Point Set Silhouettes via Local Reconstruction

Anonymous

Abstract

We present an algorithm to compute the silhouette set of a point cloud. Previous methods extract point set silhouettes by thresholding point normals, which can lead to simultaneous over- and under-detection of silhouettes. We argue that additional information such as surface curvature is necessary to resolve these issues. To this end, we develop a local reconstruction scheme using Gabriel and intrinsic Delaunay criteria and define point set silhouettes based on the notion of a *silhouette generating set*. The mesh *umbrellas*, or local reconstructions of one-ring triangles surrounding each point sample, generated by our method enable accurate silhouette identification near sharp features and close-by surface sheets, and provide the information necessary to detect other characteristic curves such as creases and boundaries. We show that these curves collectively provide a sparse and intuitive visualization of point cloud data.

1. Introduction

Point clouds acquired using laser scanners account for many of the digital 3D models in common use. However, despite much research, converting a point cloud into a quality mesh remains a difficult and costly process. This difficulty is one of the main motivations for developing geometry processing techniques which operate directly on points [20]. In this setting, even basic tasks such as rendering can be challenging, since points do not *a priori* contain normal or orientation information, and even determining which points are visible is non-trivial [26, 32]. Even with visibility resolved, displaying points without normal information can hide important surface features; see Figure 1 (left column). Methods that generate high-quality point-cloud renderings, such as splatting [40], rely on elaborate filtering especially near characteristic curves such as silhouettes.

Perception research demonstrates that rendering the characteristic curves of input models, and specifically their silhouettes, provides an effective visualization enhancement or even alternative to rendering the entire model [28]. Silhouettes are important visual cues for shape perception [28] and are very effective at conveying shapes [17, 21]. As such accurate rendering of point cloud silhouettes can enhance visualization of the cloud data in a concise yet effective way. In this paper, we provide a method for accurately and efficiently computing the silhouette set of a point cloud, *i.e.* the subset of points which best approximate the silhouette of the underlying surface. Such a set can be used to quickly and intuitively depict a point-based model (Figure 1) or adapted to other tasks that use silhouettes such as shadow volumes [9] and object tracking [35].

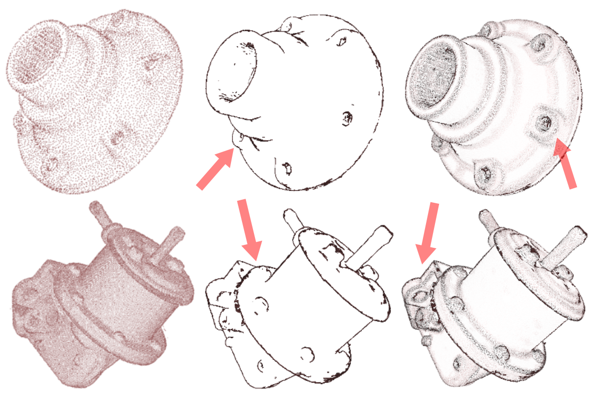


Figure 1: Surface features in raw point clouds are difficult to visualize, even with visibility resolved (left). By rendering the point set silhouettes (middle), and especially the detected sharp features (right), geometric details of the underlying shapes are better revealed.

According to the standard definition of silhouettes for a smooth surface, a point p is on the silhouette if its normal is perpendicular to the view vector at p. On a point cloud where only disconnected points are available, Zakaria and Seidel [39] proposed using *normal thresholding*, defining a point p to be on the silhouette if the scalar product between the point normal at p and the view vector is sufficiently close to zero. However, as shown in Figure 2, this method can simultaneously under- and over-detect silhouettes, introducing thick point patches instead of narrow silhouettes in low-curvature regions

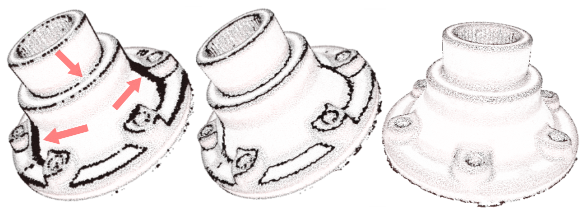


Figure 2: Normal thresholding (left) can over- and under-detect point set silhouettes. Results using our method (middle) based on SGS and local reconstruction show visible improvement on silhouette accuracy. An important note is that the camera view which produced (a) and (b) were chosen to best reveal the silhouette set returned. This view is different from the view (right) that generated the silhouettes.

and leaving gaps in high-curvature regions, especially near sharp features. Both errors reduce the utility of the resulting silhouette sets. In general, it is not possible to choose a suitable fixed threshold to simultaneously correct both types of errors.

