Approximation of Scattered Data with Multilevel B-splines

Øyvind Hjelle

August 29, 2001

Abstract

Multiresolution methods based on B-spline refinement for approximating scattered data are explored. The methods are extremely fast compared to many other methods and produce fair results. The basic schemes were adopted from Lee, Wolberg and Shin [12]. Several extensions are proposed and numerical examples are presented at the end of the report.

Contents

1 Introduction **3**

2 The Basic Schemes of Multilevel B-Splines **4**

3 Cubic C^2 Continuous Splines on Uniform Partitions **7**

4 Details of Algorithm 1 **9**
4.1 Step 4 of Algorithm 1 9
4.2 Step 5 of Algorithm 1 11
4.3 Zero-Coefficients and Data Compression 12

5 B-Spline Refinement on Uniform Partitions **12**
5.1 Refinement of Bicubic C^2 Continuous Splines 13
5.2 Refinement of Bicubic C^1 Continuous Splines 16

6 Numerical Examples and Discussion **21**

A Figures from Numerical Experiments **24**

1 Introduction

The methods explored in this report take a set of scattered data as input and produce tensor product B-spline surfaces as output. The algorithms run in a multiresolutional setting over uniform partitions such that the final surface f is composed of a sequence of surfaces at dyadic scales,

$$f = f_0 + f_1 + \cdots + f_h, \tag{1}$$

where $f_i \subset S_i,\ i = 0, \ldots, h$, and $S_0, \ldots, S_h$ is a nested sequence of subspaces of S_h,

$$S_0 \subset S_1 \subset \cdots \subset S_h. \tag{2}$$

Numerous papers have been written on this subject, especially on multiresolution analysis based on wavelets; see for example [2, 6, 13]. Multiresolution analysis with splines on non-uniform partitions was considered in [5]. Typical applications of multiresolution analysis are cartographic generalization and data compression [4].

The basic algorithms used for the results presented in this report were published in 1997 by Lee, Wolberg and Shin [12]. They called the schemes *multilevel B-splines*. The methods had been used previously for image morphing [10, 11]. Our interest is mainly scattered data interpolation and approximation, which is also the main focus in [12]. The methods are very fast compared to other methods for scattered data interpolation and approximation. They can process huge numbers of scattered data, they are numerically stable, and they produce fair results if the scattered data are evenly distributed. For clustered data unevenly distributed in the domain, the methods may generate irregularities near the scattered data in the final surface f.

There are two basic methods that produce identical results. One method is based on B-spline refinement for producing one single spline surface f from the scattered data. The other method is *adaptive* in the sense that only significant information is stored at each level of the surface hierarchy in (1). The adaptive version represents the surface compactly, as the (non-zero) spline coefficients are stored in linear arrays. It can be used to produce surfaces with large tensor product grids, for example (1 million) $\times$ (1 million), and thus deliver a surface that gives a better approximation to the input data. Interpolation of the scattered data is achieved if the tensor product grid of f_h in the finest spline space S_h is sufficiently dense. Naturally, the adaptive algorithm is slower than the non-adaptive, and evaluation of the final surface f is slower since f is represented as a sum of the surfaces f_i in (1).

The purpose of this report is twofold. We present recipes for deriving the mathematical details of the algorithms, and we present numerical experiments to show the capability of multilevel B-splines for scattered data approximation. The recipes are meant to comprise a mathematical toolbox when working with multilevel B-splines on uniform partitions such that the methods can be easily extended and adapted to specific needs in applications. In particular, we focus

on refinement operators for B-splines on uniform partitions. We assume that the reader is familiar with basic B-spline theory.

Although this report is self-contained, the paper by Lee, Wolberg and Shin [12] should be consulted for details on data structures and other aspects of implementation. We use the same mathematical notation as that of [12].

2 The Basic Schemes of Multilevel B-Splines

Given a set of scattered points $P = \{(x_c, y_c, z_c)\}$ and let Ω be a rectangular domain in the xy-plane such that (x_c, y_c) is a point in Ω. Further, let $\Theta = \Phi_0, \Phi_1, \ldots, \Phi_h$ be a dyadic sequence of uniform tensor product grids over Ω corresponding to the nested sequence of spline spaces in (2). Since Θ is a dyadic sequence, the number of grid lines in Φ_k is twice the number of grid lines in Φ_{k-1}, and the grid lines in Φ_{k-1} is a subset of the grid lines in Φ_k for $k = 1, \ldots, h$.[1].

The basic algorithm starts by calculating an initial (smooth) approximation f_0 to P defined on the coarsest grid Φ_0. To proceed to the finer levels in the multiresolutional setting of Equation (1), we quote from Lee, Wolberg and Shin [12]: The first approximation possibly leaves large discrepancies at the data points in P. In particular, f_0 leaves a deviation $\Delta^1 z_c = z_c - f_0(x_c, y_c)$ for each point (x_c, y_c, z_c) in P. The next finer tensor product grid Φ_1 is then used to define a function f_1 that approximates the differences $P_1 = \{(x_c, y_c, \Delta^1 z_c)\}$. The sum $g_1 = f_0 + f_1$ now leaves a smaller deviation $\Delta^2 z_c = z_c - f_0(x_c, y_c) - f_1(x_c, y_c)$ for each point (x_c, y_c, z_c) in P.

In general, for each level k in the hierarchy, the point set $P_k = \{(x_c, y_c, \Delta^k z_c)\}$ is approximated by a function f_k defined over the tensor product grid Φ_k, where $\Delta^k z_c = z_c - \sum_{i=1}^{k-1} f_i(x_c, y_c) = \Delta^{k-1} z_c - f_{k-1}(x_c, y_c)$, and $\Delta^0 z_c = z_c$. As more levels are included in the hierarchy, the approximation errors $\Delta^k z_c$ of the function $g_k = \sum_{i=0}^{k} f_i$ gets smaller and smaller. The final approximation $f = g_h$ is defined as the sum of functions in (1) belonging to the nested sequence of spline spaces (2).

There are many possible ways of implementing this basic scheme, but the algorithms explored later can be described by the following general pseudo code.

[1] In an algorithmic setting, this will be slightly different as the grid must be adapted to the specific type of B-spline basis that is used.

Algorithm 1

1. ***For each*** *level* $k,\ k = 0, \ldots, h$, *in the hierarchy of (1)*
2. *Let* $P_k = \{p_c^k\} = \{(x_c, y_c, \Delta^k z_c)\}$
3. ***For each*** *point* p_c^k *in* P_k
4. *Calculate the contribution from* p_c^k *to coefficients in* Φ_k *in a local neighbourhood around* p_c^k
5. *For each coefficient in* Φ_k*, compromise between the contributions from the different points in* P_k *where the local neighbourhoods overlap*
6. *Calculate the differences* $\Delta^{k+1} z_c = \Delta^k z_c - f_k(x_c, y_c)$ *for each point in* P_k
7. *Let* Φ_{k+1} *be the next finer tensor product grid*
8. *Let* $k = k + 1$
9. *GOTO 1.*

We assume that the local neighbourhood in Φ_k of a point p_c^k in Step 4 is a fixed number of coefficients in Φ_k. More specifically, the coefficients will be those corresponding to tensor product B-spline basis functions that are non-zero at the location (x_c, y_c) of p_c^k. For cubic tensor product splines (whether C^1 or C^2 continuous), there are at most (4×4) non-zero bivariate basis functions that are non-zero at a point (x, y) in Ω. Thus, the local neighbourhood of a point p_c^k includes at most 16 coefficients. Figure 1 shows the (4×4)-neighbourhoods of two scattered points p_1 and p_2 where the neighbourhoods overlap.

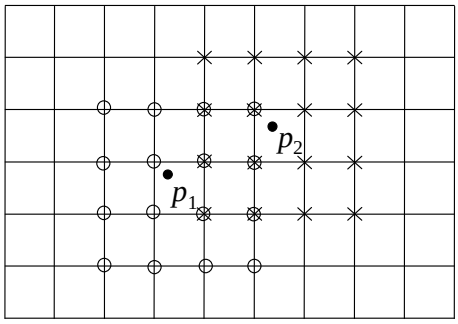

Figure 1: (4×4)-neighbourhoods of coefficients in the tensor product grid of two scattered data points p_1 and p_2.

