Eurographics Symposium on Geometry Processing (2006), pp. 1–11
Konrad Polthier, Alla Sheffer (Editors)

# WUCSE-2006-29: Smooth Surface Reconstruction using Charts for Medical Data

Paper ID 74

**Abstract**

*We present a surface reconstruction technique that constructs a smooth, $C^k$, analytic surface from scattered data. The technique is robust to noise and both poorly and non-uniformly sampled data, making it well-suited for use in medical applications. In addition, the surface can be parameterized in multiple ways, making it possible to represent additional data, such as electromagnetic potential, in a different (but related) coordinate system to the geometric one. The parameterization technique also supports consistent parameterizations of multiple data sets.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computing Methodologies]: Computer GraphicsComputational Geometry and Object Modeling

## 1. Introduction

Mesh reconstruction from data is a well-studied problem, and good solutions exist, particularly when the underlying surface is uniformly sampled [CDR05, BMR*99, MAVdF05]. Unfortunately, most medical data is not sampled uniformly. Specialized techniques exist for contour data [PSV98, AG04] but they may not produce a mesh with the correct topology. Parameterizable reconstructions for data sets of arbitrary topology are less well-studied in either case. We present a solution that produces a water-tight surface of a given topology. We are primarily interested in medical applications, but also show that the approach works for laser-scanned data sets.

Increasingly, medical data consists not just of geometry, but of additional information such as electromagnetic potential across the surface. The best parameterization for these additional data may not be the geometric one. For example, in the heart data set (Figure 1), there are three different parameterizations. The first is the geometric one, which consists of small, ellipsoidal pieces of surface. The second one is a texture map based on overlapping vertical stripes (the original images were taken by rotating the camera around the heart). The third one arises from three Photo Diode Array sensors placed around the heart which indirectly measure electric activity over time by capturing fluorescent images. A subsequent algorithm converts the image data to electromagnetic data. This algorithm was originally designed to operate on grids of a particular size; rather than re-design the algorithm, we cover the corresponding areas with quadrilateral parameterizations (see Section 8).

The input to our algorithm is a set of unlabled data points. Our approach begins by constructing *local neighborhoods* around each point — essentially one-ring approximations of the local surface. We use this local connectivity information to put the data points into a 1-1 correspondence with the appropriate domain $D$ for that topology (either a sphere or an $n$-holed torus). We then define overlapping embeddings for subsets of $D$, each of which smoothly approximates the corresponding data points. These individual embeddings are blended together to produce the final, $C^k$ surface. We then define additional parameterizations of $D$ more suited to representing other data, such as color, on the surface. These parameterizations are linked to the geometric one through $D$, but can take very different forms.

**Contributions:** Our primary contribution is an analytic surface reconstruction technique that supports subsequent representation of additional data defined on the surface. We also describe a robust method for estimating local neighborhoods and normals in irregularly sampled data. As an intermediate stage of the surface construction we produce a water-tight mesh that interpolates the data points. The final surface has guaranteed topology and continuity, and also has a very compact representation.

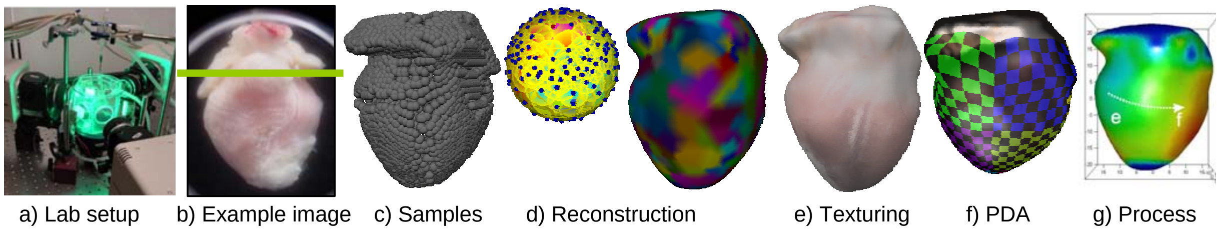

**Figure 1:** *Reconstructing a rabbit heart. From left to right: a) The capture set-up. b) Images used to perform silhouette volume carving. c) Generated samples. d) Reconstructed surface created by embedding graph of data points on the sphere, covering the sphere with charts, then fitting the charts to the data. Ovals on the sphere show chart domains, colors (right) indicate charts. e) Texture mapping with the original images. f) Parameterization for PDA images. g) Electromagnetic data calculated on surface.*

## 2. Previous work

A full review of surface reconstruction techniques is beyond the scope of this paper; we discuss examples of several different approaches.

Surface reconstruction approaches can be broken into three categories based on the *type* of data they produce — meshes, analytical surfaces, and moving least-squares surfaces. Mesh-based approaches include 3D Delauney triangulations [MAVdF05, Att97, CDR05, FR02], alpha-shapes [BBCS97], rolling-ball [BMR*99] or locally greedy [Boi84], implicit-based [HDD*92] and contour filling [AG04]. Analytical approaches build either spline-based [GLC02] or smooth implicit surfaces [SOS04]. Moving least-squares approaches [FCOS05, OBA*03] produce a *theoretically* smooth surface onto which points can be projected. This surface can be converted to a mesh by building an implicit function.

Mesh-based approaches typically build a triangulation using all of the data points, then apply optional filtering [SBS05] and simplification steps. The goal is to produce a 2D manifold mesh which is water-tight (if the data form a closed surface). Purely mesh-based approaches usually do not deal well with noise or irregularly sampled data. One option for dealing with noise is to build an implicit function from the data, using signed-distances [HDD*92] or moving least-squares [FCOS05], then re-sample the implicit function using, e.g., Marching cubes.

One of the first mesh-based approaches used a local, greedy strategy [Boi84] to add triangles to a mesh. The rolling-ball [BMR*99] approach added concepts from computational geometry to produce a much more stable algorithm with theoretical guarantees. The most theoretically sound work is based on 3D Delauney triangulation [MAVdF05, CDR05], which has reconstruction guarantees based on sampling densities. This is the epsilon-ball requirement, which essentially states that the distance between neighboring points be bounded by the distance to the nearest medial axis.

