Nonlinear Diffusion Filtering On Surfaces

Harry Biddle

Hertford College

University of Oxford

A dissertation submitted for the degree of

Master of Science in Mathematical Modelling and Scientific Computing

Trinity 2011

Abstract

Nonlinear diffusion filtering is a PDE-based method to remove noise from images that has found much success. This dissertation looks at whether nonlinear diffusion filtering can be combined with the *closest point method*, a relatively new and novel method for solving partial differential equations on surfaces. The closest point method is an embedding method that uses a simple representation of surfaces. The theory and implementation of for the closest point method is presented. We perform convergence studies that show good agreement with theory.

We discuss the use of linear and nonlinear diffusion for image processing, in particular the Perona–Malik and Gaussian schemes. We show that they can be combined with the closest point method to produce impressive results, visualised beautifully using an OpenGL raytracer designed for use with the closest point method. Some surprising and unexpected effects were discovered when moving from a plane to a three-dimensional surface. These effects are described and investigated.

This is dedicated to my parents, Stephen and Philippa,
to whom I owe everything.

Acknowledgements

I would like to extend my sincerest gratitude towards my two supervisors, Dr Colin Macdonald and Dr Tom März. Their positivity towards the project was contagious and they were always able to point me in the right direction, even despite the fact that I repeatedly misspelled their names. Hopefully the supercomputer will arrive soon.

I would also like to thank Ingrid Von Glehn for her helpful comments and encouragement each and every coffee morning, even if there was a crossword to be done.

I am eternally grateful for the opportunity given to me to have studied this course, and the support OCCAM has given me throughout the year.

This publication was based on work supported in part by Award No KUK-C1-013-04, made by King Abdullah University of Science and Technology (KAUST).

Contents

1 Introduction **1**

2 Partial Differential Equations on Surfaces **3**

2.1 Differential Operators on Surfaces 4

2.2 The Closest Point Representation of a Surface 4

2.3 Equivalence of Differential Operators 6

2.4 The Closest Point Method 8

2.4.1 The Fully-Discrete Explicit Scheme 9

2.4.2 Interpolation 10

2.4.3 Practical Implementation 12

2.4.4 Boundary Conditions 14

2.4.5 Accuracy 14

2.5 Numerical Studies: The In-Surface Heat Equation on a Circle and on a Sphere 16

3 Image Processing **18**

3.1 Gaussian Diffusion 20

3.2 Perona–Malik Diffusion 23

3.2.1 The Average Grey Level 25

3.3 Numerical Schemes for Perona–Malik Diffusion 26

3.4 Stability of the Morton and Mayers Scheme 29

4 Perona–Malik Diffusion on Surfaces **31**

4.0.1 Choosing the Bandwidth 32

4.0.2 A Raytracer for Three-Dimensional Visualisation 32

4.1 Numerical Observations 33

4.1.1 Perona–Malik Diffusion on a Circle 33

4.1.2 Edge Preservation: The Beachball 34

4.1.3 A Complicated Surface: The Stanford Bunny 36

4.2 Discussion 40

4.2.1 Grid Effects 42

4.2.2 Artefacts in the Beachball 44

4.2.3 Dropping the Interpolation Step 45

5 Conclusion **48**

References **49**

List of Figures

2.1 The closest point function may be ill-defined arbitrarily close to the surface 5

2.2 The closest point function 10

2.3 Creating a two-dimensional interpolation scheme from one-dimensional schemes 11

2.4 Convergence studies for the heat equation on the unit circle and the unit sphere 17

3.1 The *Swan* test image 19

3.2 The heat equation realised as a low-pass filter. 22

3.3 The effect of the heat equation on the frequency spectrum of a real image 22

3.4 A comparison of the effect of Gaussian diffusion and Perona–Malik diffusion on the frequency spectrum of a real image 25

3.5 The stencil for the two-dimensional Morton and Mayers scheme. 29

3.6 Gaussian diffusion and Perona–Malik diffusion on *Swan* 30

4.1 Comparing Perona–Malik on the bar and on the circle 35

4.2 Denoising a beachball 37

4.3 The variation of the average grey level for the beachball is negligable 38

4.4 Artefacts on great circles of the sphere 38

4.5 Denoising the zebra bunny 41

4.6 Investigating changing grid size with the one-dimensional Morton and Mayers scheme for Perona–Malik 43

4.7 Perona–Malik on the circle with a uniform initial condition . . . 44

4.8 Dropping the interpolation step when denoising a beachball . . . 47

4.9 Perona–Malik on the circle *without interpolation* with a uniform initial condition 47

Chapter 1

Introduction

The use of PDEs in image processing is a well-established field. Decades of research has produced a wide variety of methods to solve such problems as image denoising, image inpainting, and image segmentation [39]. With such methods well-explored in the Euclidean plane, focus has started to turn towards higher dimensions and the surfaces embedded in them.

In [8, 4] anisotropic diffusion is used to smooth irregularities in surfaces themselves, as well as functions defined on them. In [34] the closest point method is combined with an image segmentation procedure. In a very recent paper by Lai and Chan [16] a framework is presented for solving variational models for image processing on surfaces.

Work on solving partial differential equations on surface can be largely divided into two classes: those that use explicit representations of surfaces, and those that use implicit representations. Explicit representations typically begin with surfaces represented by polygon meshes, in particular triangle meshes. Research in explicit methods can be found for example in [29, 14]. Typically, patch-wise parametrisations are computed for the surface in a preprocessing stage, and the intrinsic differential operators are computed under them.

However, parameterised representations quickly become complicated. Furthemore, the scheme that arises is very problem-dependent; analysis must be carried out for each new surface. The situation is worse when we only have a set of triangle data for the surfaces; no global parametrisation is available and we can only form local approximations with some piecewise parametrisation.

The second class of methods, those which rely on implicit representations, define a surface by the zero contour of some function defined on a Euclidean domain, often a signed distance function. Differential operators on the surface are realised via a projection along the normal direction. Research involving implicit methods can be found in [28, 5, 7]. The biggest advantage of implicit methods is the ease with which changing topologies can be handled.

The closest point method [32] is a recent method that uses an implicit representation of the surface; storing for each point in space the coordinates of the closest point to it on the surface. Its advantage is its simplicity and its ability to handle a large class of surfaces, with or without boundaries. The closest point method is an *embedding* method that extends computations into a narrow band around the surface. We wish to use the closest point method to process images defined on surfaces using PDEs.

Of course, there is an issue of what we mean by an image on a surface. The sort of problem that we consider is that of an image defined on voxels. This could arise, for example, in medical imaging, where an ultrasound or magnetic scan of the body produces slices of data that can be combined into voxel data. The underlying surface could be the body organ of interest, on which we wish to extract and denoise the data.

The problem of denoising images in dimensions higher than two is not new. Three-dimensional nonlinear diffusion filters date back to at least 1992 [12]. In [9] nonlinear diffusion filtering for on voxels is achieved by treating the data as four-dimensional data sets. However, neither of these papers consider an underlying surface embedded within the data set.

The contribution made by this dissertation is in answering the following question: can nonlinear diffusion filtering be combined with the closest point method, to form an efficient and reliable tool for image processing on surfaces? Chapter 2 lays out the groundwork for defining what we mean by partial differential equations on surfaces, and presents the explicit closest point method as a way to solve them numerically. Chapter 3 introduces the use of linear and nonlinear diffusion to denoise images, and presents numerical schemes to implement them. Chapter 4 shows how linear and nonlinear diffusion can be combined with the closest point method and the interesting effects that arise from doing so. Chapter 5 concludes.

Chapter 2

Partial Differential Equations on Surfaces

The need for solving partial differential equations on surfaces is demanded by a wide variety of fields. For example, computer graphics [36, 42, 28, 10], fluid dynamics [25, 26], and biology [27]. PDEs on surfaces can model processes evolving over static surfaces, over surfaces with changing topology, or can determine the evolution of the surfaces themselves.

This dissertation concerns partial differential equations that govern the evolution of some function u defined on a static surface $\mathcal{S}$ embedded in Euclidean space $\mathbb{R}^d$. Such equations make use of *intrinsic* or *surface* differential operators. These operators mix the differentiation with the geometry of the surface itself. Extricating the geometry from these operators is a non-trivial task. It furthermore depends very much on the representation we choose for the surface.

The closest point representation is a simple representation of surfaces that can be analytically computed for many simple geometries, or numerically computed for more general ones. We will show that this representation couples easily with a wide variety of PDEs to produce a powerful tool for solving PDEs on surfaces numerically, called the closest point method. We will outline an implementation on a computer and discuss practical issues.

In this dissertation we restrict ourselves to surfaces of codimension one, although the closest point method is not limited to such surfaces [32].

2.1 Differential Operators on Surfaces

Suppose we have a surface $\mathcal{S} \subset \mathbb{R}^d$ with outward-pointing unit normal n, and a function $u : \mathcal{S} \times [0, \infty) \to \mathbb{R}$ that we would like to model the evolution of. We can define a *surface gradient* operator intrinsic to this surface as the *orthographic projection* of the gradient on to the surface

$$\nabla_{\mathcal{S}} u = \nabla \hat{u} - n(n \cdot \nabla \hat{u}). \tag{2.1}$$

Here $\hat{u}$ is any differentiable extension of u across $\mathbb{R}^d$. In particular, the vector $\nabla_{\mathcal{S}} u$ always lies tangent to the surface. We may similarly define the intrinsic analogue of the Laplacian operator. This is known as the *Laplace-Beltrami operator* and is denoted $\Delta_{\mathcal{S}}$. Rigorous definitions of such operators and other extensions involve discussion of Riemannian manifolds, metrics, tensors, and plenty more differential geometry that we avoid here. A discussion of such topics in the context of partial differential equations on surfaces can be found in [16].

Armed with such operators, we are able to define partial differential equations that hold on surfaces themselves. For example, a typical PDE would be of the form

$$\frac{\partial u}{\partial t} = f(q, u, \nabla_{\mathcal{S}} u, \Delta_{\mathcal{S}} u), \text{ for } q \in \mathcal{S}. \tag{2.2}$$

2.2 The Closest Point Representation of a Surface

The closest point representation of a surface $\mathcal{S}$ is defined over the space in which it is embedded. If we consider $\mathcal{S}$ to be a subset of points taken from $\mathbb{R}^d$, then $\mathbb{R}^d$ is the embedding space. For example, a sphere is a two-dimensional in that it requires two independent variables to parametrise it, yet lives in $\mathbb{R}^3$. So the embedding space for a sphere is $\mathbb{R}^3$.[1]

The closest point representation proceeds as follows: let cp be a function mapping points in $\mathbb{R}^d$ to points lying on the surface $\mathcal{S}$ such that for each

[1] Note that we are restricting ourselves to surfaces of codimension one.

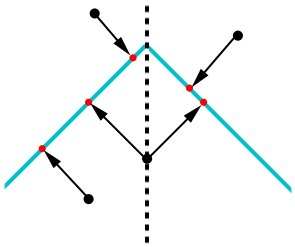


Figure 2.1: A surface (blue) with sharp edges. The closest point function may be ill-defined arbitrarily close to the surface.

$x \in \mathbb{R}^d$, $\mathrm{cp}(x)$ is the *closest point* to x on the surface. That is,

$$\mathrm{cp}(x) = \arg\min_{q \in \mathcal{S}} \|x - q\|. \tag{2.3}$$

We call $\mathrm{cp}(x)$ the *closest point function*. The surface $\mathcal{S}$ is then represented implicitly as the zero-contour of the function $x \mapsto \|\mathrm{cp}(x) - x\|$. Note that the choice of norm is arbitrary; in this dissertation the Euclidean norm will always be used.

Unfortunately, the closest point representation is not, in general, everywhere defined. The prototypical example is a circle centered at the origin. The origin is equidistant from all points on the circle, so we have no way of selecting which point on the circle the closest point function should map to.

We will see later that we do not need the closest point operator on the whole embedding space, but rather only on a narrow band around the surface. For sufficiently smooth surfaces, the closest point function is defined and smooth near the surface. In practice, such equidistant points may place an upper limit on grid size [32]. For insufficiently smooth surfaces surfaces—for example those with sharp corners—the closest point function may be ill-defined arbitrarily close to the surface (Figure 2.1). In this case, one can arbitrarily choose which closest point to take. This can result in the solution 'jumping' across the corner.

The closest point representation may be available to use analytically, as in the case of simple shapes such as circles or spheres. If the surface is given as triangular data, a minimisation procedure can be used. Research in dealing with such surfaces is ongoing. It should, however, be emphasised that in the

closest point method, the closest point representation need be computed only *once* for a given surface. Thereafter, it can be used to compute any partial differential equation on that surface, with any starting data.

2.3 Equivalence of Differential Operators

The closest point method is an *embedding method*. Suppose we are trying to solve a PDE of the form (2.2). We first form a closest point representation $\text{cp}(\cdot)$ of the surface. Then we *extend* the solution u to the embedding space $\mathbb{R}^d$ by defining a function $v = u \circ \text{cp}$. This function is constant in normal directions to the surface. We refer to this extension as a *closest point extension*. The key idea is that the *gradient and divergence of v match the corresponding surface operators of u on the surface itself*. Hence it is prudent to study the following PDE, now in the embedding space

$$\frac{\partial v}{\partial t} = f(\text{cp}(x), v, \nabla v, \Delta v). \tag{2.4}$$

We can solve this PDE using standard numerical techniques designed for Euclidean spaces. Finally, so long as the equivalence $v = u \circ \text{cp}$ is maintained, we can simply restrict v to the surface to find our solution.

To reiterate: we first form a closest point representation of the surface. This representation is used to extend the solution into the embedding space of the surface, a process known as the closest point extension. As the differential operators of this extended solution match the surface operators of the original problem at the surface, we hope to numerically solve the PDE in the embedding space and restrict the resulting solution to the surface.