From the general pseudo code in Algorithm 1 we can now estimate the complexity of the algorithm. The number of coefficients in Φ_k is a quarter of

that in the next finer grid Φ_{k+1}. Hence, if the number of coefficients of the tensor product grid Φ_h at the finest level is $(m \times n)$ and the number of points in P is N, the time complexity of Algorithm 1 is $O(N + mn) + O(N + \frac{1}{4}mn) + \cdots + O(N + \frac{1}{4^h}mn)$, which sums up to

$$O(hN + \frac{4}{3}mn). \tag{3}$$

This is extremely efficient compared to many other algorithms for scattered data approximation. For example, the optimal time complexity for computing Delaunay triangulations is $O(N \log N)$ [9]. Other methods like radial basis function interpolation [14] and the so-called Mask method [1] involve large linear equation systems, which also give time complexity beyond (3).

If the final surface f is represented with all the surfaces f_k in (1), the time complexity for evaluating f in a point (x, y) is $O(h)$ since all the surfaces must be evaluated to get the value $f(x, y) = \sum_{k=0}^{h} f_k(x, y)$. This can be made much more efficient by successive B-spline refinement of the surfaces f_k. Since the surfaces f_k belong to a nested sequence of subspaces as expressed in (2), it is clear that f can be represented as one single tensor product B-spline surface $F \subset S_h$ by a tensor product grid Ψ_h at the finest level h. Recall that Θ is a dyadic sequence of tensor product grids. The grid spacing is halved from one grid Φ_k to the next finer grid Φ_{k+1}, and the grid lines in Φ_k is a subset of the grid lines in Φ_{k+1}. Thus, a function $f_k \subset S_k$ defined by Φ_k can also be defined by a tensor product grid with the same grid spacing as Φ_{k+1} in the spline space S_{k+1}. If we denote by $\mathcal{R}_{1/2}$ the refinement operator which takes f_k from S_k to S_{k+1} by inserting grid lines halfway between the grid lines in Φ_k, the tensor product grid Ψ_h for $F == f$ can be expressed as,

$$\Psi_h = \mathcal{R}_{1/2}^h \Phi_0 + \mathcal{R}_{1/2}^{h-1} \Phi_1 + \cdots + \mathcal{R}_{1/2}^1 \Phi_{h-1} + \Phi_h = \sum_{k=0}^{h} \mathcal{R}_{1/2}^{h-k} \Phi_k, \tag{4}$$

where $\mathcal{R}_{1/2}^i$ denotes that the refinement operator is applied i times.

Instead of storing each Φ_k explicitly in Algorithm 1 and refining the final sequence Θ in a scheme following (4), Ψ_h can be derived progressively by refinement at each level of the hierarchy. This can be seen by writing Equation (4) in the alternative nested form,

$$\Psi_h = \mathcal{R}_{1/2} \left(\ldots \left(\mathcal{R}_{1/2} \left(\mathcal{R}_{1/2} \Phi_0 + \Phi_1 \right) + \Phi_2 \right) + \Phi_3 \right) \ldots) + \Phi_h. \tag{5}$$

At level $k = 1$ of Algorithm 1, the tensor product grid Ψ_1 of $g_1 = f_0 + f_1$ can be expressed as $\Psi_1 = \left(\mathcal{R}_{1/2} \Phi_0 + \Phi_1 \right)$, and in general, at level k, $\Psi_k = \mathcal{R}_{1/2} \Psi_{k-1} + \Phi_k$. Thus, Ψ_h can be derived by halving the grid spacing as we proceed from one level k to the next level $k + 1$. As the number of operations required for B-spline refinement is linear in the number of coefficients, the time complexity with these adjustments of Algorithm 1 is the same as in (3). The

space complexity, on the other hand, will be less as we do not need to store all the tensor product grids Ψ_k. More details on Algorithm 1, including adjustments with B-spline refinement, can be found in [12].

Note that the two versions of Algorithm 1, with or without B-spline refinement, produce identical results. The critical part of the algorithm is,

Step 4, how a point $p_c^k = \left(x_c, y_c, \Delta^k z_c\right)$ contributes to the coefficients of Φ_k in a local neighbourhood around p_c^k, and

Step 5, how to compromise between the contributions from different points in P_k where the local neighbourhoods overlap.

We will study these steps in detail in Section 4. In Section 5 we derive explicit formulas for the refinement operator $\mathcal{R}_{1/2}$ of the algorithm, both for C^2 and C^1 continuous bicubic B-splines. These exercises may be useful for extending the algorithm and adapt it to specific needs in applications.

3 Cubic C^2 Continuous Splines on Uniform Partitions

B-splines on uniform partitions are attractive as formulas can written down explicitly and implementation results in compact and easy-to-read code. In order to get a better understanding of multilevel B-splines in the setting of Algorithm 1, we first state some details on polynomial cubic C^2 continuous splines. C^1 continuous splines are studied in Section 5.2.

A B-spline basis function of degree three on a uniform partition $(-2, -1, 0, 1, 2)$ is defined as the piecewise polynomial,

$$B(t|-2,-1,0,1,2) = \begin{cases} (t+2)^3/6, & -2 < t \leq -1, \\ \left(-3t^3 - 6t^2 + 4\right)/6, & -1 \leq t < 0, \\ \left(3t^3 - 6t^2 + 4\right)/6, & 0 \leq t < 1, \\ (2-t)^3/6, & 1 \leq t < 2; \end{cases} \tag{6}$$

see Figure 2.

Let Φ be a uniform tensor product grid overlaid on a rectangular domain Ω. Without loss of generality we assume that $\Omega = [0, m) \times [0, n)$, where m and n are integers, and that Φ is an $(m+3) \times (n+3)$ lattice that spans the integer grid in Ω; see Figure 4. Let ϕ_{ij} be a coefficient of Φ located at position (i, j) of the integer grid defined by Φ. Further, let f be a bicubic spline function defined by Φ over Ω. Then the function value of f at a position $(x, y) \in \Omega$ is given as,

$$f(x,y) = \sum_{k=0}^{3} \sum_{l=0}^{3} B_k(s) B_l(t) \phi_{(i+k)(j+l)} \tag{7}$$

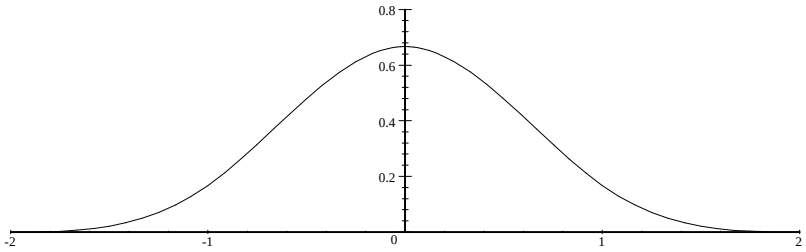

Figure 2: Uniform cubic B-spline basis function on the partition $(-2, -1, 0, 1, 2)$.

where $i = \lfloor x \rfloor - 1$, $j = \lfloor y \rfloor - 1$, $s = x - \lfloor x \rfloor$, and $t = y - \lfloor y \rfloor$. B_k and B_l are uniform cubic B-spline basis functions, which are translates of the equations in (6) covering the interval $0 \leq t < 1$,

$$\begin{aligned} B_0(t) &= (1-t)^3/6 \\ B_1(t) &= (3t^3 - 6t^2 + 4)/6 \\ B_2(t) &= (-3t^3 + 3t^2 + 3t + 1)/6 \\ B_3(t) &= t^3/6 \end{aligned} \tag{8}$$

for $0 \leq t < 1$; see Figure 3.

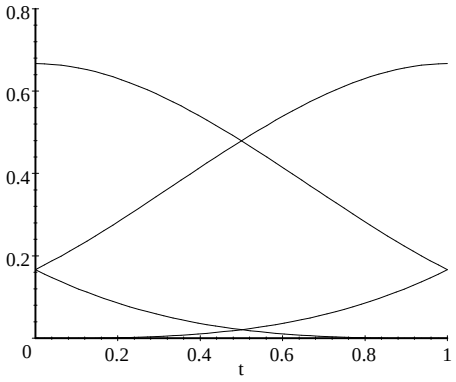

Figure 3: Uniform cubic B-spline basis functions in the C^2 continuous case on the interval $0 \leq t < 1$.