Our point set silhouette construction algorithm is driven an alternative characterization of geometric silhouettes that allows a unified treatment of silhouettes for different forms of surface models. In general, we associate with a point p on a surface model M a *silhouette-generating set* or *SGS* such that p is on the silhouette iff the viewpoint is contained in its SGS. Given appropriate selections for the SGS of a point or edge, this definition is equivalent to the traditional definition for smooth surfaces and polygon meshes. For example, on a smooth surface S, we identify the tangent plane at p as its SGS. For a mesh surface, the set difference between the union and the intersection of all the half-spaces defined by the normals of a vertex's *umbrella* (the triangles adjacent to the vertex) provides a natural SGS, though this is typically defined only on edges as a double wedge [2].

Inspired by this observation, we seek to approximate the set of planes tangent to a point p's intrinsic Voronoi cell on S (see Figure 4, which we argue describes the SGS of p. To compute the SGS of a point p in a point cloud P, we search for an optimal umbrella around it, performing a *local* reconstruction of the underlying surface. As shown in Figures 2 and 12, our construction leads to significantly more accurate silhouette extraction than normal thresholding. The local reconstruction also allows us to efficiently extract characteristic model features such as sharp edges to facilitate visualization of point clouds; see Figure 1.

Our local reconstruction algorithm is based on the assumption that the underlying surface S is a *piecewise smooth* manifold, which is smooth everywhere except at *feature curves* such as sharp edges or boundaries. For a non-feature sample point p we obtain an initial estimate of the unoriented surface normal using a simple and novel technique, and show that the error in this estimate is bounded by the sampling radius. We then construct an umbrella by performing Delaunay edge flips [14] on k nearest neighbours. Our algorithm is also able to identify feature curves in a small local neighbourhood. If p is at or near a feature, we construct a partial umbrella for each smooth patch involved. We focus on the construction of these umbrellas even in the presence of sharp features, and do not attempt to robustly estimate oriented surface normals.

In particular, we exploit the geometric insight we develop in Section 4 to identify sharp edges between smooth surface patches, as well as boundaries on the underlying sampled surface, *e.g.* in an incomplete scan of a real-world object. Many global surface-reconstruction algorithms in common use have difficulty with open manifolds or cannot reconstruct these features with high fidelity; we hope to provide a simple way to augment these methods. Our local measure performs well on samples taken from open manifolds and the produced results are shown to be comparable to those from more sophisticated methods such as PEEL [12].

Our umbrella construction does not require an oriented point cloud or extra sampling at feature curves and boundaries, though we do make lenient assumptions about the distribution of samples. We produce satisfactory results using a pair of independent parameters to indicate the density and uniformity of the sampling, using the same default values for both for all the examples in this paper. Our method is able to produce plausible local reconstructions, and thus accurate silhouettes, on inputs containing sharp features and close-by surface sheets, without resorting to expensive statistical techniques. Its formulation is based on a small number of nearest neighbours, making it well-suited to efficient implementation on architectures such as GPUs in the same manner as [27], heavily parallelized computers, out-of-core applications, and asymmetric processors, where the random global memory access required by many global reconstruction methods is difficult, costly, or impractical.

One theoretical limitation of our method is sensitivity to noise; however, we have successfully computed silhouettes on numerous noisy examples after denoising them using standard methods such as WLOP [23].

2. Related work

Silhouette extraction. Silhouette extraction for meshes is a well studied problem [25]. Image-based silhouette extraction methods, such as [10], are generally quite fast

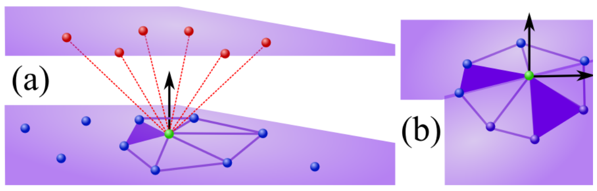


Figure 3: To generate the SGS of a point p (green) based on its k nearest neighbours, we (a) find a Gabriel triangle on p, discard neighbours distant from the triangle's plane (red), and build a Delaunay triangulation from the remainder (blue). If p lies on a sharp feature edge (b), we build separate umbrellas for each smooth patch on that edge.

at producing a set of silhouette pixels in the projection plane. Object space silhouettes [21, 33] provide additional surface information which can be used for stylized rendering and can be naturally combined with extraction and rendering of other features. In particular, by extracting the full silhouette rather than only its visible component, object-space algorithms facilitate applications such as shadow rendering [9] and collision detection [7]. When operating in object space, Glisse *et al.* show in [16] that the expected size of a mesh silhouette set under lenient assumptions is $O(\sqrt{n})$, where n is the size of the mesh. An empirical study by McGuire [30] places the average silhouette size of downloaded meshes closer to $O(n^{0.8})$. Many object-space algorithms can extract the silhouette in sublinear time.

