Smooth Approximation and Rendering of Large Scattered Data Sets

Jörg Haber* Frank Zeilfelder* Oleg Davydov† Hans-Peter Seidel*

* Max-Planck-Institut für Informatik, Saarbrücken, Germany, {haberj,zeilfeld,hpseidel}@mpi-sb.mpg.de
† Justus-Liebig-Universität Giessen, Germany, oleg.davydov@math.uni-giessen.de

Abstract

We present an efficient method to automatically compute a smooth approximation of large functional scattered data sets given over arbitrarily shaped planar domains. Our approach is based on the construction of a C^1-continuous bivariate cubic spline and our method offers optimal approximation order. Both local variation and non-uniform distribution of the data are taken into account by using local polynomial least squares approximations of varying degree. Since we only need to solve small linear systems and no triangulation of the scattered data points is required, the overall complexity of the algorithm is linear in the total number of points. Numerical examples dealing with several real world scattered data sets with up to millions of points demonstrate the efficiency of our method. The resulting spline surface is of high visual quality and can be efficiently evaluated for rendering and modeling. In our implementation we achieve real-time frame rates for typical fly-through sequences and interactive frame rates for recomputing and rendering a locally modified spline surface.

CR Categories: G.1.2 [Numerical Analysis]: Approximation—*approximation of surfaces, least squares approximation, spline and piecewise polynomial approximation*; I.3.3 [Computer Graphics]: Picture/Image Generation—*display algorithms, viewing algorithms*; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—*surface representation, splines*; E.4 [Coding and Information Theory]—*data compaction and compression.*

Keywords: scattered data approximation, least squares approximation, terrain visualization, data compression

1 Introduction

The problem of scattered data fitting is to efficiently compute a suitable surface model that approximates a given large set of arbitrarily distributed discrete data samples. This problem arises in many scientific areas and fields of application, for instance, in chemistry, engineering, geology, medical imaging, meteorology, physics, and terrain modeling. Moreover, scattered data methods play an important role in scientific visualization to get a better understanding of a given set of scattered data points and for subsequent treatment needed in many applications.

In this paper we concentrate on the problem of functional scattered data fitting, which can be described as follows: Given a finite set of points $(x_i, y_i) \in \Omega,\ i = 1, \ldots, N$, where Ω is a bounded domain in the plane, and corresponding values $z_i,\ i = 1, \ldots, N$, find a method to construct a surface $s : \Omega \mapsto \mathbb{R}$ that meets as many as possible of the following goals:

- **Approximation**: s should approximate the data, i.e. $s(x_i, y_i) \approx z_i\ (i = 1, \ldots, N)$, while offering optimal approximation order
- **Quality**: s should be of high visual quality (i.e., for "smooth data" s should be at least C^1-continuous) and have convenient properties for further processing.
- **Usability**: Large real world data sets, where N is typically at least of order 10^6, should be manageable.
- **Efficiency**: Both the computation and the evaluation of s should be fast and efficient.
- **Stability**: The computation of s should be numerically stable, i.e., the method should work for any distribution of scattered points.
- **Adaptiveness**: The local variation and distribution of the data should be taken into account.
- **Simplicity**: The method should be easy to implement.

Although many approaches have been developed in the last 30 years, the literature shows that it is a difficult task to meet all of the above goals by using one single method. In fact, the algorithms proposed in the literature typically have at least one of the following drawbacks: limitations in approximation quality, severe restrictions on the number of points, limited visual quality of the resulting surface, high computation times, and restrictions on the domain and distribution of the data.

In this paper, we develop a new approach to scattered data fitting which is based on differentiable bivariate cubic splines. We decided to construct a smooth surface since such surfaces look more pleasant and have nice properties for further processing and rendering. The method we propose belongs to the class of so-called two-stage methods [40]: In the first step of the algorithm, we compute discrete least squares polynomial pieces for local parts of the spline s by using only a small number of nearby points. Then, in the second step, the remaining polynomial pieces of s are obtained directly by using C^1 smoothness conditions. Our approach uniquely combines the following advantages: The data need not be triangulated, the domain Ω can be of arbitrary shape, no estimations of derivatives need to be computed, and we do not perform any global computations. As a result, we obtain a fast method that is applicable to large real world data, local data variations do not have an undesirable global effect, and the differentiable approximating spline is of high visual quality. Thus, we have a fully automatic method which is stable, easy to implement, and the local distribution and variation of the data are taken into account.

The spline representation of our surface model allows to employ Bernstein-Bézier techniques efficiently for evaluation and rendering of the spline. In contrast to previous methods for terrain visualization, we render smooth surfaces and thus do not need to decimate or (re-)triangulate the scattered data. Moreover, we have fast and robust algorithms for view frustum culling and computation of surface points, true surface normals, and texture coordinates.

2 Previous Work

In this section we give an overview on previous and related work in the fields of scattered data fitting and rendering of large terrain data sets. There are many different approaches to scattered data fitting, see for instance the surveys and overview in

[19, 26, 30, 35, 40]. A very active area of research are radial basis methods [2, 20, 35, 38]. However, these methods usually require solving large, ill-conditioned linear systems. Therefore, sophisticated iterative techniques are needed for the computation of the radial function interpolants [2]. An approach based on regularization, local approximation, and extrapolation has been proposed in [1].

Motivated by some classic results from approximation theory, various methods based on bivariate splines were proposed. There are several types of splines that can be used. The simplest approach is to consider tensor product splines [13, 18, 22, 23] and their generalizations to NURBS surfaces [37], which have important applications, e.g., in modeling and designing surfaces. These spaces are essentially restricted to rectangular domains. In general, tensor product methods are straightforward to apply only for data given on a grid. If the data points are irregularly distributed, there is no guarantee that the interpolation problem has a solution. Also, global least squares approximation and related methods have to deal with the problem of rank deficiency of the observation matrix. Alternatively, lower dimensional spaces and/or adaptive refinement combined with precomputation in those areas where the approximation error is too high can be employed [39, 28, 45]. In [39], parametric bicubic splines possessing G^1 geometric continuity are adaptively subdivided to approximate 3D points with a regular quadmesh structure. Multilevel B-splines are used in [28] to approximate functional scattered data.