With this definition of uniform C^2 continuous cubic splines the necessary formulae for Algorithm 1 become very simple and they are easy to implement.

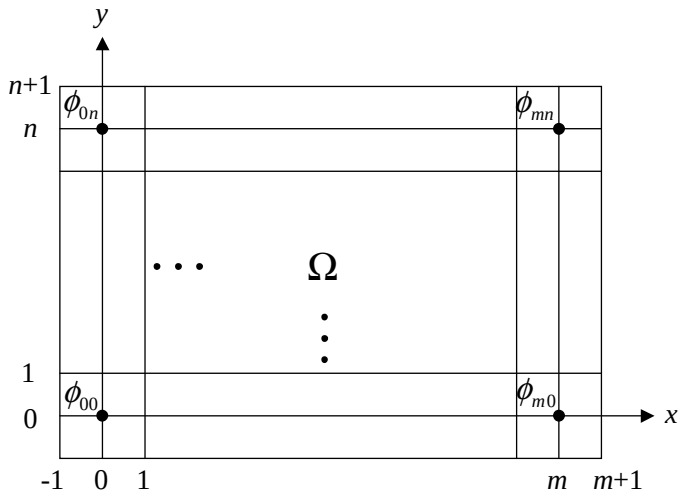

Figure 4: The tensor product grid Φ overlaid on the domain Ω.

We will also see in Section 5 that the refinement operator $\mathcal{R}_{1/2}$ takes a simple form and can be written down explicitly.

4 Details of Algorithm 1

We are now in a position to determine Step 4 and 5 of Algorithm 1. That is, how to calculate the (4×4) coefficients in the neighbourhood around a point $p_c^k = (x_c, y_c, \Delta^k z_c)$ of P_k, and how to compromise between the contributions from different points in P_k where the local neighbourhoods overlap; see Figure 1.

For simple notation we assume, without loss of generality, that $1 \leq x_c, y_c < 2$ such that $i = j = 0$ in Equation (7). Furthermore, we derive the necessary formulae for level $k = 0$ such that $p_c^k == p_c = (x_c, y_c, z_c)$ and $f_0 == f$.

4.1 Step 4 of Algorithm 1

It is natural to require exact interpolation of p_c in Step 4 of Algorithm 1 if there are no other scattered data points with a (4×4) neighbourhood of coefficients that overlap with that of p_c. In addition we may require a solution in the least square sense that minimises the square sum of the 16 coefficients. This leads to

the following standard constrained minimization problem,

$$\begin{aligned} \textit{minimise} \qquad J(\{\phi_{kl}\}) &= \sum_{k=0}^{3}\sum_{k=0}^{3}\phi_{kl}^2 && \text{(9a)} \\ \textit{subject to} \qquad h(\{\phi_{kl}\}) &= \sum_{k=0}^{3}\sum_{l=0}^{3} w_{kl}\phi_{kl} = z_c && \text{(9b)} \end{aligned}$$

where $w_{kl} = B_k(s)B_l(t)$ and $s = x - 1,\ t = y - 1$. Introducing the Lagrange multiplier λ, J and h must satisfy

$$\nabla J = \lambda \nabla h \tag{10}$$

where ∇ is the gradient operator,

$$\begin{aligned} \nabla J &= \left(\tfrac{\partial J}{\partial \phi_{00}}, \tfrac{\partial J}{\partial \phi_{01}}, \cdots, \tfrac{\partial J}{\partial \phi_{33}}\right) = (2\phi_{00}, 2\phi_{01}, \ldots, 2\phi_{33}) \\ \lambda\nabla h &= \left(\lambda\tfrac{\partial h}{\partial \phi_{00}}, \lambda\tfrac{\partial h}{\partial \phi_{01}}, \cdots, \lambda\tfrac{\partial h}{\partial \phi_{33}}\right) = (\lambda w_{00}, \lambda w_{01}, \ldots, \lambda w_{33}). \end{aligned}$$

Thus, (10) and the interpolation constraint (9b) gives a linear system of 17 equations in λ and the sixteen ϕ_{ij}'s.

$$\begin{aligned} 2\phi_{00} &= \lambda w_{00} \\ 2\phi_{01} &= \lambda w_{01} \\ &\cdots \\ 2\phi_{33} &= \lambda w_{33} \\ \sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}\phi_{ab} &= z_c. \end{aligned}$$

Eliminating λ we get a unique solution and an explicit formula for each of the 16 coefficients,

$$\phi_{kl} = \frac{w_{kl}\, z_c}{\sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}^2}, \quad (k,l) = (0,0), (0,1), \ldots, (3,3). \tag{11}$$

The coefficients at positions in the integer grid of Φ near (x_c, y_c) get larger values since they are associated with larger values of w_{kl}. Thus, $f(x,y) = \sum_{k=0}^{3}\sum_{l=0}^{3} w_{kl}\phi_{kl}$ has the value z_c at (x_c, y_c) and then tapers off smoothly away from (x_c, y_c).

Note that the choice of coefficients in (11) does not give affine invariance for f. If z_c is changed to a different z_c', then the gradient field of f is also changed. This is due to the fact that the object function J in (9a) tends to minimise the deviation of f from zero over the domain. A solution proposed in [12] is to make an initial approximation f_{-1}, for example a least squares linear fit to the

original scattered data P_0, and build the approximation function f over f_{-1}. Thus, the initial data for Algorithm 1 will be $P_0' = \{(x_c, y_c, z_c - f_{-1}(x_c, y_c))\}$ and the object function J will tend to minimise the deviation of f from f_{-1} over the domain.

An alternative way to construct the object function is to let it tend to minimise the deviation of f from z_c in the local neighbourhood of (x_c, y_c),

$$\textit{minimise} \qquad J(\{\phi_{kl}\}) = \sum_{k=0}^{3}\sum_{k=0}^{3}(\phi_{kl} - z_c)^2$$

subject to the same interpolation condition as in (9b). This gives a minimum at $J(\{\phi_{kl}\}) = 0$ with the solution

$$\phi_{kl} = z_c, \quad (k,l) = (0,0), (0,1), \ldots, (3,3).$$

That is, all the 16 coefficients are equal and f takes the constant value z_c inside the rectangular grid cell of Φ where (x_c, y_c) is located. This can be combined with the solution in (11) through a linear blend,

$$\phi_{kl} = (1-\alpha)\frac{w_{kl} z_c}{\sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}^2} + \alpha z_c, \quad (k,l) = (0,0), (0,1), \ldots, (3,3), \qquad (12)$$

with $\alpha \in [0,1]$, which also satisfy the interpolation condition $f(x_c, y_c) = z_c$. Compared to (11), the effect is that the approximation function f tapers off slower away from (x_c, y_c) if $0 < \alpha < 1$.

4.2 Step 5 of Algorithm 1

For each point $p_c = (x_c, y_c, z_c)$, Equation (12) gives a different coefficient ϕ_c for ϕ_{ij} where the local neighbourhoods of (4×4) coefficients overlap,

$$\phi_c = (1-\alpha)\frac{w_c z_c}{\sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}^2} + \alpha z_c, \quad t \in [0,1].$$

In Step 5 of Algorithm 1 we compromise between the different ϕ_c's to get a unique value for ϕ_{ij}. This can be done simply by a weighted average,

$$\phi_{ij} = \frac{\sum_c w_c \phi_c}{\sum_c w_c}, \qquad (13)$$

where the weights w_c are the values of the tensor product basis functions at the locations (x_c, y_c) of points with overlapping neighbourhoods. The contributions from points located near ϕ_{ij} in the tensor product grid now get larger values since they are associated with larger w_c.

Alternatively, as proposed in [12], ϕ_{ij} can be chosen to minimise $e(\phi_{ij}) = \sum_c \left(w_c\phi_{ij} - w_c\phi_c\right)^2$. Differentiating $e(\phi_{ij})$ with respect to ϕ_{ij}, gives

$$\phi_{ij} = \frac{\sum_c w_c^2 \phi_c}{\sum_c w_c^2}. \qquad (14)$$

Although experiments show that using (14) gives better approximation to the scattered data in most cases, the alternative with (13) is less sensitive to outliers in the data. If ϕ_{ij} has contribution from one point p_c only, both alternatives (13) and (14) reduces to $\phi_{ij} = \phi_c$. Thus, ϕ_{ij} is determined by (12) and leaves no approximation error due to the interpolation condition (9b).