For point clouds, Xu *et al.* [38] develop an image-based method using Painter's algorithm, relying on depth discontinuities to render visible silhouette pixels. Zakaria and Seidel [39] present a hybrid approach which identifies silhouette points using normal thresholding, renders them to a frame buffer, and extracts curves from the result using a thinning process. As noted earlier, normal thresholding has difficulty in regions of very high or very low curvature. In contrast to both, our method operates solely in object space, computing silhouettes as well as other characteristic curves. While our computation is slower than image-based techniques, the local reconstruction which takes most of the runtime can be performed as a preprocessing step, with the actual silhouette extraction done in real-time.

Local neighborhoods in point clouds. Many works seek to build local characterizations of the underlying surface, S, represented by a point cloud. A common technique is to derive curvature and normal information for a sample point p by principal component analysis on the covariance matrix of $Q_k(p)$, the k Euclidean-nearest neighbours of p [22, 34]. The method is subject to artifacts around close-by surfaces and sharp features. Moreover, the information computed is not sufficient for our needs. An alternative is to construct a local analytic representation of S by means of moving least squares [3] or locally optimal projection (LOP) [29], which provide a means of projecting points onto a well-defined surface. However, the problem of deciding which sample points belong to the silhouette remains. For our purpose, these two algorithms can be seen as preprocessing steps to handle noisy input data.

Many surface reconstruction algorithms characterize local neighborhoods by finding a partial or complete umbrella around surface points, *e.g.* [1, 27]. Producing a full reconstructed surface is unnecessarily complex for our purposes. For example, while methods such as Cocone [11] and T-coordinates [6] produce umbrellas similar to ours, their definition in terms of a global Voronoi diagram is at odds with our desire to restrict our domain to a small set of $Q_k(p)$. In Appendix A we show that the canonical Gabriel triangle, which anchors our reconstruction, is in fact a Cocone triangle.

Furthermore, the umbrella-construction component of these methods is designed for smooth surfaces and tends to break down in the presence of sharp features or close-by surface components – that is, samples in $Q_k(p)$ that are geodesically distant from p on the underlying surface. Our construction is specifically designed to handle such scenarios correctly. While other methods exist for detecting close-by surfaces [23], sharp features [15], and boundaries [12], we integrate these capabilities into the core of our algorithm rather than adapting an existing method as a separate step.

One way to characterize the natural neighbours of p on S is to project $Q_k(p)$ onto an estimate of the tangent plane at p, perform a planar Delaunay triangulation of the projected points, and project back onto S [19, 5]. Our method uses similar ideas, but introduces a filtering step that enables us to deal with sharp features and close-by surfaces. We then create an initial triangulation in which all the filtered neighbours share an edge with p, a technique also employed in [18], but instead of working with a projected planar triangulation, we perform Delaunay edge flips directly on the embedded mesh fragment [14].

3. Silhouettes and silhouette-generating sets

While silhouette sets are well-defined on polygon meshes and smooth surfaces as structures that separate front- and back-facing regions, it is difficult to extend these definitions to point clouds. Instead, we can define them as the set of points on the model whose tangent plane contains the view point. On smooth surfaces, this

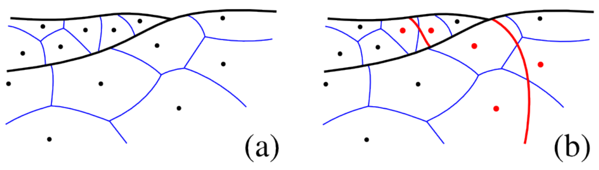


Figure 4: Finding point samples on a surface's silhouette. (a): Point samples on an underlying smooth surface S and their intrinsic Voronoi cells. (b): A silhouette curve on S. The points whose Voronoi cells are crossed by the curve (highlighted) are on the silhouette.

silhouette-generating set or *SGS* is identical to the standard definition, and it can readily be extended to both meshes and point clouds.

