EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

*Volume 21* (*2002*), *Number 3*

# Efficient Fitting and Rendering of Large Scattered Data Sets Using Subdivision Surfaces

Vincent Scheib*, Jörg Haber†, Ming C. Lin*, and Hans-Peter Seidel†

http://www.cs.unc.edu/gamma/sdsda
* Department of Computer Science, University of North Carolina at Chapel Hill, USA
† Max-Planck-Institut für Informatik, Saarbrücken, Germany

**Abstract**

*We present a method to efficiently construct and render a smooth surface for approximation of large functional scattered data. Using a subdivision surface framework and techniques from terrain rendering, the resulting surface can be explored from any viewpoint while maintaining high surface fairness and interactive frame rates. We show the approximation error to be sufficiently small for several large data sets. Our system allows for adaptive simplification and provides continuous levels of detail, taking into account the local variation and distribution of the data.*

Categories and Subject Descriptors (according to ACM CCS): G.1.2 [Approximation]: *Approximation of surfaces, Least squares approximation, Piecewise polynomial approximation*; I.3.3 [Picture/Image Generation]: *Display algorithms, Viewing algorithms*; I.3.5 [Computational Geometry and Object Modeling]: *Surface representation*.

## 1. Introduction

In recent years, technology for acquisition of three-dimensional data has reached a level of resolution that makes it hard to interactively visualize and explore the acquired data sets. For instance, aerial light detection and ranging (LIDAR) techniques are employed to analyze atmospheric composition and dynamics both on Earth and in outer space. Such techniques easily produce data sets with $10^8$ samples. Similar data sizes are obtained in other scientific applications where, for instance, seismic data, aero-magnetic data, terrain and other types of surface data are acquired. In particular, data from these applications is typically irregularly sampled. Fitting smooth surfaces to such huge scattered data sets allows us to visualize and navigate the data.

The problem of functional scattered data fitting can be defined as follows. Given a finite set of points $(x_i, y_i) \in \Omega$, $i = 1, \ldots, N$, where $\Omega \subset \mathbb{R}^2$ is a bounded domain in the plane, and corresponding values $z_i \in \mathbb{R}$, $i = 1, \ldots, N$, we want to construct a smooth surface that approximates the scattered data, taking into account local variation and distribution of the data. It should be computed and evaluated efficiently even for large scattered data sets ($N > 10^6$). Furthermore, the resulting surface should be rendered at interactive frame rates on current PC graphics hardware.

Generating very large terrain models is applicable for flight simulators, games, and other massive virtual environments. By adaptively fitting and subdividing terrain, detail is maintained where necessary and smaller meshes are stored on disk. Our method is suitable for large areas of smooth terrain (but not terrain with sharp discontinuities). Additionally, visualization applications often desire unlimited view point flexibility with high surface quality.

In this paper, we present a method to quickly construct and efficiently render surfaces for approximating large functional scattered data sets using subdivision surfaces. We combine several techniques from terrain rendering with the subdivision framework. This approach integrates the adaptive, continuous, level-of-detail offered by the subdivision framework and simplification capability of our hybrid terrain rendering method. It treats the processes of data approximation, surface triangulation, and interactive visualization in a unified framework. Our algorithm allows exploration of the resulting surfaces from any viewpoint at interactive rates. It has the following characteristics:

- It can smoothly approximate large sets of scattered data, as opposed to data on regular grids;
- It can provide adaptive, continuous level-of-detail for fast and high-quality rendering of complex data, by taking into account the data distribution and local variation;
- It can incorporate any error metric to automatically simplify the resulting meshes and seamlessly control continuous detail generation.

We demonstrate our algorithm on several data sets including scattered terrain elevation data, scientific functions, and large simulated terrain data. In the fly-through of these data sets, we observed noticeable performance improvement with higher visual quality especially around areas with highly varying detail. We also show that the resulting errors of the approximated surfaces to be sufficiently small on these data sets.

## 2. Previous Work

In this section, we survey related work in scattered data approximation and terrain rendering.

### 2.1. Scattered Data Approximation

Several scattered data fitting approaches have been proposed [15, 22, 27, 34]. Various methods are based on bivariate splines. Several types of splines can be used, for instance tensor product splines [10, 14, 18] and their generalizations to NURBS surfaces [31]. These spline spaces are basically restricted to rectangular domains. The tensor product methods are typically applied to data given on grids. In contrast, there is no guaranteed solution for interpolating data of irregular distribution. Global least squares approximation and similar methods have to deal with the problem of rank deficiency. On the other hand, lower dimensional spaces and/or adaptive refinement combined with precomputation in those areas where the approximation error is too high can be employed [24, 33, 37].

Other spline methods include those based on simplex splines [38], box splines [8, 20], or splines of finite-element type [9, 11, 28]. These local methods offer much more flexibility than tensor product splines in general. However, global least squares approximation and other global methods were often employed to alleviate the difficulting in estimating derivatives at irregularly distributed data points using FEM [11, 17, 29, 38].

A very active area of research are radial basis methods [2, 3, 16, 30]. Here, the interpolating surface $s$ is constructed as a linear combination of functions of the form $\Phi(\|\cdot-(x_i,y_i)\|)$, $i=1,\ldots,N$, where $\Phi$ is a suitable univariate function. If $\Phi$ satisfies certain requirements, the interpolation problem $s(x_i,y_i)=z_i$, $i=1,\ldots,N$, is solvable for any given data. To compute the surface, a linear system of size $N\times N$ has to be solved. Since this system can be ill-conditioned, an expensive preconditioning is required in many situations.

### 2.2. Rendering

Numerous methods for efficient rendering of terrain data have been proposed in the literature. However, these techniques usually operate on piecewise linear surface representations only. In fact, most of these approaches are restricted to data that are regularly sampled on a rectangular grid, commonly referred to as a *digital elevation map* (DEM) or *digital terrain map* (DTM) with colors associated.

