $\mathbf{W}$, blue noise sampling of the star (Figure 6.9) and heat diffusion with fixed linear weights on edges. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

QP MATLAB Implementation	**Time** (s)
Serial (`for`)	28.5 ± 0.2
Parallel (`parfor`)	8.28 ± 0.06

Table 6.5: Speed-up of QP optimization for star shape of Figure 6.9 by exploiting core-level parallelism in MATLAB.

weights for QP and ℓ_1 is very small (in the order of 10^{-3}). As in the sparser sampling case of Table 6.3, linear precision on edges is not satisfied in a strict sense, despite the fact that weights along the edges "follow" the desired pattern (Figure 6.13).

Optimization of weights at sampling locations is trivially parallelizable, since individual problems for each sample point must be solved. In Table 6.5, we compare running times of a serial implementation and a parallel implementation in MATLAB for QP on the input of Figure 6.9 which contains 9600 points, in order to demonstrate the benefit of parallelism. The experiment is run on an Intel Core i5 3.20 GHz computer with 16 GB of memory and 4 CPU cores. The parallel implementation makes use of core-level parallelism, leveraging MATLAB's `parfor` structure.

We test computation of weights for a rigid region on the star using the last configuration. The region Π is defined as the inner part of the bottom right leg of the star and points lying inside it are presented in green in Figure 6.14.

We set the rigidity parameter $\rho = 10^{-4}$ and $\theta = 5\sigma$ and solve the quadratic program (5.35) using MATLAB function `quadprog`. In this particular case, the number of sample points inside region Π is $q = 141$, the number of points inside or close to Π is $r = 395$ and optimization with quadratic programming requires approximately 2 seconds in MATLAB. We present the resulting weights in Figure 6.15 in linear scale and in Figure 6.16 in logarithmic scale. These figures include results for six representative control points and are to be compared with Figures 6.11(a)–(b) and 6.12(a)–(b) respectively. We note that weights obtained using the rigidity term satisfy exactly the same set of properties that is presented in Table 6.4.

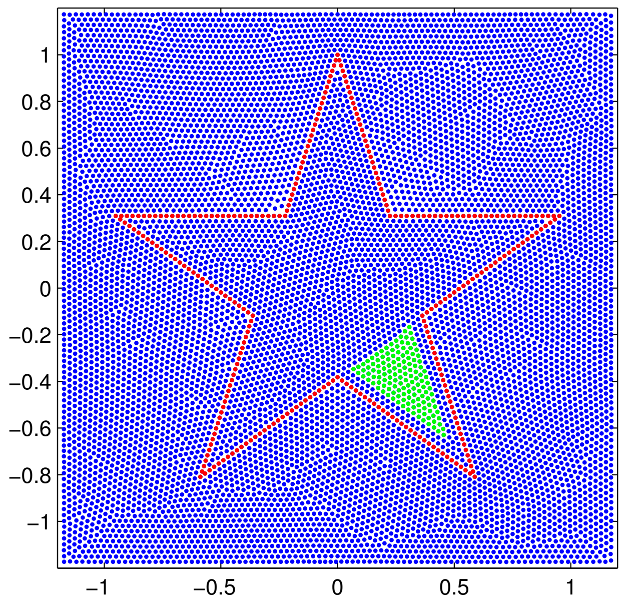


Figure 6.14: A region that should behave as rigid is defined on the star shape, using the sampling distribution of Figure 6.9. Points inside the region are shown in green.

6.2.2 Stick Figure

A shape that contains parts of different dimensionalities is the stick figure of Figure 6.17. More specifically, the figure's head constitutes a 2D area, while its sticks that represent the torso and limbs are one-dimensional structures. We define 14 control points for the stick figure: one control point at the end of each limb, two control points at the junctions of the legs and the arms and eight control points placed symmetrically at intervals of $45°$ around the head. We sample the stick figure in a similar fashion to the star shape of Section 6.2.1, i.e. we first place samples on the control points and sample the sticks regularly. This gives us a total of 238 points. Afterwards, keeping these points fixed, we distribute 6000 points in the region that surrounds the figure, using inverse mean shift with $s = 0.03$ and $\lambda = 10^{-3}$. After 2000 inverse mean shift iterations, we obtain the sampling distribution of Figure 6.18.

We calculate kernel coordinates inside the head and on the sticks of the stick figure using the sampling distribution shown in Figure 6.18, initializing weights at sampling locations via heat diffusion with fixed linear weights along sticks and setting $\sigma = 0.06$ for the Gaussian kernel. We visualize the weights obtained with each distinct optimization method in Figures 6.19–6.21 for the head in logarithmic scale and in Figures 6.22–6.24 for the sticks. In order for the results to be concise, we present weights inside the head only for control points on the right side of the head (the ones on the left side have symmetric weights), at the junction of the two arms and at the ends of the right arm and leg. Similarly, we present weights on the sticks only for control points at the ends of them and for the control point at the top of the head (the rest control points around the head have weights similar to the latter).

Coordinates obtained with quadratic programming are the most local, whereas quadratic cost minimization without non-negativity constraints gives weights that globally influence every part of the head almost equally, as shown in Figure 6.20. A very important observation is that

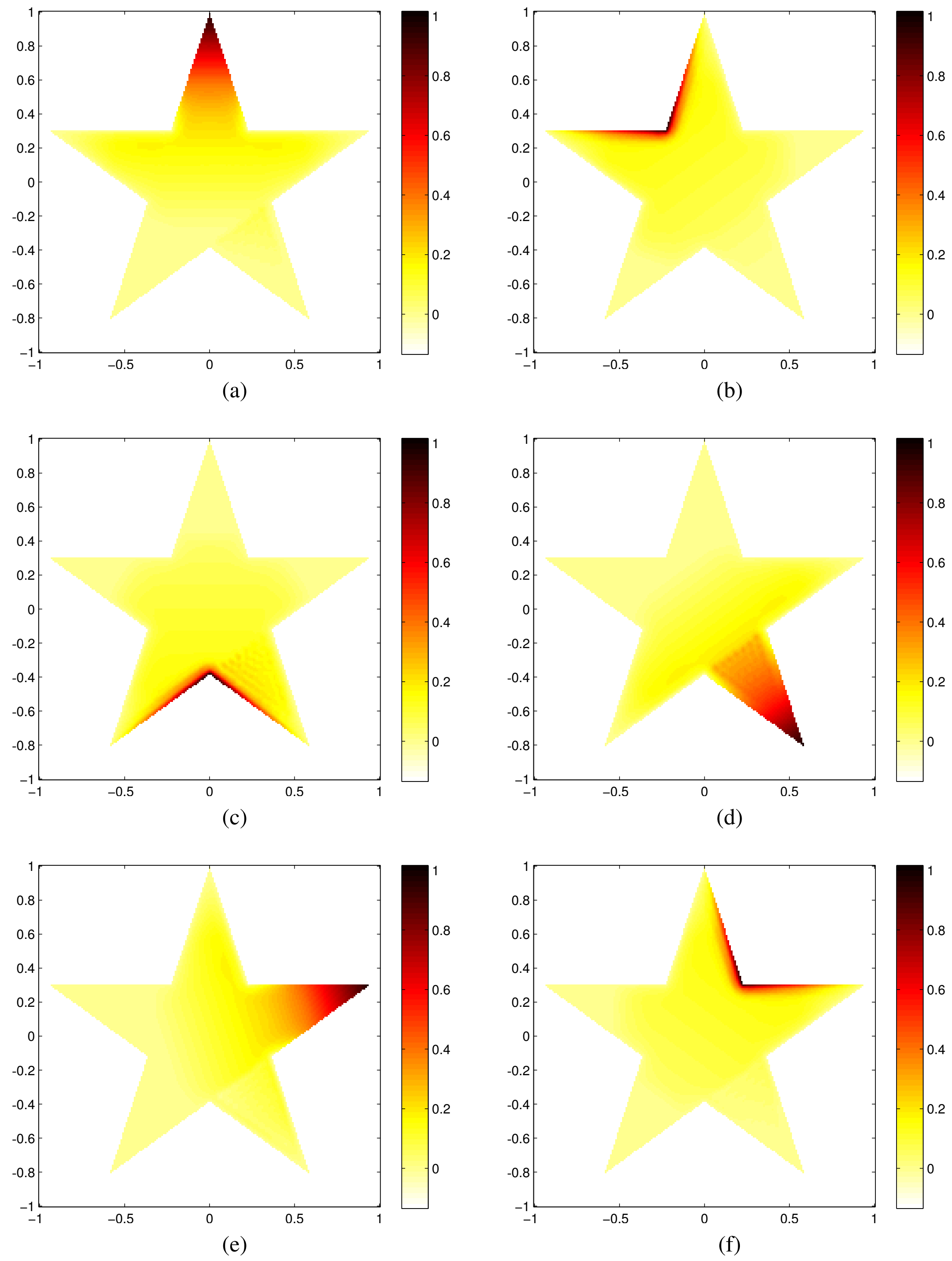


Figure 6.15: Blending weights for a five-pointed star with ten control points and a triangular rigid region at the inner part of its bottom right leg (Figure 6.14). Results for six representative control points are shown. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of **W** is done with quadratic programming, jointly for points influencing the rigid region and individually for the rest.

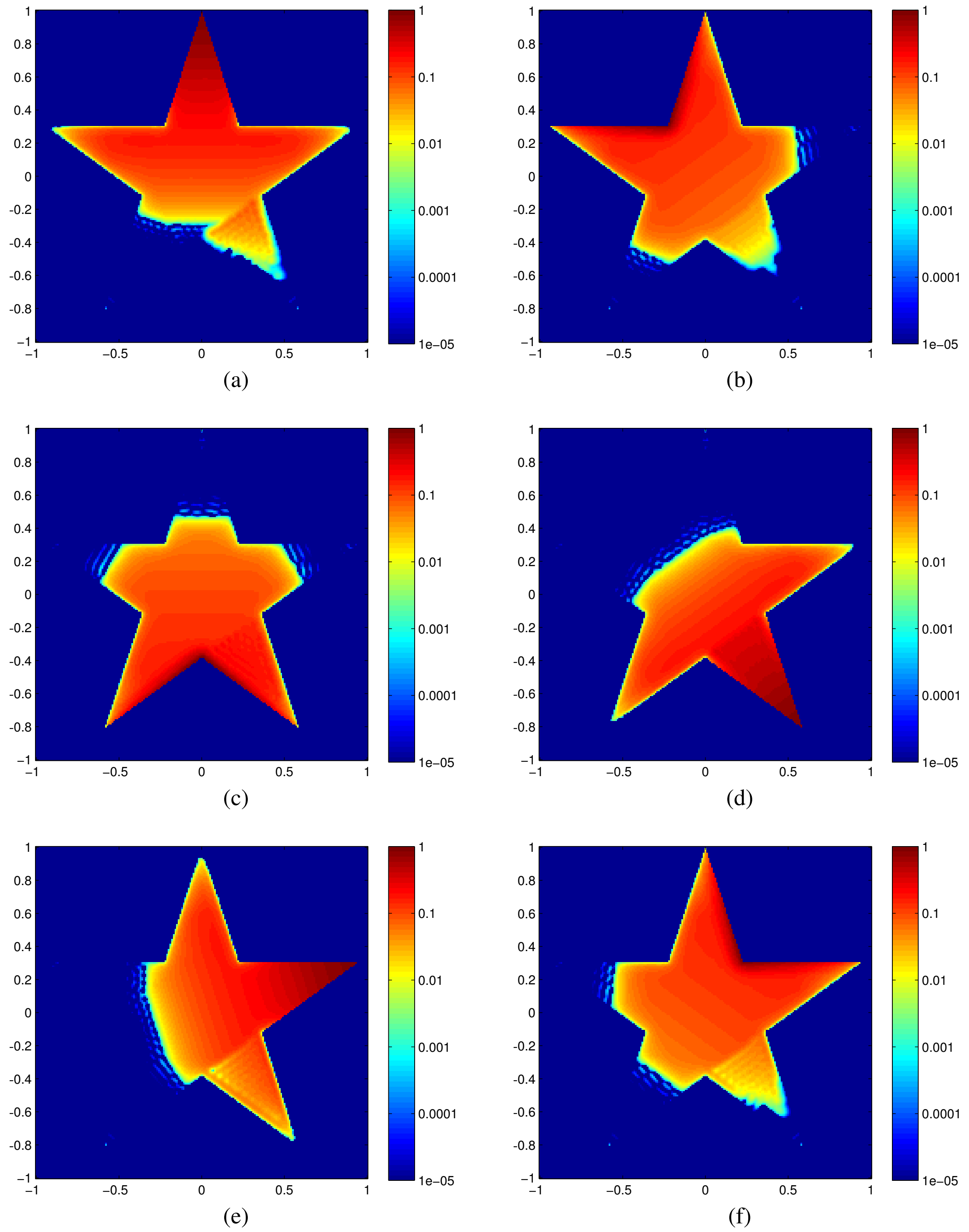


Figure 6.16: Blending weights in logarithmic scale for a five-pointed star with ten control points and a triangular rigid region at the inner part of its bottom right leg (Figure 6.14). Results for six representative control points are shown. Heat diffusion with fixed linear weights on edges is used for initialization. Optimization of **W** is done with quadratic programming, jointly for points influencing the rigid region and individually for the rest.