Other spline methods are based on box splines [9, 24], simplex splines [46], or splines of finite-element type. The simplest example of finite-element splines are continuous piecewise linear functions with respect to a suitable triangulation of the planar domain. It is well-known that the approaches based on piecewise linear functions can not exceed approximation order 2. To achieve higher smoothness and approximation order, polynomial patches of greater degree have to be considered. In particular, there are scattered data methods based on classical smooth finite elements such as Bell quintic element, Frajies de Veubecke-Sander and Clough-Tocher cubic elements, and Powell-Sabin quadratic element, see the above-mentioned surveys and more recent papers [10, 14, 31]. In general, these methods are local, and it is obvious that these splines possess much more flexibility than tensor product splines. Many methods require that the vertices of the triangulation include all data points or a suitable subset of these points obtained by a thinning procedure. Such a triangulation is not very expensive to obtain (computational cost $\mathcal{O}(N \log N)$), but the arising spline spaces can become difficult to deal with if the triangulation is complicated and its triangles are not well-shaped. In addition, the above-mentioned methods based on finite elements require accurate estimates of derivatives at the data points, which is a nontrivial task by itself assuming the data points are irregularly distributed. To overcome these difficulties, global least squares approximation and other global methods were considered, e.g., in [14, 21, 34, 46].

The basic idea of our method is related to the interpolation scheme of [33, 32]. Essential differences are, however, that we neither triangulate (quadrangulate) the data points, nor make use of any interpolation scheme as for instance in [10, 31, 32]. In particular, we do not need any estimates of z-values at points different from the given data points. Instead, we compute local least squares approximations directly in the Bernstein-Bézier form, and then settle the remaining degrees of freedom by using the standard C^1 smoothness conditions [16], which results in very short computation times. Since our method does not even require a triangulation of the data points, it is very well suited for extremely large datasets. In addition, our method allows (local) reproduction of cubic polynomials and hence offers optimal approximation order. Theoretical aspects of the method are treated in [11].

A large number of techniques for efficient rendering of terrain data has been proposed in the literature. However, these techniques usually operate on piecewise linear surface representations only. Moreover, many of these methods are restricted to data that are regularly sampled on a rectangular grid. This kind of data is commonly referred to as a *digital elevation map* (DEM) or, in the case of additional color values associated to each data point, as a *digital terrain map* (DTM).

Among the early methods based on DEM / DTM data we mention techniques such as clipping surface cells against the view frustum for ray tracing [8], extracting feature points from the data set by curvature analysis and constructing a Delaunay triangulation [41], and using quadtree data structures to accelerate ray casting [6]. More recent approaches achieve interactive frame rates for rather low resolution DTM by determining visible and occluded terrain regions [7] or by exploiting vertical ray coherence for ray casting [27]. Several methods incorporate level-of-detail (LOD) techniques to speed up rendering. Such LOD techniques can be embedded in multiresolution BSP trees [43] or combined with hierarchical visibility for culling occluded regions [42]. However, changing the LOD during an animation might result in visual discontinuities. This problem has led to the development of continuous LOD techniques [17, 29, 15]. In [29], a continuous LOD rendering is obtained through on-line simplification of the original mesh data while maintaining user-specified screen-space error bounds. A related approach is proposed in [15], where additional optimizations such as flexible view-dependent error metrics, incremental triangle stripping, and predefined triangle counts are introduced.

The methods that do not require data to be sampled on a uniform grid typically construct a *triangulated irregular network* (TIN) from the data. A multiresolution representation of arbitrary terrain data is presented in [5], where every resolution level consists of a TIN that is obtained through incremental Delaunay triangulation. The approximation error can be chosen to be constant or variable over the data domain. The approach in [25] is similar to the one before, but here the triangulation is computed on-the-fly, avoiding the storage requirements of the hierarchy.

3 Construction of the Spline

3.1 Overview and Basic Idea

Many of the methods mentioned in the previous section are based on global least squares approximation and related procedures, thus facing the problem of rank deficiency. One possibility to overcome this is by applying well known numerical techniques for rank deficient matrices, such as singular value decomposition. However, this procedure is very expensive for large coefficient matrices arising from the global methods and destroys their sparse structure. In contrast, our method only relies on *local* least squares computations, which allows us to employ the singular value decomposition efficiently.

Given a finite set of points $(x_i, y_i),\ i = 1, \ldots, N$, in a bounded domain $\Omega \subset \mathbb{R}^2$ and corresponding values $z_i,\ i = 1, \ldots, N$, we first determine a suitable space $\mathcal{S}$ consisting of smooth bivariate splines of degree 3 such that the total number of degrees of freedom (i.e., the dimension of $\mathcal{S}$) is approximately N. We construct a quadrilateral mesh covering the domain Ω, see Figure 1, and define $\mathcal{S}$ to be the space of C^1-continuous piecewise cubics with respect to the uniform triangle mesh Δ obtained by adding both diagonals to every quadrilateral. We call the union of these quadrilaterals the *spline domain* $\mathcal{Q}$. The basic idea of the method is to choose a subset $\mathcal{T}$ of triangles in Δ with the following properties:

(i) the triangles of $\mathcal{T}$ are uniformly distributed in Δ;

(ii) the polynomial patches $s|_T\,(T \in \mathcal{T})$ can be chosen freely and independently from each other;

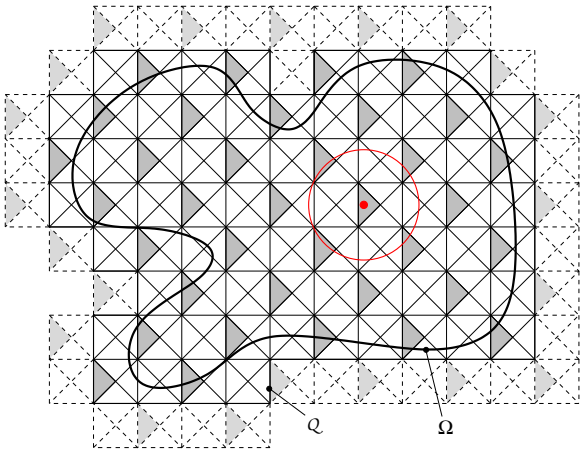


Figure 1: Layout of the Bézier triangle patches for an arbitrarily shaped domain Ω of scattered data points. In addition to the Bernstein-Bézier coefficients of the grey triangles inside the spline domain $\mathcal{Q}$, the coefficients of the light grey triangles in the dashed border cells are needed to determine all remaining triangle patches in $\mathcal{Q}$ by smoothness conditions. The red circle shows the position of one of the circles C_T.

(iii) if a spline $s \in \mathcal{S}$ is known on all triangles in $\mathcal{T}$, then s is also completely and uniquely determined on all other triangles that cover the domain;

(iv) each patch $s|_T$, where $T \in \Delta \setminus \mathcal{T}$ has a non-empty intersection with Ω, can be computed using only a small number of nearby patches corresponding to triangles in $\mathcal{T}$.