The aforementioned equivalence of the differential operators at the surface follows easily from two principles of differential geometry, as presented. They are given here as presented in [32].

Principle 1. *Suppose $v : \mathbb{R}^d \to \mathbb{R}$ is a scalar-valued function that is constant along directions perpendicular to $\mathcal{S}$. Then at the surface*

$$\nabla_{\mathcal{S}} v = \nabla v. \tag{2.5}$$

This is not a surprising result; a function constant in the normal direction changes only along the surface. This is obvious from the form $\nabla_{\mathcal{S}} v = \nabla v - n(n \cdot \nabla v)$. The second term is a directional derivative normal to the surface and so disappears.

A similar result holds for the surface divergence.

Principle 2. *Suppose $w : \mathbb{R}^d \to \mathbb{R}^d$ is a vector-valued function that is tangent to $\mathcal{S}$ at the surface itself and is furthermore tangent to all surfaces displaced a fixed distance from $\mathcal{S}$. (That is, the level sets of the distance function to $\mathcal{S}$.) Then at the surface*

$$div_{\mathcal{S}} w = div w. \tag{2.6}$$

Again, this principle has an intuitive basis: a flux which is tangent to the surface only spreads out along the surface itself.

Although simple, these two principles lie at the heart of the closest point method. We are now ready to justify the equivalence of some important differential operators at the surface.

Proposition 1. *Suppose $u : \mathcal{S} \to \mathbb{R}$ is a scalar-valued function defined on the surface. Let $v = u \circ cp$ be the closest point extension on the embedding space $\mathbb{R}^d$. Then the following equalities hold at the surface:*

1. $\nabla_{\mathcal{S}} u = \nabla v$.
2. $\Delta_{\mathcal{S}} u = \Delta v$.
3. $div_{\mathcal{S}} \left(a(u, \nabla_{\mathcal{S}} u) \nabla_{\mathcal{S}} u \right) = \nabla \cdot \left(a(v, \nabla v) \nabla v \right)$.

The first equality is immediate, since clearly $\mathrm{cp}(\cdot)$ (and hence v) is a function constant in directions normal to the surface. For the second, note that ∇v is always tangent to the level sets of the distance function of $\mathcal{S}$. Since $\Delta v = \nabla \cdot \nabla v$, applying the second principle gives the desired result.

The third equality concerns a generalised diffusive term for the heat equation that will come in useful later when studying the Perona–Malik equation. Since $a(v, \nabla v)$ is constant in normal directions to $\mathcal{S}$, the product with ∇v is still

tangent to the level sets of the distance function of $\mathcal{S}$, and the second principle again gives the desired result.

For a more rigorous treatment of the above, see [23].

This proposition is powerful in that it justifies going straight from a PDE defined on the surface—such as (2.2)—to an embedding PDE—such as (2.4)—by simply dropping the subscripts on the differential operators.

Analogous results hold for a wide range of generalisations and operators. In particular, higher-order operators such as the biharmonic operator can be handled with multiple closest point extensions. For a discussion of this see [32].

2.4 The Closest Point Method

We now investigate how to build a numerical scheme, called the *closest point method*, from the above discussion.

In this paper we focus on schemes with explicit timestepping. Suppose that we have an initial condition u_0 defined on the surface. The problem we wish to solve is

$$\frac{\partial u}{\partial t} = f(q, u, \nabla_{\mathcal{S}} u, \Delta_{\mathcal{S}} u), \qquad \text{for } q \in \mathcal{S}, \tag{2.7a}$$

$$u(q, 0) = u_0(q), \qquad \text{at } t = 0. \tag{2.7b}$$

We have already seen in the previous section the equivalence on the surface of the differential operators in the embedding space. Hence we turn our attention to the embedding PDE, with extended initial data

$$\frac{\partial v}{\partial t} = f(\text{cp}(x), v, \nabla v, \Delta v), \qquad \text{for } x \in \mathbb{R}^d, \tag{2.8a}$$

$$v(x, 0) = (u_0 \circ \text{cp})(x), \qquad \text{at } t = 0. \tag{2.8b}$$

Unlike the surface PDE, this equation lends itself easily to numerical techniques designed for Euclidean spaces. Discretising the time-derivative by finite differences of spacing δt gives the *semi-discrete* form (discrete in time, continuous in space) of an explicit scheme

$$v^{m+1} = v^m + \delta t f(\text{cp}(x), v^m, \nabla v^m, \Delta v^m). \tag{2.9}$$

The initial data for v is extended from the initial data from u. Since it is constant in normal directions to the surface, the right-hand sides of (2.7a) and (2.8a) agree. Hence after one timestep, the values of v^1 on the surface will precisely match the values of u^1 that we would have got if we had applied the semi-discrete scheme (2.9) to u. Unfortunately, we are unable to apply the same argument to subsequent timesteps, since we can no longer rely on the key property that the data is constant in normal directions to the surface.

To recover this property, we simply re-extend the values of v^1. The values off the surface—which we are uninterested in—are overwritten, and the values on the surface are unchanged. The benefit is that now we can take another explicit timestep, and be satisfied that the solution on the surface is in accordance with the semi-discrete scheme for u.

To summarise, the semi-discrete closest point scheme proceeds as follows:

1. Perform a closest point extension of the initial condition into the embedding space.
2. Evolve one explicit timestep in the embedding space.
3. Re-extend the values obtained.
4. Repeat Steps 2–3 until desired time is reached.

2.4.1 The Fully-Discrete Explicit Scheme

To fully discretise the equation, we would like to replace the second term in (2.9) with a spatial discretisation. Suppose that such a spatial discretisation is available to us. For example, we might like to replace derivatives with central finite differences. The choice of such a scheme is dependent on the form of the embedding PDE, and can be picked from the healthy selection already provided by the field of Numerical Analysis.

Armed with a numerical scheme in space and time, we construct a grid in the embedding space $\mathbb{R}^d$. The derivatives of v are replaced with numerical approximations, and with a small enough grid are sufficiently close to the true values. Hence we may simply step forward explicitly in time, re-extending our solution after each step, and in so doing obtain a consistent numerical scheme on the surface $\mathcal{S}$.

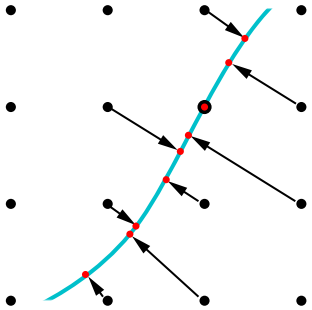


Figure 2.2: The mapping of grid points x_i (black) to their closest points $\text{cp}(x_i)$ (red) on the surface (blue). In general, neither grid points nor their closest points lie on the surface.

However, there remains a final hurdle. Closest points on the surface for each grid point do not in general themselves fall on another grid point—see Figure 2.2. Hence we are unable to extract values on the surface that are needed for the various closest point extensions. Instead, we must use an interpolation scheme. It turns out that the choice of interpolation method is key to the order of convergence of the scheme. This is discussed in Section 2.4.5.

For an explicit scheme, then, the procedure is as follows:

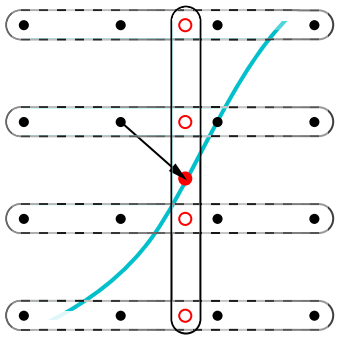
1. Perform a closest point extension of the initial condition into the embedding grid.
2. Evolve one explicit timestep in the embedding grid.
3. Re-extend interpolated surface values across the grid.
4. Repeat Steps 2–3 until desired time is reached.

Multi-stage Runge–Kutta schemes can be similarly handled, with a re-extension following each stage [18].

2.4.2 Interpolation

In a interpolation scheme, we look to find a function $f(x)$ passing through a set of data points v_j at nodes x_j, so that we can extract realistic values at some arbitrary point x that is not necessarily a node. In this dissertation we use polynomial interpolation, although schemes such as the essentially non-oscillatory (ENO) and weighted essentially non-oscillatory (WENO)—which

Figure 2.3: Creating a two-dimensional interpolation scheme from one-dimensional schemes. The desired point (red dot) is the closest point function of one of the grid points. We choose neighbouring nodes, perform $(p+1)$ interpolations horizontally, then use those values for an interpolation vertically.



handle piecewise smooth yet discontinuous solutions—have also been employed with the closest point method [18, 32].

In the closest point method, interpolation is required at the re-extension stage. For each grid point x_j, we wish to replace the value u_j with the interpolated value on the surface, namely at $\mathrm{cp}(x_j)$. Since we do this at each timestep, our interpolation scheme must be cheap computationally, yet be able to handle changing data on a fixed grid.

For polynomial interpolation, we seek to find a polynomial function $f(x)$, which takes values from our embedding space $\mathbb{R}^d$. To do this, we need only study interpolation in one spatial dimension, since higher-dimensional interpolation can be built from repeated application of a one-dimensional scheme. This is illustrated in Figure 2.3.

In one-dimension, providing $p+1$ points and seeking a p-th order polynomial is a well-posed problem. A classic representation of the (unique) solution is the *Lagrange representation*,

$$f(x) = \sum_j v_j \ell_j(x), \quad \text{where } \ell_j(x) = \frac{\Pi_j(x - x_k)}{\Pi_{k \neq j}(x_j - x_k)}. \tag{2.10}$$

The *Lagrange polynomials* ℓ_j have the property that they are one at the i-th node, and zero at all the other nodes. Since we know in advance precisely where we wish to evaluate these polynomials, we may precompute all the values $\ell(x)$, and they may thereon—in particular, during the timestepping loop—be considered as real coefficients of a linear sum in v_j.

In practice, polynomial interpolation by constructing such Lagrange polynomials is both expensive and unstable. In this dissertation, we use *barycentric interpolation*. Barycentric interpolation is an elegant rearrangement of (2.10) that is both fast and stable. Barycentric interpolation involves a precomputation stage to generate the so-called barycentric 'weights', followed by an evaluation in linear time. Importantly, the precomputed weights are independent of the data v_j. This makes barycentric interpolation particularly useful for the closest point method, where data is updated on the embedding grid after every timestep.[2] An excellent review of barycentric interpolation is found in [35].

In higher dimensions d, we can combine one-dimensional schemes as in Figure 2.3. For example, building a two-dimensional scheme from order one schemes would lead to a bilinear scheme, from order two biquadratic, and so on. Similarly to before, such an interpolation is simply a linear combination of values from $(p+1)^d$ grid points surrounding the point of interest.

In other words, if we wish to overwrite the value v_j with the interpolated value at the surface, we can simply take a linear combination of neighbouring data points. Therefore, if the data points v_j are stored in a vector, the interpolation and re-extension step can be stored as a matrix E. Moreover, this matrix can be constructed in the preprocessing stage.

2.4.3 Practical Implementation

We now bring together exactly how the explicit closest point method is performed on a computer. Firstly, for each grid point x_j we store the closest point $\mathrm{cp}(x_j)$. This provides all the information about the surface that the scheme requires. To re-emphasise: different surfaces are handled only by changing the closest point function; the code thereafter remains the same.

Knowing these points, we are able to construct our interpolation matrix E. If our numerical scheme in space is simply a linear combination of values at grid

[2] Although they mention barycentric interpolation, Ruuth and Merriman [32] use a Newton divided differences scheme. In such a scheme, a tableau of precomputed recursive differences is inserted into a simpler polynomial that can be evaluated in linear time. Crucially, however, the *precomputed quantities depend on the data* v_j [35]. It would seem rather expensive to recompute this tableau upon the arrival of new data—namely, in the timestepping loop.

points, for example as in finite differences, we can also store it as a matrix L. If $\mathbf{u}^m$ is the vector whose entries approximate the solution at a time $m\delta t$, then timestepping and re-extension can simply be achieved by the following two lines of pseudocode:

$$\tilde{\mathbf{u}}^{m+1} := \mathbf{u}^m + \delta t \mathsf{L}\mathbf{u}^m, \tag{2.11}$$

$$\mathbf{u}^{m+1} := \mathsf{E}\tilde{\mathbf{u}}^{m+1}. \tag{2.12}$$

This second line is an additional cost in the closest point method, that is not in the usual Euclidean scheme. Is it expensive? Possibly. Typically E has many more entries per row than L. For example, with a seven-point stencil in three dimensions, L has seven entries per row. If we were using trilinear interpolation ($p = 3$), then E would have 64 entries per row.

The biggest preprocessing overheads are found in building the closest point function—at least where no analytical function is known—and building the interpolation matrix E. It should be emphasised that these are *pre*processing stages and are only done once.

Naturally we can make huge computational gains by making the embedding grid as small as possible. Depending on the order of the numerical scheme and the degree of the interpolation, we need only a small band of grid points around the surface. Such a restriction is known as *banding*. An equation for precisely the bandwidth required is given in [32]. Specifically, for a scheme in d-dimensions demanding grid points only one index away (what would be a five-point stencil in two dimensions), with grid spacing at a maximum δx, and degree p interpolation polynomials, the bandwidth required is

$$\delta x \sqrt{(d-1)\left(\frac{p+1}{2}\right)^2 + \left(1 + \frac{p+1}{2}\right)^2}. \tag{2.13}$$

To summarise, the explicit closest point method—as implemented in this dissertation—proceeds as follows:

1. Preprocessing stage:

   (a) Calculate bandwidth and discard grid points outside of the band.

   (b) At each grid point x_i in the band, store the coordinates $\mathrm{cp}(x_i)$ of

the closest point on the surface.

(c) Build the matrix E that interpolates data at grid points in the band onto the surface.

(d) Extend the initial condition into the band.