Using this definition on polygon meshes, the SGS of an edge is described by the set difference between the union of the positive half-spaces defined by the faces adjacent to the edge and their intersection [2]. This region is equivalent to the double-wedge described by the usual definition of edge silhouettes. Note that if the mesh in question is not closed, the silhouette status of edges on its boundary must be specified explicitly under the standard definition; typically they are either always considered to be on the silhouette or considered as silhouette edges when their adjacent face is front-facing [8]. Both of these options are subsumed within the SGS definition, either by including an implied back-facing face ("always-on") or simply by treating the face's half-space normally ("front-facing").

For a mesh vertex the SGS is defined similarly using the planes of its umbrella triangles. A mesh vertex is on the silhouette only when at least one of its adjacent edges is on the silhouette; thus, the SGS of the vertex is the union of the SGSes of its adjacent edges.

Intuitively, to define the SGS of a point in a cloud we want to construct a local umbrella around it which approximates the underlying surface on which it was sampled. More formally, in order to define the SGS of p, we consider its relationship to the surface S that it samples. We assume that all points in P are on S; this in turn induces an intrinsic Voronoi diagram on S from the point samples. This gives us an intuition for point-cloud silhouettes: a point $p \in P$ should be on the silhouette when the silhouette curve on S passes through p's Voronoi cell; see Figure 4. Therefore, the exact SGS of p is the union of all planes tangent to points on S within p's Voronoi cell. Note here that we do not need the Voronoi cell itself. As it is impractical to construct the exact SGS of p based on this definition, we next present an approximate construction and show that it leads to high-quality silhouette extraction.

4. Local neighbourhood construction

To approximate the SGS of a point p in a point cloud P according to the definition presented in Section 3, we build an intrinsic Delaunay triangulation [14] from a subset of p's k-nearest neighbours $Q_k(p)$. We need not compute a full triangulation; any points in $Q_k(p)$ not in p's Delaunay one-ring will not affect the SGS of p in the local reconstruction and can be ignored. The supporting planes of these triangles approximate the tangent planes of the intrinsic Voronoi cell on the underlying piecewise-smooth surface S containing p, and thus describe p's SGS.

Our local triangulations are constructed in a series of four steps, each taking advantage of information obtained from the previous steps to produce a more accurate umbrella around p. We perform the following operations:

1. Normal estimation and neighbour filtering
2. Initial umbrella creation and boundary identification
3. Neighbour-based multi-umbrella creation
4. Boundary consistency enforcement

A high-level overview of the algorithm is given in pseudocode in Figure 5. Next we describe each of these steps in detail.

4.1. Initial normal estimation and neighbour filtering

In the simplest case, p lies within a smooth region of S and the intrinsic Delaunay umbrella of $Q_k(p)$ will produce an appropriate local reconstruction. However, when some members of $Q_k(p)$ are *not* in the same region of S – if they lie across a sharp feature edge, or on a close-by surface sheet – we must exclude them from our triangulation. Our first tool to achieve this is local normal estimation.

We consider first the simplest case, where p lies on a smooth region of S, relatively far from any feature curves. Let t be a triangle on p with normal n_t, and let n_p be the normal to S at p. The acute angle between the lines generated by n_t and n_p is bounded by $O(r_t/\rho_f(p))$, where r_t is the circumradius of t, and $\rho_f(p)$ is the *local feature size* at p: that is, the distance to the medial axis of S [11, Lemma 3.5].

We say that a smooth region $U \subset S$ is *well sampled* if any point $x \in U$ is closer than $\varepsilon\rho_f(x)$ to the nearest sample point, where ε is an appropriately small constant. The function $\rho(x) = \varepsilon\rho_f(x)$ is the *sampling radius*.

To estimate the local normal at p, we find the triangle with the smallest circumradius amongst those that have

```
BuildPointUmbrellas(point-cloud $P$)
    CreateOnerings($P$)
    FindMultiumbrellas($P$)
    EnforceBoundaries($P$)

CreateOnerings(point-cloud $P$)
    for $p \in P$ do
        if is-untrustworthy($t_G(p)$, $\omega$) then
            continue
        end if
        $p$.nbrs = good-nbrs($Q_k(p)$, $t_G(p)$.norm, $\omega$, $\omega_t$)
        $p$.nbrs += marginal-nbrs($Q_k(p)$, $t_G(p)$.norm, $\omega$, $\omega_t$)
        $p$.onerings += BuildOnering($p$.nbrs, $t_G(p)$.norm)
    end for

BuldOnering(point[] nbrs, vec norm)
    onering = sort-by-angle(nbrs, norm)
    onering = delaunay-edge-flip(onering, $\gamma$)
    onering.boundary = find-boundary(onering, $\varphi$)
    return(onering)

FindMultiumbrellas(point-cloud $P$)
    for $p \in P$ do
        if has-complete-onering($p$) then
            continue
        end if
        while has-open-boundary($p$) do
            $t$ = get-trusted-nbr($Q_k(p)$ $p$.onerings)
            $p$.nbrs = good-nbrs($Q_k(p)$, $t$.norm)
            $p$.nbrs += marginal-nbrs($Q_k(p)$, $t$.norm)
            $p$.onerings += BuildOnering(nbrs, $t$.norm)
        end while
    end for

EnforceBoundaries(point-cloud $P$)
    for $p \in P$ do
        if incompat-full-onering($p$) then
            $p$.onering = remove-nonrecip-edges($p$)
            $p$.onering.boundary = find-recip-boundary($p$)
        else if incompat-bdry-onering($p$) then
            $p$.onering = add-recip-edges($p$)
            $p$.onering.boundary = find-recip-boundary($p$)
        end if
        /* Most points will be ignored */
    end for
```