The approximating spline is constructed in two steps. First, we compute for every triangle $T \in \mathcal{T}$ a least squares polynomial p_T^q in its Bernstein-Bézier form by using singular value decomposition (SVD) and taking into account only points in T and several adjacent triangles. The degree q of p_T^q may vary from triangle to triangle, though not exceeding 3, and is chosen adaptively in accordance with the local density of the data points. We set $s = p_T^q$ on each $T \in \mathcal{T}$. Then, in the second step, the remaining polynomial pieces of s on the triangles $T \in \Delta \setminus \mathcal{T}$ are computed by using Bernstein-Bézier smoothness conditions. In order to guarantee property (iii), it is necessary to add some auxiliary *border cells* containing both diagonals to Δ as shown in Figure 1.

3.2 The Spline Space

Our method is implemented for arbitrary domains as shown in Figure 6. For a detailed description of the spline space and its Bernstein-Bézier representation it is sufficient to consider the square domain $\Omega = [0,1]^2$.

For given scattered points $(x_i, y_i) \in \Omega$, $i = 1, \ldots, N$, we set $n = \lfloor \sqrt{N/5} \rfloor$ and we cover the domain Ω with squares Q_{ij}, $i, j = 1, \ldots, n$ of edge length $h = 1/n$. This choice of n ensures that the dimension of the spline space approximately coincides with the number of scattered data points. In addition, a ring of square *border cells* surrounding the union $\mathcal{Q} = \bigcup Q_{ij}$ is needed to completely determine the approximating spline on $\mathcal{Q}$. A uniform triangle mesh Δ (a so-called Δ^2 partition) is obtained by adding both diagonals to every square Q_{ij} as shown in Figure 1.

We consider the space of bivariate splines $\mathcal{S}$ consisting of all cubic C^1 piecewise polynomials with respect to Δ. It is well-known that the dimension of the spline space $\mathcal{S}$ is $5n^2 + 8n + 3 \approx N$ [3]. Moreover, this space combines a number of attractive features. First, the splines of this kind are highly flexible in contrast to, e.g., tensor product splines [13]. Second, the comparatively low degree allows fast and efficient computation of the approximating splines. Third, the approximation order of the space $\mathcal{S}$ is optimal [3], i.e., any sufficiently smooth function can be approximated by a spline $s \in \mathcal{S}$ with the error $\mathcal{O}(h^4)$, which is the best possible approximation order for piecewise cubics.

For computing the cubic patches of the approximating spline, we use the well-known Bernstein-Bézier representation of a cubic polynomial p_T^3 defined on a triangle $T = [\mathbf{t}_0, \mathbf{t}_1, \mathbf{t}_2] \in \Delta$:

$$p_T^3(\mathbf{u}) \;=\; \sum_{|\alpha|=\alpha_0+\alpha_1+\alpha_2=3} B_T^{\alpha,3}(\mathbf{u})\, b_\alpha, \quad \mathbf{u} \in T. \qquad (1)$$

Here

$$B_T^{\alpha,3}(\mathbf{u}) \;:=\; \frac{3!}{\alpha_0!\alpha_1!\alpha_2!}\, \lambda_0^{\alpha_0}(\mathbf{u})\, \lambda_1^{\alpha_1}(\mathbf{u})\, \lambda_2^{\alpha_2}(\mathbf{u}),$$

are the ten Bernstein basis polynomials of degree 3, where $\lambda_\nu(\mathbf{u})$, $\nu = 0, 1, 2$, are the barycentric coordinates of $\mathbf{u}$ with respect to T. The $b_\alpha \in \mathbb{R}$ are called Bernstein-Bézier coefficients of p and represent the local degrees of freedom of the polynomial patch.

The advantages of the Bernstein-Bézier techniques include the stability of the Bernstein-Bézier basis, easily implementable smoothness conditions (see Section 3.4 below), and an efficient algorithm for the evaluation of the spline and its corresponding normal (de Casteljau algorithm).

3.3 Adaptive Least Squares Approximation

We start our algorithm by choosing a subset $\mathcal{T}$ of the triangles in Δ as in Figure 1 and initial circles C_T, $T \in \mathcal{T}$, with radius $\frac{5}{4}h$ and midpoint at the barycenter of T. This choice of the circles ensures that the domain Ω is completely covered by the union of the circles C_T, $T \in \mathcal{T}$. In the first step of our algorithm we determine the polynomial pieces of the approximating spline s on the triangles of $\mathcal{T}$. To this end we compute $n^2/2$ different local discrete least squares approximations for bivariate polynomials by using singular value decomposition.

Since we treat scattered data, there is in principle no restriction on the number m of points within a particular circle C_T. However, both too few and too many points are not desirable. Therefore, we use the initial circle only if m satisfies $M_{\min} \le m \le M_{\max}$ with $M_{\min}, M_{\max}$ chosen as described below. Thus, two different situations that require further processing can occur:

1. the number of data points within C_T is too large: $m > M_{\max}$;
2. the number of data points within C_T is too small: $m < M_{\min}$.

In the first case ($m > M_{\max}$), we thin the data inside of C_T down to at most $M_{\max}$ points. This is done by sorting all data points from within C_T into an auxiliary regular grid, which is constructed in such a way that at most $M_{\max}$ grid cells lie inside of C_T. Then we choose the most central data point from each non-empty grid cell to be taken into account for computing the local polynomial patch. Such thinning of the data is justified by the assumption that unreasonably large sets of local data points carry redundant information. By thinning the data points, we avoid expensive computation of SVD for large matrices.

In the second case ($m < M_{\min}$), we simply increase the radius of C_T until at least $M_{\min}$ scattered points are inside of C_T. The parameter $M_{\min}$ controls the minimal local approximation order of the spline surface, while $M_{\max}$ acts as a balancing parameter between detail reproduction and overall smoothness. In order to locally reproduce at least linear functions in the areas of very sparse data, we

choose $M_{\min} = 3$. Since we need at least 10 scattered data points to fit a cubic polynomial, we require $M_{\max} \geq 10$. In our tests, we found a good heuristic choice to be $20 \leq M_{\max} \leq 60$.

The process of finding the data points that lie inside of a given circle C_T is accelerated using an additional uniform grid data structure $\mathcal{G}$ constructed during the initial input of the data. This grid covers the domain Ω and its resolution is chosen such that an average number of K data points lie within each grid cell. By experiment we found that values of $10 \leq K \leq 20$ lead to a reasonable speed up. Every grid cell is assigned a list of the data points inside that cell. Thus, we can reduce the number of point-in-circle tests significantly by considering only those data points, which are associated to the cells of $\mathcal{G}$ partially or completely covered by C_T. These grid cells can be determined efficiently by using pre-computed two-dimensional bit masks that depend on the radius of C_T and the position of its midpoint relative to the grid cells of $\mathcal{G}$.