2. Processing stage, repeat until desired time is reached:

(a) Evolve one explicit timestep in the band.

(b) Re-extend interpolated surface values across the band.

A MATLAB script was built from scratch to implement the above ideas without banding. To include banding, a repository of MATLAB code which builds interpolation and differentiation matrices, built at OCCAM, was both used and contributed to during this dissertation.

2.4.4 Boundary Conditions

So far no mention has been made of boundary conditions. If we choose our surface to be without boundaries, then, at least for evolution equations such as (2.2), the only condition required is an initial profile. However, by introducing a computational band, we have introduced boundary points at its edges. There remains the question of how we populate their values.

For the explicit schemes considered in this dissertation, the choice of boundary conditions is of no consequence. Boundary values will be overwritten when we re-extend the interpolated solution, and hence any boundary values will encroach no more than one grid point into the embedding grid.

All the surfaces in this dissertation are closed surfaces. The closest point method can, however, also be applied to surfaces with boundaries. For example, a hemisphere. On such surfaces adequate boundary conditions fit for the surface PDE must be posed. These conditions can then be propagated into the embedding space using the closest point function [19].

2.4.5 Accuracy

It is a great feature of the closest point method that we are able to retain numerical schemes designed for Euclidean domains. One would hope that in

doing so, the order of accuracy of the schemes are preserved. In the closest point method, errors arise in three places:[3] from the time discretisation, the space discretisation, and the interpolation.

Suppose $\mathbf{u}$ is our numerical solution vector, and $\mathbf{x}$ the vector of grid points. We wish in particular to interpolate $\mathbf{u}$ onto the surface to approximate $u \circ \mathrm{cp}$. It is known that when interpolating smooth data on nodes separated by a distance δx, a p-th order interpolant introduces errors of order δx^{p+1} [33]. This can be loosely written as

$$(u \circ \mathrm{cp})(\mathbf{x}) = \mathsf{E}\mathbf{u} + \underbrace{\mathcal{O}(\delta x^{p+1})}_{\text{interp. err.}}, \tag{2.14}$$

where the second (scalar) term is understood to be added to each entry of the preceding vector. Suppose for example we wish to solve the heat equation $u_t = \Delta u$. First take the Laplacian of the above. Taking two derivatives of the right-hand side modifies the second term to $\mathcal{O}(\delta x^{p-1})$. If we then approximate the Laplacian of the first term using a second-order central finite difference operator L, we introduce an additional error $\mathcal{O}(\delta x^2)$.

$$\Delta(u \circ \mathrm{cp})(\mathbf{x}) = \mathsf{L}(\mathsf{E}U) + \underbrace{\mathcal{O}(\delta x^2)}_{\text{space disc. err.}} + \underbrace{\mathcal{O}(\delta x^{p-1})}_{\text{interp. err.}}. \tag{2.15}$$

Taking a forward time discretisation of spacing δt introduces a further error $\mathcal{O}(\delta t)$. If we choose our timestep spacing to be a multiple of δx^2, this is equivalently $\mathcal{O}(\delta x^2)$.

Therefore, in the case of the heat equation, when p is less than three the dominant error is due to the interpolation, whilst if p is greater than three the error is dominated by that of the spatial discretisation. Note in particular that p being zero or one leads to an inconsistent scheme.

In general, an order q-discretisation of a second-order differential operator requires an underlying interpolation degree of $p \geq q + 1$ to maintain the order of the spatial numerical scheme [21].

[3] Along with rounding errors in the computer. These creep, for example, into the closest point function.

2.5 Numerical Studies: The In-Surface Heat Equation on a Circle and on a Sphere

The heat equation serves as a simple test case for the closest point method, as well as being a good starting point for the image processing techniques introduced in Section 3. Two simple surfaces will be used: a circle, embedded in $\mathbb{R}^2$, and a sphere, embedded in $\mathbb{R}^3$.

Test problem for the circle. We first look for an initial condition with an easy explicit solution, against which we can test our numerical scheme. Writing the Laplacian in polar coordinates reveals that on a unit circle the problem is analogous to the one-dimensional heat equation on a finite bar with periodic boundary conditions. Such a problem can be readily solved by the use of Fourier series.

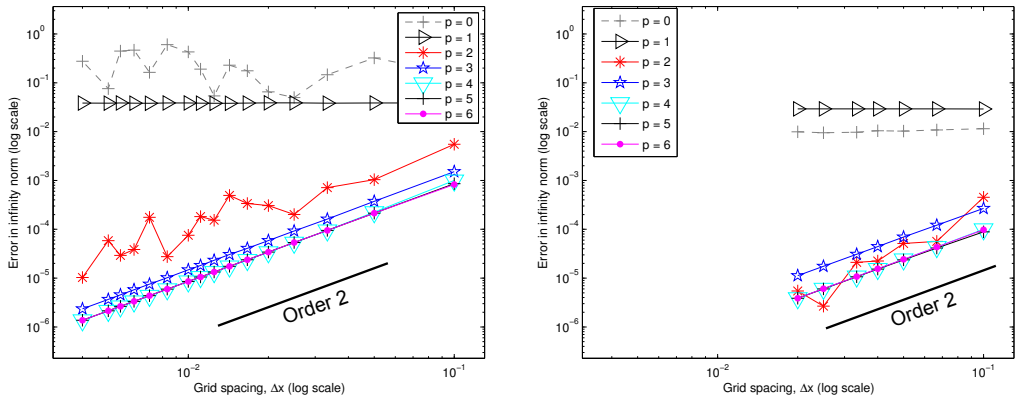
The test problem for the in-surface heat equation on a circle is taken to have initial condition $u_0(\theta) = \cos 3\theta$, for which the exact solution is $u(\theta, t) = \exp(-9t)\cos 3\theta$.

Test problem for the sphere. We look to find a similarly easy initial condition for the unit sphere. We first write the Laplacian in spherical coordinates and fix the radius r. If we find an eigenfunction (in space) of this operator, we will have a particularly simple explicit solution when we introduce time. The eigenfunctions turn out to be the spherical harmonics $Y_l^m(\theta, \phi)$, with eigenvalues $-l(l+1)$ [40].

Hence, for the in-surface heat equation on the sphere we take our initial condition u_0 to be the real part of $Y_1^1(\theta, \phi)$. The exact solution is then $u(\theta, \phi, t) = \exp(-2t)u_0(\theta, \phi)$. Whilst lower-order spherical harmonics would also have sufficed, this particular harmonic exhibits pleasant smoothness at the poles, and so sidesteps issues of junctions in the solution.

The numerical scheme for the embedding space is chosen to be an explicit Euler scheme. The size of the timesteps is chosen as a multiple of δx^2 to satisfy the stability constraint for the embedding equation.[4]

[4] Although, we notice that the use of closest point extensions typically allows the stability condition to be relaxed.



(a) Heat equation on a circle

(b) Heat equation on a sphere

Figure 2.4: Convergence studies for the heat equation on the unit circle (left) and the unit sphere (right) in the infinity norm. Various different degrees p of interpolating polynomial were used; $p = 1$ refers to bilinear/trilinear, $p = 2$ to biquadratic/triquadratic, and so on. Schemes were run up to a time $t = 1$. Timesteps are chosen as a multiple of δx^2 to ensure stability.

Figure 2.4 shows the results of the numerical simulations for various degrees p of interpolating polynomial. For p three or higher, we see good second order convergence, as expected. The theory of Section 2.4.5 suggests that bi/triquadratic interpolation is at worst $\mathcal{O}(\delta x)$, although we can see that in these test cases, the scheme outperforms this bound. Such interpolation may thus be useful where computational speed takes priority over accuracy. Notably, neither nearest-neighbour ($p = 0$) nor bilinear ($p = 1$) interpolation should be used for second-order PDEs.

Chapter 3

Image Processing

Whilst once the domain of electrical engineers and computer science, image processing is today a well-studied field in the mathematical community, drawing methods from linear algebra, numerical optimisation, functional analysis, statistics, and probability [2]. The focus of this dissertation is removing noise from an image. For images taken by either film or digital cameras, such noise from a variety of sources. The sensor of a digital camera essentially counts photons; *Schottky* (or *shot*) *noise* occurs when random variations in this counting procedure are of significance. Such noise is essentially Poisson distributed, but the numbers of photos involved are typically large enough so that one can take a Gaussian distribution. *Amplifier noise* results mainly from thermal effects causing agitation of electrons in the sensor, and also from the noise caused by the reset of capacitors. Amplifier noise is typically taken to be Gaussian distributed [22, 6].

Other types of noise include salt-and-pepper noise—a wide variety of processes that do not affect every pixel, but have a large effect on those that they do—and quantization noise–which is a uniform noise resulting from the conversion of a continuum of greyscale values to a discrete scale. In this dissertation we take our model of noise to be additive, Gaussian distributed, and affecting each pixel independently of neighbouring pixels or signal intensity.

For Gaussian noise, diffusion-like schemes are particularly helpful, since they remove high-frequency components early on in their evolution. Image smoothing using PDEs is thus typically achieved by regarding the original image as the initial condition for some parabolic process, and observing its evolution in

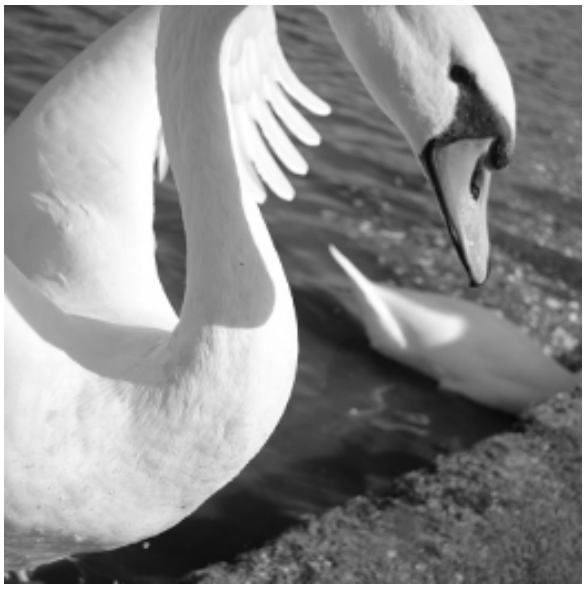


Figure 3.1: The *Swan* test image, taken by the author.

time, stopping the process when the desired result is achieved.

I begin by introducing the use of Gaussian diffusion in image processing and some key concepts, then move onto the famous Perona and Malik diffusion, summarising the theory and proposing some numerical schemes. For a summary of modern image processing using PDEs, see [39] or [38]. Throughout we frequently use a test image referred to as *Swan* (Figure 3.1).

Suppose that we begin with a greyscale two-dimensional image $I(x)$ in some rectangle $\Omega \subset \mathbb{R}^2$, where the value of I represents the amount of white at that pixel. A true image will consist of discrete pixels, but we think of it as on a continuous rectangle for the time being.

This image is assumed to be a faithful representation of the intended target. To represent image noise, we add a Gaussian distributed term $G(x)$ of mean zero. Our initial condition for our smoothing PDE is thus made up of image and noise together

$$u^0(x) := I(x) + G(x). \tag{3.1}$$

Throughout we take the range of values of I to be from zero to one. The range of u^0 varies slightly according to G, which itself varies according to its variance.

3.1 Gaussian Diffusion

A *Gaussian diffusion* scheme is the result of the evolution of this initial condition under the heat equation with homogeneous Neumann boundary conditions.[1] That is, the solution $u(x,t)$ of the boundary-value problem

$$\frac{\partial u}{\partial t}(x,t) = \Delta u(x,t) \qquad x \in \Omega,\, t > 0 \tag{3.2}$$

$$u(x,0) = u^0(x) \qquad x \in \Omega \tag{3.3}$$

$$\frac{\partial u}{\partial n}(x,t) = 0 \qquad x \in \partial\Omega,\, t > 0 \tag{3.4}$$

where $\partial\Omega$ is the boundary of Ω, with n its normal. It is well known (see, for example, [15]) that—for sufficiently bounded functions u—the solution of such a problem can be equivalently considered as the Fourier convolution of the initial condition with a Gaussian kernel $K_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{|x|^2}{2\sigma^2}\right)$, where the relation between time t and the variance σ is given by

$$\sigma = \sqrt{2t}. \tag{3.5}$$

It is illuminating to study this solution in frequency space. To do so, we take the Fourier transform $\mathcal{F}$ of the solution, and appeal to the convolution theorem, yielding

$$(\mathcal{F}u)(\omega) = \exp\left(-\frac{|\omega|^2}{2/\sigma^2}\right) \cdot (\mathcal{F}u^0)(\omega). \tag{3.6}$$

This shows that Gaussian diffusion is a low-pass filter that attenuates high frequencies in a non-trivial way.

Of course, an image $I(x)$ defined on a continuum makes little sense. In reality, our initial image will be given as pixel data. In one-dimension, a length N image would be represented as a sequence $\{u_0^0, \ldots, u_{N-1}^0\}$ defined on a uniform grid. As is the convention in image processing, we take the grid spacing δx to be one. To this image we apply a numerical scheme for Gaussian diffusion, namely the explicit Euler scheme with central second-order differences in space. Under a time discretisation of δt, this yields a processed image $\{u_i^m\}$ at a later

[1] Homogeneous Neumann boundary conditions ensure that the average grey level is preserved.

time $m\delta t$. What is the effect on the frequency spectrum?