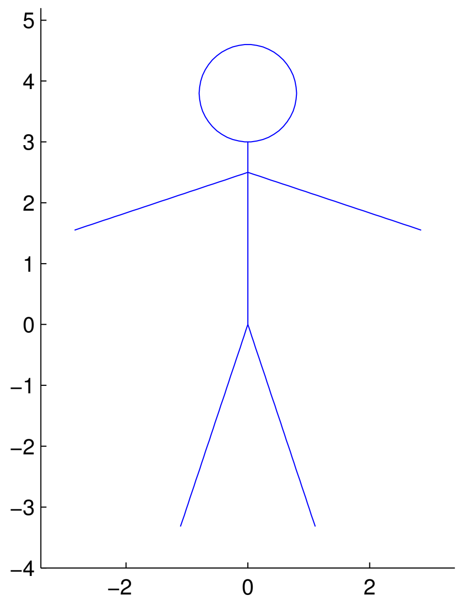


Figure 6.17: Illustration of the stick figure shape. The figure includes a circular head and one-dimensional sticks that model the neck, torso, arms and legs. The boundary of the head and the sticks are shown in blue.

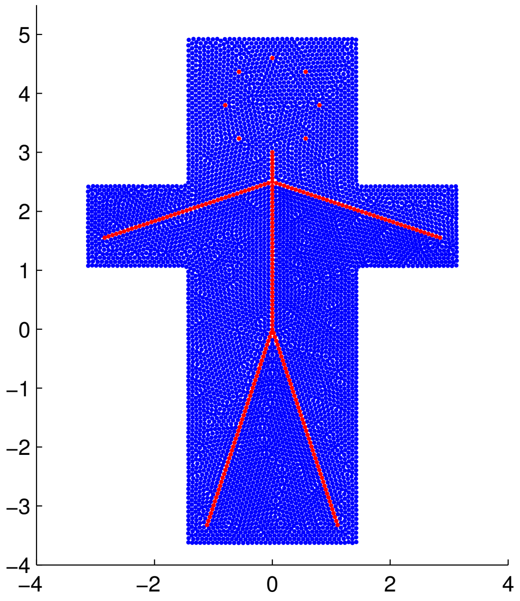


Figure 6.18: Blue noise sampling of the stick figure with 230 points on its sticks (red), 8 points as controls of the head (red) and 6000 points in the interior of the head and around the shape (blue). We show the final distribution of points after 2000 inverse mean shift iterations.

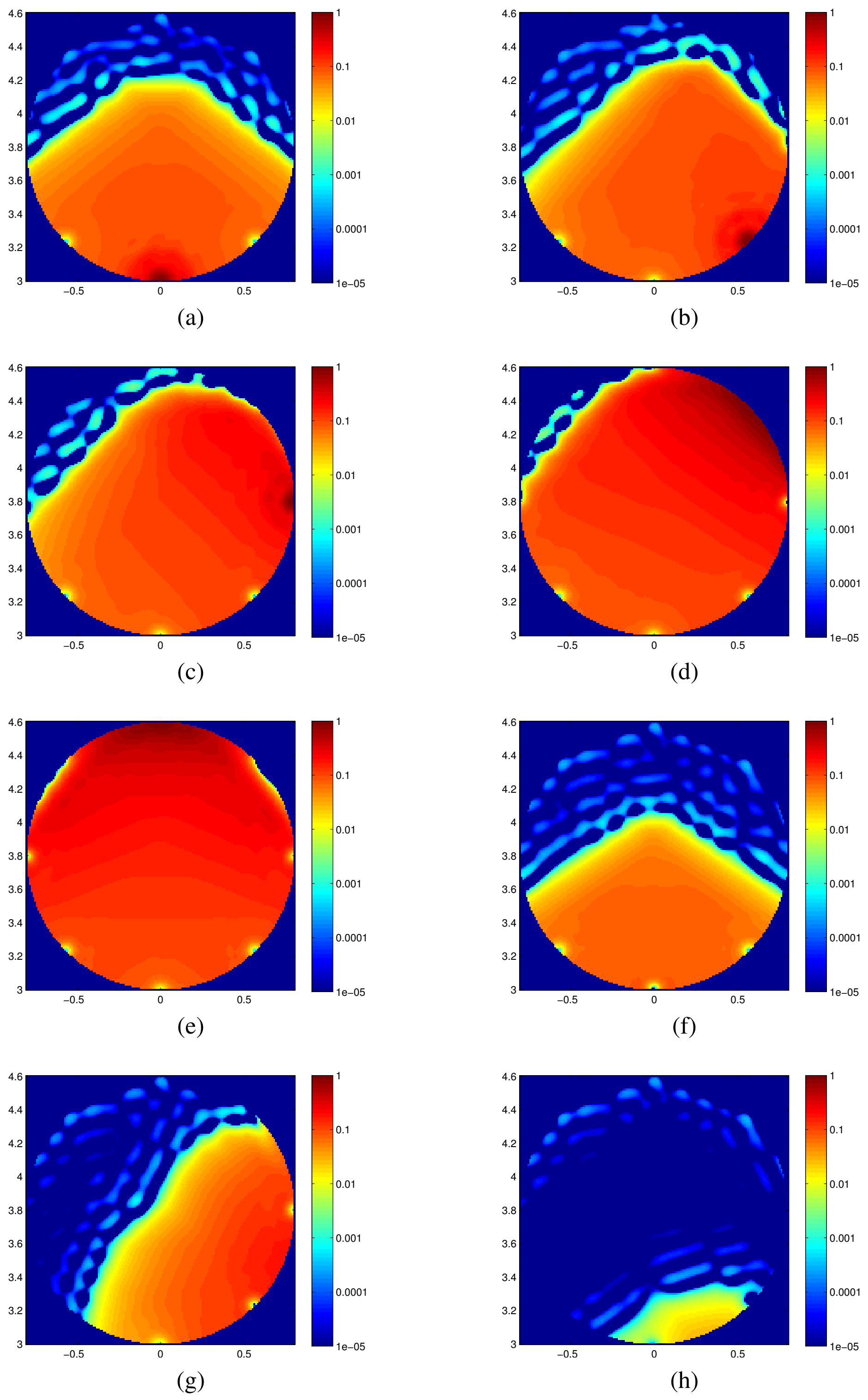


Figure 6.19: Blending weights inside the head of the stick figure with quadratic programming in logarithmic scale. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown.

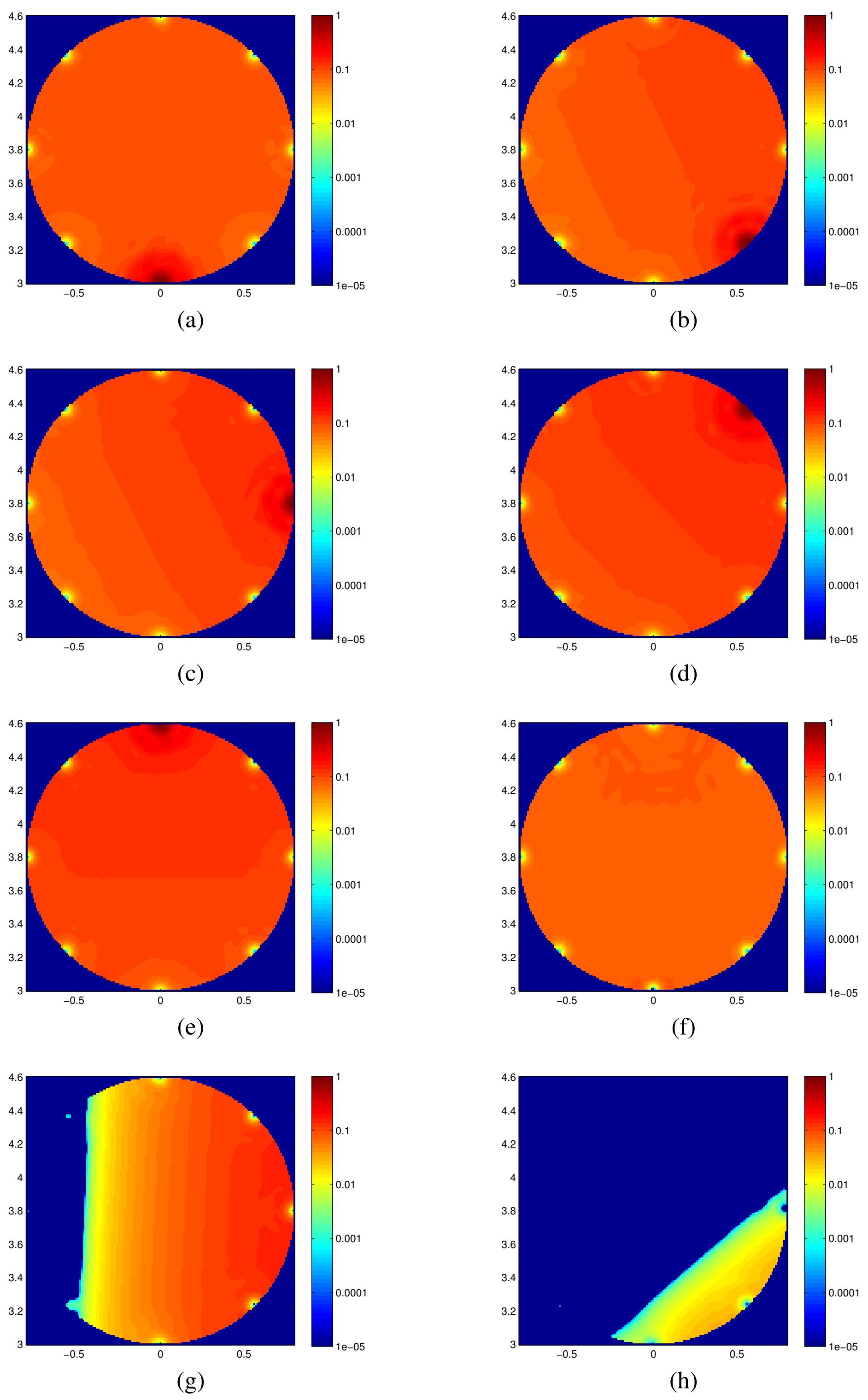


Figure 6.20: Blending weights inside the head of the stick figure with quadratic cost minimization without non-negativity constraints in logarithmic scale. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown.

