Fast Safe Spline Surrogates for Large Point Clouds

Ashish Myles

Computer Science
University of Florida
Gainesville, FL 32611

Jörg Peters

Computer Science
University of Florida
Gainesville, FL 32611

Abstract

To support real-time computation with large, possibly evolving point clouds and range data, we fit a trimmed uniform tensor-product spline function from one direction. The graph of this spline serves as a surrogate for the cloud, closely following the data safely in that, according to user choice, the data are always 'below' or 'above' when viewed in the fitting direction. That is, the point cloud is guaranteed to be completely covered from that direction and can be sandwiched between two matching spline surfaces if required. This yields both a data reduction since only the spline control points need to be further processed and defines a continuous surface in lieu of the isolated measurement points. For example, using a 20×20 *spline, clouds of 300K points are safely approximated in less than 1/2 second.*

1 Introduction

When a dense cloud of points is sampled from a surface, a number of algorithms are known to reconstruct a nearby surface from the samples (e.g. [8, 9, 18], [1, 2], [5, 6] to name just a few). Most recently, the direct manipulation and display of such clouds has become feasible [17, 15], aided by a technique that uses the point cloud as an attractor to locally enrich the cloud in a consistent fashion [10, 3]. α-shapes [7] can be used to associate a a smooth skin or accessible surface with a point set [4].

Our goal is slightly but crucially different from the above approaches. We want to compute safe and structurally simple functions whose graphs serve as surrogate surfaces with the ability to trade tightness of the fit for simplicity of the surrogate surface. In particular, we want to take advantage of higher-order approximation and the simplicity and wide acceptance of uniform cubic splines. Here *safe* means that we do not fit extremal points in the least squares sense, but guarantee that all data points are below (or all are above) the spline, according to the application, when viewed from the fixed direction (see Figures 1, 11). Here and in the following, we do not address the issue of noisy data but assume that all data are accurate and relevant or that the risk of misjudging and discarding a point as noise is too high.

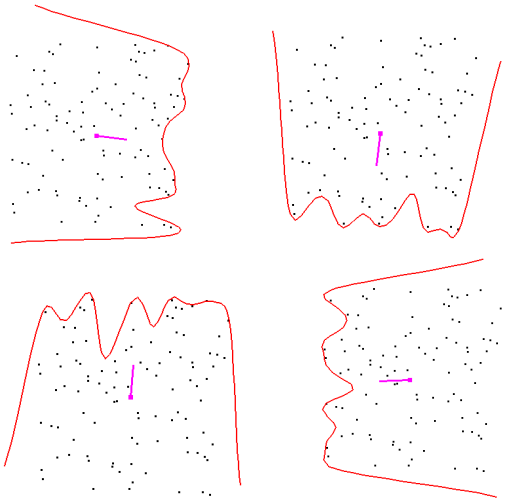

Figure 1. 2D problem. Surrogate spline functions of a cloud of $N = 100$ points in the plane for four different directions. Each spline (in one variable) has $n = 20$ knots. The direction is indicated by the line segment anchored at the center of the cloud.

Our task is both simpler and harder than the challenges listed in the first paragraph. It is simpler in that we only want to delineate the extent of the point cloud from one fixed direction. It is more challenging in that we want a *very fast and safe* method that, even for large clouds, tracks the evolution tightly and safely by a simple, standard cubic spline function.

Such a surrogate can not only serve as a high-quality imposter for display (as could almost any other surface fit that improves over flat billboards) but can be used for conserva-

tive occlusion tests or intersection tests such as for coarsely machining a mechanical part. For example, any ray within a certain cone about the viewing axis can be written as a linear function $(u(t), v(t), r(t)) = (at + b, ct + d, r(t))$ in the coordinate system $(u, v, *)$ of the surrogate spline function $f(u, v)$ (the $*$ slot is the direction of the axis); intersection then amounts to finding the roots of the spline $f(at+b, ct+d)-r(t)$ in one variable t, for which there exists an unconditionally, quadratically convergent algorithm [12]. By specifying the (density of the) knot set, we obtain surrogates of different complexity (see Figure 2). Different knot spacings allow for (local) refinement. Labeling segments of the spline as *'free'* allows to substitute energy minimization, for example where there are not enough data.

To set the stage, we describe, in Section 3, a possible but ultimately inefficient approach in one variable. Section 4 describes an alternative approach that can safely fit uniform piecewise linear functions. The idea of this intermediate scheme is leveraged and generalized in Section 5 to safely and fast fit in one variable; and in Section 6 to fit in two variables. Section 7 characterizes the cost of the algorithm.