Unfortunately, results from the continuous case do not automatically follow over to the discrete case, since the discrete Fourier transform is related non-trivially to its continuous counterpart. For theoretical purposes it is useful to work with the *semi-discrete* Fourier transform. This takes an infinite discrete signal and produces a continuum of frequency components.[2]

The scheme we use is produced from the semi-discrete form of explicit methods (2.9) by taking a central finite differencing for the Laplacian, namely.

$$u_j^{m+1} = u_j^m + \mu \left(u_{j+1}^m - 2u_j^m + u_{j-1}^m \right), \tag{3.7}$$

where $\mu = \frac{\delta t}{\delta x^2}$. Upon taking the inverse semi-discrete Fourier transform of the above, and comparing the integrals on both sides, we get after some simplification

$$U^{m+1}(k) = \left(1 - 4\mu \sin^2 \tfrac{k}{2}\right)^{m+1} U^0(k), \quad k \in [-\pi, \pi] \tag{3.8}$$

To satisfy the stability constraint of the explicit Euler scheme, we must choose our timesteps so that μ is smaller than one half. This choice ensures that the fully explicit scheme (3.7) is indeed a low-pass filter.

What implication does this have for image processing? Recall that we model noise as a Gaussian distributed random variable G added to some 'true' image I. It turns out that the Fourier transform of the Gaussian probability density function is again itself. That is, adding a Gaussian distributed value to each pixel in space is equivalent to adding a Gaussian distributed value to each frequency component. If we assume that our underlying image has predominantly low-frequency components, then attenuating high-frequency components should be more devastating to the noise than to the original image itself.

To verify the above, we perform two tests. The first uses a discrete Dirac delta function, which is known to have be a constant function in frequency space. Figure 3.2 confirms the profile of the filter (3.8). The second test is one row from the *Swan* test image. The discrete frequency spectrum is calculated before noise is added, after noise is added, and after Gaussian diffusion.

[2]To do so involves extending the image across an infinite grid. There are many ways we can do this, depending on the boundary conditions we choose to select—in our case, Neumann—but the result (3.8) holds regardless.

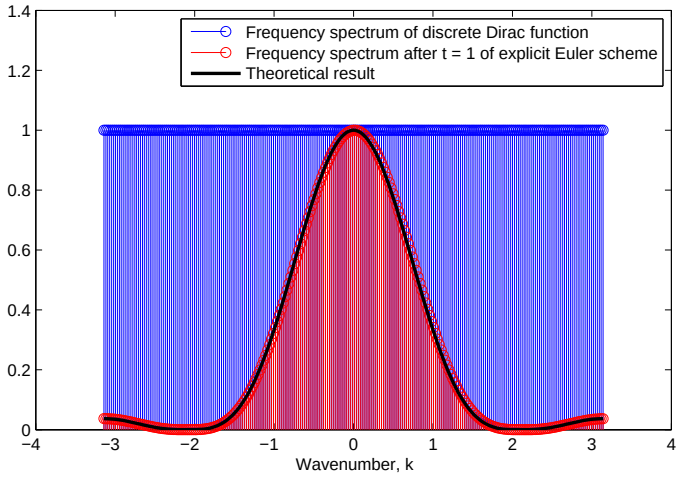


Figure 3.2: The heat equation realised as a low-pass filter. To test the effect of the explicit Euler scheme in the frequency domain, we generate an initial condition with a uniform frequency spectrum, namely a discrete Dirac function (blue). After a time $t = 1$ of the explicit Euler scheme, we can see that the high frequencies are attenuated in a non-trivial way (red). The numerical result forms a good approximation to what equation (3.8) suggests (black). All frequency spectra were generated using MATLAB's inbuilt Fast Fourier Transform.

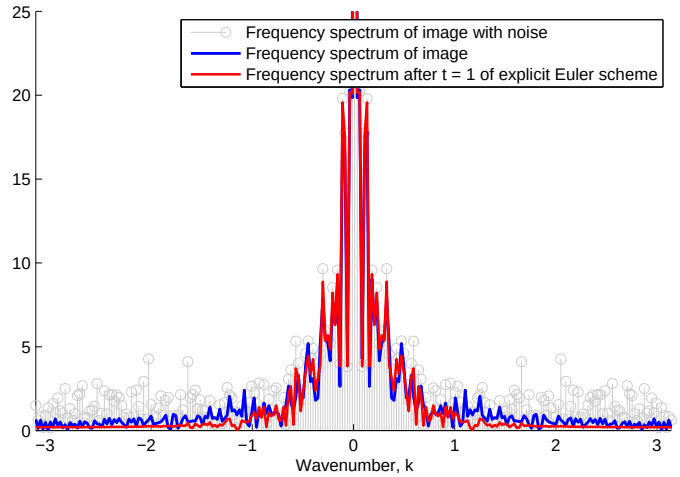


Figure 3.3: The effect of the heat equation on the frequency spectrum of a real image. We take one row of our *Swan* test image and plot the discrete frequency spectrum (blue). After adding Gaussian distributed noise to each pixel, the spectrum includes previously absent high-frequencies (grey). After a time $t = 1$ of the explicit Euler scheme, these frequencies are attenuated and the resulting spectrum (red) is a good approximation to the original. All frequency spectra were generated using MATLAB's inbuilt Fast Fourier Transform.

3.2 Perona–Malik Diffusion

We have seen how Gaussian diffusion is a low-pass filter in frequency space. Unfortunately, real images do contain high frequencies which we wish to preserve. In particular, such frequencies can arise from 'edges' in the image, that may enclose important structures. These edges are indiscriminantly blurred under Gaussian diffusion and the structure is lost. As an improvement on this, in 1987 Perona and Malik introduced a celebrated paper [30, 31] in which they propose that the diffusion coefficient be varied across the image, lessening the amount of diffusion at such edges and structures. Suppose that edges are characterised locally by rapid changes in the function u. That is, large values of $|\nabla u|$. Then we can vary the amount of diffusion across the image by introducing a variable diffusion coefficient $g(|\nabla u|)$. This modifies the heat equation (3.2) to

$$\frac{\partial u}{\partial t} = \operatorname{div}\left(g(|\nabla u|)\, \nabla u \right). \tag{3.9}$$

Large values of $|\nabla u|$ represent edges, so we would like for g to decrease monotonically from unity as $|\nabla u|$ increases. Two typical choices, suggested by the original authors, are

$$g(s) = \frac{1}{1 + \frac{s^2}{\lambda^2}}, \quad \text{and} \quad g(s) = \exp\left(-\frac{s^2}{2\lambda^2} \right). \tag{3.10}$$

In this dissertation we focus on the former. We will see that the quantity λ is a parameter that controls the sensitivity of the scheme to edges, acting as a threshold upon which the PDE changes type from forward to backward parabolic. It will help to define a *flux* function

$$\Phi(s) = sg(s). \tag{3.11}$$

Interestingly, Perona–Malik behaves differently *along* contour lines—or edges—of u than it does across them. We can find the linear diffusion in the direction of the gradient by taking the directional derivative twice. This gives

$$u_{\mathrm{nn}} := \left(\frac{\nabla u}{|\nabla u|} \right)^{\mathrm{T}} D^2 u \quad \frac{\nabla u}{|\nabla u|}. \tag{3.12}$$

Here D^2 is the Hessian matrix, with entries ∂_{ij}. Similarly the linear diffusion in the direction orthogonal to the gradient is

$$u_{\mathrm{tt}} := \left(\frac{\nabla u^{\perp}}{|\nabla u|}\right)^{\mathrm{T}} D^2 u \quad \frac{\nabla u^{\perp}}{|\nabla u|}. \tag{3.13}$$

Using these two quantities we are able to cast the diffusion term of Perona–Malik in the form

$$\operatorname{div}\big(\, g(|\nabla u|)\, \nabla u \,\big) = g(|\nabla u|)\; u_{\mathrm{tt}} \;+\; \Phi'(|\nabla u|)\; u_{\mathrm{nn}}. \tag{3.14}$$

A discussion of this can be found in [1]. Since g is strictly positive, the first term is always a diffusion *along* edges. The second term is more subtle. Indeed, for the choice (3.10a) for g, we have that

$$\Phi'(|\nabla u|) = g(|\nabla u|) \cdot \frac{1 - \frac{|\nabla u|^2}{\lambda^2}}{1 + \frac{|\nabla u|^2}{\lambda^2}}. \tag{3.15}$$

This expression changes sign as the value of $|\nabla u|$ crosses that of λ, and so the scheme is of forward-backward type perpendicular to ∇u. Where $|\nabla u| > \lambda$ the coefficient of u_{nn} is negative and equation is a *backward* diffusion perpendicular to edges. Conversely, where $|\nabla u| < \lambda$ the equation is a *forward* diffusion. Since edges correspond to large $|\nabla u|$, we expect backward diffusion across them, the parameter λ being a threshold for the value at which this occurs. This is the role of λ. Hence, Perona–Malik not only prevents edges from being smoothed, but actively sharpens them.

We have already noted that edges contribute to high-frequency components in the frequency spectrum of an image. Although a clear representation of exactly the effect that Perona–Malik has on the frequency domain is not so readily available, we can perform the same experiment as in Gaussian diffusion. Figure 3.4 shows a comparison of the effect that Gaussian diffusion and Perona–Malik diffusion has on the frequency spectrum of a real image. You can see that, at least for our particular choice of λ, Perona–Malik preserves the higher-frequencies far more than Gaussian diffusion does.

As the scheme has a scalar diffusivity, it is isotropic in a sense. However, since the diffusivity coefficient varies in perpendicular directions, Perona–Malik is

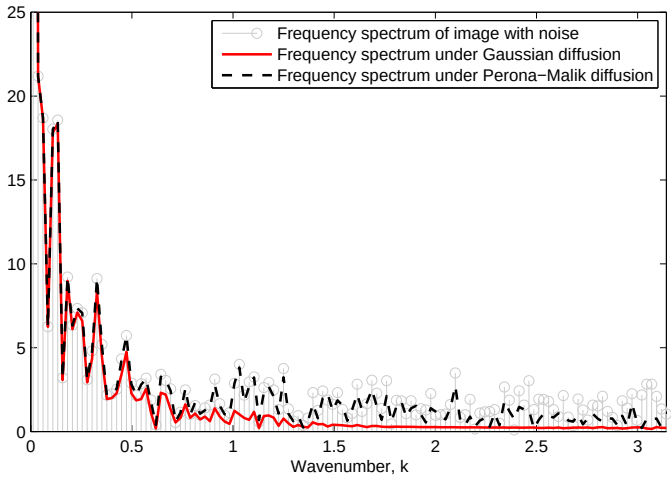


Figure 3.4: A comparison of the effect of Gaussian diffusion and Perona–Malik diffusion on the frequency spectrum of a real image. The set-up is exactly as in Figure 3.3, although the plot above focuses on the right-hand side of the frequency spectrum. Perona–Malik uses $\lambda = 0.005$. This choice leads to less attenuation of the high-frequency components.

often referred to as an *anisotropic* diffusion equation.

3.2.1 The Average Grey Level

It is often desirable in image processing for a scheme to preserve the *average grey level* of the image, defined as

$$\bar{u} = \frac{1}{|\Omega|} \int_{\Omega} u \, \mathrm{d}A, \tag{3.16}$$

where $|\Omega|$ is taken to be the area of the image, and $\mathrm{d}A$ is the infinitesimal surface element. By taking the time derivative of the above, and appealing to the form of Perona–Malik, we obtain that

$$\frac{\partial}{\partial t}\bar{u} = \frac{1}{|\Omega|} \int_{\Omega} \mathrm{div}\, \left(g(|\nabla u|)\, \nabla u \right) \mathrm{d}A. \tag{3.17}$$

To this expression we may apply the divergence theorem. Since we satisfy a homogeneous Neumann condition at the boundary, $\partial\bar{u}/\partial t$ is exactly zero and the grey level is maintained.

3.3 Numerical Schemes for Perona–Malik Diffusion

In this dissertation we consider explicit numerical schemes for Perona–Malik diffusion. These fall into the semi-discrete framework (2.9) introduced in Section 2.4. For the Perona–Malik equation (3.9), this takes the form

$$u^{m+1} = u^m + \delta t \cdot \operatorname{div}\left(g(|\nabla u^m|)\, \nabla u^m \right). \tag{3.18}$$

We wish to replace the divergence term with a spatially discretised approximation. We discuss here two choices, both in the two-dimensional case, although they can both be easily extended to three dimensions. One is the scheme proposed by Perona and Malik in their original paper; the other is adapted ([41]) from a more general scheme for parabolic equations given in Morton and Mayers [24]. Both schemes follow from the following discretisation:

$$u_{ij}^{m+1} = u_{ij}^m + \delta t \left[\frac{g_E \triangle_E u - g_W \triangle_W u}{\delta x} + \frac{g_N \triangle_N u - g_S \triangle_S u}{\delta y} \right]_{ij}^m. \tag{3.19}$$

Here N, E, S, and W stand for North, South, East, and West respectively. The coefficients g_N, g_S, g_E, g_W are taken to be approximations of the diffusion coefficient $g(|\nabla u|)$ at the boundaries of the computational cells, and the discretisations $\triangle_{\mathrm{N,S,E,W}} u_{ij}$ of the gradient are given by finite differences on a five-point grid stencil:

$$\triangle_N u_{ij} = \frac{u_{i,j+1} - u_{ij}}{\delta y}, \tag{3.20a}$$

$$\triangle_S u_{ij} = \frac{u_{i,j} - u_{i,j-1}}{\delta y}, \tag{3.20b}$$

$$\triangle_E u_{ij} = \frac{u_{i+1,j} - u_{ij}}{\delta x}, \tag{3.20c}$$

$$\triangle_W u_{ij} = \frac{u_{i,j} - u_{i-1,j}}{\delta x}. \tag{3.20d}$$

(I have tried to avoid confusion here with the Laplacian Δ—to which there is no relation—by using the symbol $\triangle$ to represent finite differences).

The two schemes that we discuss *differ only in their treatment of the coefficients* g_N, g_S, g_E, g_W.