To achieve interactive frame rates, techniques based on determining visible and occluded terrain regions [6, 7], exploiting smart data structures or vertical ray coherence for ray casting [5, 23], and incorporating level-of-detail (LOD) representations [35, 36] have been proposed to speed up rendering. To reduce visual discontinuities due to LOD switching, continuous LOD techniques have been developed by several researchers [12, 13, 25]. In [25], continuous LOD rendering is achieved with on-line simplification while maintaining user-specified screen-space error bounds [25] and improved in [12] with additional optimizations including incremental triangle stripping, predefined triangle counts, and flexible view-dependent error metrics. The computational cost of view-dependent error metrics is reduced by a method of nested spheres presented in [1] and generalized in [26].

The methods based on *triangulated irregular networks* (TINs) do not require data to be sampled on a uniform grid. Using incremental Delaunay triangulation, a multiresolution representation of arbitrary terrain data is obtained for every resolution level of TINs [4]. The approach in [21] further improves this method by computing triangulation on-the-fly to avoiding the storage requirements of the hierarchy.

Using subdivision surfaces for terrain rendering has been presented in [32]. An efficient implementation of the modified butterfly scheme is described using half-edge structures with mid-vertices. This method is demonstrated on an 8x8 terrain mesh and alone is not sufficient for large terrain visualization.

## 3. Overview

In this section we briefly describe various components of our algorithm and their interactions. Given a large number of functional scattered data points that lie on a smooth surface, our goal is to reconstruct high quality views of that surface from novel view points at interactive frame rates. By high quality, we mean smoothness that can be characterized by at least $C^1$-continuity. Modern graphics hardware acceleration offers display of millions of triangles per second, therefore a triangle mesh is a suitable output of the process.

We address this problem by taking into account the following:

- The approximated surface must fit to the scattered data points.
- The fitted surface must be smooth, at least $C^1$-continuous and preferably of higher degrees of continuity.

- Rendering of the resulting surface must not degrade its quality.
- The approximation algorithm must limit the complexity of fitted surfaces and the rendering pipeline should perform adaptive tessellation and simplification for interactive display.

The work of Haber *et al.* [19] achieves several of these goals, however the rendering is restrictively slow for views near the surface that display large areas of the surface. That work does not offer adaptive LOD, thus the entire surface is rendered at the detail required by the near view. When the view includes a large area, the detailed mesh becomes too large to efficiently display. Adaptive LOD techniques provide a method to refine the surface detail only where needed. Terrain rendering is an application where this is required – displaying high detail near the viewer while large areas approaching the horizon are simplified.

We choose a subdivision surface to fit the scattered data set and make use of terrain rendering techniques to maintain interactive frame rates for rendering large scattered data sets. The subdivision framework combined with terrain rendering techniques offers dynamic, continuous, and adaptive levels of detail with high quality rendering of smooth surfaces.

### 3.1. Adaptive Triangulation

As a preprocessing step, a planar triangulation $\mathcal{T}$ is fit to the scattered data in the domain $\Omega$. The vertices of $\mathcal{T}$ are denoted by $V_i$, $i = 1,\ldots,n$. The triangulation $\mathcal{T}$ is represented as a binary triangle tree (BTT), which consists of isosceles right triangles. The children of a triangle are created by an edge from the triangle's apex to the midpoint of the hypotenuse.

Figure 1 illustrates the construction of a BTT mesh. Triangles are divided until $N_\triangle$ or fewer points remain in any triangle. The BTT is therefore adapted to the local variation and distribution of the scattered data. We typically use $N_\triangle = 5,\ldots,15$ in our computations. Splitting a triangle may require neighboring triangles to be split to avoid T-junctions. This forced splitting continues recursively. An example is shown in bold in Fig. 1: the thickest line is split first and causes three additional splits.

### 3.2. Scattered Data Approximation

For the scattered data approximation, we use a similar approach to the one presented in [19]. However, instead of fitting the control points of uniformly distributed cubic Bernstein-Bézier triangles to compute a bivariate cubic spline surface, we now fit the control points of a subdivision surface control mesh, which is adapted to the local variation and distribution of the scattered data.

We employ local least squares computations using singular value decompositions of small matrices. Controlling the local degree of least squares polynomials according to the local density of the scattered data points and the condition number of the least squares problem ensures a numerically stable computation of the surface fitting.

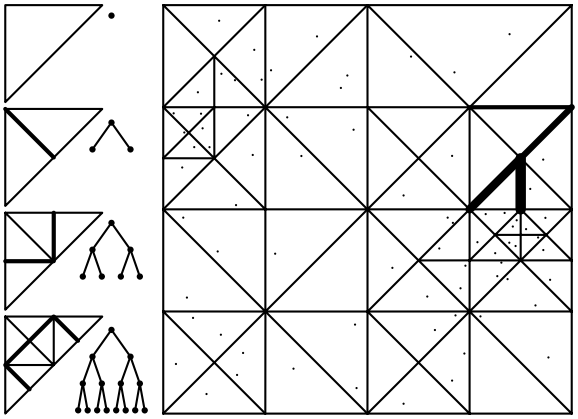

**Figure 1:** *A binary triangle tree (BTT) hierarchy is illustrated (left). A BTT is fit to scattered data with a maximum $N_\triangle$ of two points per triangle. A recursive forced split is shown in bold.*

### 3.3. Surface Rendering

The subdivision surface control mesh for large data sets is prohibitively large to be rendered directly. Therefore decimation of the mesh is required. Our mesh varies in complexity due to variable data density. We develop a hybrid rendering algorithm based on the approaches in [1, 12, 26, 39].