2 Technical Preliminaries

We denote the scattered data cloud as

$$\{\mathbf{x}_i\}, \qquad \mathbf{x}_i \in \mathbb{R}^2 \text{ or } \mathbf{x}_i \in \mathbb{R}^3, \qquad i = 0, \ldots, N.$$

Throughout this paper, f is a *uniform cubic spline* with integer knots $-3, -2, \ldots, n + 2$ and B-spline coefficients

$$b_{-1}, b_0, ..., b_{n-1}, b_n, \qquad b_{-1} = b_0 \text{ and } b_{n-1} = b_n.$$

The Greville abscissae for $b_0, ..., b_{n-1}$ are therefore $0, \ldots, n - 1$. For $b := [b_1, \ldots, b_n]^{\mathsf{t}}$ the difference operator $\Delta_i b$ is defined as

$$\Delta_i b := b_{i-1} - 2b_i + b_{i+1}.$$

Using de Boor's algorithm, it is easy to see that for all $v \in [0, n - 1]$, and $\lfloor\rfloor$ returning the next lower integer,

$$h := \lfloor v \rfloor \in \mathbb{Z}, \qquad u := v - \lfloor v \rfloor = v - h \in [0, 1],$$

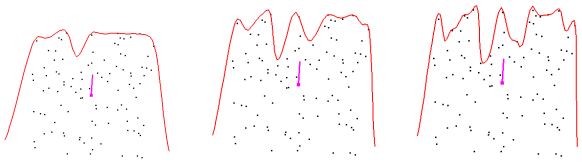

Figure 2. Fitting a 2-dimensional point cloud from the same direction with 16, 24, and 32 uniform knots.

$$f(v) = (1 - u)b_h + ub_{h+1} + \frac{1}{6}(1 - u)^3\Delta_h b + \frac{1}{6}u^3\Delta_{h+1}b. \tag{1}$$

Throughout this paper, the index h and the local variable u depend on the argument of f in precisely this manner.

Given a vector s of size $|s|$ of parameters $\{s_i\} \subset [0..n - 1]$, the evaluation of f at s can be rewritten in matrix form:

$$f(s) = \left(\mathrm{I}_s + \frac{1}{6}\mathrm{S}_s\mathrm{D}\right) b. \tag{2}$$

Here D is a three-banded $n \times n$ matrix whose rows are the second difference masks $1, -2, 1$ except for the first and last row which are first differences due to the repeated control point at each end:

$$\begin{aligned} &\mathrm{D}(i, i-1) = \mathrm{D}(i, i+1) = 1, \mathrm{D}(i, i) = -2, \\ &\quad i = 2, \ldots, n-1, \\ &\mathrm{D}(1,1) = \mathrm{D}(n, n) = -1, \quad \mathrm{D}(1, 2) = \mathrm{D}(n, n-1) = 1. \end{aligned}$$

The two-banded matrices S_s and I_s are of size $|s| \times n$ and

$$\begin{aligned} h := \lfloor s_i \rfloor, \quad u_i := s_i - h, \\ \mathrm{S}_s(i, h) = (1 - u_i)^3, \quad \mathrm{S}_s(i, h+1) = u_i^3, \\ \mathrm{I}_s(i, h) = 1 - u_i, \quad \mathrm{I}_s(i, h+1) = u_i. \end{aligned}$$

3 Trial-and-Error 2D Cloud Surrogate

Consider data points $(x_i, y_i) \in \mathbb{R}^2$ to be fit safely from the y-direction. A naive, trial-and-error method repeatedly searches for the most constraining points and interpolates them with a spline f in the hope that the remaining data points will end up below this spline.

1. **Assign parameters:** Shift and scale x_i so that they lie in the interval $[0..n - 1]$. Use the resulting values as parameters t_i of y_i.

2. **Select points:** For $j \in \{1, \ldots, n\}$, select one y_i whose parameter t_i falls into the unit interval $[j-0.5, j+0.5]$. Set $s_j := t_i$ and $f(s_j) = y_i$ to form the jth equation of the system (2). If no t_i lies in $[j - 0.5, j + 0.5]$, the jth equation enforces that the second difference vanish.

3. **Fit spline:** Solve the system.

4. **Check spline:** If some y_l lies above the spline and t_l lies in $[j - 0.5, j + 0.5]$, then replace $s_j := t_l$ and the jth equation of the system with and $f(s_j) = y_l$, and repeat step 3.

Since the matrix in step 3 is band-diagonal with at most two bands on each side of the diagonal, step 3 takes $O(n)$ time and the overall time complexity is $O(n + |s|)$. Apart from the concern of convergence, we will see that this trial-and-error heuristic becomes very expensive in the case of two variables, because the matrix in step 3 no longer has the simple banded structure. We therefore consider an alternative approach that constructs a spline that stays to one side of a piecewise linear interpolant of uniform point data. This is not the problem we set out to solve but its solution will be leveraged to obtain an efficient algorithm for the original problem.