Perona and Malik's original scheme. In Perona and Malik's original scheme, we simply evaluate the function g using the directional derivative across to that boundary, namely

$$(g_N)_{ij} = g(|\triangle_N u_{ij}|), \tag{3.21a}$$
$$(g_S)_{ij} = g(|\triangle_S u_{ij}|), \tag{3.21b}$$
$$(g_E)_{ij} = g(|\triangle_E u_{ij}|), \tag{3.21c}$$
$$(g_W)_{ij} = g(|\triangle_W u_{ij}|). \tag{3.21d}$$

Unfortunately—as Perona and Malik admit in their original paper—this scheme is not actually a discretisation of Perona–Malik diffusion. Rather, it is one for which the diagonal entries of the diffusion coefficient in (3.9) are replaced by $g(|u_x|)$ and $g(|u_y|)$. Specifically, the scheme discretises

$$\frac{\partial u}{\partial t} = \operatorname{div}\left(\begin{pmatrix} g(|u_x|) & 0 \\ 0 & g(|u_y|) \end{pmatrix} \nabla u\right). \tag{3.22}$$

Nonetheless, the results are inperceivably different from discretisations of Perona and Malik diffusion [31].

The scheme triumphs in its simplicity. However, when studying the form of (3.22), it is not clear whether the two key closest point principles of Section 2.3 apply. For Perona–Malik diffusion we appealed to Proposition 1, which applies to diffusion coefficients of the form $a(u, \nabla u)$. The above PDE does not fall into this framework. Therefore, we may not be guaranteed a method consistent with Perona–Malik diffusion on a surface when we use the above scheme with the closest point method.

Morton and Mayers scheme. In the (modified) Morton and Mayers scheme, we first construct a values of g at the centers of computational cells, and then take the approximations at the boundary to be the average across the two cells

that share it. Namely, we take

$$(g_N)_{ij} = \tfrac{1}{2}(g_{i,j+1} + g_{ij}), \tag{3.23a}$$
$$(g_S)_{ij} = \tfrac{1}{2}(g_{i,j-1} + g_{ij}), \tag{3.23b}$$
$$(g_E)_{ij} = \tfrac{1}{2}(g_{i+1,j} + g_{ij}), \tag{3.23c}$$
$$(g_W)_{ij} = \tfrac{1}{2}(g_{i-1,j} + g_{ij}). \tag{3.23d}$$

The quantities g_{ij} at the node points are given by by expanding the argument of $g(|\nabla u|)$ using central finite differences. That is,

$$g_{ij} = g\left(\sqrt{\left(\frac{u_{i+1,j} - u_{i-1,j}}{2\delta x}\right)^2 + \left(\frac{u_{i,j+1} - u_{i,j-1}}{2\delta y}\right)^2}\right). \tag{3.24}$$

The diffusion coefficients at the nodes (3.24) and boundaries (3.23), along with the discretisation (3.20) of the gradients combine to complete the discretisation (3.19) of the divergence term. From examining these various components it can be seen that the Morton and Mayers scheme is consistent with the Perona and Malik diffusion PDE (3.9).

The scheme is perhaps best summarised by MATLAB code. Here we make use of first-order differencing matrices `Dn`, `Ds`, `De`, `Dw` and averaging matrices `An`, `As`, `Ae`, `Aw` in the North, South, East, and West directions, along with second-order (central) differencing matrices `Dxc` and `Dyc` in the x and y directions. The solution is stored in a vector `I` whose entries are u_{ij}, taken row-wise. The scheme is implemented in MATLAB as

```
G = g( sqrt( (Dxc*I).^2 + (Dyc*I).^2 ) );

Unew = U + dt/dx * ( (Ae*G) .* (De*I) - (Aw*G) .* (Dw*I) ) ...
         + dt/dy * ( (An*G) .* (Dn*I) - (As*G) .* (Ds*I) );

I = Inew;
```

We must also note the stencil required for the Morton and Mayers scheme. In updating a value u_{ij}^m, we require its four neighbours $u_{i\pm1,j}$, $u_{i,j\pm1}$, on the usual five-point stencil, as well as the values $(g_{N,S,E,W})_{ij}$. These latter values are averaged quantities involving $g_{i\pm1,j}$, $g_{i,j\pm1}$. However, in each timestep, these

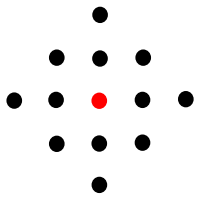


Figure 3.5: The stencil for the two-dimensional Morton and Mayers scheme.

four quantities are calculated from five-point grid stencils, each requiring their four neighbours. In total then, the stencil requires *thirteen* points, as shown in Figure 3.5. This observation will be important when we use this scheme with the closest point method.

3.4 Stability of the Morton and Mayers Scheme

For a first attempt at a condition on stability of the modified Morton and Mayers scheme, we assume that g takes a constant value across the grid. In this case, (linear) stability analysis proceeds as for the heat equation, leaving us with a slightly modified stability condition

$$\frac{\delta t}{\delta x^2} + \frac{\delta t}{\delta y^2} \leq \frac{1}{2g}. \tag{3.25}$$

Looking at the true (variable) form (3.10a) of g, we can see that g never exceeds one. Hence, the stability requirement is at worst unchanged from the requirement for the heat equation, and typically relaxed. We could incorporate this into our numerical simulations by approximating a constant g at each timestep, and then calculating a new δt on each loop. A good choice would be to use the maximum value of g across the image.

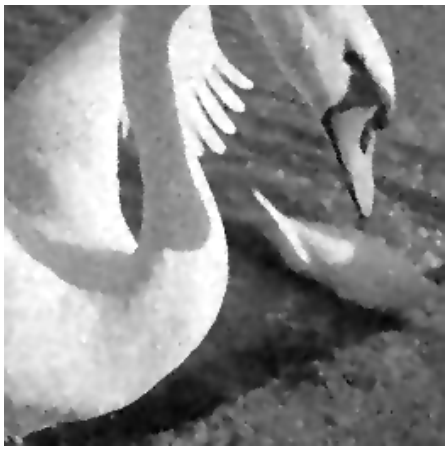
However, in practice we found that coding such a variable timestep gives little gain: g achieves the value one where the gradient is zero, and as Perona–Malik is designed to *smooth* regions, this maximum is quickly realised.

Figure 3.6 shows an implementation of the Morton and Mayers scheme for Perona–Malik diffusion on the *Swan* test image.

(a) Original

(b) Original with Noise

(c) Gaussian Diffusion



(d) Perona–Malik Diffusion

Figure 3.6: Original image (top left), image with added Gaussian noise of variance 0.1 (top right), image under a Gaussian diffusion (bottom left), image under the Perona–Malik diffusion with $\lambda = 0.02$ (bottom right), showing far better edge preservation. The image is 260 pixels by 260 pixels. The original image values vary from zero to one, the image with noise varies from slightly below zero to slightly above one. Both Gaussian and Perona–Malik diffusion were evolved up to a time $t = 2$, with a grid spacing $\delta x = 1$. Perona–Malik diffusion uses the Morton and Mayers numerical scheme.

Chapter 4

Perona–Malik Diffusion on Surfaces

To formulate Perona–Malik diffusion on a surface $\mathcal{S}$, we simply replace operators with in-surface operators:

$$\frac{\partial u}{\partial t} = \mathrm{div}_{\mathcal{S}} \left(g(|\nabla_{\mathcal{S}} u|) \nabla_{\mathcal{S}} u \right). \tag{4.1}$$

On the computer, the geometry of the surface is captured in the closest point data. This is either provided or calculated, and imported into MATLAB. To solve Perona–Malik diffusion on the surface, the Morton and Mayers scheme proceeds exactly as in the Euclidean case, except the differentiation and averaging matrices are slightly rearranged to account for the indexing of the band and we add an extra interpolation step using the interpolation matrix `E`. In two dimensions this would be:

```
G = g( sqrt((Dxc*I).^2 + (Dyc*I).^2) );

Unew = U + dt/dx * ( (Ae*G) .* (De*I) - (Aw*G) .* (Dw*I) ) ...
         + dt/dy * ( (An*G) .* (Dn*I) - (As*G) .* (Ds*I) );

I = Inew;
I = E * I;
```

4.0.1 Choosing the Bandwidth

In introducing the Morton and Mayers scheme, it was noted that the stencil in two dimensions included thirteen points. We must take note of this when designing an upper bound on the bandwidth required by the closest point method. The thirteen-point stencil (Figure 3.5) must fit around every point used in the interpolation at the surface (Figure 2.3). This has the effect of altering the bandwidth formula (2.13) to

$$\delta x \sqrt{(d-1)\left(\frac{p+1}{2}\right)^2 + \left(2 + \frac{p+1}{2}\right)^2}. \tag{4.2}$$

This is the formula that we use in the numerical results below.

4.0.2 A Raytracer for Three-Dimensional Visualisation

A visualisation in MATLAB would require the construction of a somewhat arbitrary 'plotting sphere', on which we would interpolation our solution. Instead, we use a raytracer written as part of this project. The raytracer was written in OpenGL by Jens Schneider (KAUST), with contributions and modifications from the author. It uses ideas developed by by Auer, Macdonald, Trieb, Schneider, and Westermann in [3]. First, the problem is set up in MATLAB, and the calculations for the closest point method are performed. Two files are then output by MATLAB; the first, `cp_data.txt`, contains a list of grid points, their index in the computational band, their the x, y, and z-coordinates, the coordinates of its closest point, and the distance to that closest point. This data is used to build the three-dimensional object.

For each pixel on the screen, the raytracer fires a ray along a line between the viewer's eye and that pixel. The program steps forwards along the ray incrementally, using the closest point data to determine whether it has intersected with the surface. If it does, tricubic interpolation is used to determine precisely where the surface is, and that pixel is coloured. If it does not, the pixel is left white. The colour the pixel takes depends on the incident angle with the surface and the attribute value of the point on that surface, taken from the numerical solution.

By raytracing in this way, the object can be constructed quickly without the

need for a triangulated surface.

The second file, `attr_data.txt`, contains the attribute data. Each entry of the file is a greyscale value for the point in the corresponding row of `cp_data.txt`. The attribute data is visualised on the surface as a colour value. As our images are essentially greyscale pixels, the program was modified to use nearest-neighbour interpolation and a greyscale colour scheme to visualise the attribute data.

4.1 Numerical Observations

To test the results of Perona–Malik diffusion with the closest point method, we used three 'test problems'. The first problem—*Perona–Malik on a circle*—exploits the relationship between a diffusion-like equation on a circle and on a bar, in order to check that the scheme agrees with the usual, one-dimensional Morton and Mayers scheme. We point out an interesting behaviour suggesting that, in fact, it doesn't. The second problem—*the beachball*—begins with an artificially constructed image on a sphere that tests the edge-preservation capabilities of Perona–Malik diffusion with the closest point method. The third problem—*the Stanford bunny*—demonstrates that the method can handle complex surfaces, with little change in the code itself.

4.1.1 Perona–Malik Diffusion on a Circle

To study whether Perona–Malik diffusion with the closest point method agrees with usual Perona–Malik diffusion, we perform an experiment similar to the convergence tests for the heat equation in Section 2.5. There is a strong connection between Perona–Malik diffusion on a circle and Perona Malik diffusion on a finite bar with periodic boundary conditions.

To see this, first choose an arc length parametrisation of the unit circle, $\gamma(\eta) = (\cos\eta, \sin\eta)^{\mathrm{T}}$, where η varies in the range $[0, 2\pi]$. This allows us to map values from the solution u on the unit circle onto a function v on the bar $[0, 2\pi]$ by defining $v(\eta) = (u \circ \gamma)(\eta)$. We hope that v satisfies something like Perona–Malik diffusion in one dimension. As γ is an arc length parametrisation, the

chain rule becomes

$$\partial_\eta(u \circ \gamma) = (\nabla_{\mathcal{S}} u) \circ \gamma. \tag{4.3}$$

We can use this fact in the governing equation (4.1) for u, allowing us to deduce that indeed the equation for v is Perona–Malik diffusion:

$$\frac{\partial v}{\partial t} = \partial_\eta \left(g(|\partial_\eta v|) \, \partial_\eta v \right). \tag{4.4}$$

Finally, since $v(0) = v(2\pi)$, we deduce that u satisfies the Perona–Malik equation on the circle if and only if v satisfies the one-dimensional Perona–Malik equation on the bar with periodic boundary conditions.

This provides a useful test problem for the closest point method: if we run the closest point method on the circle, and one-dimensional Perona–Malik on the bar, do we get a comparable result?

For an initial condition on the bar we select a 260-pixel-wide row from the *Swan* test image. A grid of spacing $\delta x_{\text{bar}} = \frac{2\pi}{259}$ is overlaid on the bar to accommodate each pixel. A two dimensional grid of the same spacing is set up for the unit circle. The initial condition is then mapped to the embedding grid using linear interpolation.[1] After running the closest point method on the circle, the result is interpolated back onto the bar, again using linear interpolation.

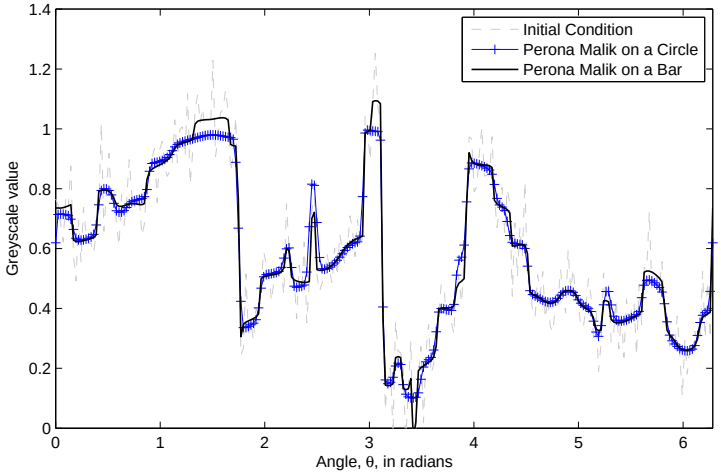
For high λ—for this data, this means in the magnitude of five and above—the problem is essentially Gaussian diffusion and excellent agreement between the bar and the circle is seen. For very low λ—in the order of δx_{bar} and below—the solution is very noisy and difficult to interpret.