After the fitting has been completed, object space error metrics are precomputed and stored in the BTT. These metrics include the error to the subdivision limit surface at the leaves, nested variance between triangles and their simplified form, and nested bounding spheres. This hierarchy can be used for view frustum culling and adaptive runtime tessellation. These object space error metrics are projected to screen space efficiently during display. Each frame involves a depth first traversal of the BTT mesh, while maintaining full connectivity information. This is required for tessellation of the subdivision surface. The BTT and subdivision surface are tessellated finely enough to satisfy a specified pixel error.

## 4. Scattered Data Fitting

Our approach to scattered data fitting borrows some ideas from the method presented in [19]. In this section, we describe the key idea of our new approach as well as the main differences to the previous method. Details of our implementation that are not significantly different from the previous method are omitted in this section for the sake of conciseness.

We start from the vertices $V_i \in \mathbb{R}^2$, $i = 1,\ldots,n$, of the triangulation $\mathcal{T}$ of $\Omega$. For each vertex $V_i$, we collect the scattered data points $(\tilde{x}_j,\tilde{y}_j)$, $j = 1,\ldots,m$, from within a certain *local area* around $V_i$. The size of such a local area is determined by the local density of the scattered data points. Each

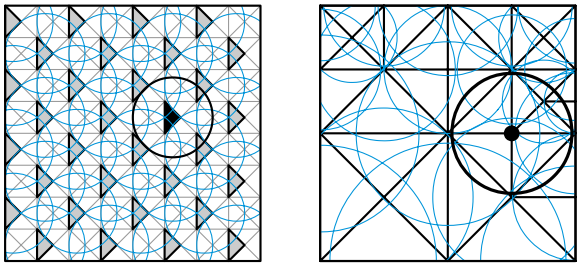

**Figure 2:** *Local fitting of Bézier triangle patches from the method presented in [19] (left) compared to our BTT vertex fitting (right). The highlighted triangle and vertex are shown with their respective initialized local area circles. Meshs are shown at densities for sampling 320 data points with* $N_\triangle = 10$.

triangle of $\mathcal{T}$ contains, by construction, a maximum of $N_\triangle$ data points. Depending on the local topology of $\mathcal{T}$, a vertex $V_i$ may have four to eight incident triangles and edges. By using the average length of the incident edges as the initial radius for a circular local area, we cover about 12 triangles on average in the local neighborhood of $V_i$ with that circle. If the number of data points inside the circle is too low (e.g. less than three, which is the minimum number to compute a linear approximation, see below), the radius of the circle is increased. Conversely, if there are too many data points, we apply thinning. This is done by binning data points into a regular grid and retaining only the most central data point from each cell. Due to the adaptive construction of the BTT, however, these situations rarely occur. The process of finding the data points that lie inside the local area around a vertex can be efficiently carried out using the same approach and data structures as described in [19].

Our method of fitting local data reflects variations in data density very well. Figure 2 compares our method with that in [19]. That work fit Bernstein-Bézier triangle patches in a regular grid, shown on the left. The 32 shaded triangles represent the patches fit directly from the data. Remaining patches are determined by smoothness constraints. The local areas used have a fixed minimum size, which guarantees coverage of all data points. Our vertex local areas, based on average edge length, are shown on the right. These meshes illustrate a situation of 320 data points. The BTT subdivides into at least 32 triangles for $N_\triangle = 10$. The number of fitting operations is variable depending on the triangulation.

We now fit a bivariate least squares polynomial $p_i^q$ of degree $q \leq 3$ to the $m$ scattered data points $(\tilde{x}_j, \tilde{y}_j)$ from the local area around vertex $V_i$ using singular value decomposition (SVD). The degree $q$ of $p_i^q$ may vary from vertex to vertex and is chosen adaptively w.r.t. the local density of the scattered data points. The number $c_q$ of unknown coefficients of $p_i^q$ 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 $m \geq c_q$.

Due to the SVD, we obtain the condition number of each local least squares problem by the ratio of the largest singular value to the smallest singular value. If this condition number exceeds a prescribed bound, we consider the problem to be ill-conditioned. This situation might appear, if the scattered data points from a local area lie close to an algebraic curve of degree $q$. In this case, we drop the degree $q$ of the least squares polynomial and re-compute the SVD again for the new degree $q' = q-1$. If the new least squares problem is ill-conditioned again and $q' > 0$, we drop the degree further down until $q' = 0$. The quality of the resulting subdivision surface is quite sensitive to the choice of the bound of the condition number. Although a high average degree of the least squares polynomials resulting from a high bound can reduce the average approximation error at the data points, our numerical simulations show that individual data points may exhibit a larger approximation error. If, however, the bound is chosen too low, more and more polynomials possess a lower degree, resulting in a lower (local) approximation order of the polynomials. We typically use values from $100, \ldots, 200$ for the bound of the condition number.

Finally, we evaluate the least squares polynomial $p_i^q$ at $V_i$ and assign the computed value as a height value to $V_i$, $i = 1, \ldots, n$. The resulting three-dimensional points in $\Omega \times \mathbb{R}$ are now interpreted as the control points of a subdivision surface control mesh $\mathcal{M}$, which can be rendered efficiently using the the technique presented in the following section.

## 5. Rendering the Subdivision Surface

To display the surface at interactive rates, the rendering algorithm places several constraints on the methods used. We have combined the subdivision framework with terrain rendering techniques. The subdivision framework provides a smooth limit surface which approximates the input data. Storing a low resolution control mesh, the limit surface is approached by recursively tessellating the mesh. The positions of vertices created by tessellation are computed using a weighted stencil of local vertices. The complexity of the polygonalized rendering of the subdivision surface can be increased until its screen projection has a sufficiently small error. However, rendering the control mesh of the subdivision surface alone proves too expensive for large data sets. Thus, an adaptive simplification technique as part of our rendering algorithm is used to simplify the control mesh where detail is not needed. An overview of this interpolation and decimation method is illustrated in Figure 3. Observe the curved surface approximating the input data on the left. The control mesh is shown in in thicker lines. On the right the control mesh is automatically simplified for fast rendering.