Contour filling is a method developed specifically for medical data consisting of one set of parallel slices [AG04, CP99, PSV98]. It can not handle multiple slicing directions, and gaps in the contours can cause the algorithms to fail.

Implicit surface [CBC*01, AG04, YDC05, SOS04] fitting is relatively robust to missing data but requires correct normals and current techniques do not handle noise directly (although filtering, either of the implicit function or the output mesh, can be applied as a post-process). They can be computationally expensive to compute, although methods exist for reducing the number of point samples used [SOS04] and for speeding up computations [OBA*03].

Spline-based approaches [EH96, HQ04, Geo] begin with defining a patch network, then assign the data points to one (or more) patches. Fitting then alternates between optimizing for the control points using a least-squares approach and projecting the data points back onto the surface to adjust parameter values. This latter step can cause folding and misassignment of data points. Finding an optimal patch network is another issue. Another, somewhat more subtle issue is the mixing of constraints — one set of constraints for keeping patches connected smoothly, the second for fitting the data.

Previous manifold approaches use hand-crafted "example" surfaces [GLC02], and rely on projection to establish a correspondence between the surface and the data set. This requires fairly accurate alignment of the example surface and the data set, and is prone to fold-overs. An alternative approach [GHQ05] works with a coarse mesh and regular sampling.

## 3. Outline of approach

Wherever possible we would like to use the original data to do the surface reconstruction. We also want to avoid having local decisions about the data connectivity unduly influence the final surface. We also want to be able to re-parameterize *any* part of the surface at will, without restricting ourselves to subsets of an initial parameterization. Finally, the reconstructed surface should be water-tight, manifold, and of the correct topology.

The surface representation we use (Section 4) consists of a specific representation $D$ of the desired genus (sphere or

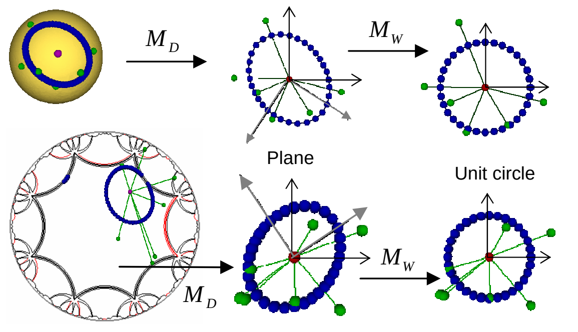

**Figure 3:** *Constructing the chart functions for the spherical (top) and n-holed (bottom) cases. $M_D$ maps from the domain to the plane, $M_W$ rotates and scales the area of interest to align it with the unit circle.*

$n$-holed tori) and a mechanism for defining local parameterizations on that representation. Embedding the domain is accomplished by defining a polynomial embedding for each local parameterization, then blending the result together. Because we have a specific representation for $D$, we can define additional, separate, local parameterizations more suited for texture mapping, data process etc. Because all of the parameterizations are on the same domain $D$ it is trivial to map back and forth between them.

```
Find local neighborhoods (5)
    Produces graph on data
Embed graph in domain D (6)
    Find valid embedding
    Adjust positions
    Adjust graph connectivity
Create charts (7)
    Ensure coverage of D
Create geometry (7.4)
    Blend function per chart
    Embed function per chart
```

**Figure 2:** *Algorithm.*

The first step of the fitting process assigns locations in the domain to each data point. We then create a set of local parameterizations based on the distribution of data points in the domain — each parameterization covers roughly the same number of points. We then fit a polynomial for each parameterization to its corresponding data. Since the parameterizations are independent, we can move both them and the data points around in the domain to reduce the error in the fit or to better distribute the points in the local parameterizations.

## 4. Surface representation

To produce a surface we begin by defining a manifold domain $D$ for each genus (sphere or $n$-holed tori) [Gri04]. The domain $D$ is then embedded by defining embeddings of *overlapping* subsets of the domain. These individual embeddings are then blended together (using the overlap information) to produce the final surface. This is analogous to the spline approach, except we are blending functions instead of control points.

The embedding $E_c$ and blend $B_c$ functions are built from polynomial or spline functions that map portions of the plane to $\mathbb{R}^3$ and $\mathbb{R}$, respectively. In order to define these functions on subsets $D_c$ of the domain $D$, we must first map $D_c$ to the plane. Let $\alpha_c : D_c \to \mathbb{R}^2$ be such a function; the entire surface is then defined by:

$$E(p) = \frac{\sum_c B(\alpha_c(p))E(\alpha_c(p))}{\sum_c B(\alpha_c(p))} \qquad (1)$$

where we define $B_c(\alpha_c(p))$ to be zero if $p \notin D_c$. We ensure that the denominator is non-zero by covering every point in the domain with at least one $D_c$, and defining the support of $B_c$ to be $c$. The continuity of $E$ depends on the minimum continuity of its constituent parts. In this paper the $\alpha_c$ and $E_c$ functions are both $C^\infty$; the blend functions are $C^3$. Unlike splines, changing the continuity of the blend function does not dramatically change the *visual* appearance of the surface because we are blending between surfaces that nearly agree already.

We define the $\alpha_c$ functions so that they map $D_c$ to a unit disk centered at the origin. The term *chart* refers to $D_c$, $\alpha_c$, and the unit disk. The embedding functions are polynomials; we cap the degree between four and seven. The blend functions are $C^k$ B-spline basis functions "rotated" around the origin [BBB87]; we could use a $C^\infty$ function if desired [YZ04]. This produces a "bump" which is one at the origin and goes to zero (along with the first $k$ derivatives) by the boundary of the disk.

The $\alpha_c = M_W \circ M_D$ consists of a domain $D$-dependent map $M_D$ from $D$ to the plane, followed by a rotation and scale $M_W$ (see Figure 3). Both $M_D$ and $M_W$ must be invertible over the area of interest. For the sphere, $M_D$ is a rotation of the sphere, followed by a stereographic projection [Gri05]. The rotation places the center of $D_c$ at the north pole and the projection then "unfolds" the sphere into the plane, taking the south pole to infinity. The rotation and scale are used to adjust the size and shape of $D_c$.