4 Broken Line Surrogate

Given a piecewise linear function ℓ with uniform knots, the goal is to construct a nearby spline f such that $f \geq \ell$. The simplicity and quality of the construction depends crucially on the following nontrivial characterization.

Lemma 4.1 *The uniform cubic spline f is bounded* below *by the piecewise linear function $\ell : [0, n-1] \to \mathbb{R}$ with breakpoints (i, ℓ_i), $i = 0, \ldots, n-1$,*

$$\ell_i := b_i + \frac{1}{6}\min\{\Delta_i b, 0\}. \tag{3}$$

Figure 3 illustrates the relationship between coefficients b_i and breakpoints ℓ_i. Readers familiar with spline theory will spot the quasi-interpolant of cubic spline interpolation. The proof of Lemma 4.1 is subsumed by the proof of Lemma 5.1, and is therefore deferred until then. Among the lower bounds, the choice in Lemma 4.1 best balances (i) minimizing the distance between ℓ and f and (ii) minimizing the unavoidable Gibbs oscillation of splines when gradients change rapidly, say at fault lines. Moreover, it is the simplest choice other than the trivial and insufficient straight line below the control polygon of f.

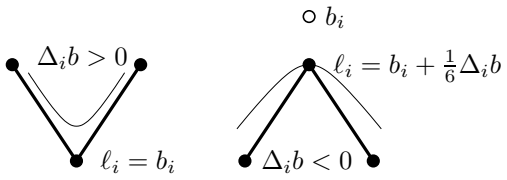

Figure 3. Interpretation of the function ℓ constructed in Lemma 4.1. (*left*) At local minima, the control polygon and ℓ coincide; (*right*) at local maxima, the control polygon defines a quasi-interpolant matching ℓ.

We can restate the lower bound relation in matrix form as follows. (We do not yet consider control points marked *free*, i.e. whose Greville abscissa falls in a unit interval $[j - 0.5..j+0.5]$ that has no associated data.) Defining the vector

$$\sigma := [\sigma_1, \ldots, \sigma_n],$$
$$\sigma_i := \text{sgn}(b, i) := \begin{cases} 1 & \text{if } \Delta_i b < 0, \\ 0 & \text{else,} \end{cases}$$

we have, with $\text{diag}(\sigma)$ denoting the matrix with diagonal σ,

$$\ell = \left(\text{I} + \frac{1}{6}\text{diag}(\sigma)\text{D}\right) b = \text{M}b. \tag{4}$$

The task is to compute the coefficients b_i so that the spline f lies above ℓ. The inversion

$$b = \text{M}^{-1}\ell$$

of the diagonally dominant matrix M is deceptively simple. There are two types of unknowns, b_i and σ_i, in Equation (4); and b_i and σ_i depend on one another.

Nevertheless, there is a very efficient heuristic to solve (4). Given a choice of signs σ_i, (4) can be solved in only $12n$ flops using a banded solver. Mimicking least squares fitting for with unknown parameters or Remez's algorithm for max-norm optimization, we alternate between choosing the signs σ_i and solving for the coefficients b_i until the σ_i agree with the signs of $\Delta_i b$.

Heuristic(σ, b)

1. Initialize $\sigma_i \leftarrow \text{sgn}(\ell, i)$.
2. Solve $\ell = \text{M}b$ for b given σ.
3. If $\text{sgn}(b, i) = \sigma_i$ for all i, stop; else, set $\sigma_i = \text{sgn}(b, i)$ and go to step 2.

Typically, this iteration converges in *two iterations* and we have never observed more than three iterations. A histogram analogous to those in Figure 9 shows a single large spike at 2 with negligible counts at 1 and 3. In particular, the number of iterations appears to be independent of n with all sign changes localized to one of several point neighborhoods. Effectively, the initialization $\sigma_i = \text{sgn}(\ell, i)$ is very close to optimal (just as the initialization of Remez's algorithm [13] by expanded Chebyshev points is observed to yield rapid convergence).

By contrast, solving the constrained optimization problem:

$$\min f - \ell \qquad \text{subject to } f \geq \ell$$

over $[0, \ldots, n]$, is nontrivial. (Recently [14] reduced this type of problem to a linear program, but this approach is still orders of magnitude slower than the proposed method.)