The majority of the surface triangles rendered will be generated by our hybrid terrain rendering algorithm, as simplified control mesh triangles. When the viewpoint is near the surface the full detail of the control mesh will be exposed. These triangles will be tessellated by the subdivision surface

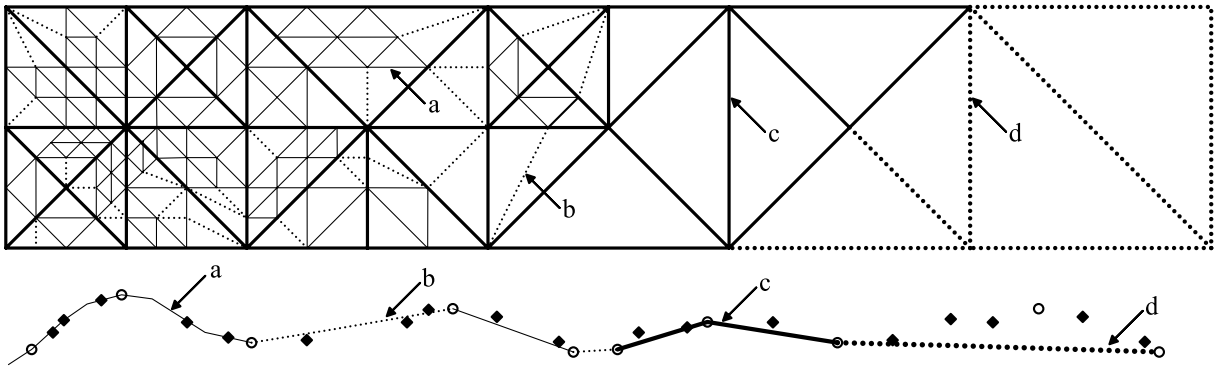


**Figure 3:** *This figure illustrates the different levels of detail in the finally rendered mesh. The top shows example mesh topology. The bottom shows a 1D example including initial data points (diamonds) and subdivision surface control points (circles). There are four types of edges distinguished by line type and labeled: a) adaptively tessellated butterfly subdivision surface – fit to the input data, b) stitching edges connecting higher subdivision levels to lower, c) subdivision surface control mesh fully exposed but satisfying screen space error metrics and not subdivided, d) simplified control mesh – distant from the eye point, and accounting for the majority of the surface area.*

until sufficient detail is achieved for screen display. These areas will be the ones the viewpoint is very near to, or areas of the surface for which the density of the scattered data points is low. In both cases, the higher order interpolation provides a higher quality surface compared to the linear interpolation of the terrain rendering algorithm alone. Any triangle that has only one or two edges tessellated will require stitching edges to avoid T-junctions and cracks.

Beyond being a suitable representation for our approximating surface, the subdivision surface must work in concert with the terrain rendering algorithm. This has led us to the selection of the modified butterfly subdivision scheme coupled with a longest edge bisection of an adaptive binary triangle tree.

### 5.1. Butterfly Subdivision Scheme

The modified butterfly subdivision surface [39] is an interpolating triangular mesh scheme. It provides $C^1$-continuity from any triangular control mesh, allowing for vertices of any valence. This is required for compatibility with the BTT method described in 5.2. For the ten point case the limit surface reproduces polynomials of degree 3. The interpolating property of this scheme ensures that all vertices are always on the limit surface. Thus, even when simplified, the control mesh will be on the curved surface we have fit to the scattered data. Figure 4 shows the stencils used to compute tessellations.

The butterfly tessellation only occurs on portions of the control mesh fully exposed by the terrain rendering algorithm. Adjacent triangles are maintained at most only one subdivision level apart. If some edges of a triangle are subdivided, while others are not, stitching edges are inserted to avoid T-junctions and cracks. These edges are not subdivided further.

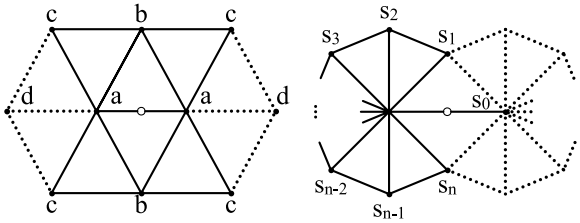


**Figure 4:** *The modified butterfly scheme ten point stencil (left), and extraordinary vertex stencil (right). A new vertex is created at the position of the circle by the weighted sum of the stencil vertices. Weights $a,\ldots,d$ and $s_0,\ldots,s_n$ are specified in* [39].

Tessellation is halted by one condition. When an edge is tessellated, the distance, on the plane perpendicular to the edge, between the new vertex and the edge is considered its error. This is a close approximation to the Hausdorff distance between the current edge and its limit surface. Projecting the error vector to the screen provides a metric in terms of pixels.

### 5.2. Binary Triangle Tree

The binary triangle tree (BTT) is well suited to terrain-like surface rendering. Notably, it offers adaptive tessellation in a manner which does not produce T-junctions or cracks. The size of the triangles can vary rapidly between areas of the mesh. Additionally, this type of subdivision is easy to implement and efficient. Unlike many terrain rendering applications using this type of mesh, we allow the highest resolution mesh to be of varying density, i.e. the binary tree is not complete. This provides for efficient storage of our maximum mesh based on the local density of input data points.