The above procedure determines for each $T \in \mathcal{T}$ a set of data points $(\tilde{x}_i, \tilde{y}_i)$, $i = 1, \ldots, m$, that is either the set of all scattered points lying in the circle C_T, or a subset of it obtained by thinning. Figure 4 shows some examples of such sets of data points. We now consider the system of linear equations

$$\sum_{|\alpha|=q} B_T^{\alpha,q}(\tilde{x}_i, \tilde{y}_i)\, b_\alpha \;=\; \tilde{z}_i, \quad i = 1, \ldots, m, \tag{2}$$

where the $\tilde{z}_i$ are the z-values at points $(\tilde{x}_i, \tilde{y}_i)$ and q is the local degree of p_T^q ($q \leq 3$). Denote by A_q the matrix of the system (2). The number M_q of unknown coefficients b_α depends on q and is equal to 10, 6, 3, or 1 if q is 3, 2, 1, or 0, respectively. Depending on m, we initially choose q so that $m \geq M_q$. If, for instance, $m \geq 10$, we choose $q = 3$.

To solve (2) in the least squares sense, we compute the singular value decomposition $A_q = UDV^\top$, where $U \in \mathbb{R}^{m,M_q}$ and $V \in \mathbb{R}^{M_q,M_q}$ are (column-) orthogonal matrices and $D = \mathrm{diag}\{\sigma_1, \ldots, \sigma_{M_q}\}$ is a diagonal matrix containing the singular values σ_j of A_q $(j = 1, \ldots, M_q)$. We use the algorithm given in [36] for the computation of the SVD. Since the dimension of A_q is at most $M_{\max} \times 10$, we compute the SVD for small matrices only, making this step fast and robust. The least squares solution of (2) can be efficiently computed as $b = VD^{-1}U^\top \tilde{z}$, since the inverse of D is again a diagonal matrix with reciprocal diagonal elements.

In general, the condition number of the system (2) is given by the ratio $\max_j\{\sigma_j\}/\min_j\{\sigma_j\}$. If at least one of the singular values σ_j is smaller than a prescribed bound $\varepsilon_{\text{cond}}$, i.e., the points $(\tilde{x}_i, \tilde{y}_i)$ lie close to an algebraic curve of degree q, we consider the system (2) to be ill-conditioned and drop the degree q of the least squares polynomial. If the initial degree was $q = 3$, this means that we consider the system (2) once again, but this time for quadratic polynomials, i.e., $q = 2$. If the system (2) for $q = 2$ is ill-conditioned again, we drop the degree further down to $q = 1$ or even $q = 0$. The bound $\varepsilon_{\text{cond}}$ is obtained from a user-specified condition number κ: $\varepsilon_{\text{cond}} := \max_j\{\sigma_j\}/\kappa$. Extensive numerical simulations show that the quality of the resulting spline surface is quite sensitive to the choice of κ, see also Figure 5. If κ is chosen too high, the spline patches constructed over the triangles in $\mathcal{T}$ tend to be of a higher degree. Although this behavior can reduce the average approximation error at the data points, our tests show that individual data points may exhibit a larger approximation error. On the other hand, if κ is chosen too low, more and more spline patches possess a lower degree, thereby decreasing the local approximation order of the spline. In our tests, we successfully use $\kappa \in [80, 200]$.

In this way, for every $T \in \mathcal{T}$ we determine a polynomial $p_T^q = s|_T$ on T. If the degree q of p_T^q is less than three, the cubic Bernstein-Bézier representation (1) of

$$p_T^q(\mathbf{u}) \;=\; \sum_{|\alpha|=q} B_T^{\alpha,q}(\mathbf{u})\, a_\alpha, \quad \mathbf{u} \in T, \tag{3}$$

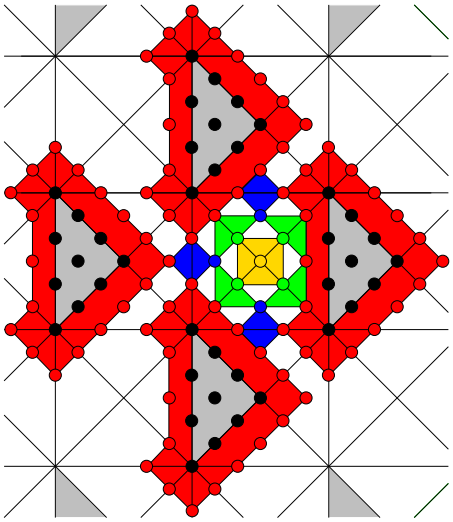


Figure 2: Remaining Bernstein-Bézier coefficients are determined due to C^1-conditions in the order: red, blue, green, and yellow.

is finally obtained by degree raising. The corresponding relations between the coefficients of p_T^q in its two representations (1) and (3) result from the following equation [16]:

$$\left(\sum_{|\alpha|=q} B_T^{\alpha,q}\, a_\alpha\right)(\lambda_0 + \lambda_1 + \lambda_2)^{3-q} \;=\; \sum_{|\alpha|=3} B_T^{\alpha,3}\, b_\alpha\,.$$

3.4 C^1-Conditions

In the second step of the algorithm, we determine the approximating spline s on the remaining triangles $T \in \Delta \setminus \mathcal{T}$ that have non-empty intersection with Ω.

For computing the remaining coefficients of s we use the well-known C^1 smoothness conditions for two adjacent triangular Bézier patches [16]. By using these simple formulas, we compute the Bernstein-Bézier coefficients of the polynomial pieces of the smooth spline s on the triangles $T \in \Delta \setminus \mathcal{T}$ that have a non-empty intersection with Ω. In particular, we do not need to perform this step for the remaining triangles in the border cells. This computation works step by step as illustrated in Figure 2. Here, so-called *domain points* in the part of the domain surrounded by four triangles in $\mathcal{T}$ are shown. Each domain point represents a Bernstein-Bézier coefficient of the spline. In particular, the points shown in black correspond to the Bernstein-Bézier coefficients on the triangles in $\mathcal{T}$. We first use these coefficients and the above C^1 smoothness conditions to compute the coefficients corresponding to the red points. Then we compute successively the coefficients shown in blue, green, and orange. Note that in the last step there are several possibilities to compute the coefficient shown in orange. We just choose one of them since the result is unique as follows by a standard argument.