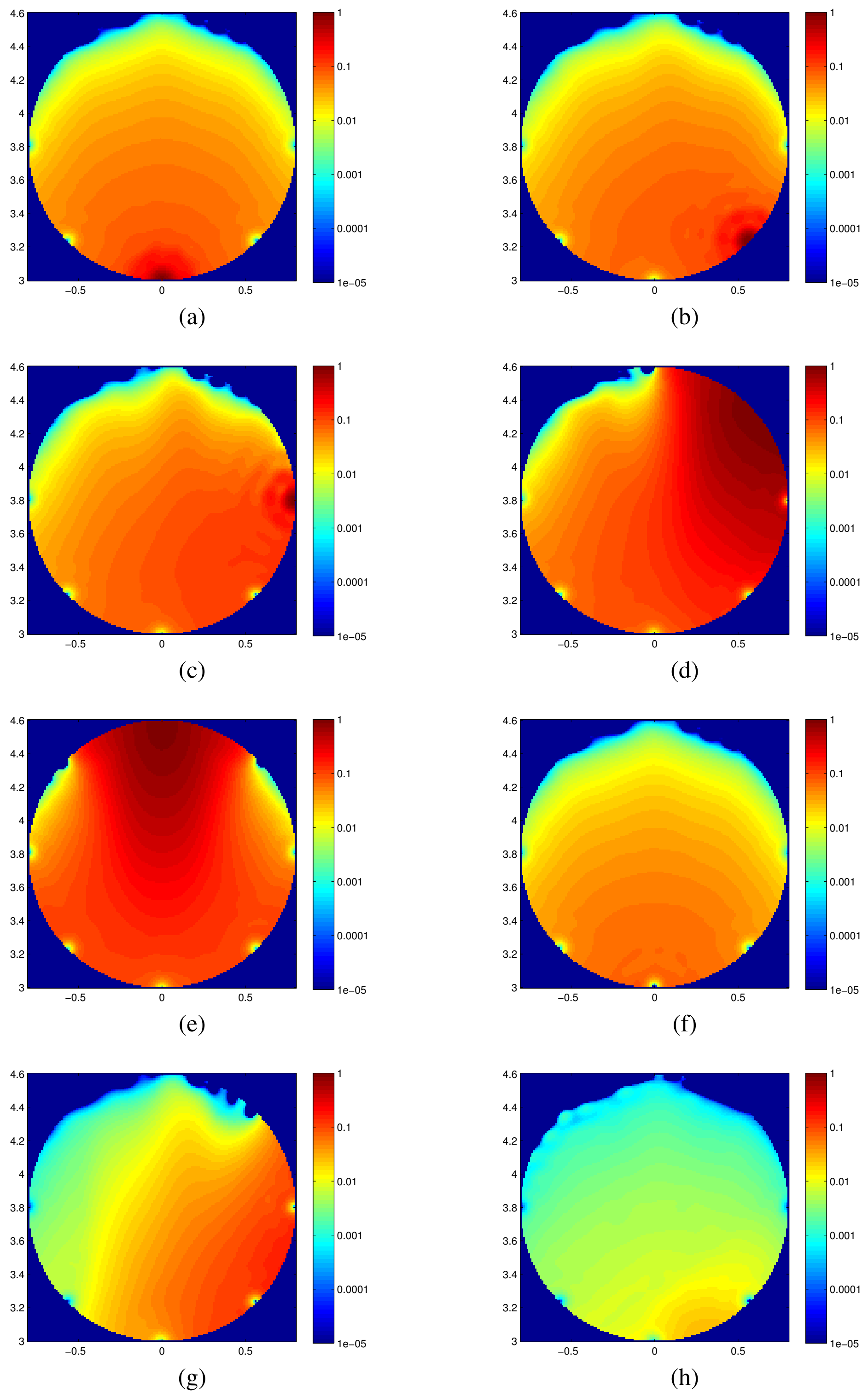


Figure 6.21: Blending weights inside the head of the stick figure with ℓ_1 optimization in logarithmic scale. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown.

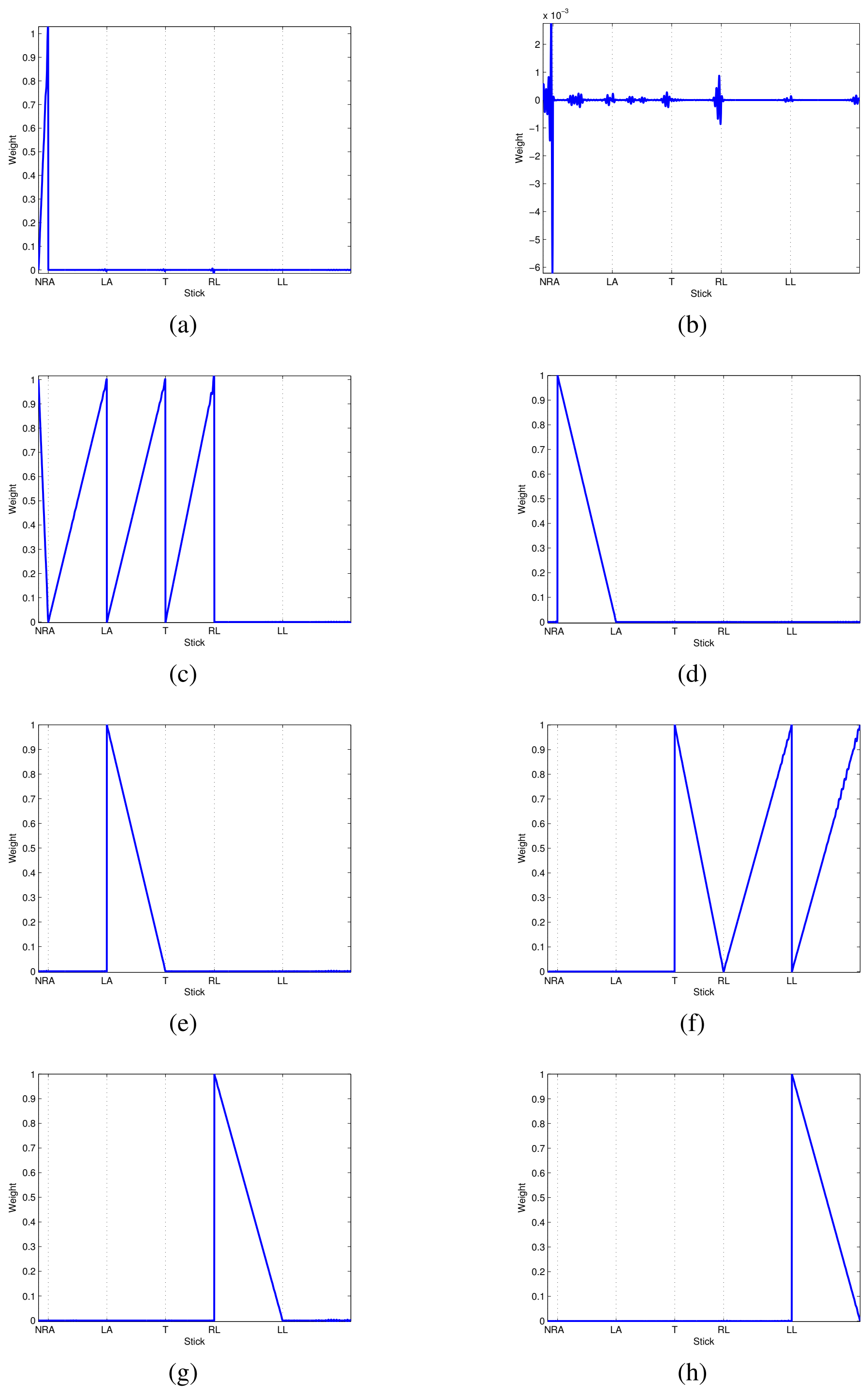
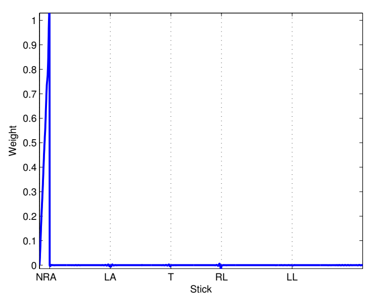
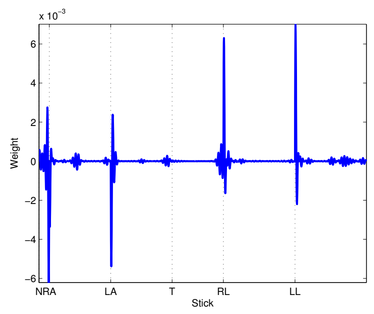


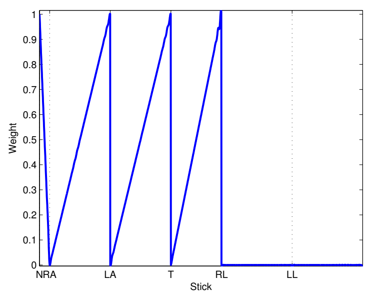
Figure 6.22: Blending weights along the sticks of the stick figure with quadratic programming. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown. The horizontal axis is split into distinct sticks—which are not consecutive—for illustration purposes. Their start is marked with their acronym; N: neck, RA: right arm, LA: left arm, T: torso, RL: right leg, LL: left leg.



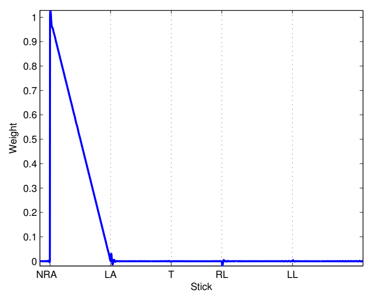
(a)



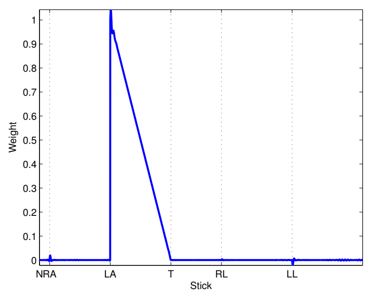
(b)



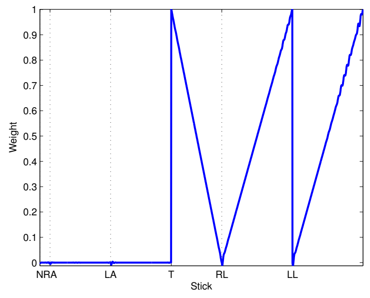
(c)



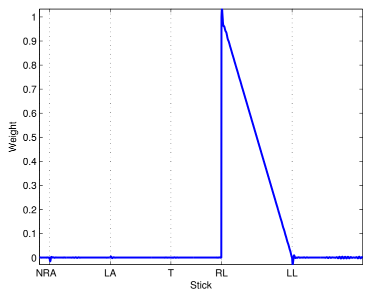
(d)



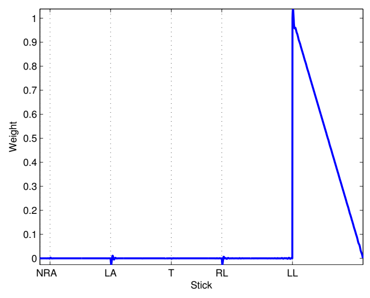
(e)



(f)



(g)



(h)

Figure 6.23: Blending weights along the sticks of the stick figure with quadratic cost minimization without non-negativity constraints. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown. The horizontal axis is split into distinct sticks—which are not consecutive—for illustration purposes. Their start is marked with their acronym; N: neck, RA: right arm, LA: left arm, T: torso, RL: right leg, LL: left leg.