For $n$-holed tori, the domain is a $4n$-sided polygon in the hyperbolic plane [Gri04]. This domain simplifies to the tiled Euclidean plane when $n = 1$. $M_D$ is a Linear Fractional Transform that takes the center of $D_c$ to the origin. Let $p = r(\cos\theta + i\sin\theta)$ be the center of the disk. Then the centering transform is:

$$T(p) = \begin{bmatrix} \cos-\theta + i\sin-\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -r \\ -r & 1 \end{bmatrix} \qquad (2)$$

Again, $M_W$ adjusts the size and orientation of $D_c$. The charts can be scaled until they begin to "wrap" around the handles.

## 5. Local neighborhoods

Our eventual goal is to embed the data points in the appropriate domain. A good embedding preserves the local sur-

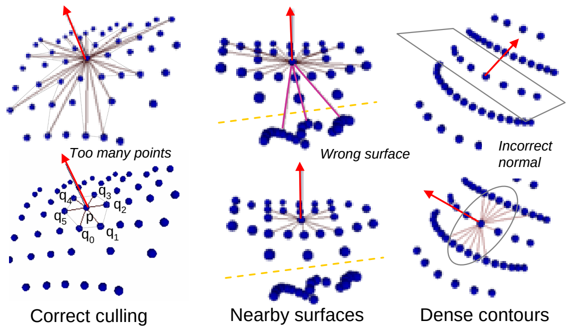

**Figure 4:** *Examples of incorrect (top) and correct (bottom) local neighborhoods and normals.*

face structure — *i.e.,* if the points $q_i$ form a ring around $p$ on the surface, then they should form a similar ring after being embedded. For meshes, ensuring that the orientation of the one-ring around each point is preserved (no folding) ensures that the global embedding is correct. Our goal here is to create the equivalent of a one-ring structure (the *local neighborhood*) around each data point. These local one-ring structures, when combined together, form a graph over the data points which we then use to embed the graph in a manner very similar to the mesh-based approach.

Essentially, the local neighborhood $q_i$ of a point $p$ should consist of an angle-ordered subset of the nearby points that tightly surround $p$. The $q_i$, projected onto the tangent plane at $p$, should form a polygon enclosing $p$. Moreover, other data points should project *outside* of the polygon. For regularly sampled data the $q_i$ will usually be the Delauney neighbors of $p$. Note that there are many possible polygons, depending on the normal orientation, and even for a fixed normal the polygon is not necessarily unique. the within Figure 4 shows some examples.

We first discuss normal estimation, then give the algorithm for finding the $q_i$ given a normal. We use the Singular Value Decomposition (SVD) with an adaptive neighborhood size to find the tangent plane and normal for the majority of points; however, this can fail catastrophically in areas of both high curvature and irregular sampling (Figure 4, right). This is because the best-fit plane will be one that is perpendicular to the desired tangent plane. In these cases, we use a more expensive algorithm to find a normal who's local neighborhood correctly spans the contours.

**Initial normal, SVD:** We use a combination of Mitra's [MN03] and Tang's [TM99] techniques to find the best set of neighbors $k$ from which to estimate the normal using SVD. The best $k$ depends on the sampling density, curvature, and noise [MN03,DS05]. Flat, noisey, or poorly-sampled areas require a bigger $k$, but larger values of $k$ perform poorly in areas of high curvature. We search for the $k \in [6,50]$ that

```
Calculate normal $\hat{N}$ using SVD
Calculate local neighborhood $q_i$
Calculate normal $\hat{N}_{q_i}$ from $p$ and $q_i$
    Angle-weighted average of triangle normals $q_i, p, q_{i+1}$
If $\hat{N}$ and $\hat{N}_{q_i}$ agree, done, use $\hat{N}$, $q_i$
Else try 20 evenly distributed normals $\hat{N}_i$
    Calculate $q_i$, $\hat{N}_{q_i}$ for $\hat{N}_j$
    Re-calculate $q'_i$ and $\hat{N}'_{q_i}$ from $\hat{N}_{q_i}$ and rate $q_i$
Return best $\hat{N}'_{q_i}$, $q'_i$
```

**Figure 5:** *Calculating the local neighborhood.*

minimizes:

$$E = \frac{\lambda_1 - \lambda_2}{(\lambda_1 + \lambda_2)} + \frac{\lambda_3}{(\lambda_1 + \lambda_2)} \qquad (3)$$

where the $\lambda_i$ are the Eigenvalues, sorted in descending order, of the $k \times 3$ matrix made from $q_i - p$, where the $q_i$ are the $k$ closest points. The first term measures how circular the $k$ set is, the second term the planarity. The normal $\hat{N}$ is the third Eigenvector.

Once we have a normal $\hat{N}$ we use it to calculate the local neighborhood, and from that the normal $\hat{N}_{q_i}$ of the local neighborhood. If $\hat{N}$ and $\hat{N}_{q_i}$ are similar ( $\hat{N} \cdot \hat{N}_{q_i} > 0.9$ ) then the SVD normal is assumed to be correct. If $\hat{N}$ and $\hat{N}_{q_i}$ do not agree, then the data is either anisotropically sampled, near a sharp feature or boundary, contains points from a disjoint area on the surface, or a combination of the above.

**Initial normal, search:** If the SVD normal check fails, then we try a set of twenty normals $\hat{N}_j$ evenly-distributed on the sphere, and pick the one that results in the best local neighborhood. Since the set $\hat{N}_j$ is sparse, we actually iterate once, replacing $\hat{N}_j$ with the local neighborhood normal $\hat{N}_{q_i}$. The evaluation function we use balances how well-distributed the $q_i$ are in the ring around $p$, and how well the $q_i$ approximate all of the nearby points (see Appendix A).

**Local neighborhood:** This is not necessarily unique and can be sparse (few $q_i \approx 8$) or dense. We use an ad-hoc algorithm which culls out points that are sufficiently out of the plane that they may belong to another part of the surface, and points that are "blocked" by closer ones. If exact, oriented normals are provided then we can cull out points with normals in the opposite direction as well. See Appendix A.

The normal $\hat{N}_{q_i}$ is the angle-weighted sum of the face angles $q_i, p, q_{i+1}$. We also detect if the point $p$ lies on an edge by the fact that the face normals cluster into two distinct (greater than 30 degrees) directions; in this case, we set $\hat{N}_{q_i}$ to be the average of the two face-normal cluster directions.