4.3 Zero-Coefficients and Data Compression

When a coefficient ϕ_{ij}^k in the tensor product grid Φ_k does not belong to any (4×4)-neighbourhood of coefficients of any data point in P, then ϕ_{ij}^k has no influence on $f(x_c, y_c)$ for any point (x_c, y_c, z_c) in P. Thus, ϕ_{ij}^k can be assigned an arbitrary value, without affecting the approximation error. The natural approach in this case is simply to assign zero to ϕ_{ij}^k. At level $k = 0$ of Algorithm 1, the tensor product grid Φ_0 can be made sufficiently coarse such that any ϕ_{ij}^0 of Φ_0 belongs to at least one (4×4)-neighbourhood of coefficients around data points in P_0. But as Φ_k gets denser at increasing levels k, more and more coefficients of Φ_k falls outside the (4×4)-neighbourhoods, and more ϕ_{ij}^k's must be assigned zero. This may cause anomalies, especially if the scattered data are unevenly distributed in the domain. The approximation function $f = \sum_k f_k$ is pulled more and more towards the data points locally with increasing levels k, while the neighbourhoods with no data are not changed.

On the other hand, the fact that more and more coefficients can be assigned zero at the finer levels can be utilised for data compression in an adaptive version of Algorithm 1. The tensor product grids Φ_k, which become more and more sparse with increasing k, can be represented compactly in linear arrays by storing only those coefficients that are non-zero. Thus, the grid Φ_h at the finest level may be extremely dense, for example, (1 million$\times$1 million) grid lines.

Note that with this approach we cannot benefit from computational savings of using B-spline refinement. In that case the refinement operator $\mathcal{R}_{1/2}$ fills in non-zero coefficients such that the tensor product grids Φ_k at each level represent full matrices of coefficients; see the next sections. The adaptive algorithm also involves sorting at each level k of the hierarchy such that the overall computational complexity becomes $O(N \log N)$; see [12] for details.

5 B-Spline Refinement on Uniform Partitions

In this section we derive explicit expressions for the refinement operator $\mathcal{R}_{1/2}$ introduced in Section 2. With the restrictions to uniform partitions, the refinement operator results in simple formulae which can be implemented compactly in optimised code. Two cases are considered; cubic C^2 continuous splines and cubic C^1 continuous splines on tensor product grids. The motivation for considering C^1 continuous splines is the fact that they can, in some sense, possess more shape preserving properties than C^2 continuous splines (on the expense of smoothness); see for example [8]. A general algorithm for B-spline refinement,

the so-called "Oslo algorithm", without restriction to uniform partitions was developed by Cohen, Lyche and Riesenfeld [3].

5.1 Refinement of Bicubic C^2 Continuous Splines

The refinement operator $\mathcal{R}_{1/2}$ takes f_k from the spline space S_k to S_{k+1} by inserting grid lines halfway between the grid lines in Φ_k to obtain Φ_{k+1}. If we denote by $R_i(t),\ i = 0, 1, 2, 3$, the univariate B-spline basis functions in the refined space, then the $R_i(t)$ are (dyadic) scalings of the basis functions in (8),

$$R_i(t) = B_i(2t)$$

on the interval $0 \le t < 0.5$,

$$\begin{aligned} R_0(t) &= \frac{1}{6}(1-2t)^3 \\ R_1(t) &= 4t^3 - 4t^2 + \frac{2}{3} \\ R_2(t) &= -4t^3 + 2t^2 + t + \frac{1}{6} \\ R_3(t) &= \frac{4}{3}t^3; \end{aligned} \tag{15}$$

see Figure 5.

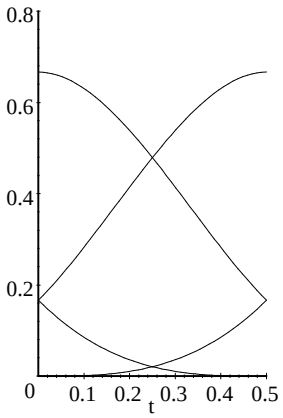

Figure 5: The univariate cubic B-spline basis functions $R_i(t) = B_i(2t),\ i = 0, 1, 2, 3$ in the refined space S_{k+1} in the C^2 continuous case.

A univariate spline function $f(t)$ can now also be expressed in the refined space S_{k+1} on the interval $0 \le t < 0.5$,

$$f(t) = \sum_{i=0}^{3} r_i R_i(t) = \sum_{i=0}^{3} c_i B_i(t), \tag{16}$$

where $\{r_i\}$ and $\{c_i\}$ are spline coefficients in S_{k+1} and S_k respectively. Inserting from the equations (8) and (15), and sorting the terms by degree in t in both spaces we get,

$$\begin{aligned}
& \left(-\frac{4}{3}r_0 - 4r_2 + 4r_1 + \frac{4}{3}r_3\right) t^3 + (2r_2 - 4r_1 + 2r_0)\, t^2 + (-r_0 + r_2)\, t \\
& + \left(\frac{1}{6}r_0 + \frac{2}{3}r_1 + \frac{1}{6}r_2\right) \\
= \quad & \left(-\frac{1}{6}c_0 - \frac{1}{2}c_2 + \frac{1}{2}c_1 + \frac{1}{6}c_3\right) t^3 + \left(\frac{1}{2}c_2 - c_1 + \frac{1}{2}c_0\right) t^2 \\
& + \left(-\frac{1}{2}c_0 + \frac{1}{2}c_2\right) t + \left(\frac{1}{6}c_0 + \frac{2}{3}c_1 + \frac{1}{6}c_2\right).
\end{aligned}$$

Comparing term by term for each degree in t, we get a linear system of 4 equations in the unknowns r_i,

$$\begin{bmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -4 & 2 & 0 \\ -\frac{4}{3} & 4 & -4 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{6}c_0 + \frac{2}{3}c_1 + \frac{1}{6}c_2 \\ -\frac{1}{2}c_0 + \frac{1}{2}c_2 \\ \frac{1}{2}c_0 - c_1 + \frac{1}{2}c_2 \\ -\frac{1}{6}c_0 + \frac{1}{2}c_1 - \frac{1}{2}c_2 + \frac{1}{6}c_3 \end{bmatrix}.$$

The solution gives the coefficients in the refined space in terms of the known coefficients in the original space,

$$\begin{bmatrix} r_0 = \frac{1}{2}(c_0 + c_1) \\ r_1 = \frac{1}{8}(c_0 + 6c_1 + c_2) \\ r_2 = \frac{1}{2}(c_1 + c_2) \\ r_3 = \frac{1}{8}(c_1 + 6c_2 + c_3) \end{bmatrix}$$

Note that r_2 is just a "translate" of r_0, and likewise for r_3 and r_1. This is due to symmetry of the B-spline basis function around a grid point; see Figure 2. r_0 and r_2 correspond to new grid points in the refined grid I' halfway between existing grid points in the original grid I, and r_3 and r_1 correspond to new grid points in the same position as those in the original grid. If we denote by ϕ'_{2i} a coefficient associated with a grid point in I' which coincides with that of ϕ_i in I, then the coefficients in I' are obtained from those in I by (see also Figure 6),

$$\phi'_{2i} = \frac{1}{8}(\phi_{i-1} + 6\phi_i + \phi_{i+1}) \tag{17a}$$

$$\phi'_{2i+1} = \frac{1}{2}(\phi_i + \phi_{i+1}). \tag{17b}$$

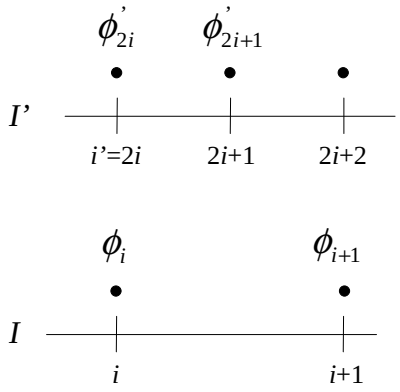

Figure 6: New coefficients (primed) of the refined grid I' in the univariate C^2 case.