The most interesting results are seen when λ is around one half. Figure 4.1 shows the solutions for two stopping times. Excellent agreement is shown when the scheme is stopped at a time $t = 0.01$. At a later time $t = 0.2$, however, it seems as if there is excessive sharpening on the circle at localised regions. We will return to this observation in Section 4.2.

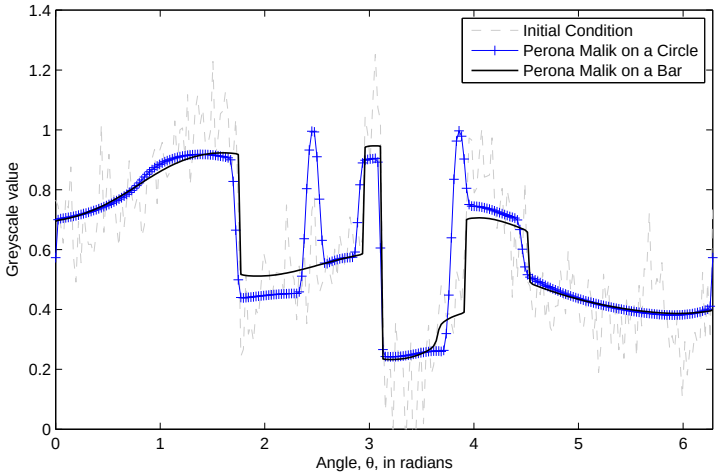
4.1.2 Edge Preservation: The Beachball

To test the edge-detecting capabilities of Perona–Malik with the closest point method, we construct the following problem: consider a sphere in spherical

[1] Linear interpolation gives a more faithful representation of gradients than, say, cubic.



(a) $t = 0.01$



(b) $t = 0.2$

Figure 4.1: Comparing Perona–Malik on the bar and on the circle. The initial condition is taken to be a 260-pixel-wide row from our *Swan* test image. Perona–Malik is solved on the bar $[0, 2\pi]$ with a grid spacing $\delta x_{\text{bar}} = \frac{2\pi}{259} \approx 0.0243$, using a parameter $\lambda = 0.5$. There is excellent agreement for short times (top). After a longer time (bottom) the solutions diverge at some regions, but agree at others.

coordinates (θ, ϕ, r). Form an initial condition by adding Gaussian noise to the following function:

$$\begin{cases} 1 & \text{for } \theta \text{ such that } \cos 3\theta > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

This produces an image reminiscent of a beachball. There are clear edges in the underlying image. Recovery of these edges will test our method. The initial condition and the result after Perona–Malik diffusion can be seen in Figure 4.2.

The image shows very good edge-preservation qualities in comparison to the Gaussian scheme, even at the poles of the sphere where the edges come together. This is an encouraging result; the preservation of junctions in the image was presented as a notable feature of Perona–Malik diffusion by the original authors, so it is especially pleasing to see the good performance in this respect when using the closest point method.

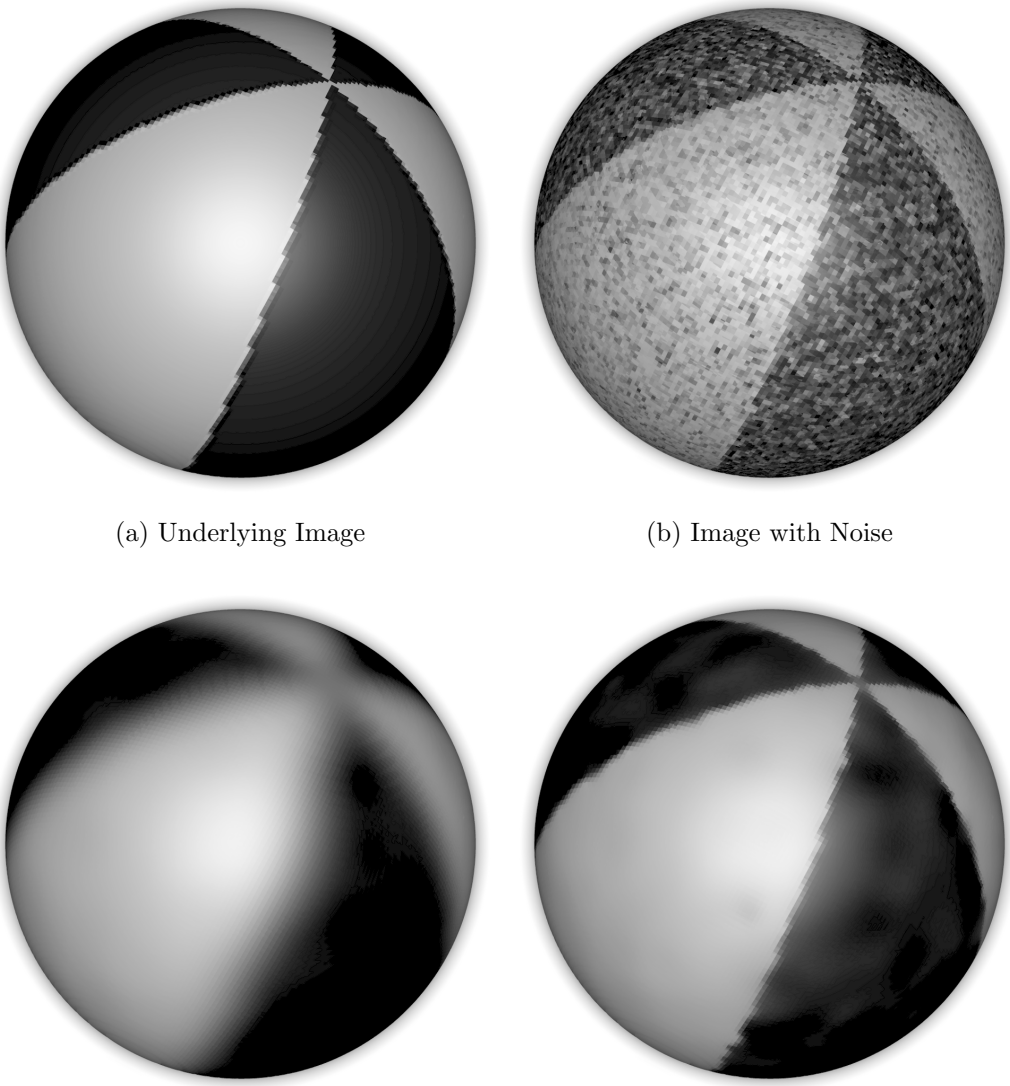
In Section 3.2.1 we stated that the Perona–Malik diffusion preserves the average grey level. Is this conservation observed when using the closest point method? To test this, average grey levels were estimated at each timestep by averaging the grey levels across all points in the computational band. Pleasingly, Figure 4.3 shows that the variation is indeed negligable, varying only between 0.4995 and 0.5001. It would be interesting to subject this same test to a more complex image.

Upon manipulating the beachball solution using the raytracer, a persistent artefact was noted. Namely, sharper 'bands' along (perpendicular) great circles aligned with the grid. We return to this observation in Section 4.2.

4.1.3 A Complicated Surface: The Stanford Bunny

One of the advantages of the closest point method that has been extolled throughout this dissertation is the ability to deal with complex surfaces. However, up to now, the method has only been demonstrated on simple surfaces such as circles and spheres.

To test the closest point method on something more challenging, we turn to the animal mascot of computer graphics: the Stanford bunny [37]. The bunny



(a) Underlying Image

(b) Image with Noise

(c) Gaussian Diffusion

(d) Perona–Malik Diffusion

Figure 4.2: Denoising a beachball. The sphere is the unit sphere, embedded in a grid of spacing $\delta x = 0.0125$. To form our initial condition we add Gaussian noise of variance 0.5 to the underlying image (top). Both schemes are evolved for $t = 0.002$. Perona–Malik diffusion uses a parameter $\lambda = 4$ (bottom). The image shows good edge preservation despite significant noise.

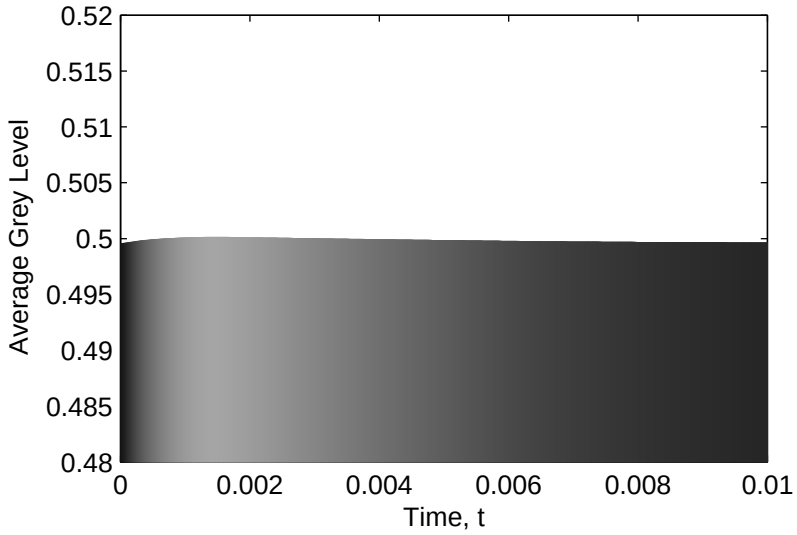


Figure 4.3: The variation of the average grey level for the beachball is negligable. Values can be in the range zero to one, but were found to vary only between 0.4995 and 0.5001.

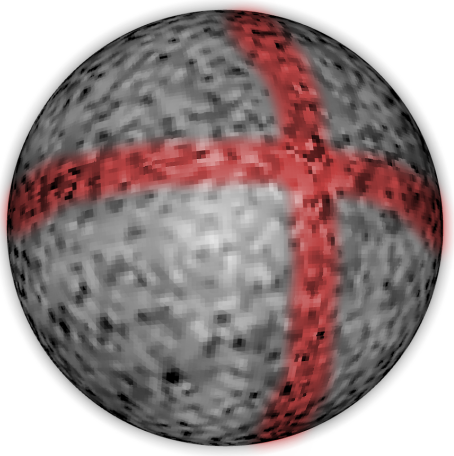


Figure 4.4: There is more sharpening on great circles. The initial condition was pure noise, of variance one. The parameters of the scheme were chosen to highlight the effect: λ is two and the final time is $t = 0.0013$. The sphere is the unit sphere on an embedding grid of spacing $\delta x = 0.0125$. The image is artificially coloured to highlight where the effect is found.

is imported in to MATLAB as a data file, each row of which relates to one node in the grid. The row contains the index of the node, the x, y, and z-coordinates of the node, the closest point function at the node, and the Euclidean distance to that closest point.

This data was kindly provided by Colin Macdonald, generated using a triangulation algorithm described in [20, 21].

For an initial condition we divide the x-coordinates into stripes of equal width, alternating between white and black. This produces a striped bunny. To this is added Gaussian noise. The usual schemes for Perona–Malik and Gaussian diffusion are then run,[2] and the results are written to files for use with our OpenGL raytracer.

In implementing Perona–Malik on the bunny, there were some unexpected difficulties. Perona–Malik used in the way described above consistently outputs a uniformly grey solution, even after short times. Closer investigation reveals that this is caused by persistent negative values in the image.

These negative values come from two sources: one, in the addition of a (possibly negative) Gaussian noise to voxels that are already zero; and two, in the overshoot of the tricubic interpolation. Why were these not problems for the beachball case? Whilst the initial condition did contain negative values, the simple, convex nature of the sphere presumably overcomes the effect of interpolation overshoot.

To remedy this problem on the bunny, we simply test, at the initial condition and *at each timestep*, for voxels that are either negative or that exceed one. We truncate such pixels, overwriting negative values with zero values, and overwriting values above one with the number one itself. This is the same approach used by [3], where similar problems were encountered when simulating fluid flow on the surface of a three-dimensional horse.

Figure 4.5 shows the original striped ('zebra') bunny, the bunny with noise added, the result of Gaussian diffusion, and the result of Perona–Malik diffusion with the truncation as described above.

The results show an impressive recovery of the original image and serve to

[2]Namely, for Perona–Malik the closest point method with tricubic interpolation and the modified Morton and Mayers scheme; for Gaussian diffusion the closest point method with tricubic interpolation and the explicit Euler method with central second-order finite differences.

display the power of the closest point method to arbitrary surfaces. Indeed, the only change in the code was the different closest point data, and two lines of code to handle the truncation.

A particularly important feature to take note of is the preservation of the locations of the boundaries. The problem of dislocating edges from their original locations is a well known issue with linear diffusion filtering, and is indeed something that Perona–Malik diffusion is intended to rectify [39, 38]. A dislocation of structures could be an issue if, to take our original example, the image data is from a medical scan on the surface of a bodily organ, and structures are moved from their original locations. It is pleasing to see no perceptible dislocation of the edges in the image, despite the complex geometry.