Statistics of this algorithm are given in Table 1, and a more complete analysis in the supplemental materials. In particular, it should be noted that *any* algorithm works on simple data sets such as the bunny, even with irregular sampling and added noise. However, this is not the case for data sets with high curvature and convex regions.

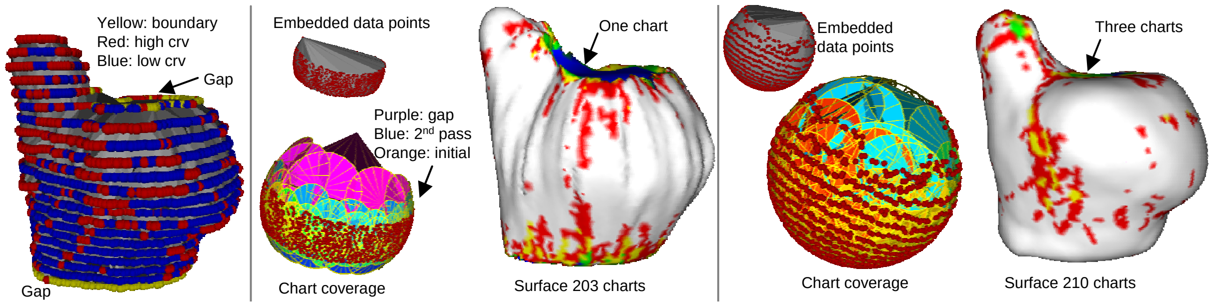

**Figure 6:** *Data points on left; notice missing region both on top and on bottom. Middle: Place points on the sphere, build mesh using QHull, cover with charts, embed surface. Right: Re-do embedding with updated neighborhood information from the first QHull mesh. Both surfaces use the same polynomial degree (max 4). Embedded surface color indicates coverage — blue is one chart, green two, yellow three, red four, and white more than four.*

| Data | # points | # Random | # Bdry | $k$ | Time |
|---|---|---|---|---|---|
| Bones | 49701 | 847 | 134 | 27.59 | 0.99 |
| Bunny | 34834 | 2466 | 90 | 20.88 | 0.66 |
| Garg | 250003 | 40630 | 6 | 18.61 | 0.73 |
| Heart | 3631 | 485 | 25 | 20.65 | 0.69 |
| Vase | 1476 | 474 | 0 | 10.74 | 0.99 |

**Table 1:** *Total number of points, number where $\hat{N}$ and $\hat{N}_{q_i}$ disagree, and the number of boundary points (no complete neighbor set). $k$ is the SVD neighborhood size. Timings are per data point, in milliseconds, Pent.-M 1.5GHz, 768MB Ram.*

## 6. Embedding the data points

In this section we assign a location, $\mathbb{D}(p)$, in the domain $D$ for each point $p$. This is equivalent to the parameterization assignment in spline fitting, except that we do not assign points on a per-patch basis. Instead, each chart embedding will determine which set of points (and parameter values to use) based on the chart function $\alpha_c$ (Section 7.4).

The goal of the embedding function is to distribute the data points evenly in the domain, respecting the local neighborhood connectivity. Once an initial set of locations is found, and an approximate surface constructed, we can further adjust the data point's domain locations by letting them "slide" around in $D$. This is analogous to the usual spline fitting process of iterating between fitting with fixed parameter points, and re-projecting to adjust the parameter values.

Finding the initial locations is domain dependent; we first give the algorithm for the sphere, then the $n$-holed torus.

**Sphere:** We use Saba's spherical embedding [SYGS05], modified to take our local neighborhood data, to initially place the points on the sphere. We use edge-length weights, which try to preserve distances to neighbors. Once the points are placed on the sphere we use QHull [BDH96] to build a water-tight mesh on the embedded data (see Figure 6). By moving the mesh vertices to the original 3D locations, we get a mesh that interpolates the original data points (see Figure 12).

Gaps in the original data tend to produce large, uncovered areas in the domain (see Figure 6). While the chart construction process (Section 7) will correctly cover them, the placement is poorly distributed. These large areas also introduce inaccuracies when interpolating (Section 7.4). The gaps arise because any data point with partial neighbors gets "pulled" in only one direction.

To better distribute the data points we use the connectivity of this first mesh to update the local neighborhood of the boundary points. Essentially, we re-run the local neighborhood process, this time adding in points that are also close as measured by graph distance on the QHull mesh. If a non-boundary point's neighborhood radius changes dramatically we mark it as being on a boundary. Re-running the spherical embedding with the new neighborhood information gives a more uniform distribution of points (right of Figure 6).

***N*-holed torus:** To embed an $n$-holed torus we first find a system of loops [EW05] in the surface, then place these loop lines at the boundaries of our $4n$-holed polygon (see Figure 7). We can then adjust the parameterization by moving each point towards the center of its neighbors, being careful to use the closest copy in Hyperbolic, not Euclidean, space. Unlike the sphere case, this iterative approach is very stable and quickly converges.

Unfortunately, all of the $n$-holed parameterization approaches require a manifold mesh [DS95]. To get around this, we use an intermediate implicit surface construction step [Ju04]. This algorithm was modified to a) return a single, connected component, b) ensure the mesh is manifold and c) return a list of which vertices were in the original data set. The input to this algorithm is the local neighborhood disks. The output is a mesh that contains a subset of original points plus additional points introduced at oct-tree boundaries.

We embed the implicit mesh in the domain, which gives us the domain locations of some of the original points. To solve for the remaining points, we fix the locations of the known points in the domain, then solve for the remaining locations by placing each unpinned point in the center of its known neighbors.

**Adjusting the locations:** is a straight-forward optimization step where we allow the point to slide in the domain until it minimizes the projection error of an overlapping chart embedding — essentially moving the point until the error is in the surface normal direction.

## 7. Making charts

In this section we specify the $\alpha_c$ chart functions and fit them, via the embedding functions $E_c$, to the data points. The goal is to cover the domain with charts, without having any chart get too "big" — we want to bound the number of data points each chart covers. This lets us use lower-order polynomials for the $E_c$ functions.

The user provides three parameters: the number of data points each chart should overlap, the chart spacing, and the maximum amount of total curvature allowed per chart. The more the charts overlap, the smoother the result.