We now turn to the bivariate case to find the coefficients of the refined tensor product grid Φ_{k+1} in terms of the coefficients of Φ_k. Let $f(s,t) = \sum_k \sum_l c_{kl} B_k(s) B_l(t)$ denote a uniform tensor product B-spline surface in the space defined by Φ_k. This equation can be rewritten as

$$f(s,t) = \sum_k B_k(t) \left[\sum_l c_{kl} B_l(s) \right].$$

Notice that, for each row number k in Φ_k, the sum in the square brackets describes a uniform B-spline curve in the variable s. Thus, each row in the tensor product has a corresponding B-spline curve. We can now refine these curves for all k using the univariate refinement operator in (17),

$$f(s,t) = \sum_k B_k(t) \left[\sum_l r'_{kl} R_l(s) \right] = \sum_k \sum_l r'_{kl} R_l(s) B_k(t). \tag{18}$$

The result so far is the original spline surface expressed by a new tensor product grid Φ'_{k+1} where new grid lines in the t-direction have been inserted halfway between the grid lines of Φ_k. Similarly as above we can rewrite (18) as

$$f(s,t) = \sum_l R_l(s) \left[\sum_k r'_{kl} B_k(t) \right],$$

such that for each column l in Φ'_{k+1}, the sum in the square brackets describes a uniform B-spline curve in the variable t. Again, these curves can be refined by the univariate refinement operator in (17),

$$f(s,t) = \sum_l R_l(s) \left[\sum_k r_{kl} R_k(t) \right] = \sum_k \sum_l r_{kl} R_l(s) R_k(t).$$

Thus, refinement has been done in both directions of the tensor product grid such that $\{r_{kl}\}$ are coefficients of the refined grid Φ_{k+1}. It is now an easy exercise to determine the bivariate refinement operator $\mathcal{R}_{1/2}$ from the univariate operator (17). From the coefficients on the right hand side of (17a) and 17b we make the matrices $M_1 = [1/8, 6/8, 1/8]$ and $M_2 = [1/2, 1/2]$. The following matrix products does the job,

$$
\begin{aligned}
M_1^T M_1 &= \begin{bmatrix} 1/8 \\ 6/8 \\ 1/8 \end{bmatrix} \begin{bmatrix} 1/8 & 6/8 & 1/8 \end{bmatrix} = \begin{bmatrix} 1/64 & 6/64 & 1/64 \\ 6/64 & 36/64 & 6/64 \\ 1/64 & 6/64 & 1/64 \end{bmatrix} \\
M_1^T M_2 &= \begin{bmatrix} 1/8 \\ 6/8 \\ 1/8 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} 1/16 & 1/16 \\ 6/16 & 6/16 \\ 1/16 & 1/16 \end{bmatrix} \\
M_2^T M_1 &= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/8 & 6/8 & 1/8 \end{bmatrix} = \begin{bmatrix} 1/16 & 6/16 & 1/16 \\ 1/16 & 6/16 & 1/16 \end{bmatrix} \\
M_2^T M_2 &= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}.
\end{aligned}
$$

Similar as in the univariate case, let $\phi'_{2i,2j}$ correspond to a grid point in Φ_{k+1} which coincides with ϕ_{ij} in Φ_k. Then the coefficients of Φ_{k+1} are obtained from those in Φ_k by the matrix products above,

$$
\begin{aligned}
\phi'_{2i,2j} &= \frac{1}{64}[\phi_{i-1,j-1} + \phi_{i-1,j+1} + \phi_{i+1,j-1} + \phi_{i+1,j+1} \\
&\qquad +6\left(\phi_{i-1,j} + \phi_{i,j-1} + \phi_{i,j+1} + \phi_{i+1,j}\right) + 36\phi_{i,j}] \\
\phi'_{2i,2j+1} &= \frac{1}{16}\left[\phi_{i-1,j} + \phi_{i-1,j+1} + \phi_{i+1,j} + \phi_{i+1,j+1} + 6\left(\phi_{i,j} + \phi_{i,j+1}\right)\right] \\
\phi'_{2i+1,2j} &= \frac{1}{16}\left[\phi_{i,j-1} + \phi_{i,j+1} + \phi_{i+1,j-1} + \phi_{i+1,j+1} + 6\left(\phi_{i,j} + \phi_{i+1,j}\right)\right] \\
\phi'_{2i+1,2j+1} &= \frac{1}{4}\left[\phi_{i,j} + \phi_{i,j+1} + \phi_{i+1,j} + \phi_{i+1,j+1}\right].
\end{aligned}
$$

The refinement operator $\mathcal{R}_{1/2}$ can now be implemented as four separate functions operating on the coefficient matrix Φ_k to obtain the new refined coefficient matrix Φ_{k+1}.

5.2 Refinement of Bicubic C^1 Continuous Splines

The B-spline basis functions for C^1 continuous polynomial splines, which are outlined below, do not possess the symmetry property of basis functions in the C^2 case. We must now consider two B-spline basis functions,

$$
B(t|-1,-1,0,0,1) = \begin{cases} \left(-5t^3 - 9t^2 - 3t + 1\right)/2, & -1 \le t < 0, \\ (1-t)^3/2, & 0 \le t < 1, \end{cases} \tag{19}
$$

on the partition $(-1,-1,0,0,1)$, and

$$B(t|-1,0,0,1,1) = \begin{cases} (t+1)^3/2, & -1 < t \le 0, \\ \left(5t^3 - 9t^2 + 3t + 1\right)/2, & 0 \le t < 1, \end{cases} \tag{20}$$

on the partition $(-1,0,0,1,1)$; see Figure 7.

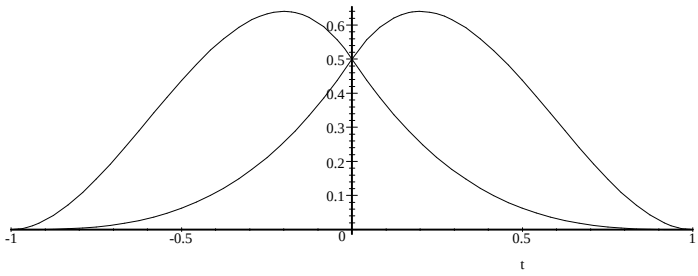

Figure 7: Cubic B-spline basis functions on the partitions $(-1,-1,0,0,1)$ and $(-1,0,0,1,1)$.

The translates $B(t+i)$ of (19) and (20) covering the interval $0 \le t < 1$ are given as,

$$\begin{aligned} B_0(t) &= (1-t)^3/2 \\ B_1(t) &= \left(5t^3 - 9t^2 + 3t + 1\right)/2 \\ B_2(t) &= \left(-5t^3 + 6t^2\right)/2 \\ B_3(t) &= t^3/2; \end{aligned} \tag{21}$$

see Figure 8.

In the univariate case we now associate two coefficients with each grid point. Thus, given a grid point with index i, then the coefficients "left" and "right" of i are ϕ_{2i-1} and ϕ_{2i} respectively. In the refined partition the grid point index i' that correspond to an old grid point i is $i' = 2i$. Thus, the coefficients "left" and "right" of an existing grid point in the refined grid are ϕ'_{4i-1} and ϕ'_{4i} respectively, and the coefficients in new positions "left" and "right" of a new grid point are ϕ'_{4i+1} and ϕ'_{4i+2}.

The tensor product spline function f at a position $(x,y) \in \Omega$ is given as,

$$f(x,y) = \sum_{k=0}^{3}\sum_{l=0}^{3} B_k(s)B_l(t)\phi_{(i+k)(j+l)}$$

where $i = 2\lfloor x \rfloor - 1$, $j = 2\lfloor y \rfloor - 1$, $s = x - \lfloor x \rfloor$, and $t = y - \lfloor y \rfloor$, and B_l and B_k are the basis functions in (21).