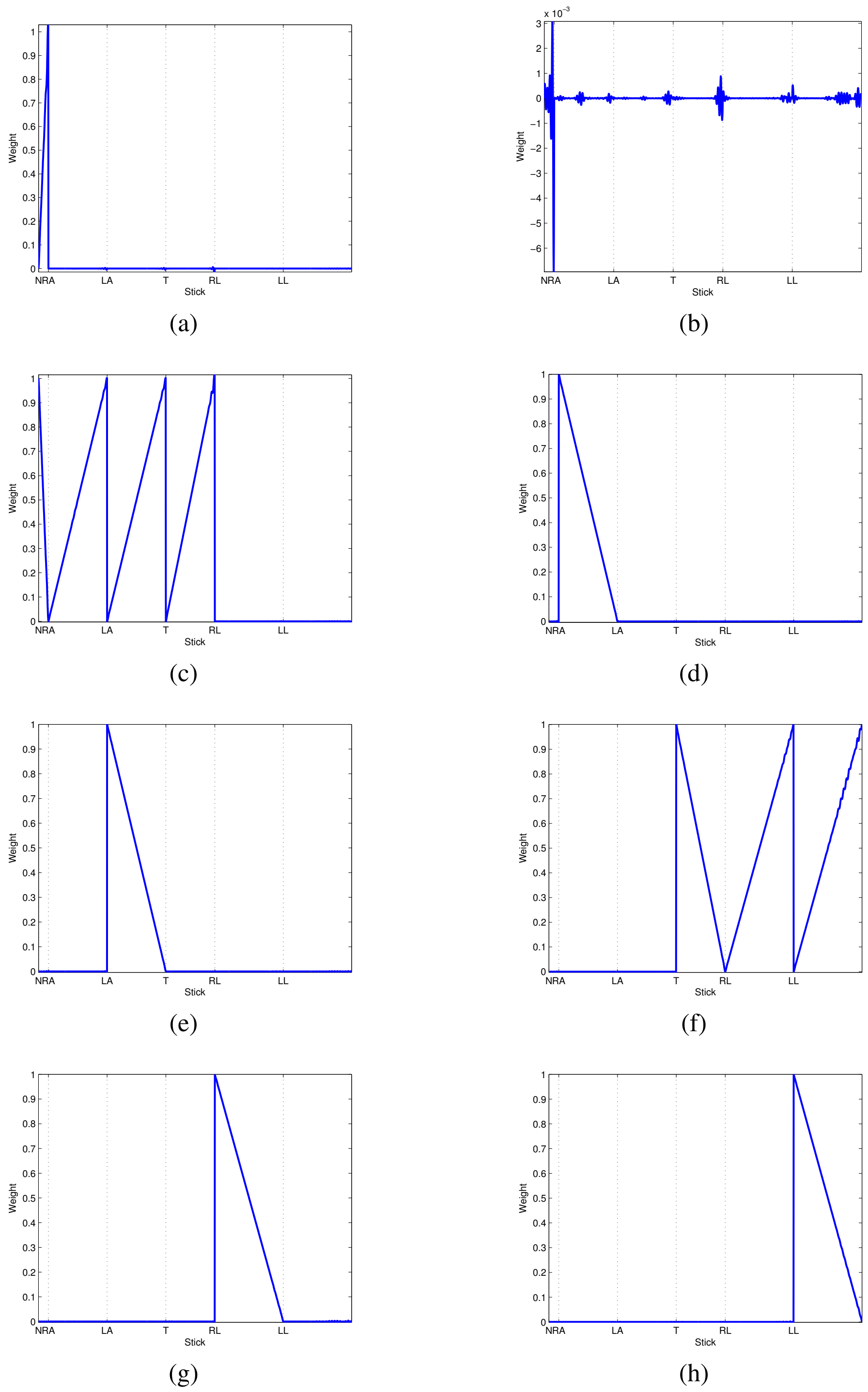


Figure 6.24: Blending weights along the sticks of the stick figure with ℓ_1 optimization. Heat diffusion with fixed linear weights on sticks is used for initialization. Results for eight representative control points are shown. The horizontal axis is split into distinct sticks—which are not consecutive—for illustration purposes. Their start is marked with their acronym; N: neck, RA: right arm, LA: left arm, T: torso, RL: right leg, LL: left leg.

Properties	**QP**	**QEC**	ℓ_1
Reproduction	Yes	Yes	Yes
Partition of unity	Yes	Yes	Yes
Lagrange property	Yes	Yes	Yes
Linear precision on edges	No	No	No
Non-negativity	No	No	No

Table 6.6: Properties of weights derived with different optimization methods for $\mathbf{W}$, blue noise sampling of the stick figure (Figure 6.18) and heat diffusion with fixed linear weights on sticks. QP stands for quadratic programming and QEC for minimization of quadratic cost only with equality constraints.

control points other than the ones on the boundary of the head have weights that assume large values inside the head irrespective of the optimization method (Figures 6.19(f)–(h), 6.20(f)–(h), 6.21(f)–(h)). Even the control points at the end of the arms and legs influence a great part of the figure's head. Since we fix weights to linear values along sticks throughout heat diffusion iterations, control points at the ends of sticks are "favored" during heat diffusion over control points around the head, as weights for the latter are diffused only from a single sample point (the one lying at the control handle) instead of several sample points (the ones lying on sticks that are adjacent to the control handle).

Table 6.6 summarizes the properties that are satisfied (up to a certain precision) with each optimization method on the stick figure. Reproduction, partition of unity and Lagrange property are very accurately satisfied with all methods. While non-negativity is not satisfied with any method, we note that the magnitude of all negative weights for QP and ℓ_1 is rather small (in the order of 10^{-3}). Linear precision on sticks is not satisfied in a strict sense with any method, but weights along sticks are quite close to the ideal linear values.

6.3 Interface for Interactive Weight Design

6.3.1 Interface Outline

We implement an interaction interface in MATLAB, where the user can modify the weights and inspect the resulting deformation of 2D objects. The interaction window is split into two basic plots. In the left plot, the object along with the control points are shown. In the right plot, user-driven weights $\mathbf{W}$ of individual control points are visualized and the user can redesign them by painting. The workflow for weight design is the following:

- The shape together with its sampling distribution of points is loaded. The user can either choose to draw some high-level structures such as lines, which requires that a new sampling distribution is computed for the shape, or load an already existing sampling distribution from a previous session.

- The weights at sampling locations are initialized through heat diffusion and a first round of optimization is run, in order for the interaction to begin with valid weights.
- The user selects a particular control point, for which she would like to modify the control weights. The current weights for this particular point are then plotted on the right side.
- The user paints these weights by dragging the mouse over certain parts of the plot. A small, spatially smooth increment is applied to the weights in order to compute their new desired values.
- Once the user finishes painting, all painted weights are re-optimized, using the currently selected optimization method. The updated weights of the selected control point are shown on the right. The user can then select another control point and follow the same steps, further modifying the weights.
- The left plot of the window, which shows the processed image, also contributes to interaction. Interlaced with atomic painting moves, the user can move the control points and deform the object based on the current weights using linear blend skinning, so as to decide whether the resulting deformations appeal to her.
- At any point during interaction, the user can reset the shape to its rest pose by clicking a reset button and restart applying a different sequence of transformations to control handles.
- There is also a radio button pane which allows the user to change the optimization method at interaction time. Once the user selects a different optimization method from the one that is currently used, the initial heat diffusion weights are used as input to optimization, so any painting moves that the user had performed intermediately are disregarded. Since optimization is performed for the whole object, this feature of the interface is not interactive for large input shapes.
- The above procedure is continued, until the user is satisfied with the achieved deformation of the original object.

6.3.2 Rectangular Shape

As a first experiment, we deform a rectangular 2D object, with four control points at its corners. We use a 256×256 image of a checkerboard pattern with distinguishable corners to aid visualization of deformations and define a sparser 29×29 grid of sampling locations. This grid extends a little beyond the image boundaries, so that the computed kernel coordinates are exact even near the image boundary (cf. Figure 3.1). We initialize the weights using heat diffusion with linear weights on the edges of the image and present some indicative screenshots of the interaction window during an interaction session, where quadratic programming is used for optimization. To render the deformed versions of the image, MATLAB function `griddata` is utilized.

The interaction window is initialized in the form that we present in Figure 6.25. No control point has been selected, so no weights are shown yet. The first operation of the user is to select the upper right control point, whose weights are then shown on the right (Figure 6.26). This control

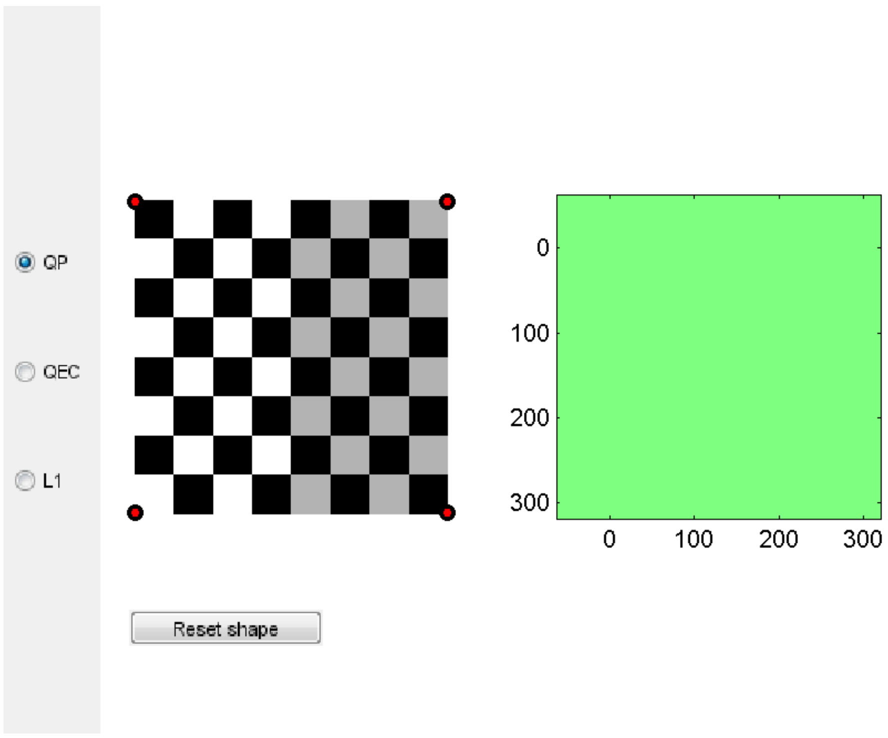


Figure 6.25: Starting screen of interaction interface.

point is dragged slightly downwards and to the right, resulting in the deformed image of Figure 6.27. The next operation is a painting move at the middle right part of the grid, which leads to a modification of the weights and a subsequent update of the deformation (a careful observation of the checkerboard pattern of Figure 6.28 at the region corresponding to the modified weights reveals a slight change). The user then chooses to reset the image to its original form, as shown in Figure 6.29. This brings only the image back to its rest pose, whereas the painting move previously performed on the weight of the upper right control point is not undone. Finally, the user applies a large translation to the lower right control point, so that a fold-over appears in the result, shown in Figure 6.30.

6.3.3 Star Shape

We experiment further with deforming the stap shape of Section 6.2.1. Again, a checkerboard pattern is used to add texture to the shape and help inspect the deformations. We sample the star and its surrounding region with the distribution of Figure 6.9 and run heat diffusion with fixed linear weights along the edges of the star to initialize weights at sampling locations. Due to non-convexity of this particular shape, rendering with `griddata` causes artifacts in the image of the deformed shape. Instead, we implement rendering of the deformed shape via linear interpolation based on a triangular mesh that is defined on the grid of pixels of the undeformed image, using MATLAB function `tri2grid`.