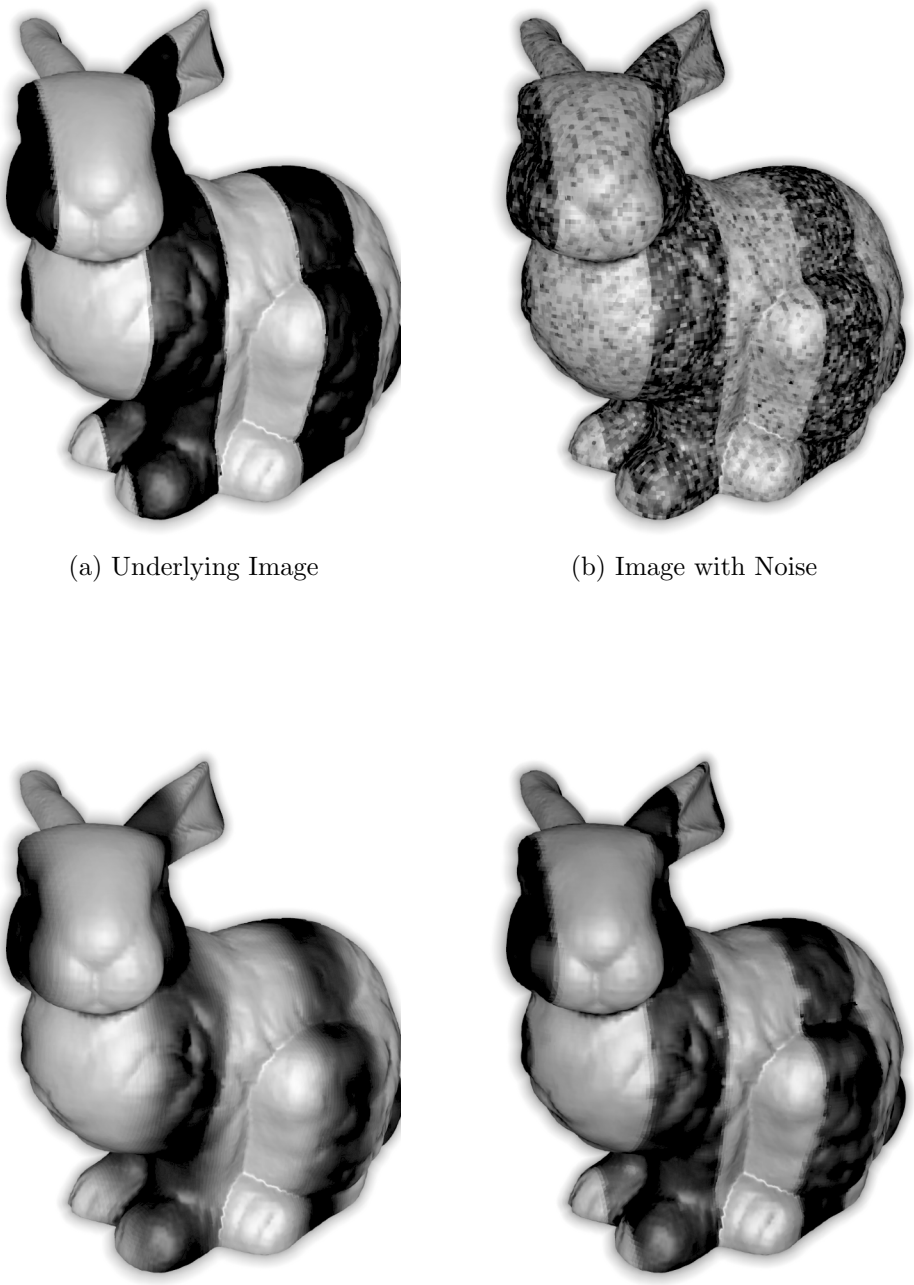
The average grey level was tested before and after the Perona–Malik scheme, in the same manner as the beachball. It was found to have increased very slightly from 0.5114 to 0.5118.

No artefacts of the kind seen in the beachball were noticed on the bunny.

4.2 Discussion

In the numerical tests above we made a number of observations. In **Perona–Malik on a circle** it was noted that whilst agreement between the circle and the bar was good for short times, for longer times there seemed to be localised regions of excessive sharpening on the circle. When studying the **beachball**, excellent edge preservation, particularly at boundaries, was observed, and the average grey level was found to be constant. However, a persistent artefact was noted along great circles of the sphere. In the **Stanford Bunny** it was demonstrated that the method can handle complex geometries, with good edge preservation and no noticeable dislocation of edges. However, it was found necessary to truncate the solution at each timestep.

In this section we first look to explain why we see the regions of excessive sharpening on the circle and the sphere. We then explore the impact that our choice of interpolation has on the solution, and suggest that the method might be better off *without any interpolation at all*.



(a) Underlying Image

(b) Image with Noise

(c) Gaussian Diffusion

(d) Perona–Malik Diffusion

Figure 4.5: Denoising the zebra bunny. The bunny is enclosed in the box $[-1, 1]^3$, and the data is given on an embedding grid of spacing $\delta x = 0.0125$. Schemes were evolved up to a time $t = 0.001$. Perona–Malik uses a parameter $\lambda = 5$.

4.2.1 Grid Effects

In Section 4.1.1 we compared the Morton and Mayers scheme on the bar in one dimension to the scheme with the closest point method on the circle. Whilst good agreement was found, we noted that there seemed to be excessive sharpening in some parts of the image by the scheme on the circle. Closer inspection of Figure 4.1 reveals two regions in particular: near $\theta = \frac{\pi}{4}$ and $\theta = \frac{3\pi}{4}$. Upon running further numerical tests, disagreement between the bar and the circle was consistently observed around *these* values of θ.

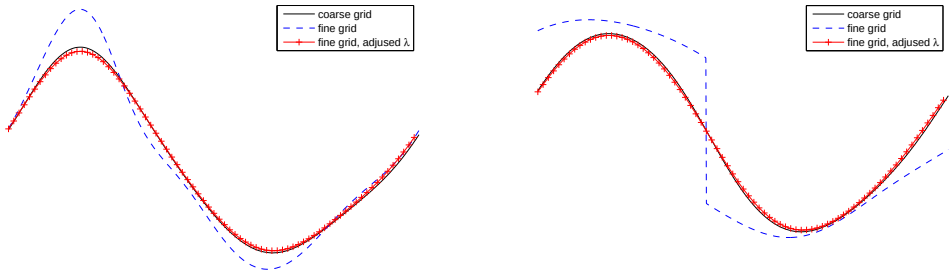
Recall that λ represents a gradient threshold. Typically, gradients in an image vary very little, and when λ is around these values, as indeed we would want it to be, small changes in the gradients of the image lead to very different results. In any numerical scheme that approximates a continuous equation, we would hope that using the *same* parameters on a grid of *different* spacing would lead to the same solution. Indeed, this is the essence of a convergence test.

Unfortunately, this is not true for many numerical schemes for Perona–Malik diffusion. In [11] Esedoḡlu considers a continuum limit of Perona and Malik's *original* numerical scheme, and proposes that as the grid spacing δx changes, λ should be scaled like $\delta x^{-1/2}$ to obtain a meaningful limit.[3] That is, upon making the grid k times finer, one should use a parameter $\sqrt{k}\lambda$ to ensure consistency.

In Section 4.1.1 the embedding grid for the circle was of the same spacing as for the bar. However, the surface itself does not always pass orthogonally through the embedding grid. In particular, at $\theta = \frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$, the circle passes diagonally roughly through grid points. The 'effective' grid spacing here, then, is more like $\sqrt{2}\delta x$ than δx. Hence, as we move around the circle, the effective grid spacing changes from δx to around $\sqrt{2}\delta x$. Could this be causing the regions of sharpness?

Unfortunately, the analysis Esedoḡlu carries out doesn't carry over easily to the Morton and Mayers scheme that is used in this dissertation. His findings do, however, suggest that a similar scaling might be applicable in the Morton

[3] In doing so, Esedoḡlu hopes to find an equation that captures the essence of Perona–Malik, without the issues of well-posedness. His proposed equation is a system of heat equations laid over the image domain, with nonlinear, Neumann-like boundary conditions coupling them together. Each heat equation represents a region, with its boundary an edge in the image. This is an interesting interpretation of Perona–Malik diffusion.



(a) 100th Row of *Swan* (b) 210th Row of *Swan*

Figure 4.6: Investigating changing grid size with the one-dimensional Morton and Mayers scheme for Perona–Malik. The initial conditions are rows from the Swan test image—namely the 100th (left) and the 210th (right). We first solve on a 'coarse', pixel-level grid (black) with $\lambda = 8$. We then solve on a 'fine' grid, three times finer (blue and red). Agreement is visibly better after adjusting λ to be $\sqrt{3}$ times bigger. Both schemes are run up to a time $t = 0.01$, although the agreement was also found to persist for longer times.

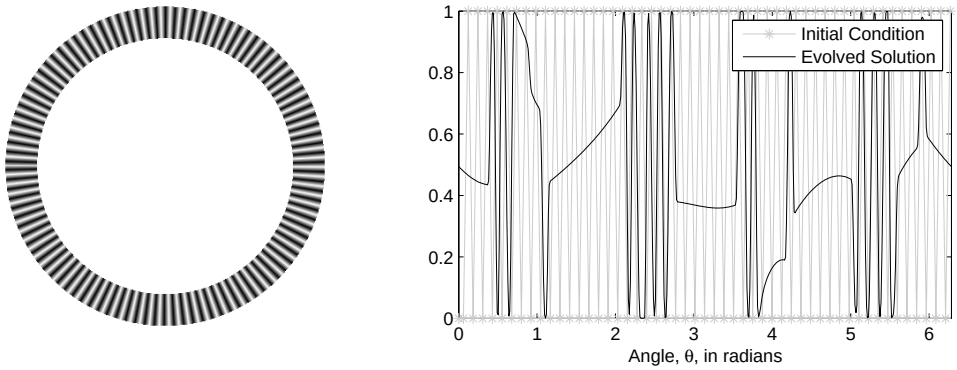
and Mayers case.

To investigate this, consider the Perona–Malik equation on the bar $[0, 1]$. Again, a row from the Swan test image provides the initial condition. We first run the Morton and Mayers scheme on the natural grid for the bar, namely, a grid point at each pixel. Then, a second, finer grid is chosen, onto which the initial condition is (linearly) interpolated. Morton and Mayers is run using the same λ as the coarse scheme, and then using a scaled λ.

Figure 4.6 shows the results for two different slices of the Swan test image. In both cases, scaling λ like $\delta x^{-1/2}$ leads to far better agreement between the coarse and fine grids. This scaling was found to persist for a variety of λ's, initial conditions, and stop times.

This strongly suggests that such a scaling is necessary in the one-dimensional case to ensure consistency.

We would now like to investigate our conjecture that the effective grid spacing—or, equivalently, the effective λ—varies around the circle. To test this, consider an initial condition on the circle that is evenly spread. We pick an integer N, and at N angles around the circle alternate the initial condition between zero and one. This produces a uniformly stripey circle. If the 'effective' λ does indeed vary around the circle, we should expect a varying amount of smoothing



(a) Initial Condition (b) Evolved Solution

Figure 4.7: Perona–Malik on the circle with a uniform initial condition. The unit circle is in an embedding grid of spacing 0.002. The initial condition alternates between one and zero at 512 points around the circle (left), and is evolved for $t = 0.02$ using the Morton and Mayers scheme with $\lambda = 1.5$. Despite the evenly spread initial condition, the solution shows considerably sharper regions around $\theta = \frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$.

across the image.[4]

Indeed, the results of Figure 4.7 show far more sharpening at angles near $\frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$ than elsewhere. This suggests that the variation in the effective λ here is causing the discrepancies between the method on the bar and on the circle.

The outcome of this discussion is that we may be better off varying the λ across the surface, depending on the grid alignment.

4.2.2 Artefacts in the Beachball

When viewing the beachball solution in the raytracer it is evident that there are curious artefacts in the image, manifesting themselves as sharper bands lying along perpendicular great circles. In an effort to isolate this problem, the same code for the beachball was run on a sphere of pure Gaussian noise. The result of this is displayed in Figure 4.4. The same artefacts were definitely present, although they are subtle and difficult to notice unless the object can be manipulated in 3D.

[4]Numerical tests confirmed—or at least, supported—that Perona–Malik one-dimensional schemes, where the λ is correct throughout, *do* result in evenly spread solutions.

The effect of a varying λ described above could provide a conjecture for these artefacts. However, it should be noted that it is not clear whether the origins are computational, or rather a quirk of the raytracer. A good test for this would be to rotate the solution before it is output from MATLAB. If the artefacts rotate with the solution it would suggest that the origins are computational.

4.2.3 Dropping the Interpolation Step

In the closest point method a re-extension is needed at each timestep to ensure that the data remains constant in normal directions to the surface. This is realised via an interpolation. In including this step we ensure that the PDE we are solving on the surface is the correct one. However, there are also a number of disadvantages to this repeated interpolation:

- There is a computational expense involved in the interpolation, in particular in building the matrix E.
- The interpolation may contribute to the varying effective λ effect designed above. Without interpolation, there is no knowledge of the surface, and hence no variation in the effective grid spacing.
- It would seem that the use of bi/tricubic interpolation would smooth the solution, when in fact we are working with quite discontinuous data.

One might ask whether the constancy of the solution along normal lines is preserved by the Perona–Malik equation, even in the absence of such interpolation. If it were, the surface PDE would remain approximately correct. In [13], Greer shows that $\mathcal{O}(\delta x^{\frac{3}{2}})$ convergence is possible for the heat equation on the sphere without using interpolation. In [34] Tian, Macdonald, and Ruuth seek a steady state solution of gradient descent equations, and suggest dropping the interpolation step as an 'initial phase' for efficiency gains.

Of course, dropping the interpolation comes at some cost:

- We will no longer be solving the surface PDE, but rather something different.
- We must concern ourselves with the boundary of the computational band, since the values there are no longer overwritten at each timestep.