Equation (4) changes slightly when a point ℓ_i is marked free. To avoid working with an underconstrained system

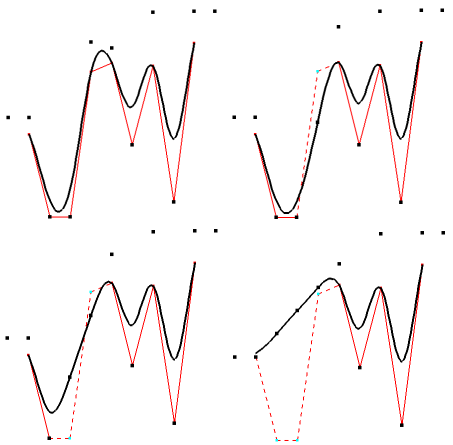

Figure 4. Inversion with 0,1,2 or 3 free breakpoints and energy minimization.

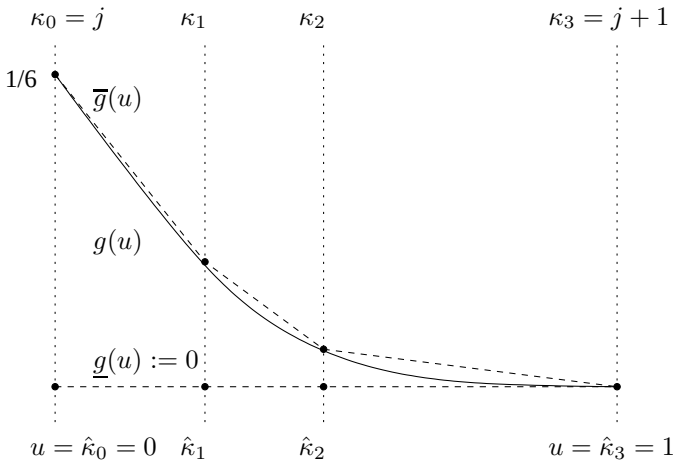

Figure 5. Bounding $g(u) = (1-u)^3$ with two piecewise linear functions with specified local break knots $\hat{\kappa}_i := \kappa_i - \lfloor \kappa_i \rfloor \in [0,1)$.

and seizing on the chance to improve the spline surrogate, we can replace the constraint row in M by another linear equality constraint that does not make the system singular. For example, Figure 4 shows the effect of one, two and three free points when the corresponding second differences are set to zero. Since the change only amounts to replacing the corresponding ith row of M with $[\ldots, 0, 1, -2, 1, 0, \ldots]/6$ and setting $\ell_i = 0$, the complexity of the inversion process is not affected.

5 2D Scattered Points Surrogate

The surrogate computed in the previous section will now be generalized to a lower bound with arbitrary breakpoints $(t_i, \ell(t_i))$. We can then ignore the piecewise linear nature of ℓ and view $(t_i, \ell(t_i))$ as data $\mathbf{x}_i$. Our immediate goal though is to compute coefficients b_i so that ℓ lies above a constraining piecewise-linear function Λ with knots $t_i \in [0..n-1]$, i.e.

$$f \geq \ell \geq \Lambda.$$

Lemma 5.1 *Let $\ell : [0..n-1] \to \mathbb{R}$ be a piecewise linear function whose knots $\{s_i\}$ include but is not restricted to the integers $0, \ldots, n-1$. Then the uniform cubic spline f with knots $0, \ldots, n-1$ is bounded below by ℓ if ℓ has breakpoints*

$$\begin{aligned}\ell(s_i) := (1-u)b_h + ub_{h+1} \\ + (1-u)^3 \frac{1}{6} \min\{\Delta_h b, 0\} \\ + u^3 \frac{1}{6} \min\{\Delta_{h+1} b, 0\}. \end{aligned} \quad (5)$$

Proof We may focus on a subsequence $\{\kappa_i\}$ of s between two knots $j, j+1$, $j \in 0, ..., n-2$ of the spline. For $v := \kappa_i$ and local $\hat{\kappa}_i := \kappa_i - \lfloor \kappa_i \rfloor \in [0,1)$, $u := v - \lfloor v \rfloor \in [0,1]$ and $g(u) := \frac{1}{6}(1-u)^3$ and $g(1-u) := \frac{1}{6}u^3$, we have

$$\begin{aligned} f(v) - \big((1-u)b_h + ub_{h+1}\big) \\ = g(u)\Delta_h b + g(1-u)\Delta_{h+1} b. \end{aligned}$$