Figure 5: A pseudocode overview of our method.

both the point p and its nearest neighbour as vertices. This triangle, which we denote $t_G(p)$, is necessarily a *Gabriel triangle*; its smallest open circumball is empty of sample points [13, Lemma 4.12]. Since this canonical Gabriel triangle is the only one of interest to us, we take the liberty of referring to it as *the* Gabriel triangle. If q is the nearest neighbour to p, Mederos *et al.* [31] identify t_G as the triangle $[p,q,u]$ that has the largest (necessarily acute) angle at u. Appendix A shows that this triangle provides a good approximate normal in a well sampled smooth surface patch.

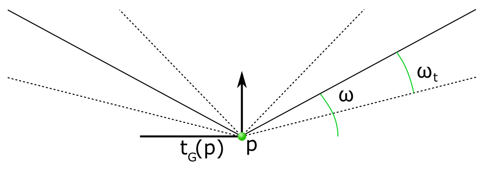


Figure 6: Angle bounds for neighbour filtering. Members of $Q_k(p)$ that fall within the wedge with half-angle ω_t are considered *marginal*, and must satisfy a distance constraint to be selected in the first pass.

If p lies in the interior of a well-sampled smooth surface patch, all samples in an umbrella on p should lie close to the tangent plane of p. The sampling radius ensures that $\|p-q\| = O(\varepsilon)\rho_f(p)$ for any neighbour q of p. The angle between $\overline{pq}$ and the tangent plane to p is $O(\varepsilon)$, and does not vary with the local feature size [11, Lemma 3.4]. Similarly, the angle between the tangent plane and the plane of t_G is $O(\varepsilon)$. This motivates our use of a constant angle threshold, ω, to filter the points in $Q_k(p)$ which are candidates for being neighbours on the same surface patch. This angle threshold reflects the implicit parameter ε governing the sampling radius.

Given the canonical Gabriel triangle, we discard all edges that form an angle greater than ω with the triangle's plane. In smooth areas of the surface this filtering discards most unrelated samples, such as those coming from close-by surface sheets.

When $Q_k(p)$ lies in a well-sampled region, filtering by ω produces a high-quality set of neighbours and is sufficient by itself. However, when $Q_k(p)$ does not conform to our assumed sampling density, we may inadvertently select points which are barely within the cone described by ω but geodesically distant. Further, if the surface curvature is high, we may also *reject* desirable neighbours which are barely outside of the ω cone. To address these cases, we introduce another parameter $\omega_t < \omega$ to describe *marginal* edges where our confidence in the ω criterion is weaker. If the angle between an edge pq and the plane of $t_G(p)$ falls within $[\omega-\omega_t, \omega+\omega_t]$, we accept q only when $|pq|$ is sufficiently small; we find that the condition $|pq| < \gamma r_{t_G(p)}$ works well in all of our examples, where γ is introduced in Section 4.3 as a parameter used to bound the circumradius of triangles in an umbrella.

Filtering is even more challenging in the presence of sharp feature edges, as shown in Figure 7; these edges can be seen as an extreme case of undersampling. First, even filtering by both ω and ω_t may keep points on surfaces across sharp (approximately right-angle) edges, requiring an extra filtering step (Section 4.2). Second, the Gabriel triangle *itself* may include samples on both

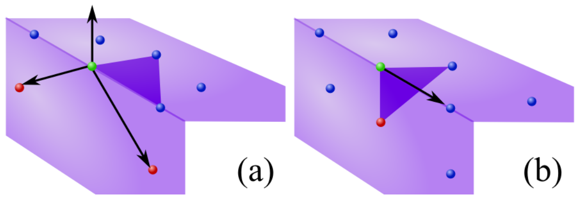


Figure 7: Filtering on the Gabriel normal at a point p (green) on a sharp edge may (a) include points on the opposite surface from the Gabriel triangle which pass the ω-test; also, (b) the Gabriel triangle itself may cross the edge.