### 7.1. Placing chart centers

The goal of the chart center algorithm is to place the centers at equally spaced distances along the surface (see left of figure 9). In places with high curvature we place the centers closer together.

The input to the algorithm is the desired geodesic radial spacing $r$ and a maximum curvature $C$. $r$ can be calculated from a desired number of points $G_n$ using an estimate [MN03] of the sampling density $\rho$.

$$r \;=\; s_r\sqrt{G_n/(\pi\rho)} \qquad (4)$$

For the figures in this paper we set $s_r$ to be 0.6; this results in charts that overlap substantially.

To place a new chart center, we look for a data point which is not within $r$ of an existing center and is at a distance $2r$ from as many existing centers as possible. To keep the algorithm efficient, we only look in an expanding front around the current set of centers. We seed the algorithm with data points on the boundary and points with high curvature.

When a center is selected it marks all of the points within the distance $r$ (in graph distance) as covered using breadth-first search across the local neighborhood graph. The marking process is terminated early if the total accumulated curvature is greater than the allowable curvature. Our curvature measure is the second half of Equation 3; therefore it measures curvature on cylinders as well as peaks and valleys. It is also area-normalized; a bigger value should be used only if the data set has noise.

Once the chart centers are selected we assign each data point to its closest center. We then construct a group adjacency structure, similar to a Delauney triangulation, by marking two centers as adjacent if they share a boundary. This information is used to place charts in the domain and control their size.

### 7.2. The chart functions

For each chart center $p_c$ we create a chart centered at the domain location $\mathbb{D}(p_c)$. To determine the orientation and initial $xy$ scaling of the chart, we map all of $c$'s neighbor's centers to the plane using $M_D$. We apply Singular Value Decomposition to find the rotation and (non-uniform) scale $s_x, s_y$ that best maps the neighbors to the boundary of the unit disk. Next, we adjust the overall scale until the chart covers the specified number of points, scaling down if the total accumulated curvature is bigger than the specified bound.

Once an initial set of charts is defined we repeat the chart center algorithm, this time pre-marking as covered any points already in a chart. All existing chart centers are used when finding the chart adjacency structure. We repeat making charts and finding chart centers until all of the data points are covered. Typically, one to three iterations suffices.

We consider a point to be covered if it is significantly in the interior of some chart. More specifically, a domain point $\mathbb{D}(p)$ is only considered well-covered if there is a chart $c$ such that $\alpha_c(\mathbb{D}(p)) - (0,0) < 0.9$.

### 7.3. Filling in gaps

The previous process ensures that every data point $\mathbb{D}(p)$ is covered by at least one chart. As a final step we ensure that the entire domain $D$ is covered, adding in the biggest possible uniformly-scaled chart that covers any missing regions (the purple charts in Figure 6, middle). We let the chart center move as well as change scale, moving the center towards the maximally uncovered area.

**Guaranteeing coverage:** An uncovered region must be adjacent to either a) a chart that overlaps no other charts or b) the intersection of two chart boundaries. Sampling the boundary of each chart a few times finds the first case, and sampling pairs of charts for intersections finds the second case.

### 7.4. Fitting charts

Although mathematical smoothness is guaranteed, *visual* smoothness is a function of both the smoothness of the individual embedding functions and how well they agree where they overlap. Although it is possible to globally fit all of the functions at once, this can be computationally expensive. We prefer, instead, to fit each function individually, relying on shared data point constraints to enforce similarity. If the data distribution is dense and uniformly sampled then a straight-forward least-square approach ($Ax = b$, $x = [x_{ij,0\leq i,j<K}]^T$

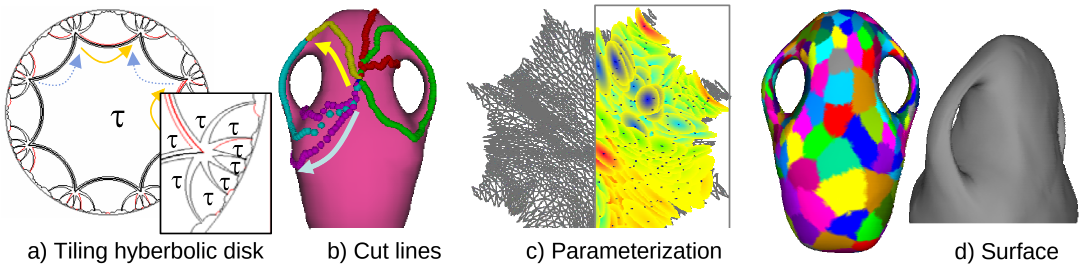

**Figure 7:** *2-holed tori. a) Tiling the Hyberbolic disk with an 8-sided polygon. We keep the center and adjacent copies of τ. b) Cut lines on the data set; yellow and blue arrows match arrows in τ. c) The mesh after cutting and relaxation (70 iterations). Chart coverage, right: blue dots are chart centers, colors indicate chart order. d) The reconstructed surface, colored by chart.*

the coefficients of the $K$th order polynomial) suffices. Let $\{p_i : ||\alpha_c(\mathbb{D}(p_i)) - (0,0)|| < 1.2\}_c$ be the set of data points that map to chart $c$, where we take points just outside of the chart as well. Let $(s,t) = \alpha_c(\mathbb{D}(p))$. We define each row of the matrix $Ax = b$ by

$$[\cdots s^i t^j \cdots][\cdots x_{ij} \cdots]^T = [p] \tag{5}$$

To choose $K$, we find the smallest $K$ such that $\sum_{p\in\{p_c\}} ||E_c(\alpha_c(\mathbb{D}(p))) - p|| < \varepsilon$. We set ε to be some fraction (1.2) of the average of the neighborhood edge lengths in the set $\{p\}_c$.

## 7.5. Non-uniform data

In the case of non-uniformly sampled data we modify Equation 5 to account for both unevenly distributed data points and to interpolate across areas with no data. Contour samples present particular problems because the surface is free to undulate between rows of samples; naively adding additional samples between contour rows can actually exacerbate this problem.