Since, on $[0..1]$, g is non-negative and concave upward, it is bounded below by 0 and above by the piecewise linear interpolant $\overline{g}$ of g at the $\hat{\kappa}_i$ (see Figure 5). Then, combining negative numbers with upper bounds and positive numbers with lower bounds (zero), we get

$$\begin{aligned} & f(v) - ((1-u)b_h + ub_{h+1}) \\ &= g(u)\left(\min\{\Delta_h b, 0\} + \max\{\Delta_h b, 0\}\right) + \\ &\quad g(1-u)\left(\min\{\Delta_{h+1} b, 0\} + \max\{\Delta_{h+1} b, 0\}\right) \\ &\geq \overline{g}(u)\min\{\Delta_h b, 0\} + \overline{g}(1-u)\min\{\Delta_h b, 0\}. \end{aligned}$$

□

As in the simpler bound in Section 4, $\ell(t)$ is computed in time linear in $|s|$. Since for $s = (0, ..., n-1)$, we have $u = 0$ for all breakpoints, Lemma 4.1 follows. The lower bound is similar to the one derived by [11] and [16]. However, those bounds rely on tabulated pre-optimized piecewise-linear bounds for $g(u)$ with breakpoints at uniformly spaced $\hat{\kappa}$. Our new piecewise linear bound does not require precomputation since $\overline{g}(u)$ is easily determined online by the simple evaluation of g at $\hat{\kappa}$. Therefore, our new bound can adapt to breakpoints on the fly.

We rewrite (5) in matrix form with D, b, S_s and I_s as defined in Section 2 and σ as defined near (4):

$$\ell = \left(\mathrm{I}_s + \frac{1}{6}\mathrm{S}_s \mathrm{diag}(\sigma)\mathrm{D}\right) b = \mathrm{M}b. \quad (6)$$

This system has $|t|$ rows, i.e. as many as there are break knots t_i. Since for our applications $N = |t| >> n$, we need to select n equations, or, equivalently, n knots t_j^* of the t_i. To do so, we associate each t_i with the closest integer, and hence spline Greville abscissa, and select exactly one t_j^* for each integer $j = 0, \ldots, n$. We encode this in a selection matrix C_λ of size $n \times N$). All its entries are zero except for $\mathrm{C}_\lambda(j, \lambda_j) = 1$ if λ_j is the index of the selected knot. The resulting system

$$(\mathrm{C}_\lambda \mathrm{M})b = \Lambda \tag{7}$$

can be solved for b in $O(n)$ operations since $\mathrm{C}_\lambda\mathrm{M}$ is banded with at most two bands on each side.

In addition to the two vectors of unknowns, b and σ, we now have to determine λ. This selection is updated every time the choice of b and σ is valid. If any $\Lambda(t_k) > \ell(t_k)$ then the k with maximal $\Lambda(t_k) - \ell(t_k)$ is selected to replace the equation in row round(t_k) in (7).

Since the knots of Λ can be chosen exactly as needed, and the equations depend only on the breakpoint values, we need no longer view Λ as a piecewise linear function. Rather, it can be *any, unordered* data $\mathbf{x}_i$ of the form (knot, value). From here on, we will therefore consider

Λ a collection of scattered data points.

The heuristic is summarized as follows.

Heuristic(σ, b, λ) non-uniform data

1. Initialize λ_j as the point in Λ on the interval $[j - 0.5..j + 0.5]$ with the maximal value:

   $\lambda_j := \max\{y_i : (t_i, y_i) \in \Lambda \text{ and round}(t_i) = j\}$.

2. Initialize all $\sigma_j = 1$.
3. Solve $\ell = \mathrm{C}_\lambda \mathrm{M} b$ for b given σ and λ.
4. If $\mathrm{sgn}(b, j) = \sigma_j$ for all j, continue; else, set $\sigma_j = \mathrm{sgn}(b, j)$ and go to step 3.
5. For the current b, determine the i so that $\Lambda(t_i) - \ell(t_i)$ is maximal and positive. If no i exists, stop. Otherwise, update $\lambda_{\lfloor t_i \rfloor} = i$ and go to step 3.

The complexity of the heuristic is $O(n + |t|)$ where t is the vector of break knots of Λ. The constant of the big O notation is the number of iterations until convergence. In almost all cases, this number is less or equal to 10 and only occasionly have we encountered a case where the number is up to 20 (see Figure 9, *right*). If any control point does not have any associated ℓ breakpoints, it is free and treated as in the previous section. The results of this method are illustrated in Figures 1 and 2.

6 Surface Surrogates

Suppose we wanted to interpolate a subset of $m \times n$ data $\mathbf{x}_i \in \mathbb{R}^3$ (satisfying the Schoenberg-Whitney conditions) by a spline surface with $m \times n$ control points. The corresponding interpolation matrix is of size $mn \times mn$. If the parameter values associated with $\mathbf{x}_i \in \mathbb{R}^3$ do not have special structure the cost of interpolation is $O(m^3n^3)$ and the overall cost of mimicking the naive procedure in Section 3 is $O(m^3n^3 + N)$. However, if the data are on a u, v grid, the interpolation matrix factors into two banded matrices and we can tensor interpolation, reducing the overall cost to $O(mn + N)$ as follows.