In Figure 6.31, we show an instance of the interaction window right after the control point at the corner of the bottom right leg of the star has been clicked. The initially chosen optimization method is ℓ_1. The user left-clicks and drags this point (which corresponds to translating it) to

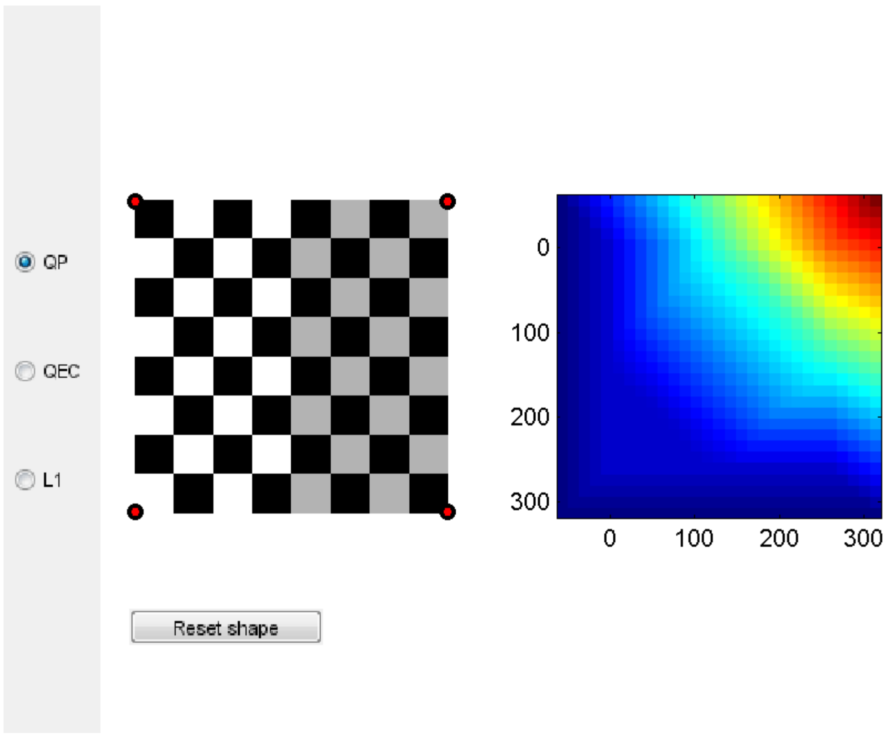


Figure 6.26: The upper right control point has been selected and its corresponding weights are plotted on the right.

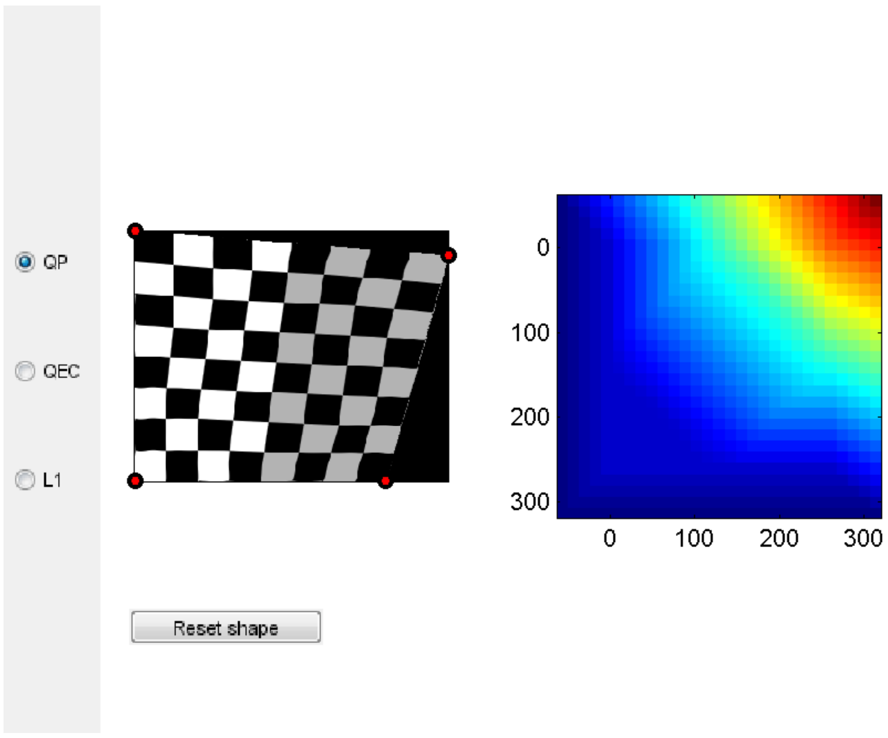


Figure 6.27: The upper right control point has been dragged and the current weights have been blended with linear blend skinning to produce the presented deformation of the pattern.

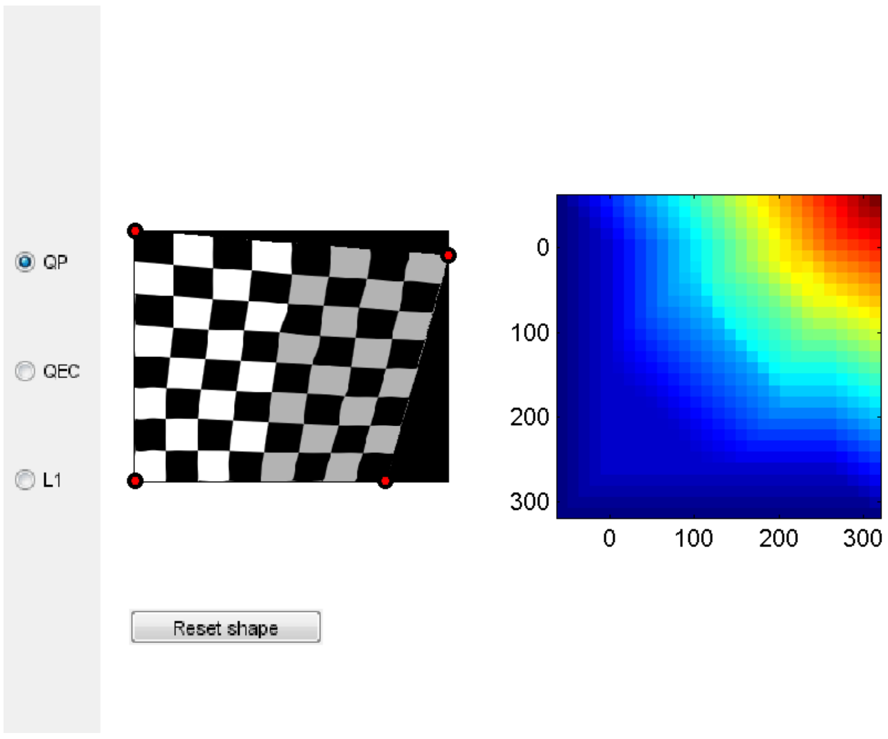


Figure 6.28: The user has painted some weights and they have been re-optimized, changing the deformed image as well.

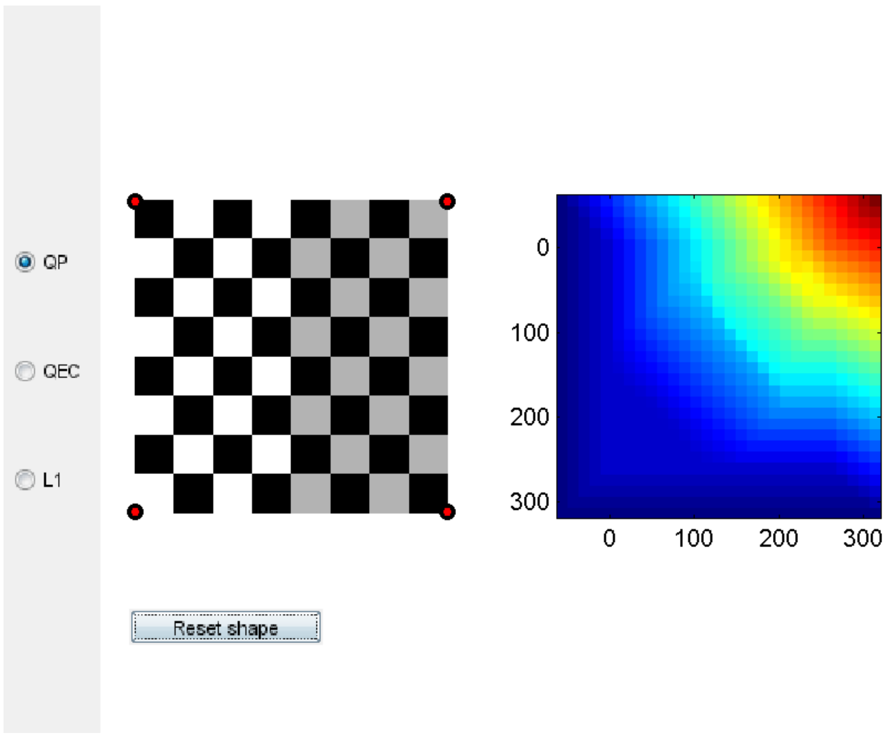


Figure 6.29: The user has clicked the reset button placed under the shape and all previous deformations are undone, bringing the checkerboard image pattern back to its original form.

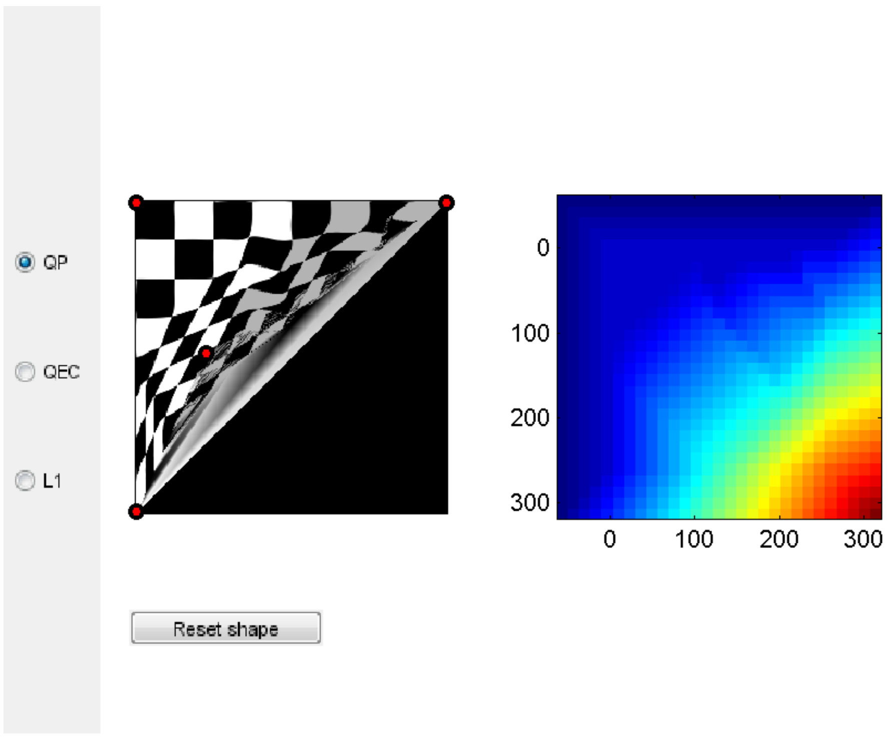


Figure 6.30: The lower right control point has been translated to further deform the image. Convexity of the deformed boundary holds no longer and hence a fold-over appears.

produce the deformation of Figure 6.32 and then right-clicks it and drags the pointer (which corresponds to rotating it) to produce the deformation of Figure 6.33. Another move is made by rotating the control point at the end of the top leg (Figure 6.34). A different method, namely quadratic programming, is selected for optimization, which yields a reoptimization of weights across the entire shape and consequently a modified deformation with the new weights (Figure 6.35). Finally, a transformation of a concave control point, namely a translation, is applied in Figure 6.36.

We provide a comparison of deformation results for the star shape with and without a region that should behave as rigid. The rigid part of the star is the same as in Section 6.2.1, namely the inner part of the bottom right leg (Figure 6.14). Figure 6.37 presents the deformed versions of the star after the control point at the end of the bottom right leg has been translated inwards. A careful observation of the deformed checkerboard pattern for the “simple” and the “rigid” version reveals that squares in the bottom right leg of the latter are deformed in a more rigid fashion, preserving their right angles better.