The least-squares approach to surface fitting works best if the data points are evenly distributed in the domain. Rather than resample everywhere, we weight the original samples by their local density, and only resort to interpolated samples where there is no data. More specifically, we place a grid over the unit circle and count the number of samples that fall into each gird cell. If there are $n$ samples in a grid cell, we weight each of those samples by $1/n$. Any empty grid cell is filled with an interpolated sample. The grid we use is actually one cell bigger on all sides than the unit circle and has the corner points culled (see Figure 8). The number of grid cells should be at least as big as the maximum number of allowable coefficients. We used a $(K+2)\times(K+2)$ grid spacing for a maximum polynomial order of $K$.

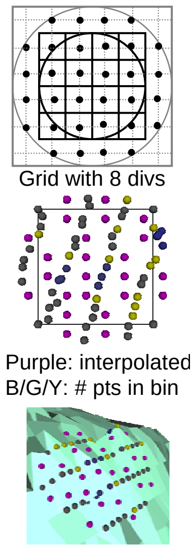

**Figure 8:** *Binning data. Bins with centers inside the big circle are kept. Small circle is unit disk. Colors indicate weights.*

## 7.6. Interpolating data

The interpolation function takes in a point on the domain and returns a point in $\mathbb{R}^3$. Currently, we are only using linear interpolation of three points. To find those points, we tessellate the domain using either the 3D convex hull (sphere) or a 2D Delauney triangulation, adding additional points along the boundary to ensure the polygon edges lie on the boundary ($n$-holed tori). This produces two meshes with identical topology, one of which approximates the surface. We project the domain point onto the mesh embedded in $D$ to get the face and barycentric coordinates, then use the corresponding 3D mesh vertex locations to compute the point's interpolated location.

## 8. Additional parameterizations

We are not limited to the parameterizations given in Section 4; we can use any invertible mapping of the sphere or Hyberbolic disk. For the texture mapping parameterizations of the heart we use Gnomonic mapping extended to convex polygons to create polygonal patches; a similar approach works in the $n$-holed tori case using a version of barycentric coordinates on the Klein-Beltrami model [Gri04]. The polygonal patches match the images projected onto the geometry; vertical stripes for the texture map images, square patches for the PDA images. We build blend functions in the same manner as before, but define the texturing or PDA function to be black where the domain is uncovered.

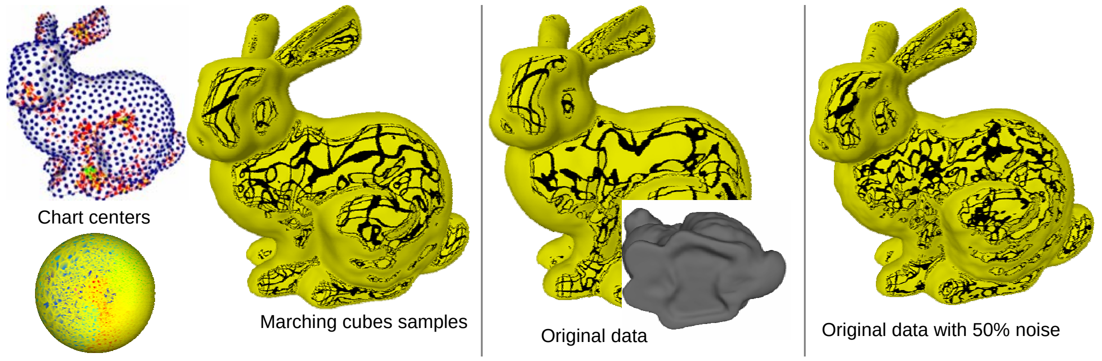

**Figure 9:** *Three different bunny data sets. For all three, the average chart covers 300 points, has maximum curvature 10, and maximum polynomial degree 6. Left: Marching cubes sampled from PolyMender, 45860 samples, 1501 charts. Upper left shows chart centers, lower left shows chart locations on the sphere. Middle: Original data, 34834 samples, 1163 charts. Right: Original data with random noise added to each data point. Amount of noise was between zero and 50 percent of the average edge length in the original zippered mesh.*

## 9. Implementation issues

To speed up the local neighborhood computations we build a KD-tree, which supports closest point searches in $O(m \log n)$ time, where $m$ is the number of neighbors and $n$ is the number of points [AMN*98]. We also use a KD-tree built from the face centers to use as a first guess when interpolating data points.

We make extensive use of the QHull library [BDH96] both to perform interpolation and to tessellate [Gri05] the embedding.

We keep a list of which charts overlap and evaluate Equation 1 starting with a point in a particular chart. To find the overlaps we map the boundary of chart $i$ into chart $j$ via $\alpha_j \circ \alpha_i^{-1}$ and check for intersections with the unit disk. We break the boundary into sections and check each section, recursively splitting it. We use the derivatives of the map [SB], to conservatively bound each section with a triangle.

**Hyperbolic domain:** The $n$-holed torus domain is the Hyperbolic disk, tiled with an infinite number of copies of the $4n$-sided polygon. In practice, we only work with the interior copy and all of the ones that are adjacent to it, mapping points back into the interior copy (see Figure 7).

## 10. Results and remarks

We have tested our surface reconstruction on a range of data sets: unstructured medical data (Figure 1), contour sets (Figure 10), dense laser scans of varying sizes (Figure 9, 12), Marching cubes samples (Figure 9) and data samples from smooth surfaces (Figure 7). Surface reconstruction ranges from under a minute (the heart) to several hours (the gargoyle). We used the default parameter values given throughout the paper, except for the three data-dependent parameters listed in Table 3.

| Data | LN | Embed | Charts | Fit | Eval | Total |
|---|---|---|---|---|---|---|
| Bones | 4.48s | 5m | 3m | 4m | 3m | 16m |
| Bunny | 2.30s | 48s | 70s | 7m | 1m | 10m |
| Garg | 18.3s | 123s | 1.5h | 15m | 2m | 2.5h |
| Heart | 0.25s | 3s | 10s | 3s | 2s | 20s |
| Vase | 0.15s | 45s | 70s | 10s | 5s | 2m |

**Table 2:** *Approx. timings, Pent-M 1.5GHz, 768 MB Ram.*

As an intermediate step this approach produces a watertight mesh of the correct topology. The metric used to triangulate the data, however, is only loosely based on the geometry of the original data. For dense, evenly sampled data sets the resulting mesh is surprisingly good (left of Figure 12) but the result is somewhat more noisy, although still acceptable, on other data.