1. Use the naive curve-fitting method in Section 3 along the u direction m times to obtain m intermediate spline curves with n control points each.

2. Apply the same method to the resulting $m \times n$ array of control points along the v direction n times to obtain an array of n spline curves with m control points each. The result is the $m \times n$ array of control points for the surface.

To apply this cheaper solution to our ungridded data, we exploit the fact that the splines output by the heuristics in Sections 4 and 5 stay above the linear interpolant of their input point set. We proceed as follows (see Figure 6).

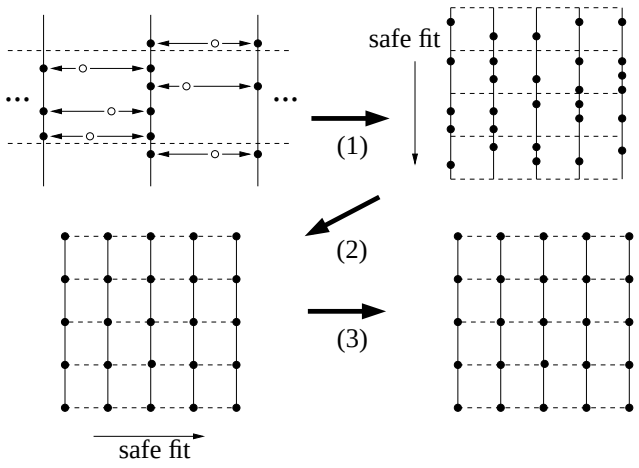

Figure 6. Surrogate construction. (1) Project all the points to their neighboring u-parameter lines. (2) Safely fit the resulting non-uniform data along the u direction. (3) Safely fit the resulting uniform data along the v direction.

Reduction to gridded data

1. Project all the points in the cloud to their two neighboring u-parameter lines.
2. For each of the m u-parameter lines, use the general **Heuristic(σ, b, λ)** of Section 5 to fit a spline curve with n control points on the non-uniform data. This results in an $m \times n$ array of control points, which can also be thought of as uniform data for the next step.
3. For each of the n v-parameter lines, use the simpler **Heuristic(σ, b)** for piecewise lines from Section 4 with m control points on the uniform data. The resulting $m \times n$ control points define the safe spline surrogate.

Exchanging u and v and averaging symmetrizes the heuristic but is not necessary. The tensored surrogate computation has time complexity of $O(mn + N)$. The resulting surface stays above the bilinear interpolant of two projections of each point which, in turn, is guaranteed to be above the original point before projection. Therefore, the resulting surface stays above the input point cloud.

Similarly, since the surface surrogate is to one side of the bilinear interpolant of two projections of each point, applying the construction to one direction and its opposing direction creates a pair of *non-intersecting* surfaces that sandwich the point cloud. Figure 12 shows cross sections of such a two-sided fit.

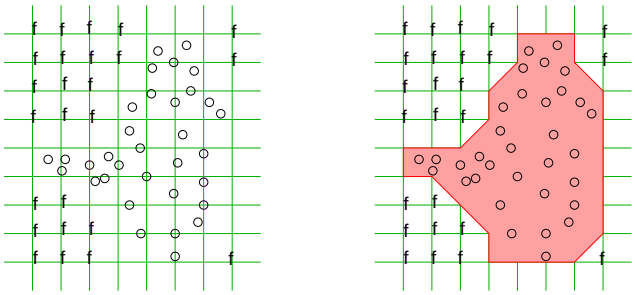

Figure 7. (*left*) Determining free grid points (labeled f) that will not influence the shape of the surrogate spline. (*right*) Piecewise linear trim lines (see also Figure 11).

A problem inherent to tensor-product splines is *aliasing*. The B-spline footprint (support) betrays the underlying grid of knot lines by zig-zaging along them. This becomes visible along the boundary of the orthogonally projected point set (Figure 8). Piecewise linear trimming along and diagonal to grid cells of the outer boundary is a simple answer (see Figure 7). Of course, along sharp interior jumps, aliasing and the Gibbs (overshooting) effects are still visible from side views (see Figure 10, *middle*).