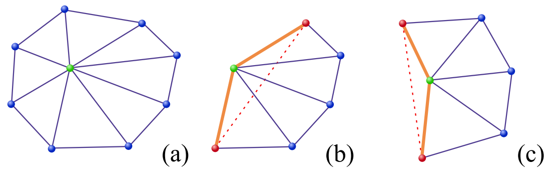


Figure 8: Boundary detection in the initial triangulation. Most initial triangulations have no boundaries (a). When the distribution of $Q_k(p)$ is severely biased, fold-overs may occur (b); these create *convex* boundaries (orange). Even if fold-overs do not occur, *concave* boundaries are created (c) when a triangle's angle on p exceeds φ. Dashed red lines are removed from the umbrella.

sides of the edge. In the latter case, only a small proportion of neighbours will pass the Gabriel normal filter; such points are dealt with in the second phase of our algorithm, described in Section 4.4. If we trust $t_G(p)$, we sort the remaining points by angle in its supporting plane and proceed as below.

4.2. Initial umbrella creation

At this stage we have an estimate of the relevant neighbours in $Q_k(p)$. The process of constructing an umbrella for p also drives our boundary detection. We work with the radial edges from p to its neighbours. We sort them in counterclockwise order according to their projection on the plane defined by t_G, thus defining an umbrella at p. See Figures 8 and 9(a) for examples.

We must account for the possibility that p itself lies on a boundary between surface patches. In this case our initial umbrella will be a *partial umbrella* on the patch containing the Gabriel normal. The construction of the remaining surface patch(es) is described in Section 4.4. Some boundaries are identified in the initial triangulation; this is described in Section 4.3.

This in turn lets us make a second, more aggressive pass on the remaining neighbours, as shown in Figure 9(a): we discard any edges whose *adjacent triangles' normals* form an angle greater than ω with the Gabriel normal. However, we do not remove an edge if the resulting triangle would also fail this criterion. (Removing an edge is akin to an edge flip, but we only preserve triangles incident to p.) We also avoid removing

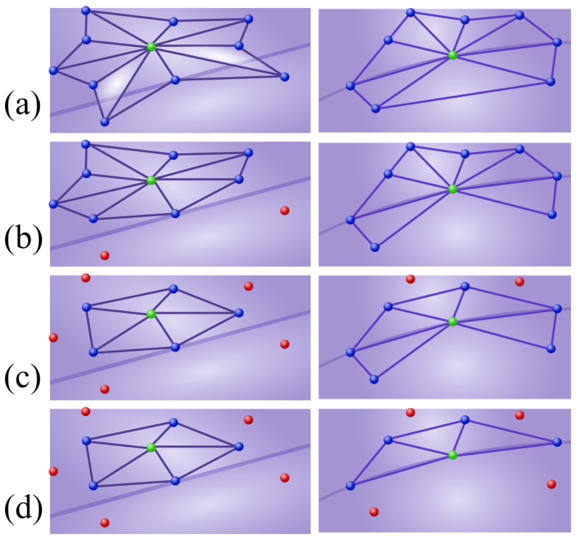


Figure 9: Umbrella creation near (left) and at (right) a feature line. We perform (a) triangle removal, (b) boundary detection, (c) Delaunay edge flipping, and (d) boundary expansion. See text for details.

boundary edges, as defined in Section 4.3. This process eliminates most samples remaining on other surface patches, as well as distant neighbours on smooth surfaces with relatively high curvature.

Next we apply the extrinsic Delaunay edge-flipping algorithm from [14], removing edges adjacent to p which are not locally Delaunay, shown in Figure 9(c). At each step, we examine p's one-ring for newly-created concave boundaries; boundary edges are never flipped.

4.3. Boundary detection

Our boundary detection is based upon an additional sampling assumption. A *local uniformity constraint* is some criterion that limits the number of samples that can appear in a small region. Any algorithm that attempts to create an umbrella on p using only points from $Q_k(p)$ is at least implicitly assuming some local uniformity constraint. Otherwise, all k nearest neighbours could be confined to a tiny disk close to p such that no reasonable umbrella could be constructed.

We express our local uniformity assumption as a bound on the minimum distance between sample points. We assume $\|e_G\| > \delta\varepsilon\rho_f(p)$, where $0 < \delta < 1$, and ε governs the sampling radius, as above.