Our current implementation errs on the side of many small charts with substantial overlap. We have also experimented with larger charts that overlap less. Results are visually similar for dense meshes and no noise, but unwanted undulations can arise in noisy data sets. This is primarily because charts make local decisions about how to approximate the data, and those decisions may not agree, leading to smooth, but noticable, changes between charts. Globally fitting the surface reduces this problem, but this is an expensive solution and introduces additional questions about what kinds of additional "agreement" constraints to add.

The algorithm is relatively insensitive to neighborhood sizes and polynomial degrees, with the exception of areas of high curvature. In this case, stretching a chart over the area can cause very poor behaviour, which is only partially mitigated by increasing the degree of the polynomial. These areas are better covered by a small number of charts, each with limited "bending" needed.

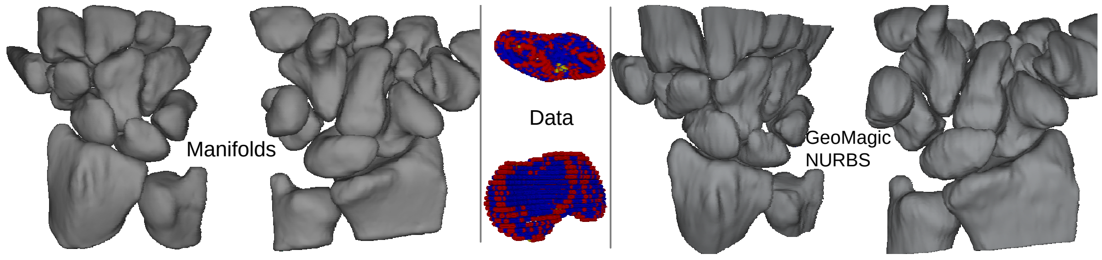

**Figure 10:** *The bones of the hand constructed from CT scans. Left: Our reconstruction. Middle: Example data showing contour structure. Right, NURBS surfaces constructed using commercial software (Geomagic).*

| Data | Pts | Crv | Deg | Charts | Overlap | Fit |
|---|---|---|---|---|---|---|
| Bones | 100 | 12 | 4 | 3009 | 5.14 | 0.22/0.21 |
| Bunny | 300 | 10 | 6 | 1164 | 5.85 | 0.21/0.23 |
| Garg | 500 | 25 | 8 | 3505 | 4.60 | 0.61/0.59 |
| Heart | 100 | 15 | 4 | 224 | 6.17 | 0.31/0.30 |
| Vase | 20 | 15 | 4 | 355 | 4.82 | 0.19/0.18 |

**Table 3:** *Surface information. Points, curvature, and degree are input parameters. Overlap is the average number of charts overlapping each data point. Fit is the average fit per chart, followed by the total fit for the surface, given as a percentage of the average edge neighbor length.*

## Appendix A: Local neighborhood

Refer to Figure 11. The $q_i$ are the local neighborhood, the $d_k$ all points in the $k$ neighbor set and $l_q = \max ||q_i - p||$. Project the $d_k$ and $q_i$ into the tangent plane at $p$. Let $\perp d$ and $\angle d$ be the distance to and angle in the tangent plane, respectively. Sort the $q_i$ by angle. For every point $d_k$ find the pair of points $q_{i_k}$ and $q_{i_{k+1}}$ that bracket $d_k$. The error metric for that neighborhood is:

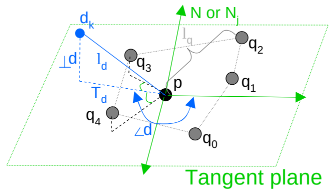

**Figure 11:** *Labeling for local neighborhood.*

$$l_{i_k} = ||q_{i_k} - p|| \quad t = \frac{\angle d_k - \angle q_{i_k}}{\angle q_{i_k+1} - \angle q_{i_k}} \tag{6}$$

$$E_N = \sum_{d_k} \perp d_k \min\left(1, \frac{(1-t)l_{i_k} + t l_{i_k+1}}{||d_k - p||}\right) \tag{7}$$

To find the $q_i$, bin the $d_k$ by $\angle d$, keeping only the closest point for each bin (32 divisions). Let $d_l$ and $d_r$ be the points bracketing $d_k$, and $\gamma$ the angle between $d_l$ and $d_r$. We cull $d_k$ if:

**Plane culling:** The angle $\gamma$ is small and the distance $\perp d$ relative to $T_d$ is big. Specifically, $\gamma < 0.6\pi$ and $\perp d/T_d > 0.75$, or $\gamma < 0.3\pi$ and $\perp d/T_d > 0.5$, or $\gamma < 0.2\pi$ and $\perp d/T_d > 0.3$.

**Blocked culling:** If $\gamma > \pi/2$ then $d_k$ is not culled. If $d_k$ is very close to $d_l$ or $d_r$ (within $\pi/16$) and farther away, it is culled. If $d_k$ is twice as far away as both $d_l$ and $d_r$, it is culled. If $\gamma < 0.25\pi$ and $d_k$ is farther away than $d_l$ and 1.5 farther than $d_r$ (or vice-versa) then it is culled. Increasing the allowable angles (the $\gamma$s) results in a more sparse set of neighbors.

The culling order is found by randomly walking through the current set of points, testing each. If a point meets the criteria then it is culled and the random walk re-started.

## References

[AG04] Akkouche S., Galin E.: Implicit surface reconstruction from contours. *The Visual Computer 20*, 6 (2004), 392–401.

[AMN*98] Arya S., Mount D. M., Netanyahu N. S., Silverman R., Wu A. Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM 45*, 6 (1998), 891–923.

[Att97] Attali D.: r-regular shape reconstruction from unorganized points. In *SCG '97: Proc. symp. on Computational geometry* (1997), ACM Press, pp. 248–253.

[BBB87] Bartels R., Beatty J., Barsky B.: *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[BBCS97] Bernardini F., Bajaj C. L., Chen J., Schikore D. R.: A triangulation-based object reconstruction method. In *SCG '97: Proc. symp. on Computational geometry* (1997), ACM Press, pp. 481–484.

[BDH96] Barber C. B., Dobkin D. P., Huhdanpaa H. T.: The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software 22*, 4 (Dec 1996), 469–483. http://www.qhull.org.