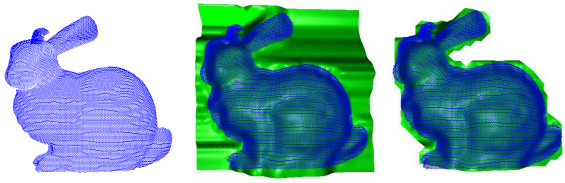

Figure 8. (*left*) Range Data, (*middle*) surface fit from below and (*right*) trimmed surface.

7 Test Cases and Timing

This method is parallelizable in that, in Step 2, each of the m sets of computation are independent of one another and, in Step 3, each of the n computations. Our implementation reviewed below, however, was on a single processor.

To test our theory and implementation, we created a sequence of surrogates for the union of two moving spherical scatter clouds. On this synthetic data, we vary N and n, the number of data points and the $n + 2 \times n + 2$ control points. Since we are using a heuristic, we computed all numbers by averaging over 10 different configurations and 20 surrogate computations. The times in Tables 1, 2 include
(i) the projection of the point cloud to a plane parallel to the fitting direction for parameter assignment;
(ii) the projection of the points to two nearby parameter lines; in addition to
(iii) the heuristic to obtain the control points.
The computational environment is a generic desktop PC with a 2.4GHz Pentium 4 with 512Mb RAM, running Linux.

N	10K	30K	50K	100K	300K
msecs	14.5	42.5	71.0	141	445

Table 1. Time in milliseconds for constructing a spline surrogate for a cloud of N points ($N = 10,000$ to $300,000$) and a 20×20 auxiliary (i.e. $n = 20$).

n	10	20	30	40	50	60
msecs	42.5	42.5	45.0	51.0	55.5	64.5

Table 2. Time for constructing an $n \times n$ spline surrogate for a cloud of $N = 30K$ points.

As expected, the times reported in Table 1 increase lin-

early in the size, N, of the cloud. Indeed, Table 2 indicates that for the typical $N >> n$, and an $n \times n$ spline, the linear complexity in N dominates time quadratic complexity in n.

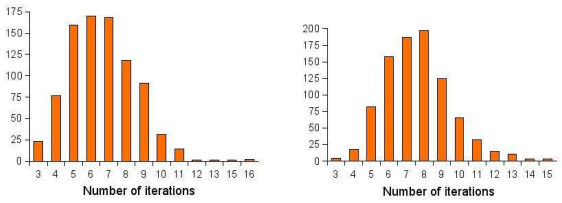

Figure 9. Histogram of the total number of iterations (LDL decompositions of the matrix) of the general Heuristic(σ, b, λ) of Section 5 when computing a surrogate surface for $N = 30K$, $N = 300K$ data points and $n \times n$ control points, $n = 20$.

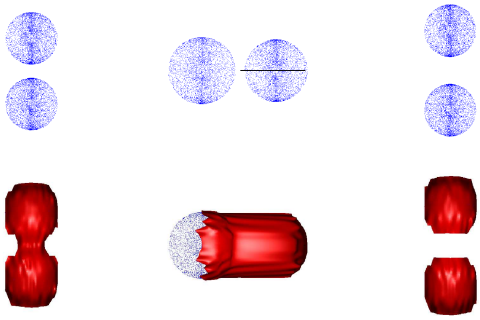

Figure 10. Real-time safe surrogate spline for two moving spherical scatter clouds. (*middle*) Although not visible from the front, a side view shows the aliasing of tensor-product splines at the steep jump between the two clouds when the direction is skew to the line through the centers of the two point clouds.

8 Summary

By computing a simple safe spline surrogate, in real-time for moderately sized point clouds of $N = 30K$ points, we are able to safely replace a large unstructured discontinuous cloud of point data by a structured smooth spline defined by a uniform $n \times m$ grid of spline control points. As the standard higher-order representation of scientific computing, this surrogate can then be used for real time safe tracking, compression, comparison over time and other computations and queries of the point cloud.

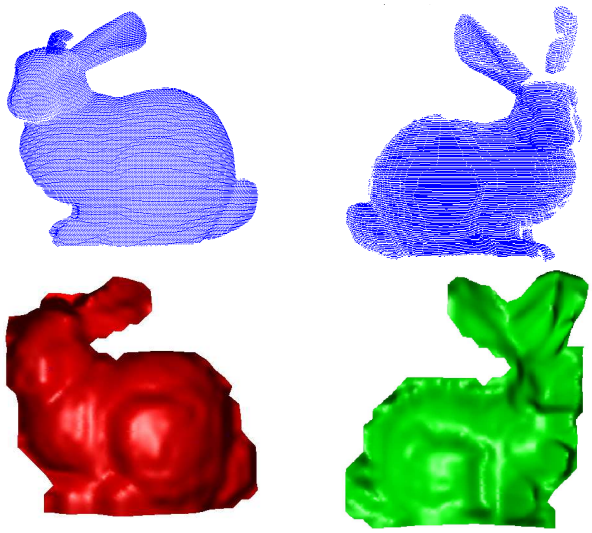

Figure 11. Real-time directionally safe surrogates for range data of the Stanford bunny ($N = 40,256$). (*left*) Safe front fit (58ms), (*right*) Safe back fit (59.5ms).