4 Rendering

To efficiently render our spline surface, we adapted several techniques from [8, 17, 29, 44] to smooth Bézier spline surfaces. We start by overlaying the domain Ω with a uniform, axis-aligned *render grid*. The resolution of this grid is adjustable, we typically use about 30×30 grid cells. Each of the grid cells c_{ij} is assigned a minimum and a maximum z-value $z_{ij}^{\min}$ and $z_{ij}^{\max}$, which are taken

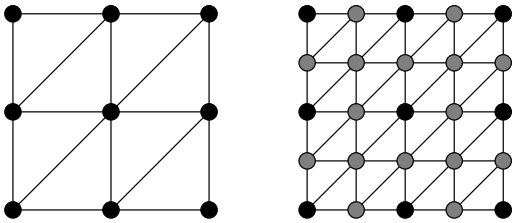


Figure 3: Sub-grids and resulting triangle meshes corresponding to LOD = 1 (left) and LOD = 2 (right). The spline surface is evaluated at the positions • for level one. For level two, the spline needs to be evaluated only at the new positions •.

from the minimum and the maximum Bernstein-Bézier coefficient of all the Bézier triangles covered by c_{ij}, respectively. Since the spline surface is known to lie completely inside the convex hull of its Bernstein-Bézier control points, we can use the boxes b_{ij} defined by the grid cells c_{ij} and their associated z-values $z_{ij}^{\min}$ and $z_{ij}^{\max}$ as bounding boxes for a cheap visibility test. Our rendering algorithm is organized as follows:

```
for each box b_ij
  cull b_ij against view frustum;
  if b_ij not completely culled
    mark b_ij as visible;
    compute bounding rectangle of projection
       of b_ij into screen space;
compute LOD from max bounding rect extent;
for each visible box b_ij
  if LOD ≠ b_ij.LOD
    evaluate spline on uniform sub-grid of
       level LOD over c_ij;
    draw triangle strips into b_ij.displayList;
    b_ij.LOD = LOD;
  execute b_ij.displayList;
```

The culling of the axis-aligned bounding boxes b_{ij} is performed in a conservative but cheap way: Each of the eight vertices of a box is projected into screen space. The box is rejected from visibility if and only if all of the projected vertices lie completely inside the same of one of the four half planes above, below, left, or right of the viewport. An early acceptance of the box is found, if any of the projected vertices lies inside the viewport. We found that this conservative test performs better than a more accurate test that includes 2D line clipping for each of the twelve edges of a box.

For each box b_{ij} that passes the culling test, we compute the bounding rectangle of the projected vertices. The maximum width / height of all these bounding rectangles is used to compute the global level-of-detail (LOD) for the current viewpoint. Obviously, it would be more efficient to have an individual LOD for each box: grid cells that are projected onto only a few pixels need not be subdivided as finely as grid cells that cover a large area of the viewport. However, such an adaptive LOD might result in visible gaps between adjacent grid cells, since the common boundary curve of the spline surface between two grid cells might be represented by a different number of linear segments for each cell. To overcome these problems, we are currently investigating the applicability of view-dependent sampling and triangle mesh generation as suggested for instance in [15].

After the global LOD has been computed, the spline surface is evaluated within every visible box b_{ij} on a uniform sub-grid over the domain of the grid cell c_{ij}. The resolution of the sub-grid is determined by the LOD. In our implementation we use a discrete LOD such that incrementing the LOD by one doubles the resolution of the sub-grid along both of its dimensions (see Figure 3). This approach has the advantage that all surface points and normals from level j can be re-used for level $j+1$. Thus the spline surface has to be evaluated only at the new positions for a higher level. In the case of going down from level $j+1$ to level j, we do not need to evaluate the spline at all – all the surface points and normals we need have already been computed. Since the number of triangles quadruples when going from level j to level $j+1$, it might seem reasonable to split every triangle into only two triangles, instead. In this case, however, it is not possible to generate optimal triangle strips for efficient rendering: Either the triangle strips are running diagonally, so that additional expensive texture binding calls are needed, or the triangle strips become much shorter than in our current approach.

To avoid possible visual discontinuities ("popping effects") when changing the LOD during an animation, we blend the surfaces S_{new} (corresponding to the new LOD) and S_{old} (corresponding to the previous LOD) over k successive frames using the alpha buffer and a blend function provided by our graphics hardware. Since we achieve real-time frame rates for rendering, we found that $k = 20$ yields visually smooth transitions from S_{old} to S_{new}.

The evaluation of the spline within every visible box b_{ij} is done using the de Casteljau algorithm [12]. In contrast to tensor product patches of the same degree, the de Casteljau algorithm for bivariate triangle patches takes about half as many multiplications and additions to compute a point on the spline surface. If, like in our case for proper shading, the surface normal has to be computed as well, the advantage of bivariate triangle patches becomes even more evident: In addition to the costs of evaluating a surface point, we need only three multiplications and additions each to compute the (unnormalized) exact normal vector in that point.

5 Results

In order to verify the working of our method, we performed numerical tests on several real world scattered data sets which vary widely in size and quality. All simulations were run on an `sgi` Octane with a 300 MHz R12k processor and 1 GB main memory. The approximation quality of a spline is measured using the one-sided Hausdorff distance $d_H : \mathbb{R}^3 \times \mathcal{S} \to \mathbb{R}$ between the scattered data points $(x_i, y_i, z_i) \in \mathbb{R}^3$ and the spline $s \in \mathcal{S}$. To facilitate comparison of different data sets, we divide the maximum approximation error by the length δ of the diagonal of the scattered data's bounding-box. The results of our simulations are summarized in Table 1.

The results in Table 1 clearly show that both the runtime and the memory usage of our algorithm is linear in the number N of scattered data points. The approximation quality of the spline depends on the quality of the input data: The more densely the scattered data points lie, the more least squares polynomial patches of higher order will be created in general, thereby increasing the local approximation order of the spline. Our simulations show that large real world scattered data can be approximated well within some seconds: The maximum approximation error for our largest data set (588 km^2 with 360 meters elevation range) is about 14 meters (relative error $5.2 \cdot 10^{-4}$). In particular, in the case of huge and dense data sets, the deviation of the approximating spline to the majority of data points is less than one meter.