Each triangle $T_i$ in the tree has an object space error metric $\sigma_i$ and bounding radius $r_i$ associated with it. These nested

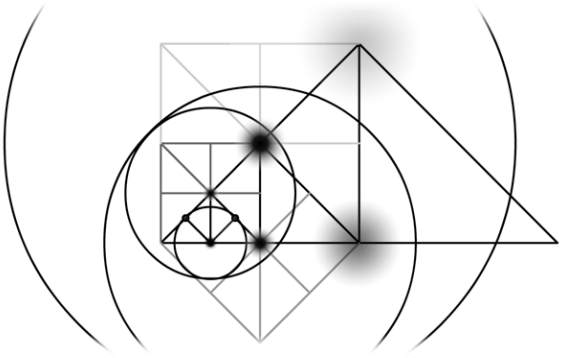

**Figure 5:** *An illustration of nested circles in 2D on a BTT. Centers are marked by soft dots of increasing size higher in the tree. Not all BTT edges are shown, only edges connected to centers.*

values are computed in a bottom up order, each node represents the maximum value of its self and its children. For each leaf triangle, $\sigma_i$ is initialized with an approximate error to the subdivision limit surface. The three edges of $T_i$ are subdivided and the maximum distance from the new vertices to $T_i$ is used. The bounding sphere of each triangle $T_i$ is centered at its apex $A_i$ and encloses the bounding spheres of its children: $r_i = \max_j(||A_i - A_j|| + r_j)$ for children apex vertices $A_j$ and radii $r_j$. Leaf triangles are initialized with $r_i = 0$. This is illustrated in figure 5.

Traversal of the BTT is performed as a depth first search, stopping if a triangle's bounding radius falls completely outside of the view frustum or the object space error is negligible when projected to the screen:

$$(\xi\sigma_i + r_i)^2 \; < \; ||A_i - e||^2$$

where the pixel error threshold $\xi$ is specified as the minimum size of a pixel in world space and $e$ is the position of the eye point. More detail can be found in [26].

Triangles are always enabled as pairs of both left and right children – splitting one right triangle and replacing it with two smaller ones. Additionally, the base of a triangle which is being split must be shared with a triangle of equal depth. If it is not, the opposite triangle is first split. This may result in a recursive split across several triangles. At each step in the BTT traversal, each triangle will have neighbors which vary in depth by no more than one level. Explicit connectivity information for face-face, face-vertex, and vertex-edge relationships is maintained during this traversal. The result is the portion of the BTT to be rendered, which is also a BTT. This is transferred to the subdivision subsystem. Any exposed leaf triangles will be subdivided as necessary.

## 6. Comparison and Results

In this section we discuss the performance of our prototype implementation. We analyze the surface approximation error as well as rendering speed and visual quality. We also compare the performance of our method with that of the Bézier patch implementation [19] on several of our data sets.

### 6.1. Data Sets

We used several data sets to test and profile our implementation. Each data set has an associated fly-through animation which is played back from a script file. Each data set has a short name used to refer to it:

- SciVis: $10,000$ data points from simulation of a magnetic drive unit with four poles and known holes in the non-square domain;
- Sparse: $45,000$ sparse data points taken from $12 \times 12$ km$^2$ hillside country;
- Dense: $736,000$ dense contour points from $12 \times 12$ km$^2$ terrain with a large river;
- Fractal: $1,000,000$ evenly distributed random samples over a fractal terrain;
- Fractal4: $4,000,000$ data points obtained by tiling Fractal terrain data set four times.

### 6.2. Approximation Error

Some error between the rendered approximation surface and the input data points is expected. We present the results based on several error metrics.

A Hausdorff distance is used to compute the maximum error. This is divided by the diagonal of the scattered data's bounding box, providing a normalized metric between data sets. Similarly, a root mean square (RMS) value of the average error is computed and normalized for each set. To understand the distribution of the errors, the percentage of data points above certain errors are recorded. These error values are specified as percentages of the scattered data's span in height. Table 1 lists error values for our data sets.

The internal data points are compared to a fully subdivided control mesh. We exclude data points near the border of the domain from the error metrics. These regions are unable to be properly subdivided by the butterfly scheme. Each iteration of the butterfly subdivision brings the mesh nearer to the limit surface. However, subsequent iterations contribute less than early iterations. Our implementation subdivides until system memory is exhausted. The larger data sets, Dense and Fractal, are subdivided only twice while only the control mesh is used for Fractal4. The error values reported are conservative for these data sets.

Terrain data sets, covering large areas of predominantly rolling hills, are handled well by our system. Extremely high resolution data, including hundreds of points across a tree top, requires filtering to meet our smooth surface requirement. Mathematical functions are generally fit well, unless they contain discontinuities.

### 6.3. Rendering Performance

We have tested our implementation on several PC workstations. Figure 6 shows the average frames per second for the

| N | Precomp (sec) | | Memory Usage (MB) | | | Max Abs Error | | RMS Error | | Data w/ Err > 5% | | Data w/ Err > 1% | | Data w/ Err > 0.1% | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Our | Prev | Our | Pre/F | Pre/R | Our | Prev | Our | Prev | Our | Prev | Our | Prev | Our | Prev |
| 10K | 0.187 | 0.234 | 13 | 2 | 344 | 0.010279 | 0.01791 | 0.000887 | 0.001738 | 0.00% | 10.1% | 48.75% | 43.19% | 85.40% | 100.0% |
| 45K | 0.562 | 0.765 | 16 | 6 | 348 | 0.009883 | 0.00240 | 0.001546 | 0.000227 | 0.00% | 0.00% | 49.02% | 5.602% | 85.94% | 79.39% |
| 736K | 12.98 | 12.17 | 83 | 80 | 422 | 0.001144 | 0.00085 | 0.000069 | 0.000031 | 0.00% | 0.00% | 1.909% | 0.390% | 54.83% | 47.23% |
| 1M | 19.11 | 19.99 | 116 | 110 | 452 | 0.009841 | 0.00141 | 0.000100 | 0.000027 | 0.00% | 0.00% | 0.526% | 0.026% | 66.15% | 36.47% |
| 4M | 81.48 | 45.00 | 530 | 435 | $> 2G$ | 0.000801 | 0.00093 | 0.000045 | 0.000011 | 0.00% | 0.00% | 0.011% | 0.007% | 32.00% | 27.08% |