The natural choice for the boundary values at the edges of the computational band would be a Neumann condition,

$$\frac{\partial u}{\partial n} = 0. \tag{4.6}$$

One way to implement this would be to increase the bandwidth slightly to include 'ghost points'. At each timestep, the values at the ghost points are overwritten by values at the surface, providing an approximate Neumann condition. Unfortunately, this leaves us no better than before; we are still left with an interpolation step, which in practice is vary similar to the one used in the closest point method.

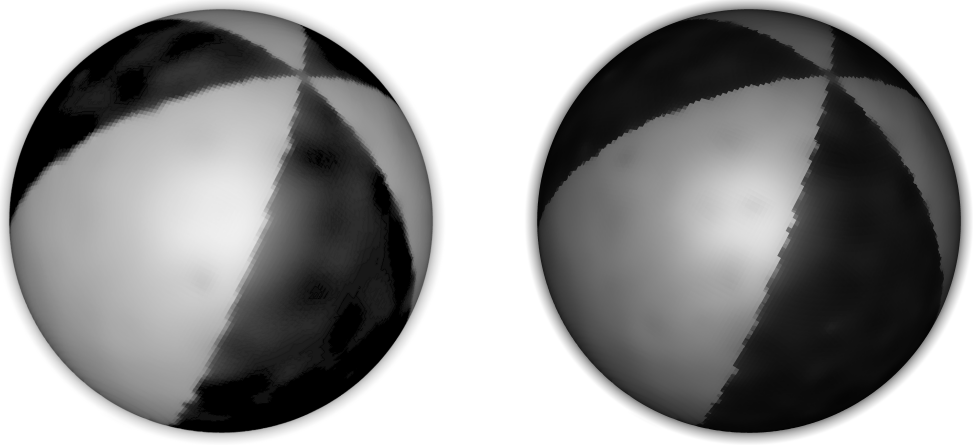
Alternatively, we could impose Dirichlet conditions, keeping the values at the boundary of the computational band unchanged throughout. We eventually discard these points, so it does not matter that the noise persists there so long as it does not affect the solution at the surface. Such a Dirichlet condition is trivial to code; the matrices `Dxf`, etc., are constructed as Dirichlet, so we need only delete the line `I = E * I;`.

Surprisingly, impressive results are obtained by doing just this. See Figure 4.8 for a comparison of the beachball problem with and without interpolation. This may be because the Perona–Malik equation regards the boundary as a 'region' in the image: the gradient remains high at the boundary and so, as long as λ is chosen wisely, the noise is not diffused towards the surface.

There may be another advantage of dropping the interpolation step. There may be less of problem with a varying effective λ; see Figure 4.9.

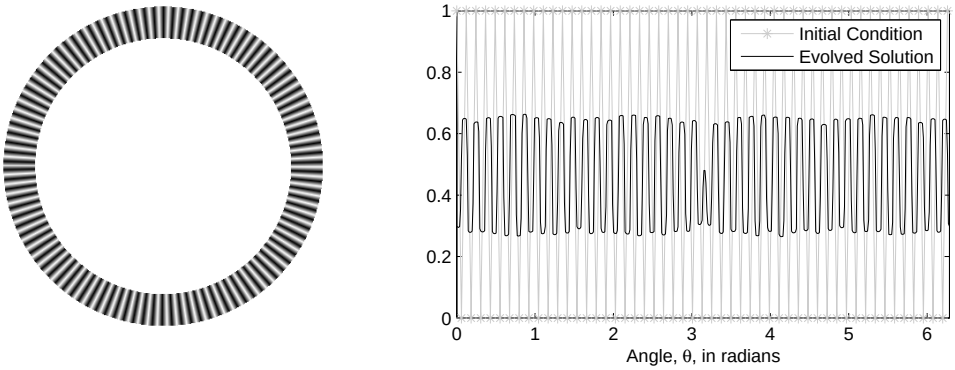
Such decisions as to whether the re-extension step should be dropped would require further analysis. Whilst the results of Figure 4.8 are impressive, it should be noted that the parameters to produce this image were chosen by hand, and it could well be that there exist parameters for which the closest point method outperforms the method where there is no re-extension.

One would also expect that in regions of high curvature, normal lines may intersect close to the surface, merging in a non-trivial way and affecting the normality of the solution after a certain time period.



(a) Closest Point Method (b) Without Re-Extension

Figure 4.8: Dropping the interpolation step when denoising a beachball. The initial condition and parameters are the same as in Figure 4.2. (Left) Perona–Malik diffusion using the closest point method. (Right) good results are obtained even without the re-extension step.



(a) Initial Condition (b) Evolved Solution

Figure 4.9: Perona–Malik on the circle *without interpolation* using a uniform initial condition. The problem and parameters are exactly as in Figure 4.7. The solution is quite evenly spread. This result persisted for a variety of λ's and stop times. This suggests that dropping the interpolation step may be at least a partial remedy to the problem of a varying effective λ.

Chapter 5

Conclusion

The remit of this dissertation was to answer the following question: can nonlinear diffusion filtering be combined with the closest point method, to form an efficient and reliable tool for image processing on surfaces? This is an important and relevant question. The number of recent papers on image processing in higher-dimensions and on surfaces suggest that this is becoming an active area of research, particularly in computer graphics and medicine. This dissertation has demonstrated that nonlinear diffusion filter can be combined with the closest point method, and that impressive results can be obtained with minimal modification to existing code. The closest point method has advantages over other methods—such as those that rely on explicit representations and surface paramaterisations—in its simplicity and ease of adaptability to a wide range of surfaces.

The closest point method was explored and presented. Convergence studies were constructed both from scratch and using a repository of closest point code at OCCAM, showing excellent agreement with theory. The use of of linear and nonlinear diffusion was understood and presented.

For the image processing on surfaces, beautiful visualisations were achieved using an OpenGL raytracer designed as part of the project. The calculations themselves were done in MATLAB. The numerical scheme suggested by the original authors of Perona–Malik diffusion may be unsuitable for the closest point method, and so a scheme inspired by Morton and Mayers [24] and [41] was constructed.

A range of example problems for the Morton and Mayers scheme with the

closest point method were presented. They showed that the method reproduces the excellent edge-preserving properties of Perona–Malik diffusion in Euclidean spaces, with little extra addition to code. Numerical tests suggested that properties such as a constant average grey level persist when using the closest point method. This was true of both the simple surface of a sphere, and also the complex geometry of the Stanford Bunny. We also discovered that in the case of an surface such as the bunny, it is necessary to truncate values, in particular, discarding those below zero.

In running these test problems some interesting and unexpected effects were discovered. Namely, a variation in the amount of diffusion across the perimeter of the circle in a regular fashion. Investigating these effects numerically and turning to the literature allowed us to construct a better picture of how these effects arise. However, the exact details remain exclusive. At this point a strong theoretical approach is required.

This dissertation also highlighted some possibly related artifacts seen on the surface of the sphere. Further tests are needed to determine whether the origins of this effect are computational or a quirk of the visualisation. Further standardisation of the numerical experiments in this dissertation would benefit from introducing an algorithm to select parameters. Such algorithms are available [31, 43, 17] and would remove the human element in this choice.

A natural next step for the method would be to consider different interpolation schemes. For all the numerical results in the previous section, bi/tricubic interpolation was used. However, under Perona–Malik diffusion solutions tend to become almost piecewise constant, resulting in regions of little change linked by large jumps [11]. The result is highly discontinuous data. One would expect, then, a benefit from using interpolation schemes that deal well with discontinuous data, such as WENO interpolation.

References

[1] Luis Alvarez, Pierre-Louis Lions, and Jean-Michel Morel. Axioms and fundamental equations of image processing. *Archive for Rational Mechanics and Analysis*, 123:199–257, 1993.

[2] Gilles Aubert and Pierre Kornprobst. *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*. Springer, 2001.

[3] Stefan Auer, Colin B. Macdonald, Jens Schneider, Rüdiger Westermann, and Marc Treib. Real-time fluid effects on surfaces using the closest point method. In preparation.

[4] Chandrafit L. Bajaj, Guo Liang Xu, Rafit L. Bajaj, and Guoliang Xu T. Anisotropic diffusion of subdivision surfaces and functions on surfaces. *ACM Transactions on Graphics*, 22:2003, 2002.

[5] Marcelo Bertalmìo, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *Journal of Computational Physics*, 174(2):759–780, 2001.

[6] Alan Conrad Bovik. *Handbook of image and video processing*. Academic Press, 2005.

[7] Li-Tien Cheng, Paul Burchard, Barry Merriman, and Stanley Osher. Motion of curves constrained on surfaces using a level set approach. *Journal of Computational Physics*, 175:2002, 2000.

[8] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. Anisotropic geometric diffusion in surface processing. In *In T. Ertl, B. Hamann, and A. Varshney, editors, Proceedings of IEEE Visualization 2000*, 2000.

[9] Gerardo Anchez-Ortiz Daniel, Daniel Rueckert, and Peter Burger. Knowledge-based anisotropic diffusion of vector-valued 4-dimensional cardiac MR images. In *the British Machine Vision Conference, BMVC'96*, 1996.

[10] Julie Dorsey and Pat Hanrahan. Digital materials and virtual weathering. *Scientific American*, 282(2):46–53, 2000.

[11] Selim Esedoḡlu. An analysis of the Perona–Malik scheme. *Communications on Pure and Applied Mathematics*, 54:1442–1487, 2001.

[12] Guido Gerig, Olaf Kübler, Ron Kikinis, and Ferenc A. Jolesz. Nonlinear anisotropic filtering of MRI data. 11(2):221–232, November 1992.

[13] John Greer. An improvement of a recent Eulerian method for solving PDEs on general geometries. *Journal of Scientific Computing*, 29:321–352, 2006. 10.1007/s10915-005-9012-5.

[14] Xianfeng Gu and Shing-Tung Yau. Global conformal surface parameterization. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 127–137. Eurographics Association, 2003.

[15] Gunter Hellwig. *Partial differential equations*. B.G. Teubner, Stuttgart, 1977.

[16] Rongjie Lai and Tony F. Chan. A framework for intrinsic image processing on surfaces. *Computer Vision and Image Understanding*, In Press, Corrected Proof, 2011.

[17] Der-Shan Luo, Michael A. King, and Stephen Glick. Local geometry variable conductance diffusion for post-reconstruction filtering. *Nuclear Science, IEEE Transactions on*, 41(6):2800–2806, dec 1994.

[18] Colin B. Macdonald. *The closest point method for time-dependent processes on surfaces*. PhD thesis, Simon Fraser University, 2008.

[19] Colin B. Macdonald, Jeremy Brandman, and Steven J. Ruuth. Solving eigenvalue problems on curved surfaces using the closest point method. 2011. To appear in Journal of Computational Physics.

[20] Colin B. Macdonald and Steven J. Ruuth. Level set equations on surfaces via the closest point method. *Journal on Scientific Computing*, 35(2–3):219–240, June 2008. doi:10.1007/s10915-008-9196-6.

[21] Colin B. Macdonald and Steven J. Ruuth. The implicit closest point method for the numerical solution of partial differential equations on surfaces. *SIAM Journal on Scientific Computing*, 31:4330–4350, 2009.

[22] Ron Mancini. Op amps for everyone, 2002. Texas Instruments Design Reference.

[23] Thomas März. On closest point operators. In Preparation.

[24] Keith W. Morton and David Francis Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 1994.

[25] Tim G. Myers and Jean P. F. Charpin. A mathematical model for atmospheric ice accretion and water flow on a cold surface. *International Journal of Heat and Mass Transfer*, 47(25):5483–5500, 2004.

[26] Tim G. Myers, Jean P.F Charpin, and S. Jon Chapman. The flow and solidification of a thin fluid film on an arbitrary three-dimensional surface. *Physics of Fluids*, 14(8):2788–2803, August 2002.

[27] Facundo Mmoli, Guillermo Sapiro, and Paul Thompson. Implicit brain imaging. *NeuroImage*, 23:179–188, 2004.

[28] Stanley J. Osher and Ronald P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 1st edition, October 2002.

[29] Lucero Lopez Perez, Rachid Deriche, and Nir A. Sochen. The Beltrami flow over triangulated manifolds. In *ECCV Workshops CVAMIA and MMBIA*, volume 3117 of *Lecture Notes in Computer Science*, pages 135–144. Springer, 2004.

[30] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. In *Proceedings of IEEE Computer Society Workshop on Computer Vision*, pages 16–22, November 1987.

[31] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:629–639, July 1990.

[32] Steven Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *Journal of Computational Physics*, 227:1943–1961, January 2008.

[33] Endre Süli and David Francis Mayers. *An introduction to numerical analysis*. Cambridge University Press, 2003.

[34] Li Tian, Colin B. Macdonald, and Steven J. Ruuth. Segmentation on surfaces with the closest point method. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3009–3012, November 2009.

[35] Lloyd N. Trefethen and Jean-Paul Berrut. Barycentric Lagrange interpolation. *Society for Industrial and Applied Mathematics Review*, 46(3):501–517, 2004.

[36] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 289–298, New York, NY, USA, 1991. ACM.

[37] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[38] Joachim Weickert. A review of nonlinear diffusion filtering. In *Proceedings of the First International Conference on Scale-Space Theory in Computer Vision*, SCALE-SPACE '97, pages 3–28, London, UK, 1997. Springer-Verlag.

[39] Joachim Weickert. *Anisotropic Diffusion in Image Processing*. B.G. Teubner Stuttgart, 1998.

[40] Eric W. Weisstein. Spherical harmonic. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SphericalHarmonic.html`.

[41] Maciej Wielgus. Perona–Malik equation and its numerical properties. Bachelor's Thesis, Warsaw University.

[42] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Computer Graphics*, pages 299–308, 1991.

[43] Terry S. Yoo and James M. Coggins. Using statistical pattern recognition techniques to control variable conductance diffusion. In *Information Processing in Medical Imaging*, pages 459–471. Springer-Verlag, 1993.