Once a spline approximation to a scattered data set has been computed, this spline surface can be efficiently evaluated for rendering using the techniques described in Section 4. On our Octane with a Vpro/8 graphics board we achieve real-time frame rates of 30–60 fps in typical fly-through sequences with up to 100k Gouraud-shaded and textured triangles per frame. Moreover, it is possible to locally modify the data, e.g., for animation or modeling, and render the adapted spline surface at interactive frame rates. We have created an animation sequence of a surface that has been

N	avg. density (data / km^2)	memory usage	CPU	percentage of patches of degree				$\max \frac{\lvert d_H \rvert}{\delta}$	percentage of data with $\lvert d_H \rvert$		
				3	2	1	0		> 10 m	> 5 m	> 1 m
9,704	81	1.8 MB	0.3 s	61.9%	8.4%	13.7%	16.0%	$3.1 \cdot 10^{-3}$	3.3%	17.5%	95.3%
45,324	278	4.2 MB	1.4 s	95.6%	1.4%	2.8%	0.2%	$2.4 \cdot 10^{-3}$	9.0%	28.5%	84.7%
736,577	5,001	48 MB	19.9 s	77.6%	20.4%	1.4%	0.6%	$8.5 \cdot 10^{-4}$	0.002%	0.06%	8.4%
2,958,078	5,020	189 MB	92.9 s	78.1%	20.6%	0.8%	0.5%	$5.2 \cdot 10^{-4}$	0.002%	0.07%	8.5%

Table 1: Results of smooth approximation for different real world scattered data sets. Timings are given for the construction of the spline; see Section 5 for details on operating platform and Hausdorff distance d_H. The diameter of the scattered data's bounding-box is denoted by δ.

constructed from 736k scattered data points. In an area that covers about 0.4% of the surface, the scattered data points are moved from frame to frame. Due to the locality of the method, we are able to recompute the Bernstein-Bézier coefficients of the polynomial pieces for each frame and render the spline surface at about 10 fps.

An additional area of application for our method is (lossy) geometric compression (see Figure 6). Storing the $5n^2$ Bernstein-Bézier coefficients that have been computed as described in Section 3.3 allows to reconstruct the complete spline due to C^1-conditions. For $n = \lfloor \sqrt{N/5} \rfloor$, this requires storage of approximately N floating point numbers only, since the position of the Bernstein-Bézier coefficients is implicitly defined by our scheme[1]. Since the scattered data require storage of $3N$ floats, we obtain a compression ratio of about $3:1$. By decreasing the resolution n of the spline grid, we can control the compression ratio. Unlike many other compression techniques, our method becomes faster when using higher compression ratios. Compressing three million scattered data points from an area of 588 km^2 with an elevation range of 360 meters by a ratio of $30:1$ takes about 15 seconds and yields a maximum approximation error of 20 meters. Although our method will probably not achieve the same reconstruction quality as state-of-the-art geometric compression or mesh reduction techniques [4], it can compete very well with such techniques concerning run-time complexity. Most of these techniques operate on polygonal meshes. If, however, the input data are given as unorganized scattered data, it takes $\mathcal{O}(N \log N)$ time to compute a triangulation. Since our method computes the Bernstein-Bézier coefficients of a smooth approximation of the scattered data in $\mathcal{O}(N)$ time, it might be the method of choice for very large N.

6 Conclusion and Future Work

We have presented an efficient method to automatically compute smooth approximations of large sets of unorganized scattered data points. The method is based on the construction of a differentiable bivariate spline with respect to a uniform triangle mesh over an arbitrarily shaped planar domain. For a uniformly distributed subset of triangles we compute local polynomial least squares approximations by using singular value decomposition (SVD) of small matrices. The smooth approximating spline is constructed by gluing together these patches using Bernstein-Bézier smoothness conditions. We emphasize the following key features of our method:

- We develop a completely local approach, which means that we do not use any global optimization or other techniques involving computation with large portions of the data set.
- We employ the rank-revealing features of SVD to control the polynomial degree of the initial patches, which allows to take into account the local variation and distribution of the data points.
- The algorithm does not make use of any interpolation scheme. In particular, no estimation of derivatives is needed.
- Our method offers optimal approximation order and the constructed spline is by its nature C^1-continuous. In addition, the spline surface does not have artifacts like, for instance, peaks or flat spots close to the data points.
- The use of a uniform triangle mesh also contributes to great savings in computation time. As a result, the overall complexity of the algorithm is linear in the number of scattered data points.

The numerical examples with millions of scattered points of real world data sets show that the approximating spline can be computed very fast, the approximation error is very small, the resulting surfaces are of high visual quality and can be easily used for further processing in the context of rendering, modeling, and geometric compression.

In addition we note that the most expensive step of our algorithm is the computation of local discrete least squares approximations (more than 95 % of the overall computation time). Since these approximations are computed independently from each other, this step can be easily parallelized, thus leading to savings proportional to the number of processors on a multi-processor machine.

Considering these results, a variety of generalizations and applications of the new method can be thought of. In particular, the local adaptiveness of the method can be increased by designing multilevel spline spaces and adjusting the local dimension of the spline space adaptively to the local distribution of data points. However, because of the additional computational complexity, it is not clear if this approach will increase the overall efficiency of the algorithm. Other natural and important questions that include generalizations of the method to the reconstruction of surfaces of higher smoothness and more general topology or to the reconstruction of trivariate functions using, e.g., Bézier tetrahedra are currently under investigation.

[1] In addition, the number n and the min/max coordinates of the scattered data in the xy-plane have to be stored.

Acknowledgment

The data used for the simulations and images was taken from the digital terrain model, scale 1:5000 (DGM 5) by kind permission of "Landesamt f¨ur Kataster-, Vermessungs- und Kartenwesen des Saarlandes" under license numbers G-07/00 (9/26/00) and D 90/2000 (10/17/00).

References

[1] E. Arge, M. Dæhlen, and A. Tveito. Approximation of Scattered Data Using Smooth Grid Functions. *J. Computational and Applied Math.*, 59:191–205, 1995.

[2] M. D. Buhmann. Radial Basis Functions. *Acta Numerica*, pp. 1–38, 2000.

[3] C. K. Chui. *Multivariate Splines*. SIAM, 1988.

[4] P. Cignoni, C. Montani, and R. Scopigno. A Comparison of Mesh Simplification Algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[5] P. Cignoni, E. Puppo, and R. Scopigno. Representation and Visualization of Terrain Surfaces at Variable Resolution. *The Visual Computer*, 13(5):199–217, 1997.

[6] D. Cohen and A. Shaked. Photo-Realistic Imaging of Digital Terrains. In *Computer Graphics Forum (Proc. Eurographics '93)*, volume 12, pp. C363–C373, 1993.

[7] D. Cohen-Or and A. Shaked. Visibility and Dead-Zones in Digital Terrain Maps. In *Computer Graphics Forum (Proc. Eurographics '95)*, volume 14, pp. C171–C180, 1995.

[8] S. Coquillart and M. Gangnet. Shaded Display of Digital Maps. *IEEE Computer Graphics and Applications*, 4(7):35–42, July 1984.

[9] M. Dæhlen and V. Skyth. Modelling Non-rectangular Surfaces using Box-splines. In D. C. Handscomb, *Mathematics of Surfaces III*, pp. 287–300. 1989.

[10] W. A. Dahmen, R. H. J. Gmelig Meyling, and J. H. M. Ursem. Scattered Data Interpolation by Bivariate C^1-piecewise Quadratic Functions. *Approximation Theory and its Applications*, 6:6–29, 1990.