**Table 1:** *Performance analysis and comparison to a previous Bézier patch implementation. Precomp lists the time to sort and fit scattered data points. The previous work uses substantially more memory durring rendering (Pre/R) than for fitting (Pre/F), while our algorithm uses the same memory footprint for both fitting and rendering. Max absolute and RMS error values are normalized by the data's bounding-box diagonal.*

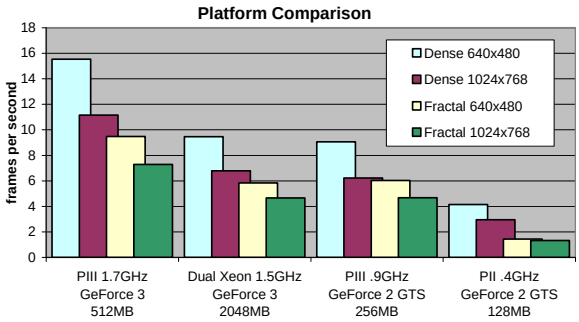


**Figure 6:** *Average frames per second for two data sets and resolutions for scripted fly-through animations.*

fly-through animations. Two large data sets and two resolutions are compared. Our application's bottleneck is the CPU, as seen when comparing platforms with equivalent graphics cards.

Data gathered for performance analysis, and the accompanying videos, is generated from scripted animation paths. A wide variety of viewpoints are explored, both near and far from the surface. Additionally, slow and fast viewpoint movements and rotation are included.

The graphs in Figure 7 show timings taken from two data sets. They represent frame time, view angle, and triangles displayed during the playback of a scripted animation path. The most influential factor on performance of both implementations is the vertical viewing angle. This is shown in the center graph as degrees deviation from the horizontal plane. The performance of the previous implementation is severely affected by views which cover large areas of the surface. These views correspond with near-horizontal views. The top graph displays the computation time in seconds for each frame of the animation. Note the large variation in frame time for the previous implementation. The bottom graph shows, on a log scale, the triangles rasterized for each frame. The old implementation frequently renders two orders of magnitude more triangles than the new implementation.

The strong computational expense of the previous implementation is due to its lack of adaptive tessellation. That implementation renders Bézier patches at discrete levels of detail. When high detail is required near the viewpoint, all patches are forced to that LOD. If many of these are visible, the triangle count becomes unmanageable. Our adaptive implementation is not as affected by changes in viewpoint or orientation. Additionally, because the new method generates the mesh every frame, there is no reliance on temporal coherence. The previous implementation drops in frame rate noticeably when quickly switching between LODs.

We note that although our method is more conservative with triangles, newer hardware acceleration products lessen this bottleneck. We were unable, however, to compare our implementation with the previous implementation on various platforms. The display time memory requirements of the previous method were too large for our workstations on our large data sets.

The butterfly tessellation represents only a small fraction (2%) of the computational cost of our system. A large portion of time is spent building and resetting the rendered portion of the BTT, and preparing data structures to be used by the subdivision surface subsystem. Methods exist that address these issues, which we did not pursue.

### 6.4. Visual Comparison

Figures 8 and 9 compare still images of the two implementations. Their textured surfaces are very similar (but not identical). However, viewing the wireframe of the meshes indicates significant differences. Figure 9 clearly shows the adaptive tessellation, adding detail to near geometry and silhouettes. The combination of subdivision methods can be seen in Figure 10. The BTT mesh is refined by the butterfly subdivision for surface points near the camera.

During animations some visual popping is apparent even with a one-pixel error tolerance. This is due to insertion of new vertices with significantly deviant normals. An error metric sensitive to normals can be used, however we found this too slow when viewing very large data sets.

## 7. Conclusion and Future Work

We have described a method for computing a smooth surface for approximation of large scattered data sets. The resulting subdivision surface can be rendered efficiently from