Figure 6.38 shows the sampling distribution for the star in the case where the user has drawn a line on it. We utilize MATLAB function `getline` to provide the interface with the line-drawing feature. The spacing of points near the line (whose regularly spaced samples are shown in red) has been successfully adjusted by inverse mean shift so that these points do not fall too close to points on the line. In this case, the parameters of the algorithm are set to $s = 1$ and $\lambda = 1$ (larger values are due to larger scale of the shape) and 5000 iterations are run to get the final sampling distribution.

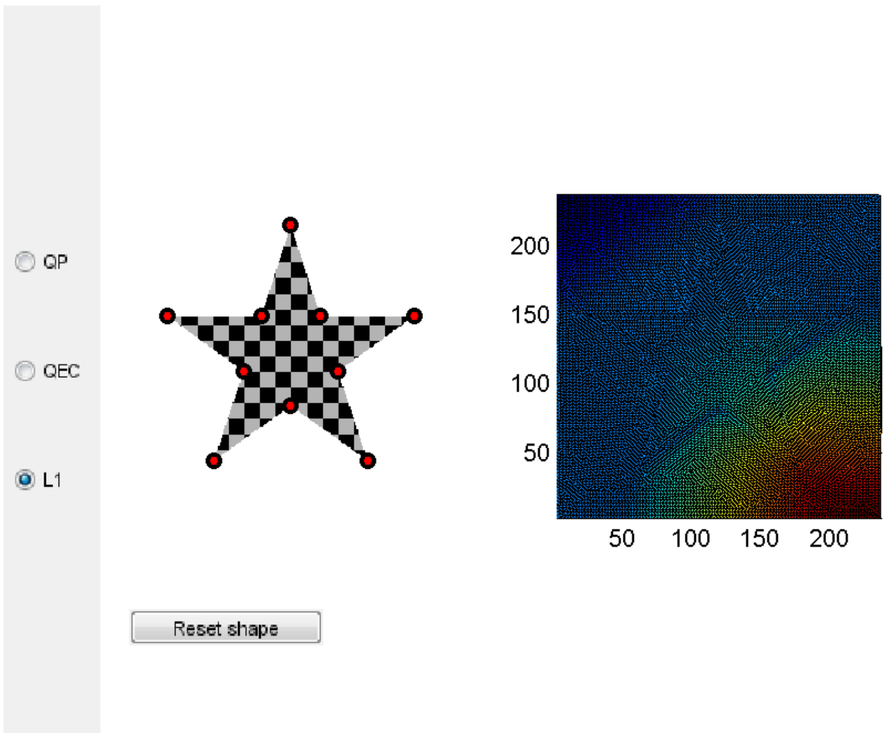


Figure 6.31: The control point at the bottom right leg of the star has been selected and its corresponding weights are plotted on the right.

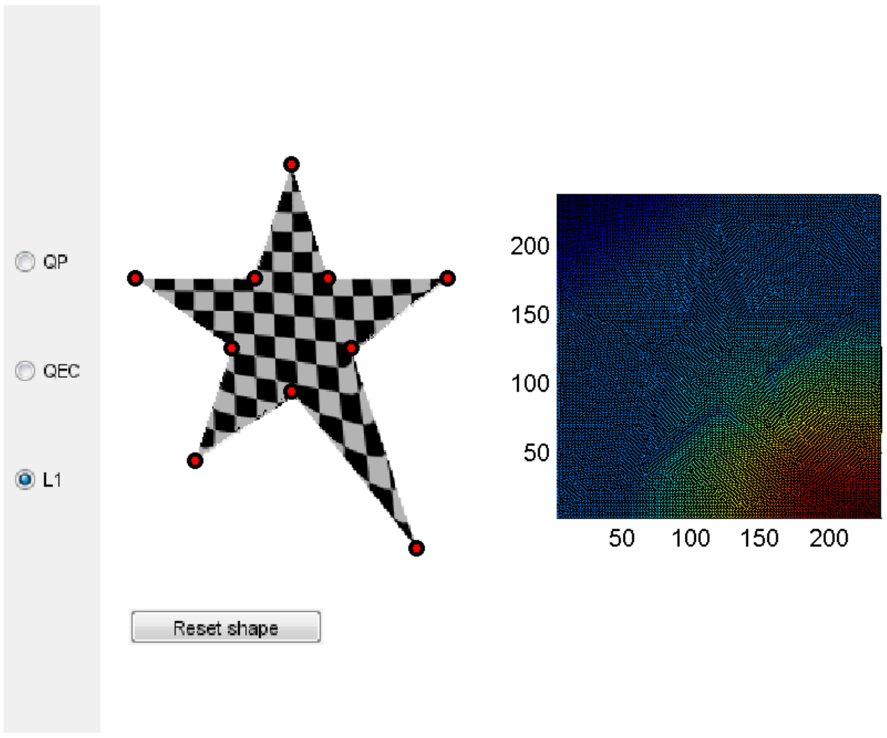


Figure 6.32: The control point at the end of the bottom right leg of the star has been translated outwards.

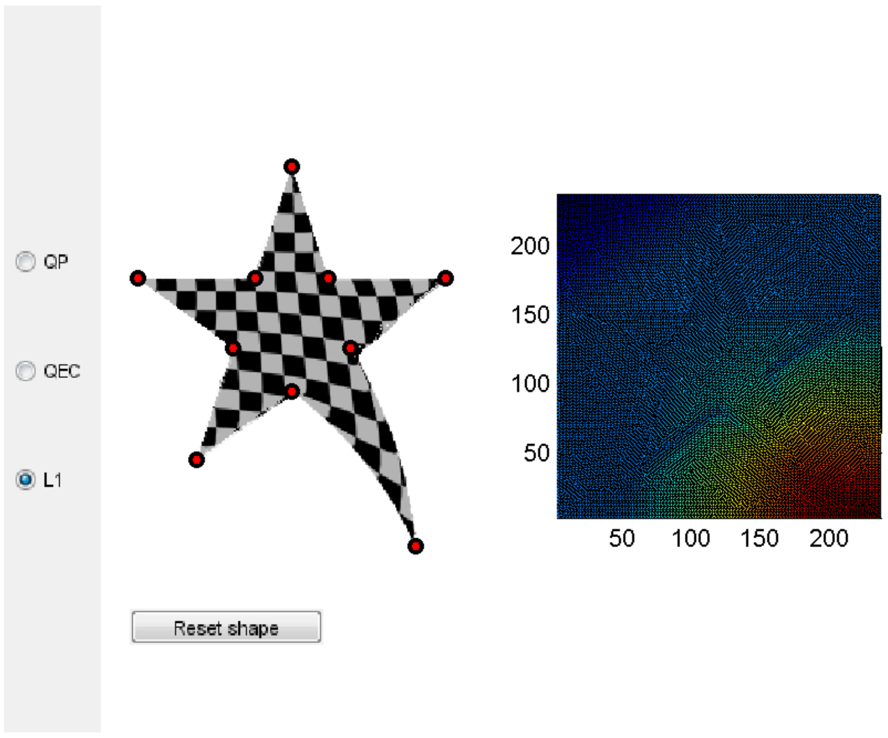


Figure 6.33: The control point at the end of the bottom right leg of the star has been rotated clockwise.

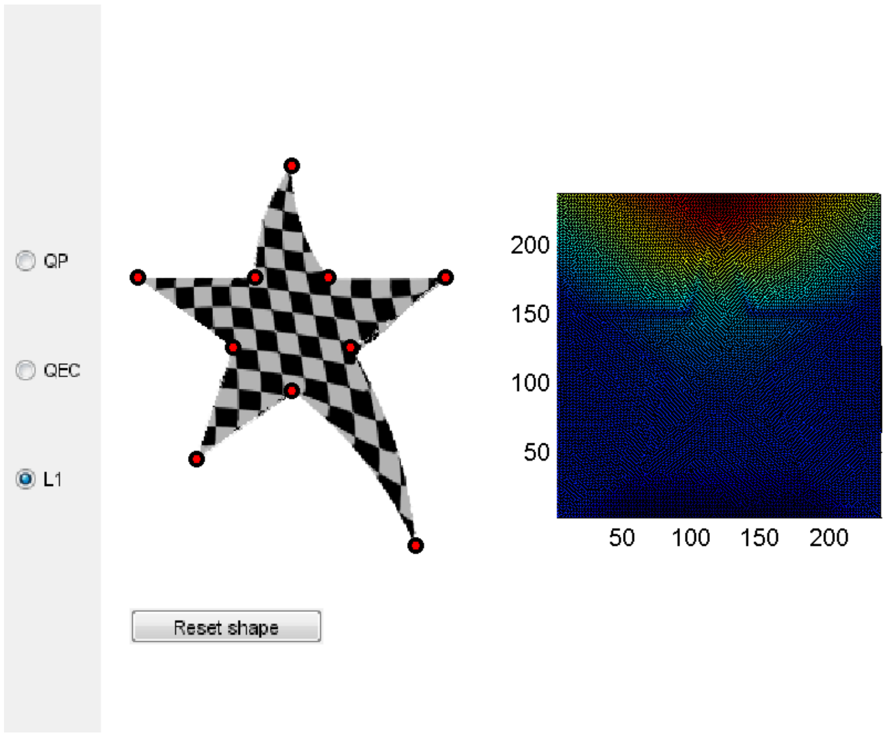


Figure 6.34: The control point at the end of the top leg of the star has been rotated clockwise.

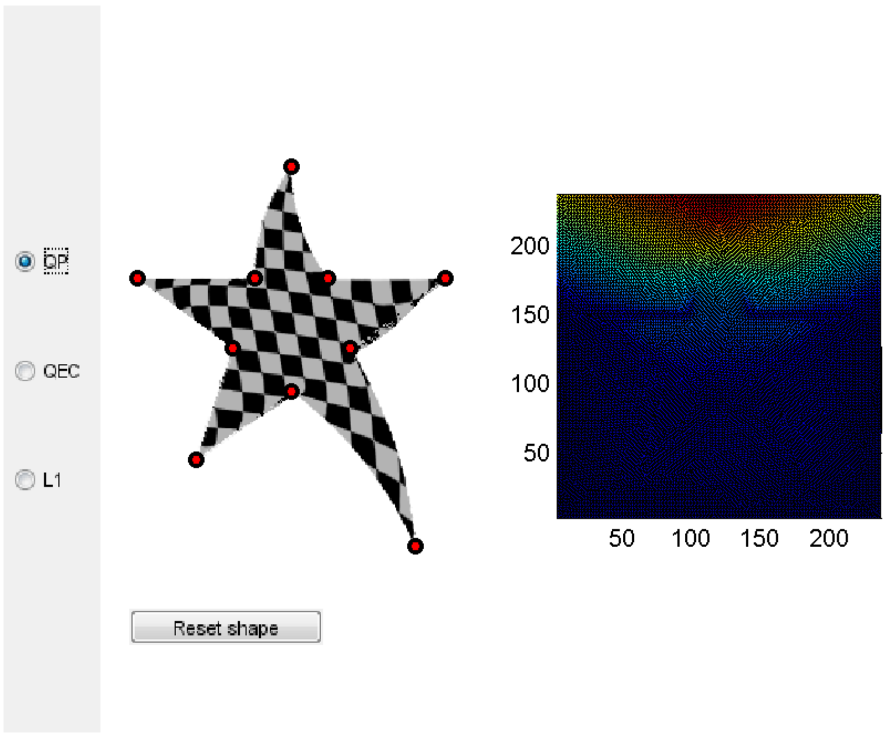


Figure 6.35: Change of selected method for optimization to quadratic programming.