[11] O. Davydov and F. Zeilfelder. Scattered Data Fitting by Direct Extension of Local Polynomials with Bivariate Splines, 2001. in preparation.

[12] P. de Casteljau. Outillages M´ethodes Calcul. Technical report, Andre Citroen Automobiles, Paris, 1959.

[13] P. Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, 1993.

[14] P. Dierckx, S. Van Leemput, and T. Vermeire. Algorithms for Surface Fitting using Powell-Sabin Splines. *IMA Journal of Numerical Analysis*, 12(2):271–299, 1992.

[15] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. In *Proc. IEEE Visualization*, pp. 81–88, 1997.

[16] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 4. edition, 1993.

[17] R. L. Ferguson, R. Economy, W. A. Kelley, and P. P. Ramos. Continuous Terrain Level of Detail for Visual Simulation. In *Proc. Image V Conference 1990*, pp. 145–151, 1990.

[18] D. R. Forsey and R. H. Bartels. Surface Fitting with Hierarchical Splines. *ACM Transactions on Graphics*, 14(2):134–161, April 1995.

[19] R. Franke. Scattered Data Interpolation: Test of Some Methods. *Mathematics of Computation*, 38(157):181–200, January 1982.

[20] R. Franke and H. Hagen. Least Squares Surface Approximation using Multiquadrics and Parameter Domain Distortion. *Computer Aided Geometric Design*, 16(3):177–196, 1999.

[21] R. H. J. Gmelig Meyling and P. R. Pfluger. Smooth Interpolation to Scattered Data by Bivariate Piecewise Polynomials of Odd Degree. *Computer Aided Geometric Design*, 7(5):439–458, August 1990.

[22] B. F. Gregorski, B. Hamann, and K. I. Joy. Reconstruction of B-spline Surfaces from Scattered Data Points. In *Proc. Computer Graphics International 2000*, pp. 163–170, 2000.

[23] G. Greiner and K. Hormann. Interpolating and Approximating Scattered 3D Data with Hierarchical Tensor Product Splines. In A. Le M´ehaut´e, C. Rabut, and L. L. Schumaker, *Surface Fitting and Multiresolution Methods*, pp. 163–172. 1996.

[24] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise Smooth Surface Reconstruction. In *Computer Graphics (SIGGRAPH '94 Conf. Proc.)*, pp. 295–302, 1994.

[25] R. Klein, D. Cohen-Or, and T. H¨uttner. Incremental View-dependent Multiresolution Triangulation of Terrain. *The Journal of Visualization and Computer Animation*, 9(3):129–143, July–September 1998.

[26] P. Lancaster and K. Šalkauskas. *Curve and Surface Fitting*. Academic Press, 1986.

[27] C.-H. Lee and Y. G. Shin. A Terrain Rendering Method Using Vertical Ray Coherence. *Journal of Visualization and Computer Animation*, 8(2):97–114, 1997.

[28] S. Lee, G. Wolberg, and S. Y. Shin. Scattered Data Interpolation with Multilevel B-Splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, July 1997.

[29] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hughes, N. Faust, and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Computer Graphics (SIGGRAPH '96 Conf. Proc.)*, pp. 109–118, 1996.

[30] S. K. Lodha and R. Franke. Scattered Data Techniques for Surfaces. In H. Hagen, G. Nielson, and F. Post, *Proc. Dagstuhl Conf. Scientific Visualization*, pp. 182–222, 1999.

[31] M. Morandi Cecchi, S. De Marchi, and D. Fasoli. A Package for Representing C^1 Interpolating Surfaces: Application to the Lagoon of Venice's Bed. *Numerical Algorithms*, 20(2,3):197–215, 1999.

[32] G. N¨urnberger, L. L. Schumaker, and F. Zeilfelder. Local Lagrange Interpolation by Bivariate C^1 Cubic Splines. In *Proc. Conference on Curves and Surfaces*, 2001. in print.

[33] G. N¨urnberger and F. Zeilfelder. Local Lagrange Interpolation by Cubic Splines on a Class of Triangulations. In *Proc. Conf. Trends in Approximation Theory 2000*, pp. 341–350, 2001.

[34] R. Pfeifle and H.-P. Seidel. Fitting Triangular B-splines to Functional Scattered Data. In *Proc. Graphics Interface '95*, pp. 80–88, 1995.

[35] M. J. D. Powell. Radial Basis Functions for Multivariable Interpolation. In J. C. Mason and M. G. Cox, *Algorithms for Approximation of Functions and Data*, pp. 143–168. 1987.

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2. edition, 1992.

[37] H. Qin and D. Terzopoulos. D-NURBS: A Physics-Based Framework for Geometric Design. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, March 1996.

[38] R. Schaback. Improved Error Bounds for Scattered Data Interpolation by Radial Basis Functions. *Mathematics of Computation*, 68(225):201–216, January 1999.

[39] F. J. M. Schmitt, B. B. Barsky, and W. Du. An Adaptive Subdivision Method for Surface-Fitting from Sampled Data. In *Computer Graphics (SIGGRAPH '86 Conf. Proc.)*, pp. 179–188, 1986.

[40] L. L. Schumaker. Fitting Surfaces to Scattered Data. In G. G. Lorentz, C. K. Chui, and L. L. Schumaker, *Approximation Theory II*, pp. 203–268. 1976.

[41] D. A. Southard. Piecewise Planar Surface Models from Sampled Data. In *Proc. Computer Graphics International '91)*, pp. 667–680, 1991.

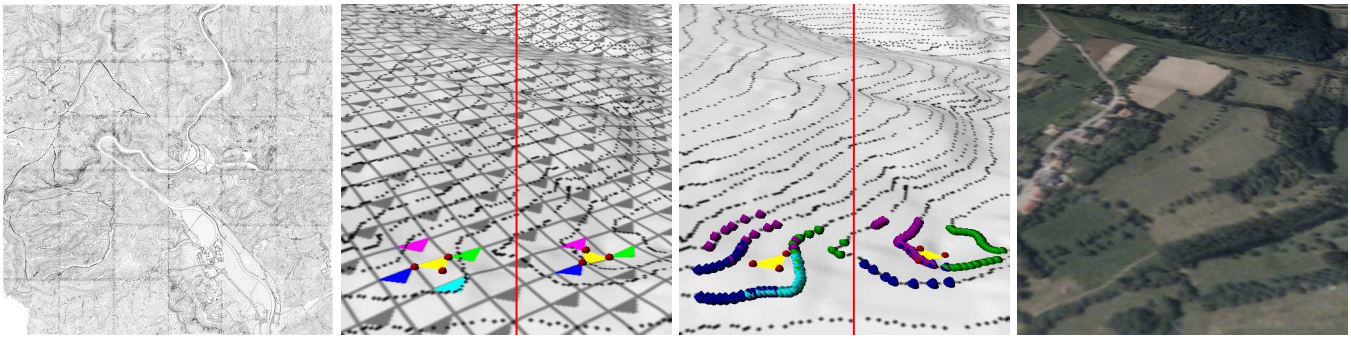
[42] A. J. Stewart. Hierarchical Visibility in Terrains. In *Rendering Techniques '97 (Proc. 8th EG Workshop on Rendering)*, pp. 217–228, 1997.