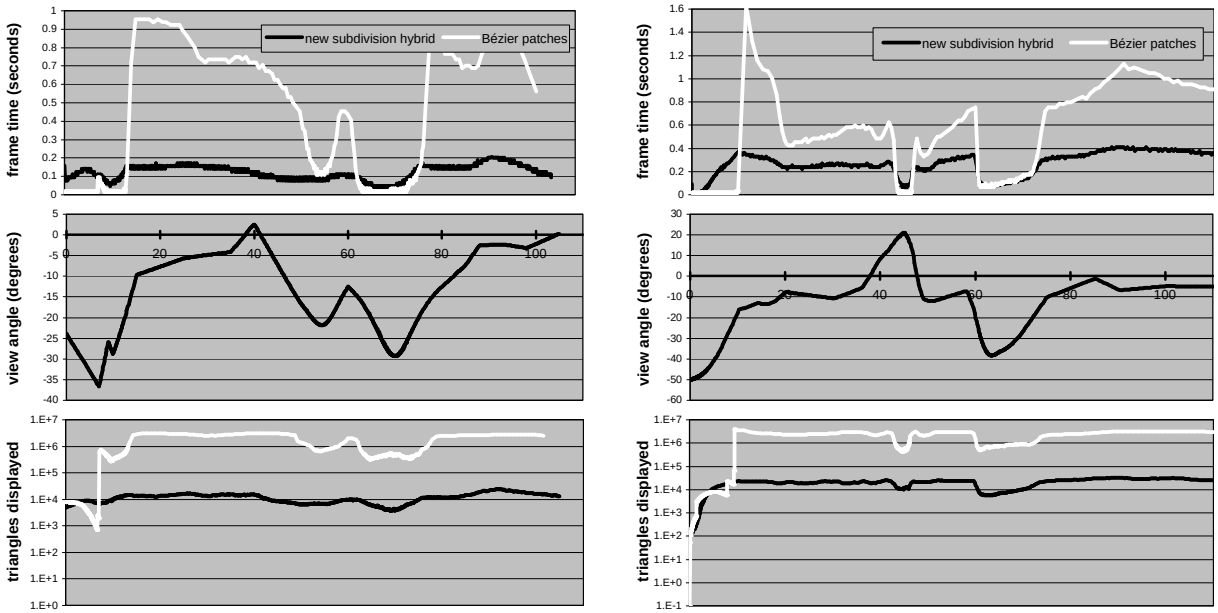


**Figure 7:** *Frame computation times and triangle counts for two fly-through animations of the Dense and Fractal data sets (left and right, respectively). Our new implementation (black) is compared with the Bézier patch method* [19] *(white). Values are plotted over time in seconds of the animations. These are the same as seen in the accompanying videos. Note the strong impact viewing angle has upon the Bézier patch method.*

arbitrary viewpoints using our hybrid rendering algorithm. An adaptive, continuous, level-of-detail is provided by taking into account the distribution and local variation of the scattered data. We have shown our method to result in small approximation errors and interactive rendering frame rates for several large scattered data sets.

Further performance gain for rendering is expected using just released PC graphics boards that support display of subdivision surfaces. Our CPU bound implementation will benifit from this feature. Additionally, hardware subdivision reduces the overall bandwidth used on the graphics card.

We expect that tiling methods would benefit this algorithm. Instead of using error metrics on every triangle, they could be done on groups. Tiling methods have been explored for Bézier patchs and subdivision surfaces. Additionally, subdivision could be performed uniformly across a tile. Although more difficult to implement, these methods may significantly improve performance.

**Acknowledgements**

BTT and subdivision code by Turner and Sharp at www.gamasutra.com provided a starting point for our implementation. Thanks also to Dinesh Manocha for advice. This research is supported in part by NSF DMI-9900157, ONR N00014-01-1-0067 and Intel.

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

**References**

1. J. Blow. Terrain Rendering at High Levels of Detail. In *Proc. of the 2000 Game Developers Conference*, March 2000.

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

3. J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects With Radial Basis Functions. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 67–76, Aug. 2001.