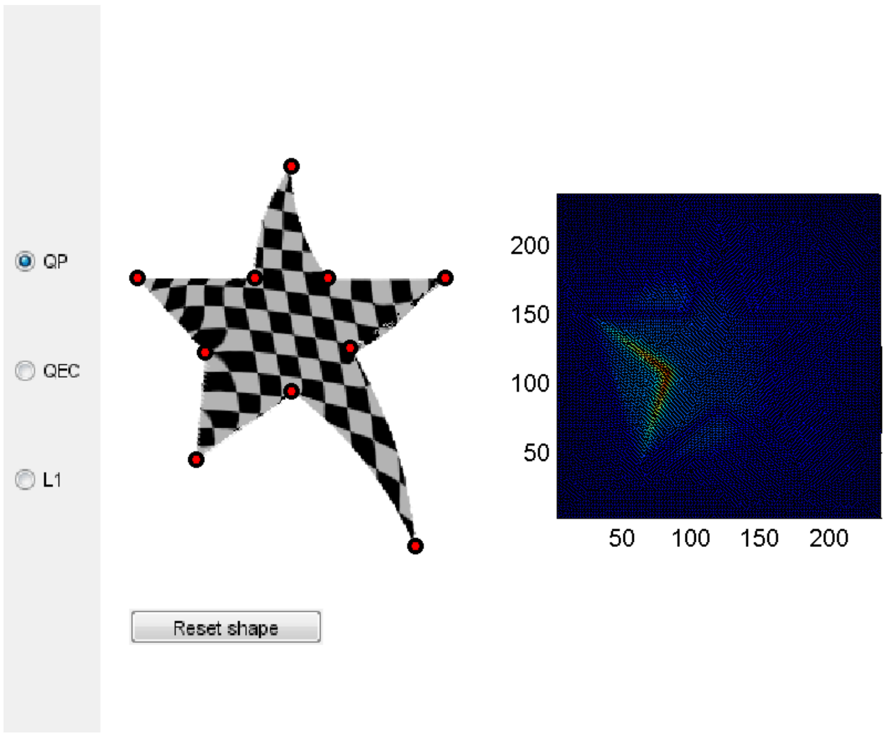
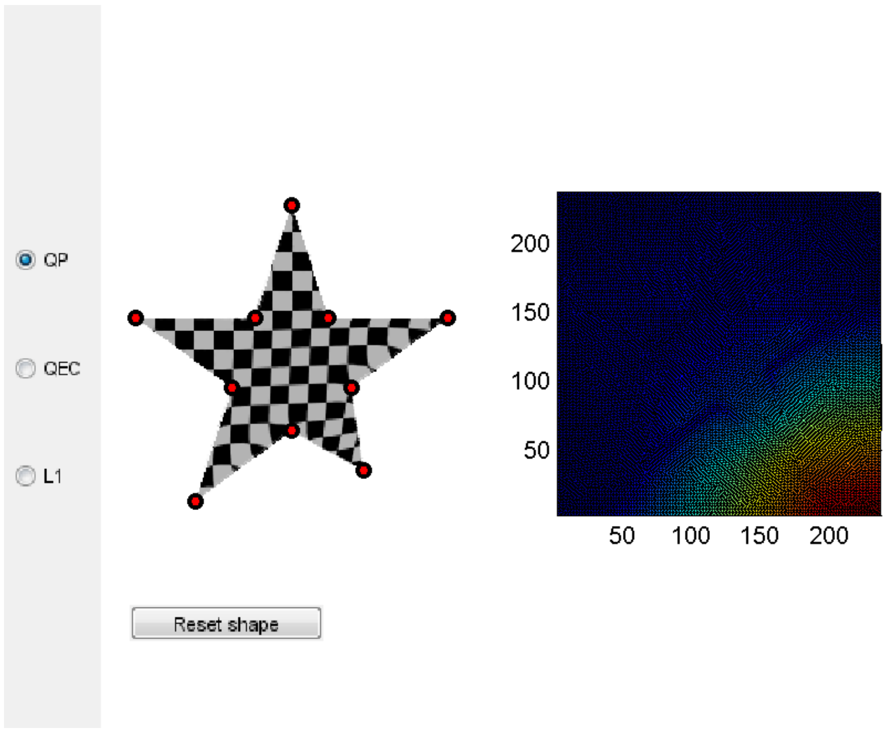
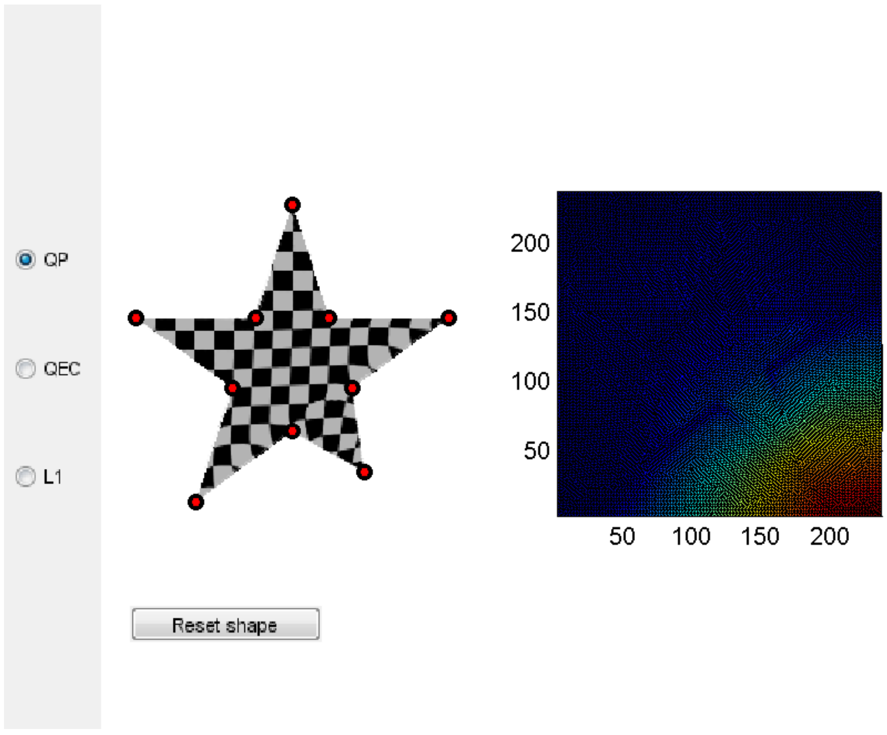


Figure 6.36: The control point at the junction of the two left legs of the star has been translated outwards.



(a)



(b)

Figure 6.37: Effects of using a rigid region on deformation of the star shape. (a) shows the deformed shape without any rigid part defined, whereas (b) is obtained using the rigid region of Figure 6.14.

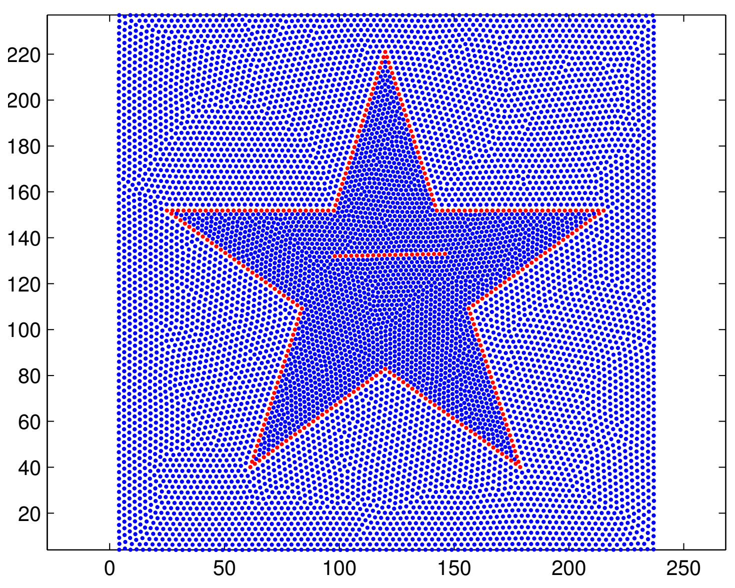


Figure 6.38: Blue noise sampling of the star shape with 300 sample points on edges (red), 21 points along the user-defined line (red) and 9300 more points for the rest of the shape's interior and exterior. The final distribution of points has been obtained after 5000 inverse mean shift iterations.

6.3.4 Shapes in Images

Our deformation interface can handle general 2D shapes that are contained in images. In this case, a cage has to be defined on the image, in such a way that it encloses the shape. The user can define this cage as a closed polygonal line interactively or she can load a cage constructed in a previous interaction session. Afterwards, sampling is performed in the interior of the cage using inverse mean shift, again allowing the user to load the sampling distribution of a previous session. The remaining features of the interface are similar to the ones presented for the cases of the rectangular and star shapes in Sections 6.3.2 and 6.3.3.

In Figure 6.39, we present the deformation of a butterfly, whose initial pose is illustrated together with the user-defined cage of 26 control points in Figure 6.39(a). We have used optimization with ℓ_1-norm cost to compute weights at sampling locations with our kernel coordinates method and compare our result to bounded biharmonic weights [1]. More specifically, three control points at the upper part of the butterfly's wing, marked green in Figure 6.39(a), are translated to new positions in an attempt to shrink that part of the wing. We present the sum of these points' weights in Figure 6.39(b) for our method and in Figure 6.39(c) for bounded biharmonic weights. The latter method achieves better locality, but lacks the reproduction property which is guaranteed by kernel coordinates. The deformation results are illustrated in Figure 6.39(d) for kernel coordinates and Figure 6.39(e) for bounded biharmonic weights. Judging from the deformed shape, the two methods produce a similar deformation, albeit kernel coordinates suffer from some delicate artifacts in the hind limbs of the butterfly.

We provide a comparison of running times of the two aforementioned methods for the butterfly input in Table 6.7. We used both a serial and a parallel implementation for both methods: for bounded biharmonic weights, weights for individual control handles are computed in parallel,