[43] C. Wiley, A. T. Campbell III, S. Szygenda, D. Fussell, and F. Hudson. Multiresolution BSP Trees Applied to Terrain, Transparency, and General Objects. In *Proc. Graphics Interface '97*, pp. 88–96, 1997.

[44] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL Programming Guide*. Addison–Wesley, Reading, MA, 1999.

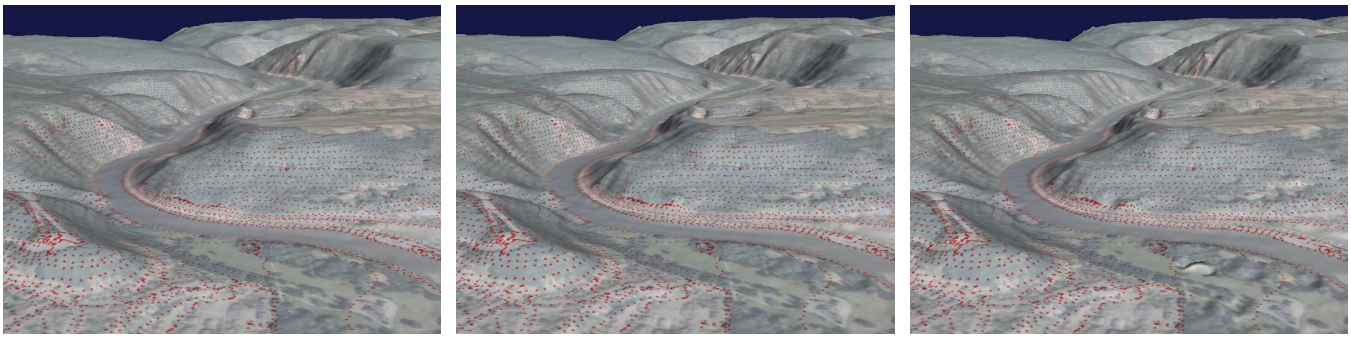
[45] W. Zhang, Z. Tang, and J. Li. Adaptive Hierarchical B-Spline Surface Approximation of Large-Scale Scattered Data. In *Proc. Pacific Graphics '98*, pp. 8–16, 1998.

[46] J. Zhou, N. M. Patrikalakis, S. T. Tuohy, and X. Ye. Scattered Data Fitting with Simplex Splines in Two and Three Dimensional Spaces. *The Visual Computer*, 13(7):295–315, 1997.



a) data distribution b) two configurations of triangles c) associated scattered data points d) surface with applied texture

Figure 4: Stages of the approximation process. a) Distribution of 736,577 scattered data points. b) Perspective view onto the surface with projected spline grid and data distribution. Two different sets of triangles $T \in \mathcal{T}$ (blue, magenta, green, cyan) are shown on the left and on the right. The Bézier points over these triangles are computed from the given scattered data points by local least squares approximation (see Section 3.3). Figure c) depicts the corresponding scattered data points (color coded) that have been used in this local least squares fitting for each triangle. Once the Bernstein-Bézier representation over the blue, magenta, green, and cyan triangles is available, the Bernstein-Bézier representation over the yellow triangles $T \in \Delta \setminus \mathcal{T}$ can be computed from the surrounding triangles using the C^1 smoothness conditions (see Section 3.4). d) Final result with applied texture.

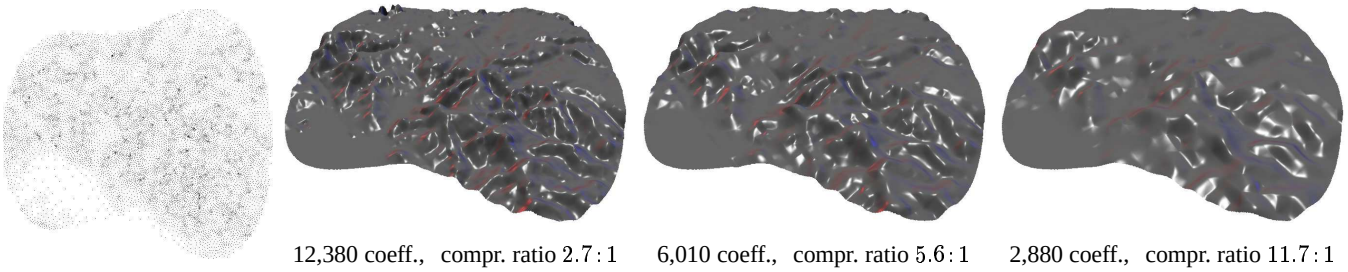


$\kappa = 20$
$\max |d_H| = 21.1\text{m}$, $\text{avg}\,|d_H| = 0.67\text{m}$

$\kappa = 150$
$\max |d_H| = 14.3\text{m}$, $\text{avg}\,|d_H| = 0.39\text{m}$

$\kappa = 1000$
$\max |d_H| = 67.2\text{m}$, $\text{avg}\,|d_H| = 0.34\text{m}$

Figure 5: Influence of the condition number κ on the approximation quality of the surface. Left: κ is too low, the average local approximation order of the spline decreases; Middle: κ is chosen appropriately; Right: κ is chosen far too high, thereby reducing the average approximation error but exhibiting high errors at individual data points in regions of extremely sparse data (see bottom right corner of image). See Sections 3.3 and 5 for details on κ and the Hausdorff distance d_H, respectively.



12,380 coeff., compr. ratio $2.7:1$ 6,010 coeff., compr. ratio $5.6:1$ 2,880 coeff., compr. ratio $11.7:1$

Figure 6: Scattered data compression for an arbitrarily shaped domain: Distribution of $N = 12,359$ scattered data points and approximating spline surfaces with a varying number of degrees of freedom (left to right). For a rectangular domain, the compression ratio is about three times higher than the ratio N : #coefficients, cf. Section 5. For an arbitrarily shaped domain, we need to store one additional integer number for every triangle $T \in \mathcal{T}$ (see Section 3) to uniquely determine the position of the Bernstein-Bézier coefficients. Therefore, the compression ratios given above are slightly lower than for the rectangular domain case.