[BMR*99] Bernardini F., Mittleman J., Rushmeier H., Silva C., Taubin G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics 5*, 4 (/1999), 349–359.

[Boi84] Boissonnat J.-D.: Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3*, 4 (1984), 266–286.

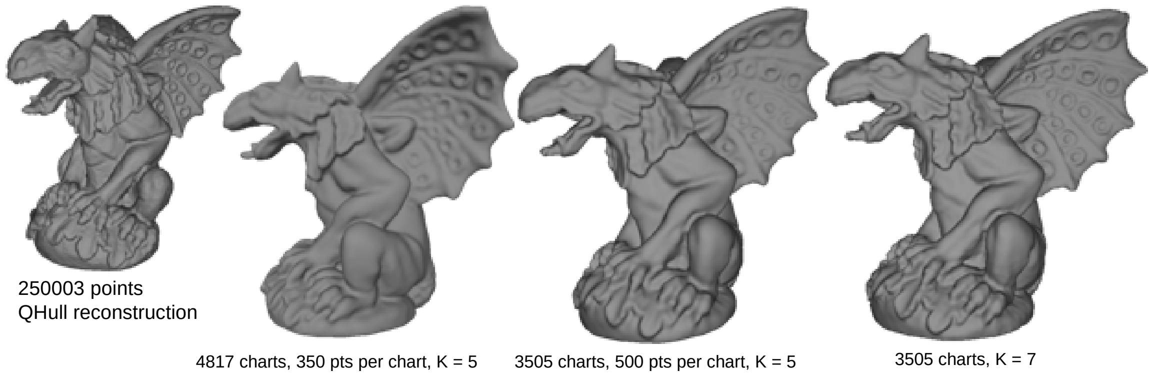
250003 points
QHull reconstruction
4817 charts, 350 pts per chart, K = 5
3505 charts, 500 pts per chart, K = 5
3505 charts, K = 7

**Figure 12:** *Gargoyle data set, 250003 points, 3505 charts. Upper left: Mesh reconstructed using QHull (the original ply file was corrupted and only had a partial set of faces). Middle: Reconstruction using maximum 5 degree polynomials. Right: Same charts, but with maximum degree 7.*

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), pp. 67–76.

[CDR05] CHENG S.-W., DEY T. K., RAMOS E. A.: Manifold reconstruction from point samples. *Proc. ACM-SIAM Sympos. Discrete Algorithms* (2005).

[CP99] CONG G., PARVIN B.: An algebraic solution to surface recovery from cross-sectional contours. *Graphical Models and Image Processing 61*, 4 (July 1999), 222–243.

[DS95] DEY T. K., SCHIPPER H.: A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete and Computational Geometry 14* (1995), 93–110.

[DS05] DEY T. K., SUN J.: *Normal and Feature Estimations from Noisy Point Clouds*. Tech. Rep. OSU-CISRC-7/50-TR50, Ohio State, July 2005.

[EH96] ECK M., HOPPE H.: Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *Proceedings of SIGGRAPH 96* (Aug. 1996), pp. 325–334.

[EW05] ERICKSON J., WHITTLESEY K.: Greedy optimal homotopy and homology generators. In *SODA* (2005), pp. 1038–1046.

[FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C. T.: Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3 (2005), 544–552.

[FR02] FUNKE S., RAMOS E. A.: Smooth-surface reconstruction in near-linear time. In *SODA '02: Proc. ACM-SIAM symp. on Discrete algorithms* (2002), SIAM, pp. 781–790.

[GLC02] GRIMM C., LAIDLAW D., CRISCO J.: Fitting manifold surfaces to 3d point clouds. *Journal of Biomechanical Engineering 124* (February 2002), 136–140.

[Gri04] GRIMM C.: Parameterization using manifolds. *International J. of Shape Modelling 10*, 1 (June 2004), 51–80.

[Gri05] GRIMM C.: Spherical manifolds for adaptive resolution surface modeling. In *Graphite* (Nov. 2005).

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proceedings of SIGGRAPH 92* (1992), vol. 26, pp. 71–78.

[HQ04] HE Y., QIN H.: Surface reconstruction with triangular b-splines. In *Geometric Modeling and Processing* (2004), p. 279.

[Ju04] JU T.: Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3 (2004), 888–895.

[MAVdF05] MEDEROS B., AMENTA N., VELHO L., DE FIGUEIREDO L. H.: Surface reconstruction for noisy point clouds. In *Third Eurographics Symposium on Geometry Processing* ( 2005), pp. 53–62.

[MN03] MITRA N. J., NGUYEN A.: Estimating surface normals in noisy point cloud data. In *SCG '03: Proc. of symp. on Computational geometry* (2003), pp. 322–328.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Transactions on Graphics 22*, 3 (2003), 463–470.

[PSV98] PONTIER S., SHARIAT B., VANDORPE D.: Implicit surface reconstruction from 2d ct scan sections. In *Computer Graphics International* (1998).

[SB] STAUNING O., BENDTSEN C.: http://www.imm.dtu.dk/nag/proj_km/fadbad/.

[SBS05] SCHALL O., BELYAEV A., SEIDEL H.-P.: Robust filtering of noisy scattered point data. In *Symposium on Point - Based Graphics 2005* (June 2005), pp. 71–78.

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. on Graphics 23*, 3 (Aug. 2004), 896–904.

[SYGS05] SABA S., YAVNEH I., GOTSMAN C., SHEFFER A.: Practical spherical embedding of manifold triangle meshes. *Shape Modelling International* (June 2005), 256–265.

[TM99] TANG C.-K., MEDIONI G.: Robust estimation of curvature information from noisy 3d data for shape description. In *IEEE Int'l Conf. Computer Vision* (1999), pp. 126–433.

[Geo] WWW.GEOMAGIC.COM: Geomagic commercial software.

[YDC05] YANG Z., DENG J., CHEN F.: Fitting unorganized point clouds with active implicit b-spline curves. *The Visual Computer 21*, 8-10 (2005), 831–839.

[GHQ05] GU, X., HE, Y., QIN, H.: Manifold Splines. *Solid and Physics Modeling*, 3 (2006).

[YZ04] YING L., ZORIN D.: A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. on Graphics 23*, 3 (Aug. 2004), 271–275.