A good triangle, t on p has its circumradius bounded by $O(\varepsilon\rho_f(p))$, and by a straight-forward geometric argument (as shown by Kil and Amenta [27], for example), the largest angle in t is bounded above by $\alpha = \pi - O(\delta)$ Thus the largest angle in any triangle is governed by a constant parameter that is independent of the local feature size. This is our parameter φ.

Before performing Delaunay edge flipping, we first check for boundaries (Figure 8). *Concave* boundaries are identified with triangles whose angles on p exceed φ. *Convex boundaries* are defined by triangles whose angle in the counter clockwise ordering around p exceeds π. We mark these explicitly here to ensure correct behaviour from the edge-flipping algorithm.

We also use excessively large triangle circumradii as indicators of the presence of a boundary. We should have $r_t < O(\frac{1}{\delta})||e_G||$, where $||e_G||$ is the distance to the nearest neighbour of p. However we have found this particular method of detecting boundaries to be too sensitive to variations in the sampling uniformity. Instead, we have had better success requiring $r_t < \gamma r_{t_G}$. Where γ is another constant parameter whose value reflects an expectation on the local uniformity of the sampling.

Finally, if any boundary edges have been detected, we attempt to *enlarge* them by examining the circumradii of the associated triangles, as in Figure 9(d). If a triangle t on a boundary edge has $r_t > \gamma r_{t_G}$, we disregard the sample on that edge, and make the other edge of t incident to p a boundary edge.

4.4. Alternate normals and supplemental umbrellas

We now have an estimate of the local surface around each point with a trustworthy Gabriel triangle. If that point is on a smooth surface patch, we expect it to have a triangulation without boundary. However, if p is on a boundary between smooth surface patches, the umbrella we have just constructed will only inform us about a single patch, and we must build umbrellas on its other adjacent patches using information from neighbouring points. Furthermore, if the Gabriel triangle, $t_G(p)$ is untrustworthy, we must obtain an estimate of p's normal from one of its neighbours. We decide that $t_G(p)$ is untrustworthy if less than half of the points in $Q_k(p)$ make an angle smaller than ω with the plane of t_G.

We address both problems in a second pass over the input, this time considering both points whose umbrellas have a boundary and points with untrustworthy Gabriel triangles. In either case we proceed as before, with the following modifications.

Rather than obtain a normal estimate from $t_G(p)$, we instead choose the computed normal from a *trustworthy* neighbour of p. This is a point in $Q_k(p)$ that has a single umbrella without boundary. The closest trustworthy neighbour to p gives us a trusted normal even when $t_G(p)$ is not reliable.

Filtering $Q_k(p)$ on this normal is more restrictive when p already has a partial umbrella. We reject points that would be admissible in an already-constructed umbrella, *unless* they lie on that umbrella's boundary (indicating that they too lie on feature edges). However, if a point lies on the boundaries of two partial umbrellas, it cannot lie on a third if S is manifold, and therefore it must be rejected.

We build partial umbrellas using the algorithm of Section 4.2 with these additional criteria until all boundary points have been included in a partial umbrella or no more trusted neighbours remain in $Q_k(p)$. In some cases, one of the new umbrellas will not have a boundary; this occasionally happens in areas of high curvature and sparse sampling, where $t_G(p)$ might be misleading because the sampling assumptions do not hold. In these cases, we simply discard the complete umbrella when it contains the fewest vertices of all p's umbrellas, and accept it (discarding the others) otherwise.

4.5. Enforcing boundary consistency

We have now constructed a local umbrella around each point p which is consistent with our characterization of the underlying surface (based on ω and γ) and incorporates our estimates of boundary and feature curves passing through p. However, aside from normal information in the cases of boundary points and untrustworthy Gabriel triangles, we have not incorporated any information contained in p's neighbours into its triangulation. For a well-sampled smooth surface this is generally sufficient; however, when the actual structure of our input point cloud does not satisfy our local uniformity assumptions this may lead to visual artifacts such as inconsistent or even spurious boundary detection where the input sampling breaks down.

Rather than attempting to enforce umbrella consistency across the whole point set as in the work of Kil and Amenta [27], we instead perform a third high-level pass over the input, identifying and correcting obvious inconsistencies near detected boundaries. This simple step significantly increases quality with minimal performance cost. Again, we are able to ignore most points in the input, instead focusing on points with boundaries that are incompatible with their neighbours' umbrellas. This may occur when a point's neighbour across a boundary edge does not itself have a boundary, as in Figure 10(b-c), or when two neighbours have boundary edges but do not agree on which edges those should be, as in Figure 10(d).