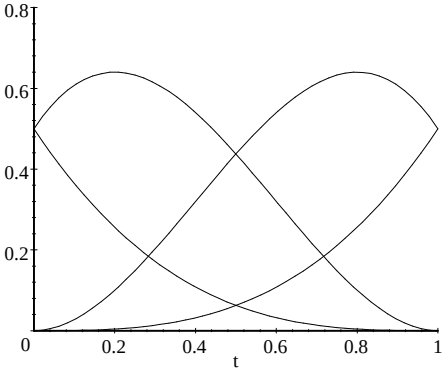

Figure 8: Cubic B-spline basis functions in the C^1 continuous case on the interval $0 \leq t < 1$.

The univariate B-spline basis functions $R_i(t) = B_i(2t),\ i = 0, 1, 2, 3$, in the refined space are now,

$$\begin{aligned} R_0(t) &= \frac{1}{2}(1-2t)^3 \\ R_1(t) &= 20t^3 - 18t^2 + 3t + \frac{1}{2} \\ R_2(t) &= -20t^3 + 12t^2 \\ R_3(t) &= 4t^3, \end{aligned}$$

on the interval $0 \leq t < 0.5$; see Figure 9.

We proceed exactly as in the C^2 continuous case and get the following formulae for the refinement operator in the univariate case (see also Figure 10),

$$\phi'_{4i-1} = \frac{3}{4}\phi_{2i-1} + \frac{1}{4}\phi_{2i} \quad (\text{“left” of existing grid point}) \tag{22a}$$

$$\phi'_{4i} = \frac{1}{4}\phi_{2i-1} + \frac{3}{4}\phi_{2i} \quad (\text{“right” of existing grid point}) \tag{22b}$$

$$\phi'_{4i+1} = \frac{1}{8}\phi_{2i-1} + \frac{5}{8}\phi_{2i} + \frac{1}{4}\phi_{2i+1} \quad (\text{“left” of new grid point}) \tag{22c}$$

$$\phi'_{4i+2} = \frac{1}{4}\phi_{2i} + \frac{5}{8}\phi_{2i+1} + \frac{1}{8}\phi_{2i+2} \quad (\text{“right” of new grid point}) \tag{22d}$$

Note that all these four expressions must be considered while the C^2 case only involved the two expressions in (17) due to symmetry of the basis function.

Again we proceed as for C^2 continuous splines to obtain the refinement operator $\mathcal{R}_{1/2}$ in the bivariate case. Four coefficients are now associated with each

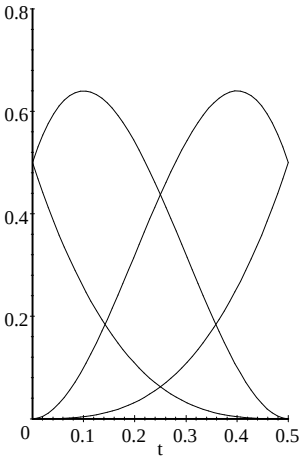

Figure 9: The cubic B-spline basis functions $R_i(t) = B_i(2t),\ i = 0, 1, 2, 3$ in the refined space S_{k+1} in the C^1 continuous case.

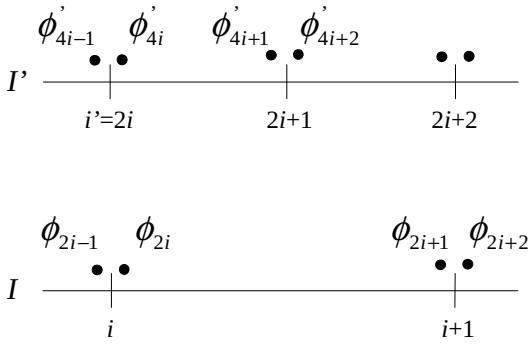

Figure 10: New coefficients (primed) of the refined grid I' in the univariate C^1 case.

point in the tensor product grid. From the four equations (22a) – (22d) we make the matrices $M_1 = [3/4, 1/4]$, $M_2 = [1/4, 3/4]$, $M_3 = [1/8, 5/8, 1/4]$ and $M_4 = [1/4, 5/8, 1/8]$. We form sixteen matrix products $M_1^T M_1,\ M_1^T M_2,\ M_1^T M_3, \ldots, M_4^T M_4$, and the coefficients of Φ_{k+1} in the refined space S_{k+1} are obtained from those of Φ_k in S_k by the following sixteen equations,

$$
\begin{aligned}
\phi'_{4i-1,4j-1} &= \frac{1}{16}\left[9\phi_{2i-1,2j-1} + 3(\phi_{2i-1,2j} + \phi_{2i,2j-1}) + \phi_{2i,2j}\right] \\
\phi'_{4i,4j} &= \frac{1}{16}\left[\phi_{2i-1,2j-1} + 3(\phi_{2i-1,2j} + \phi_{2i,2j-1}) + 9\phi_{2i,2j}\right] \\
\phi'_{4i+1,4j+1} &= \frac{1}{64}[\phi_{2i-1,2j-1} + 5\left(\phi_{2i-1,2j} + \phi_{2i,2j-1}\right) + 25\phi_{2i,2j} \\
&\quad +2\left(\phi_{2i-1,2j+1} + \phi_{2i+1,2j-1}\right) + 10\left(\phi_{2i+1,2j} + \phi_{2i,2j+1}\right) + 4\phi_{2i+1,2j+1}] \\
\phi'_{4i+2,4j+2} &= \frac{1}{64}[4\phi_{2i,2j} + 5\phi_{2i+2,2j+1} + 2\left(\phi_{2i+2,2j} + \phi_{2i,2j+2}\right) + \phi_{2i+2,2j+2} \\
&\quad +25\phi_{2i+1,2j+1} + 10\left(\phi_{2i+1,2j} + \phi_{2i,2j+1}\right) + 5\phi_{2i+1,2j+2}] \\
\phi'_{4i-1,4j} &= \frac{1}{16}\left[3\left(\phi_{2i-1,2j-1} + \phi_{2i,2j}\right) + 9\phi_{2i-1,2j} + \phi_{2i,2j-1}\right] \\
\phi'_{4i,4j-1} &= \frac{1}{16}\left[3\left(\phi_{2i-1,2j-1} + \phi_{2i,2j}\right) + \phi_{2i-1,2j} + 9\phi_{2i,2j-1}\right] \\
\phi'_{4i-1,4j+1} &= \frac{1}{32}[3\phi_{2i-1,2j-1} + 15\phi_{2i-1,2j} + \phi_{2i,2j-1} + 5\phi_{2i,2j} + 2\phi_{2i,2j+1} + 6\phi_{2i-1,2j+1}] \\
\phi'_{4i+1,4j-1} &= \frac{1}{32}[3\phi_{2i-1,2j-1} + \phi_{2i-1,2j} + 15\phi_{2i,2j-1} + 5\phi_{2i,2j} + 2\phi_{2i+1,2j} + 6\phi_{2i+1,2j-1}] \\
\phi'_{4i-1,4j+2} &= \frac{1}{32}\left[6\phi_{2i-1,2j} + 2\phi_{2i,2j} + 15\phi_{2i-1,2j+1} + 5\phi_{2i,2j+1} + \phi_{2i,2j+2} + 3\phi_{2i-1,2j+2}\right] \\
\phi'_{4i+2,4j-1} &= \frac{1}{32}\left[6\phi_{2i,2j-1} + 2\phi_{2i,2j} + \phi_{2i+2,2j} + 15\phi_{2i+1,2j-1} + 5\phi_{2i+1,2j} + 3\phi_{2i+2,2j-1}\right] \\
\phi'_{4i,4j+1} &= \frac{1}{32}\left[\phi_{2i-1,2j-1} + 5\phi_{2i-1,2j} + 3\phi_{2i,2j-1} + 15\phi_{2i,2j} + 2\phi_{2i-1,2j+1} + 6\phi_{2i,2j+1}\right] \\
\phi'_{4i+1,4j} &= \frac{1}{32}\left[\phi_{2i-1,2j-1} + 3\phi_{2i-1,2j} + 5\phi_{2i,2j-1} + 15\phi_{2i,2j} + 2\phi_{2i+1,2j-1} + 6\phi_{2i+1,2j}\right] \\
\phi'_{4i,4j+2} &= \frac{1}{32}\left[2\phi_{2i-1,2j} + 6\phi_{2i,2j} + 5\phi_{2i-1,2j+1} + 15\phi_{2i,2j+1} + \phi_{2i-1,2j+2} + 3\phi_{2i,2j+2}\right] \\
\phi'_{4i+2,4j} &= \frac{1}{32}\left[2\phi_{2i,2j-1} + 6\phi_{2i,2j} + 5\phi_{2i+1,2j-1} + 15\phi_{2i+1,2j} + \phi_{2i+2,2j-1} + 3\phi_{2i+2,2j}\right] \\
\phi'_{4i+1,4j+2} &= \frac{1}{64}[2\left(\phi_{2i-1,2j} + \phi_{2i+1,2j+2}\right) + 10\left(\phi_{2i,2j} + \phi_{2i+1,2j+1}\right) \\
&\quad +5\left(\phi_{2i-1,2j+1} + \phi_{2i,2j+2}\right) + 25\phi_{2i,2j+1} + 4\phi_{2i+1,2j} + \phi_{2i-1,2j+2}] \\
\phi'_{4i+2,4j+1} &= \frac{1}{64}[2\left(\phi_{2i,2j-1} + \phi_{2i+2,2j+1}\right) + 10\left(\phi_{2i,2j} + \phi_{2i+1,2j+1}\right) \\
&\quad +5\left(\phi_{2i+1,2j-1} + \phi_{2i+2,2j}\right) + 4\phi_{2i,2j+1} + 25\phi_{2i+1,2j} + \phi_{2i+2,2j-1}].
\end{aligned}
$$