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

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

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

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

8. M. Dæhlen and V. Skyth. Modelling Non-rectangular Surfaces

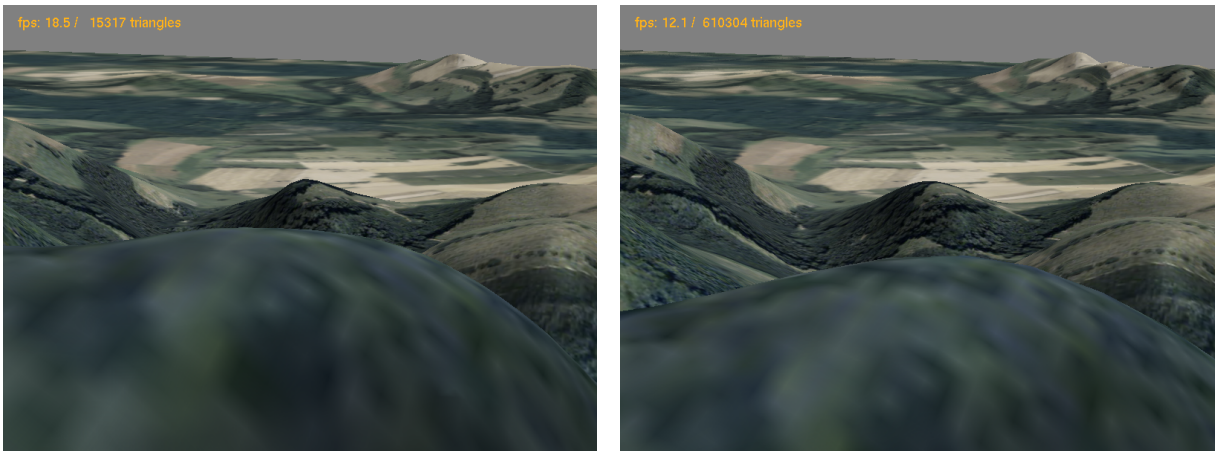


**Figure 8:** *Still Images of Sparse Data Set: our implementation (left); Bézier patch implementation (right).*

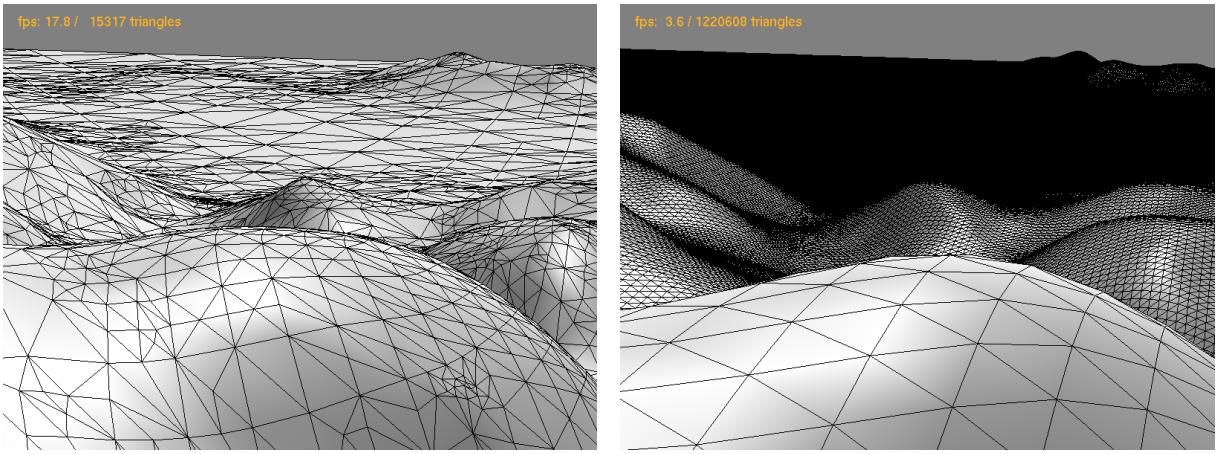


**Figure 9:** *The Corresponding Wireframe: our implementation (left); Bézier patch implementation (right).*

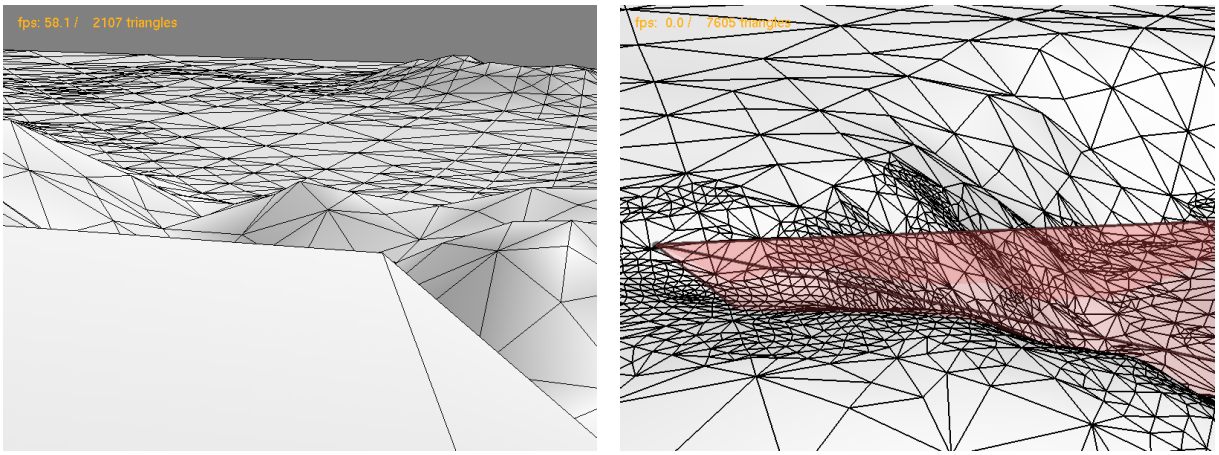


**Figure 10:** *Our implementation without butterfly subdivision (left). Outside point of view of mesh tessellation (right).*

using Box-splines. In D. C. Handscomb, editor, *Mathematics of Surfaces III*, pages 287–300. Clarendon Press, 1989.

9. 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.

10. P. Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, Oxford, 1993.

11. 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.

12. 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*, pages 81–88, 1997.

13. R. L. Ferguson, R. Economy, W. A. Kelley, and P. P. Ramos. Continuous Terrain Level of Detail for Visual Simulation. In *Proc. of the 1990 Image V Conference*, pages 145–151. Image Society, Inc., June 1990.

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

15. R. Franke. Scattered Data Interpolation: Test of Some Methods. *Mathematics of Computation*, 38(157):181–200, Jan. 1982.

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

17. R. H. J. Gmelig Meyling and P. R. Pfluger. Smooth Interpolation to Scatterd Data by Bivariated Piecewise Polynomials of Odd Degree. *Computer Aided Geometric Design*, 7(5):439–458, Aug. 1990.

18. G. Greiner and K. Hormann. Interpolating and Approximating Scattered 3D Data with Hierarchical Tensor Product Splines. In A. Le Méhauté, C. Rabut, and L. L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, pages 163–172. Vanderbilt University Press, 1996.

19. J. Haber, F. Zeilfelder, O. Davydov, and H.-P. Seidel. Smooth Approximation and Rendering of Large Scattered Data Sets. In *Proc. IEEE Visualization 2001*, pages 341–347; 571, Oct. 2001.

20. 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.)*, pages 295–302, July 1994.

21. R. Klein, D. Cohen-Or, and T. Hüttner. Incremental View-dependent Multiresolution Triangulation of Terrain. *The Journal of Visualization and Computer Animation*, 9(3):129–143, July – Sept. 1998.

22. P. Lancaster and K. Šalkauskas. *Curve and Surface Fitting*. Academic Press, London, 1986.

23. 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.

24. 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.

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

26. P. Lindstrom and V. Pascucci. Visualization of Large Terrains Made Easy. In *Proc. IEEE Visualization 2001*, pages 363–370, 2001.

27. S. K. Lodha and R. Franke. Scattered Data Techniques for Surfaces. In *Proc. Dagstuhl Conf. Scientific Visualization*, pages 182–222, 1999.

28. 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.

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

30. M. J. D. Powell. Radial Basis Functions for Multivariable Interpolation. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation of Functions and Data*, pages 143–168. Oxford University Press, 1987.

31. 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.

32. D. Rose, M. Kada, and T. Ertl. On-the-Fly Adaptive Subdivision Terrain. In *Proc. Vision, Modeling, and Visualization (VMV 2001)*, pages 87–92, 2001.

33. 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.)*, pages 179–188, 1986.

34. L. L. Schumaker. Fitting Surfaces to Scattered Data. In G. G. Lorentz, C. K. Chui, and L. L. Schumaker, editors, *Approximation Theory II*, pages 203–268. Academic Press, 1976.

35. A. J. Stewart. Hierarchical Visibility in Terrains. In *Rendering Techniques '97 (Proc. 8th Eurographics Workshop on Rendering)*, pages 217–228, 1997.

36. 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*, pages 88–96, 1997.

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

38. 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.

39. D. Zorin, P. Schröder, and W. Sweldens. Interpolating Subdivision for Meshes with Arbitrary Topology. In *Computer Graphics (SIGGRAPH '96 Conf. Proc.)*, pages 189–192, 1996.