9 Acknowledgments

Work supported in part by NSF Grants DMI-0400214 and CCF-0430891.

References

[1] Amenta, Choi, and Kolluri. The power crust, unions of balls, and the medial axis transform. *CGTA: Computational Geometry: Theory and Applications*, 19, 2001.

[2] N. Amenta, S. Choi, and R. K. Kolluri. The powercrust. In D. C. Anderson and K. Lee, editors, *Proceedings of the Sixth Symposium on Solid Modeling and Application (SM-01)*, pages 249–260, New York, June 6–8 2001. ACM Press.

[3] N. Amenta and Y. J. Kil. Defining point-set surfaces. *ACM Trans. Graph.*, 23(3):264–270, 2004.

[4] H. Cheng, T. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. *Discrete Computational Geometry*, 25:525–568, 2001.

[5] T. K. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. In *IEEE Symposium in Parallel and Large Data Visualization and Graphics*, pages 19–27, 2001.

[6] T. K. Dey and S. Goswami. Tight cocone: A water-tight surface reconstructor. *Journal of Computing and Information Science in Engineering*, 3:302–307, 2003.

[7] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graphics*, 13(1):43–72, Jan. 1994.

[8] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28(Annual Conference Series):295–302, July 1994.

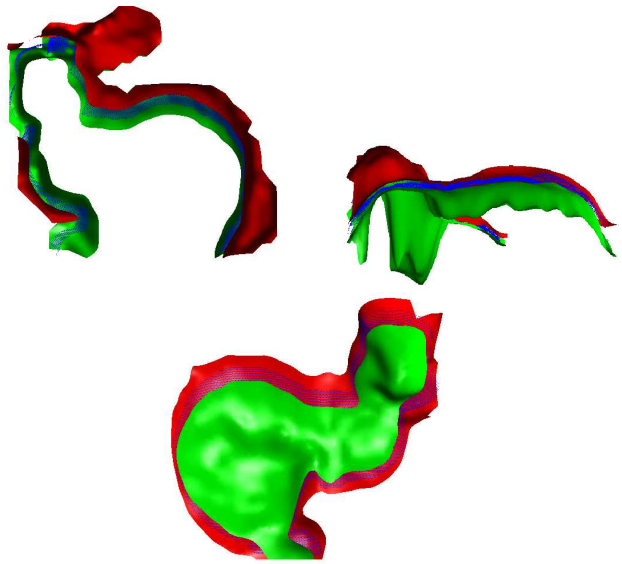

Figure 12. Three slices through the ensemble of safe front fit, range data cloud and safe back fit. Front and back fits do not intersect but yield a proper bunny sandwich.

[9] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. *Computer Graphics*, 30(Ann. Conf. Series):313–324, 1996.

[10] D. Levin. Mesh-independent surface interpolation. In H. Brunnett and Mueller, editors, *Geometric Modeling for Scientific Visualization*, pages 37–49. Springer-Verlag, 2003.

[11] D. Lutterkort and J. Peters. Tight linear envelopes for splines. *Numerische Mathematik*, 89(4):735–748, Oct. 2001.

[12] K. Moerken and M. Reimers. An unconditionally convergent method for computing zeros of splines and polynomials. *submitted for publication*, 2005. http://heim.ifi.uio.no/∼martinre/publications.html.

[13] F. D. Murnaghan and J. W. Wrench, Jr. The determination of the Chebyshev approximating polynomial for a differentiable function. *Mathematical Tables and Other Aids to Computation*, 13(67):185–193, July 1959.

[14] A. Myles and J. Peters. Threading splines through 3d channels. *Computer Aided Design*, 37(2):139–148, 2004.

[15] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In J. Hodgins and J. C. Hart, editors, *Proceedings of ACM SIGGRAPH 2003*, volume 22(3) of *ACM Transactions on Graphics*, pages 641–650. ACM Press, 2003.

[16] J. Peters and X. Wu. Sleves for planar spline curves. *Computer Aided Geometric Design*, 21(6):615–635, 2004. http://authors.elsevier.com/sd/article/S0167839604000615.

[17] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[18] T. Várady, R. R. Martin, and J. Cox. Reverse engineering of geometric models - an introduction. *Computer-aided Design*, 29(4):255–268, 1997.