6 Numerical Examples and Discussion

The algorithms have been tested on a variety of different types of measurement data: interpreted seismic data representing geological faults and horizons, seabed data obtained from multi-beam echo sounders, data from 3D scanning devices, different GIS-data representing terrain, and parametrised 3D scattered data.

Common to all these data sets are that they contain noise and outliers. Recall that exact interpolation of a data point p_c is achieved if there are no other scattered data points with a corresponding (4×4) neighbourhood of coefficients which overlap with that of p_c. Thus, the tensor product grid at the finest level in the surface hierarchy should not be too dense as this would yield exact interpolation of the data and result in oscillations and anomalies as explained earlier. Even if the data were accurate without noise and outliers, exact interpolation may cause local peaks near the scattered data.

Some of the data sets mentioned above have some nice properties which make them well suited for approximation with multilevel B-splines. They are typically well distributed and they are densely sampled in the domain. Thus, many scattered data points contribute to each coefficient in the tensor product grid by Equation (13) and (14). Even if there are coincident points in the domain with different z-values, these equations ensure that the spline surface possesses a good weighted average of the scattered data points.

A spline surface with 4097×8193 coefficients was created from a large data set consisting of approximately 53 million data points. The data set was a mixture of terrain data and seabed data from multi-beam echo sounders from a huge area of $137\texttt{km} \times 300\texttt{km}$ from the coast of Norway. Thus, the size of each grid cell was approximately 33.4×36.6 meters. This gives an average of 1.6 points per grid cell, and since the data were well distributed in most of the domain, many scattered data points contributed to each spline coefficient. The algorithms performed remarkably well on this data set, though some irregularities could be observed in areas where the scattered data were sparse. (We have no images from this experiment.)

The terrain data in Figure 11 consists of approximately $48,000$ points from digitised contours over a domain of $14.9\texttt{km} \times 27.8\texttt{km}$. Since the data are digitised contours, they are not well distributed in the domain. If the tensor product grid is made too dense, for example 1025×2049 coefficients as in Figure 14, many of the contours are visible in the 3D surface as "terraces". In this case each grid cell is approximately 14.5×13.6 meters and the average deviation (measured vertically) of the spline surface from the given data is 0.23 meters. If the tensor product grid is reduced to 257×513 coefficients with grid cells 58.1×54.4 meters, leaving an average error of 1.77 meters, many of the visual irregularities would disappear. This would be sufficient for visualization, but for terrain analysis in GIS-applications, one must probably operate at one dyadic level lower with grid cells 116.2×108.7 meters, which makes the model smoother (see Figure 13). This leaves an average error of 3.93 meters. The terrain models shown are C^2 spline surfaces and the plots are scaled by a factor of four in the z-direction.

On an 800 Mz PC with 256 Mb RAM, the CPU-time used to create the spline surface with 129×257 coefficients was 1.7 seconds, and 3.7 seconds was used to create the finest surface with 1025×2049 coefficients (cf. Equation 3).

The 3D scattered data in Figure 15, sampled from a head and a foot, were parametrised over rectangular domains by a method developed by Floater [7]. Each point in the parameter domain corresponds to a scattered data point in 3D space. The rectangular boundary of the domains correspond to boundaries of the physical models (below the neck in the head-model and above the ankle in the foot-model). Due to the (shape preserving) parametrisation method that was used, the points are unevenly distributed in the parameter domain; see the figures in the middle. Thus, extremely dense tensor product grids were necessary to recover all details in the models - for example, the foot-model in the figure is represented by 16384×16384 B-spline coefficients. The adaptive approach explained in Section 4.3 was in this case used to limit storage requirements. Some irregularities are visible in the 3D plots near the boundaries where the tensor product grids are dense compared to the density of the parametrised points.

Acknowledgments *This work was supported by the Research Council of Norway through basic funding to SINTEF. The author thanks Tor Dokken, Michael Floater and Kai Hormann for fruitful discussions on multilevel splines and related topics during this work; and Kyrre Strøm for useful help when deriving explicit formulae for refinement operators.*

References

[1] E. Arge, M. Dæhlen, and A. Tveito. Approximation of scattered data using smooth grid functions. *Journal of Computational and Applied Mathematics*, 59:191–205, 1995.

[2] C. K. Chui. *An Introduction to Wavelets*. Academic Press, Boston, 1992.

[3] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-Splines and subdivision techniques in computer aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14(2):87–111, 1980.

[4] M. Dæhlen and Ø. Hjelle. Compact representation of seismic sections. Technical Report 910125-3, SINTEF, 1992.

[5] M. Dæhlen and T. Lyche. Decomposition of splines. In T. Lyche and L. L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 135–160. Academic Press, New York, 1992.

[6] I. Daubechies. *Ten Lectures on Wavelets*. SIAM Publications, Philadelphia, 1992.

[7] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[8] J. E. Lavery. Univariate cubic L_p splines and shape-preserving, multiscale interpolation by univariate cubic L_1 splines. *Computer Aided Geometric Design*, 17(4):319–336, 2000.

[9] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[10] S. Lee, K.-Y. Chwa, J. Hahn, S. Y. Shin, and G. Wolberg. Image metamorphosis using snakes and free-form deformations. In *Computer Graphics, SIGRAPH'95*, volume 3, pages 439–448, 1995.

[11] S. Lee, G. Wolberg, K.-Y. Chwa, and S. Y. Shin. Image metamorphosis with scattered feature constraints. *IEEE Transaction on Visualization and Computer Graphics*, 2(4):337–354, 1996.

[12] S. Lee, G. Wolberg, and S. Y. Shin. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):229–244, 1997.

[13] Y. Meyer. *Ondolettes et Operateurs*. Hermann, Paris, 1990.

[14] M. J. D. Powell. The theory of radial basis function approximation in 1990. In W. Light, editor, *Advances in Numerical Analysis, Vol II*, pages 105–210. Oxford Science Publications, 1992.

A Figures from Numerical Experiments

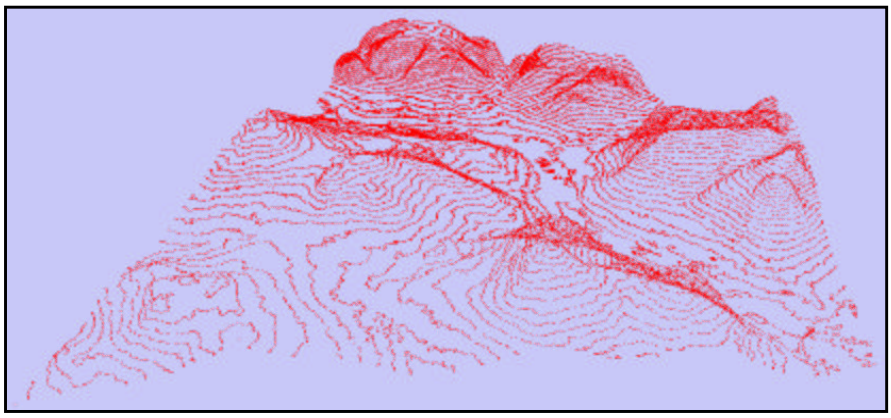

Figure 11: Contour data as input to the surface model shown in the next figures.