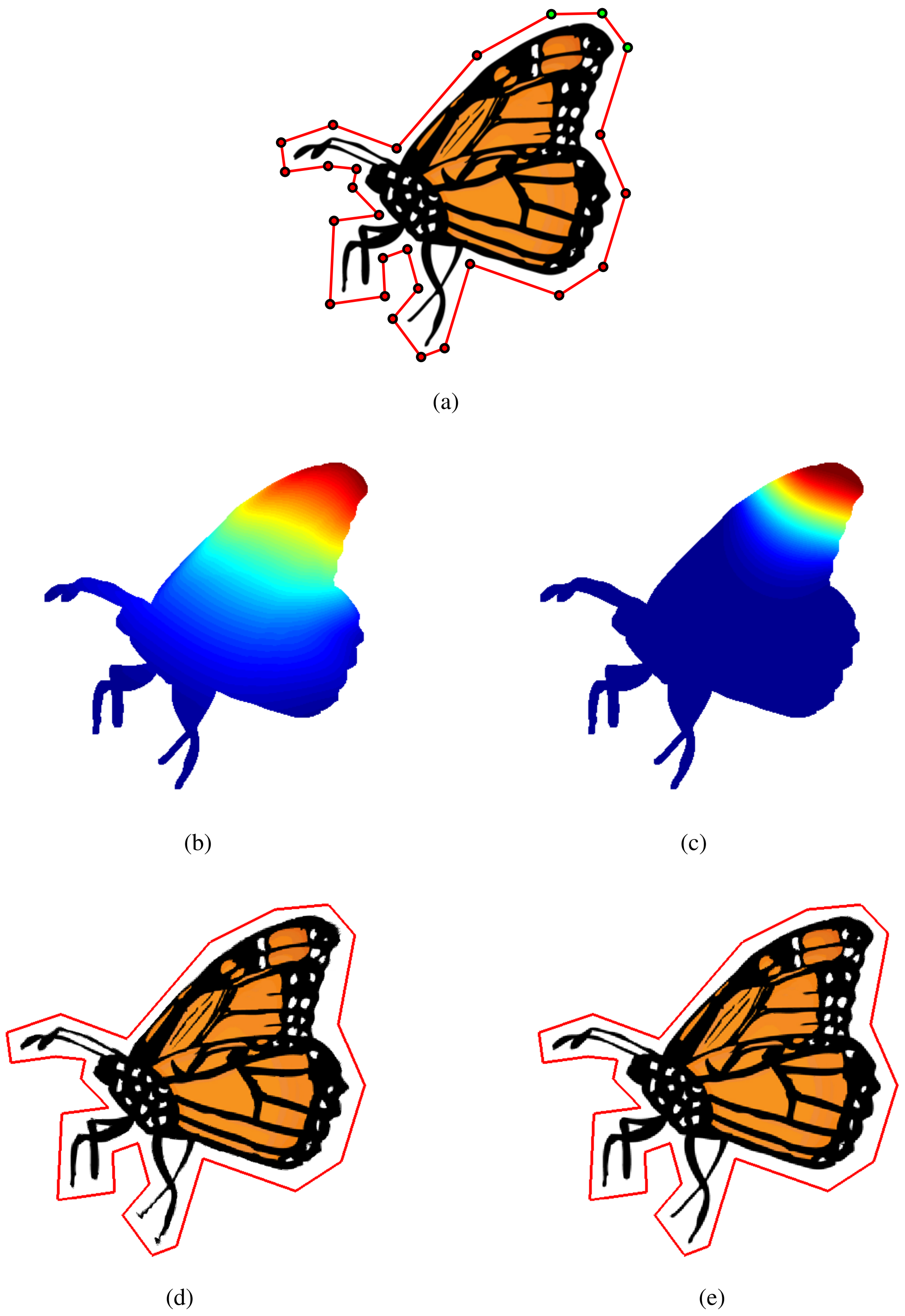


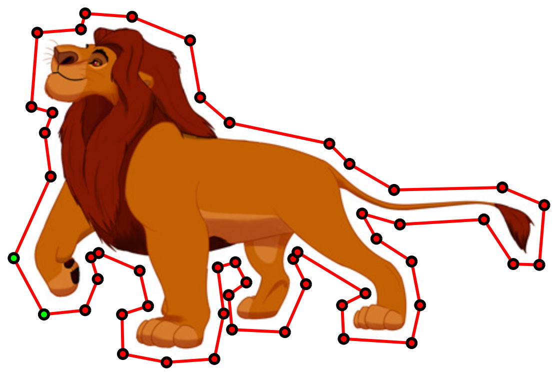
Figure 6.39: Deformation of a butterfly. (a) shows the original pose of the butterfly along with the user-defined cage around it. Control points in green are manipulated by the user. (b) and (c) demonstrate weights obtained with kernel coordinates (ℓ_1 optimization) and bounded biharmonic weights respectively, as a sum over all green control points. (d) and (e) present the deformed versions of (a) with the aforementioned methods after the three green control points have been translated.

Type of Implementation	**KC** - ℓ_1 (s)	**BBW** (s)
Serial (`for`)	11.3	29.9
Parallel (`parfor`)	9.2	11.4

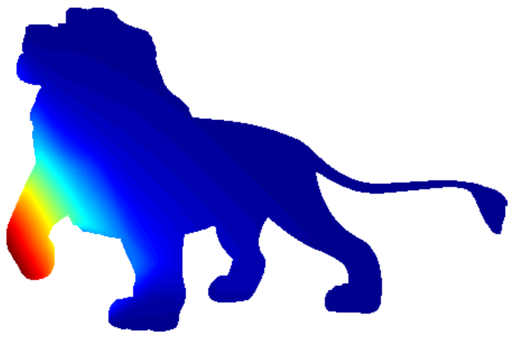
Table 6.7: Computation timings in MATLAB for kernel coordinates (KC) with ℓ_1 optimization and bounded biharmonic weights (BBW), using the butterfly shape of Figure 6.39.

while for kernel coordinates, computation is parallelized at the level of point samples. Our method is faster in both cases, even though the overhead for setting up parallel computations in MATLAB diminishes the returns to a greater extent for our method than for bounded biharmonic weights.

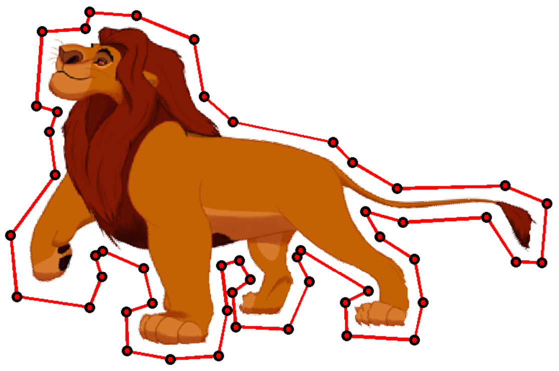
Figure 6.40 demonstrates the deformation of an image of Mufasa, using kernel coordinates with ℓ_1 optimization. Two control points near the paw of Mufasa's front right leg (marked green in Figure 6.40(a)) are translated forward and Mufasa moves his leg in that direction (Figure 6.40(c)).



(a)



(b)



(c)

Figure 6.40: Deformation of Mufasa. (a) shows the original pose of Mufasa along with the user-defined cage around him. The two control points in green are manipulated by the user. (b) demonstrates weights obtained with kernel coordinates (ℓ_1 optimization) as a sum over the two green control points. (c) presents the deformed version of (a) after the green control points have been translated.

7 Conclusion and Outlook

This thesis has presented kernel coordinates as a mesh-less method to compute generalized barycentric coordinates for control handles that are used as a metaphor to deform shapes. Our method is based on radial basis interpolation at point samples with dual kernel functions, which is leveraged to produce coordinates of the same smoothness level as that of the basis functions across the entire shape. We avoid expensive global optimization that has been employed in most previous approaches, and optimize separately for weights of each point, penalizing deviation from an initial value which is obtained with a simple heat diffusion process from cage elements. This alternative, decoupled optimization is amenable to parallel implementation, which provides a significant advantage in speed compared to the aforementioned methods and paves the way for interactive processing of weights by the user. More thoroughly, the main contributions of the thesis are the following:

- The theoretical framework for radial basis interpolation using dual kernel functions is presented and its generic properties are established, focusing on the Gaussian kernel. We provide proofs for direct "inheriting" of these properties by interpolated coordinate functions, and determine the optimal empirical value of the standard deviation of the Gaussian RBF for the case of regular sampling.
- The conditions under which local extrema can occur in radial basis interpolation with Gaussian kernel are examined, providing indicative examples of interpolated functions that give rise to this behavior.
- A closed-form expression is derived for the Jacobian matrix of the mapping that is induced by using kernel coordinates as weights for linear blend skinning.
- An efficient heat diffusion algorithm is defined to initialize weights at sampling locations in a shape-aware fashion and guide subsequent optimization properly, ensuring linear weights on the cage boundary.
- We explore various formulations of weight optimization at individual points, experiment-

ing with different objectives and sets of constraints. The quadratic objective combined with pure equality constraints (which correspond to reproduction and partition of unity) after omitting non-negativity constraints admits the fastest solution, using the closed form that we have derived. However, this comes at the expense of weights assuming negative values of large magnitude at certain parts of the shape, causing unintuitive deformations. Proper quadratic programming including non-negativity as inequality constraints is far more expensive, yet the resulting weights are guaranteed to be non-negative and they vanish away from the respective control handles, achieving better locality. A similar behavior is also observed for weights obtained using the ℓ_1-norm cost without non-negativity constraints. This optimization problem can be recast to a linear program which is solved faster than the aforementioned quadratic program, and the resulting weights are close to being non-negative (any negative weight has negligible magnitude).

- Constraints on weights for preservation of certain structural features are determined. We derive linear variation of weights at points along lines which should be preserved under translation-only transformations at handles. For parts of the object that should behave rigidly, we implement soft constraints by minimizing a quadratic energy that couples weights of all points inside or close to these parts and penalizes large gradients.
- We use a sampling algorithm based on inverse mean shift to distribute points uniformly in the interior of the cage, so that they are in congruence with samples spaced regularly on cage elements.
- An interface is implemented in MATLAB for interactive 2D shape deformation. Its key features include interactive cage construction, interactive manipulation of weights by the user, change of optimization method, application of translations and rotations to control handles for generation of new poses, and a reset option to restore the original pose of the shape. This interface has been used to create the deformation results presented in the thesis.

There are several interesting directions for future work in kernel coordinates. First, a more thorough analysis of the effect of optimization on initial weights computed via heat diffusion can be performed, examining the conditions under which interior locality of the weights is preserved after imposing the property-related constraints. In addition, different radial basis functions may be used for interpolation instead of the Gaussian on which we have focused our method, exploring the suitable range for their respective parameters. Last, we have exploited parallelism for optimization only at a CPU core level, which limits the relative speed-up to the number of cores of the machine. There is further margin for improving speed by leveraging thread-level parallelism on a GPU.

Bibliography

[1] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, “Bounded biharmonic weights for real-time deformation,” *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, vol. 30, no. 4, pp. 78:1–78:8, 2011.

[2] Y. Wang, A. Jacobson, J. Barbic, and L. Kavan, “Linear subspace design for real-time shape deformation,” *ACM Trans. Graph.*, vol. 34, no. 4, pp. 57:1–57:11, July 2015.

[3] J. Zhang, B. Deng, Z. Liu, G. Patanè, S. Bouaziz, K. Hormann, and L. Liu, “Local barycentric coordinates,” *ACM Trans. Graph.*, vol. 33, no. 6, pp. 188:1–188:12, November 2014.

[4] C. Lessig, M. Desbrun, and E. Fiume, “A constructive theory of sampling for image synthesis using reproducing kernel bases,” *ACM Trans. Graph.*, vol. 33, no. 4, pp. 55:1–55:14, July 2014.

[5] M. S. Floater, “Mean value coordinates,” *Comput. Aided Geom. Des.*, vol. 20, no. 1, pp. 19–27, March 2003.

[6] T. Ju, S. Schaefer, and J. Warren, “Mean value coordinates for closed triangular meshes,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 561–566, July 2005.

[7] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, “Harmonic coordinates for character articulation,” *ACM Trans. Graph.*, vol. 26, no. 3, July 2007.

[8] T. Igarashi, T. Moscovich, and J. F. Hughes, “As-rigid-as-possible shape manipulation,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1134–1141, July 2005.

[9] A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine, “Fast automatic skinning transformations,” *ACM Trans. Graph.*, vol. 31, no. 4, pp. 77:1–77:10, July 2012.

[10] Z. Levi and D. Levin, “Shape deformation via interior RBF,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 7, pp. 1062–1075, July 2014.

[11] R. Poranne and Y. Lipman, "Provably good planar mappings," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 76:1–76:11, July 2014.

[12] N. Mitra, M. Wand, H. Zhang, D. Cohen-Or, and M. Bokeloh, "Structure-aware shape processing," in *Eurographics State-of-the-art Report*, 2013, pp. 175–197.

[13] R. Gal, O. Sorkine, N. Mitra, and D. Cohen-Or, "iWires: An analyze-and-edit approach to shape manipulation," *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, vol. 28, no. 3, pp. 33:1–33:10, 2009.

[14] S. Schaefer, T. McPhail, and J. Warren, "Image deformation using moving least squares," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 533–540, July 2006.

[15] M. Balzer, T. Schlömer, and O. Deussen, "Capacity-constrained point distributions: A variant of Lloyd's method," *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, vol. 28, no. 3, pp. 86:1–86:8, 2009.

[16] E. Candès and J. Romberg, "l1-MAGIC: Recovery of sparse signals via convex programming," October 2005.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

KERNEL COORDINATES

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
SAKARIDIS	CHRISTOS

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 02.04.2016	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.