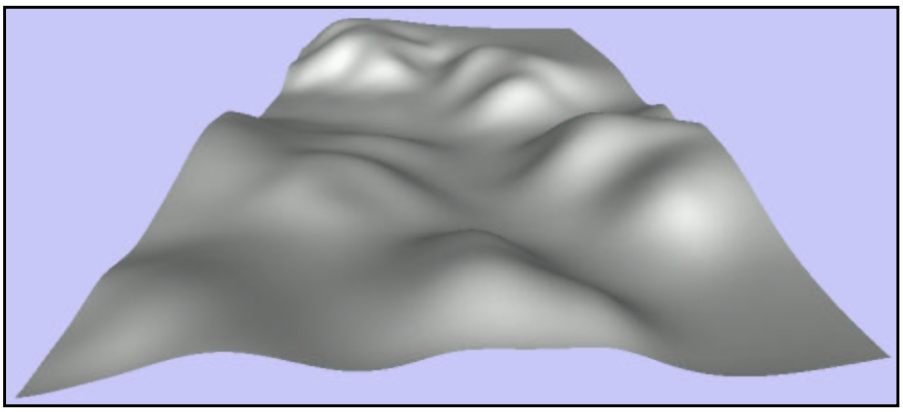

Figure 12: Surface model at a coarse (and smooth) level with 33×65 B-spline coefficients.

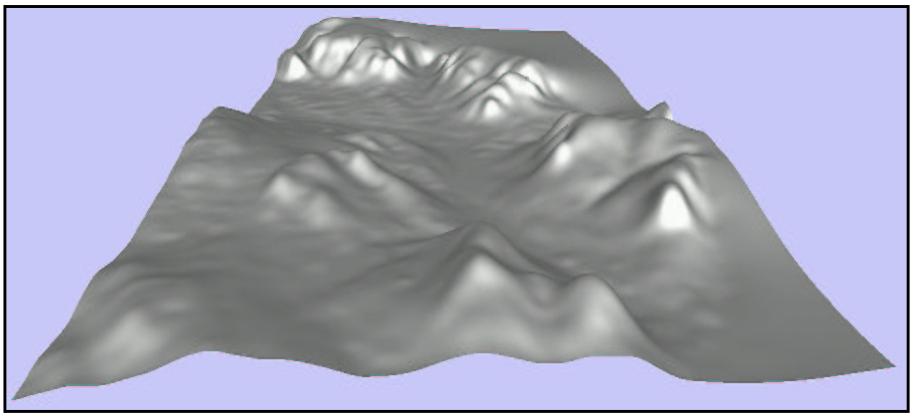

Figure 13: Surface model with 129×257 B-spline coefficients.

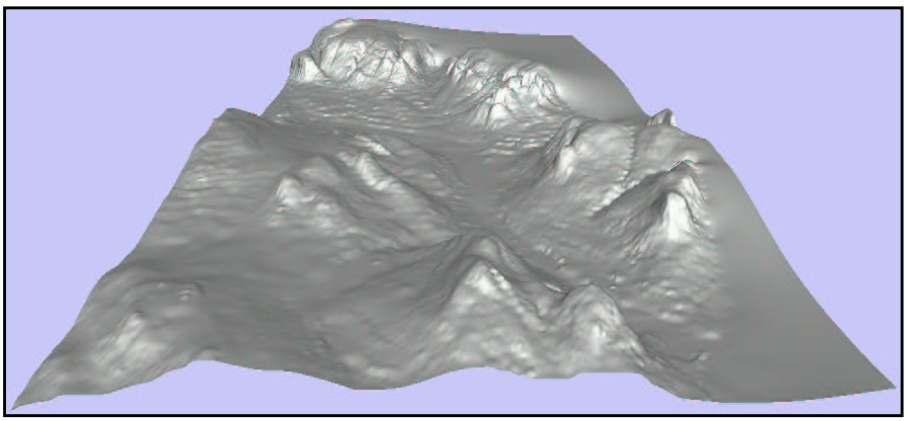

Figure 14: Surface model with 1025×2049 B-spline coefficients. This is too dense relative to the density of the contour data shown in Figure 11. There are anomalies in the 3D model visible as "terraces" along the contour data.

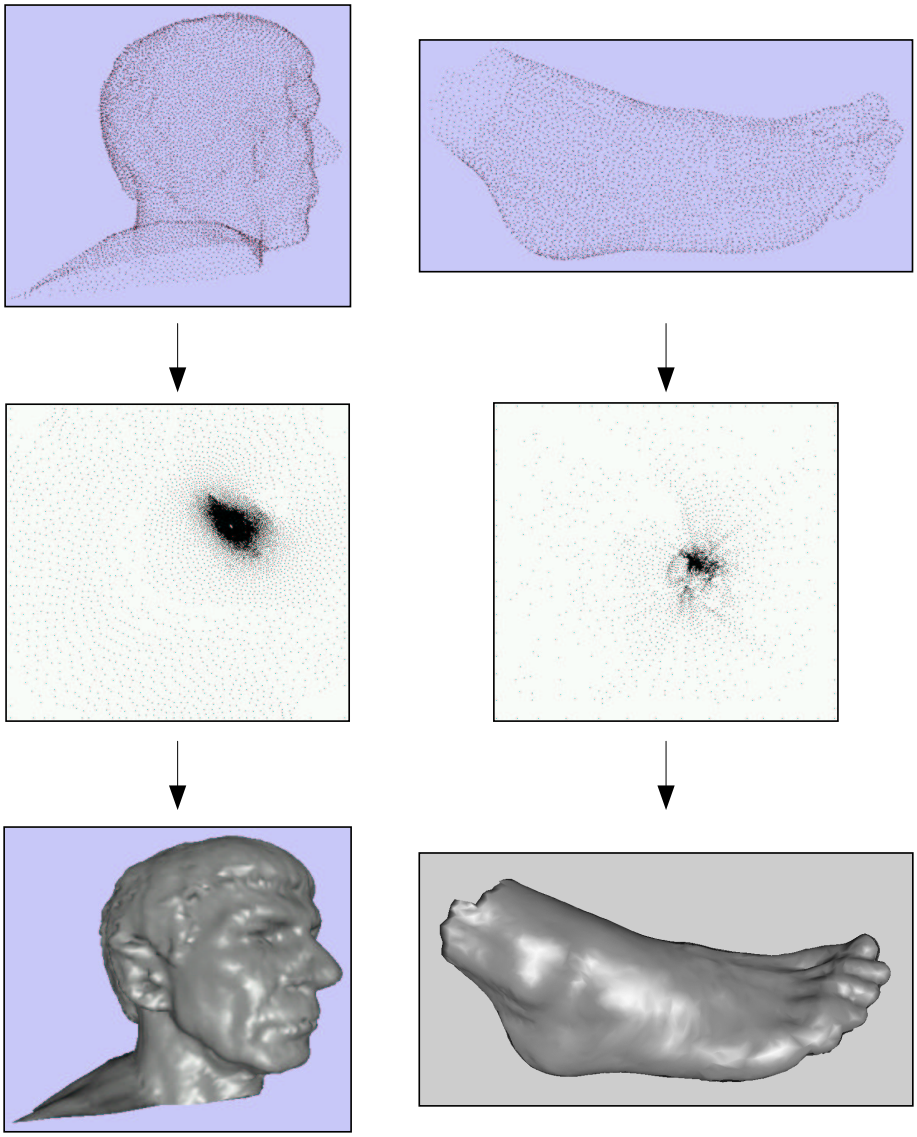

Figure 15: 3D measurement data from a head and a foot parametrised over rectangular domains. The adaptive Multilevel B-spline scheme was used to create the 3D models. The foot-model consists of 16384×16384 B-spline coefficients.