The first case may not indicate a problem at all; when a crease joins a smooth surface as in Figure 10(a), a sharp edge will terminate rather than meet another feature curve. Thus, if p has multiple partial umbrellas with consistent boundary edges, we simply accept it.

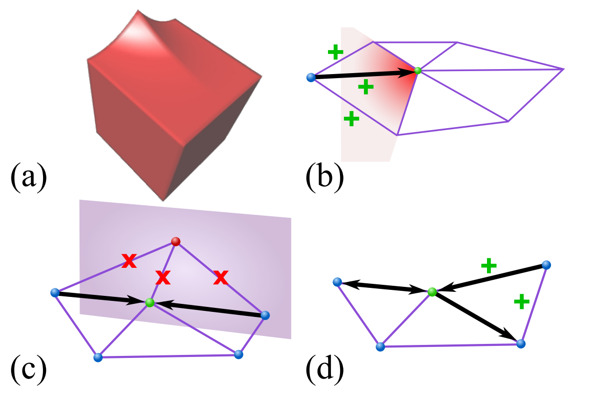


Figure 10: Not all disappearing sharp features indicate errors (a, taken from [36]. When a point p has an inconsistent boundary (b), we search $Q_k(p)$ for points that lie in the indicated area and contain reciprocal edges. Sparse sampling of boundaries may lead to point umbrellas with non-reciprocal edges on geodesically-distant sheets (c), or adjacent boundary vertices which are inconsistent (d). Our method addresses both problems by enforcing boundary edge reciprocity.

Otherwise, we identify points in $Q_k(p)$ whose umbrellas contain p, and consider those edges when rebuilding p's umbrella. We call these *reciprocal edges*, and our general strategy in the third pass is to add missing reciprocal edges and remove spurious reciprocal edges when necessary to ensure consistency between adjacent boundary points.

We identify and address two cases: points with full onerings but at least two incoming boundary edges, and points with non-reciprocal boundary edges. The first case often occurs when a surface boundary approaches another sheet of the surface: points on the boundary may erroneously include points on the sheet in their onerings. In this case, we mark incoming boundary edges and search between them for onering neighbours without reciprocal edges. We remove these neighbours and construct a boundary consistent with the incoming boundary edges. See Figure 10(c).

The second case, where one or more of a point's open boundary edges connect to neighbours without reciprocal boundary edges, often occurs at a sparsely sampled boundary. Here we wish to correct the boundary, taking neighbour information into account. If the point has incoming boundary edges, we update its boundary to contain those edges, adding reciprocal edges as needed, as shown in Figure 10(d).

However, most of these points occur in regions of high curvature and low sample density, where our φ assumption does not hold, and spurious and isolated boundaries are often detected. For these points, we find reciprocal edges in $Q_k(p)$ to fill the boundary, as shown in Figure 10(b).

5. Point set silhouette and feature extraction

We first describe our method for calculating the silhouette set of a point cloud and constructing local silhouette arcs. Then we present our preliminary attempt at point set feature extraction from the local umbrellas.

Silhouette extraction and silhouette arcs. We slightly modify the Hough-space silhouette algorithm in [33] to handle the SGS-approximating umbrellas created in Section 4. We refer the reader to [33] for details and only focus on the necessary modifications. To take advantage of the spatially and temporally coherent nature of Hough-space silhouette extraction and update, we store the Hough transforms of each SGS face in an augmented octree as described in [33]. Rather than consider every edge in p's umbrella, we store all faces associated with p together and test them as a group. This increases the complexity of testing the octree's edge bounding volumes against the v-sphere in initial silhouette extraction, but only by a constant factor.

Once we have identified a set of silhouette points, drawing silhouette *edges* between them is straightforward. Silhouette edges in each one-ring are easily identified but form a superset of the silhouette we wish to draw. Borrowing language from [27], we cull these edges into a more conservative set by drawing only *consensus silhouette edges*: We render a silhouette edge pq if and only if it exists and is a silhouette edge in the umbrellas of both p and q.

Note that we draw only local silhouette arcs, not full silhouette loops. The latter depend on global properties of point connectivity and must meet certain topological criteria[2] which we cannot guarantee from purely local constructions. It may be possible to augment our SGS construction with extra information and build consistent silhouette loops; we address this in our discussion of future work. For now we consider our silhouette arcs to be a first step towards a geometric solution.

Feature detection and emphasis. To aid visualization of point clouds, we adapt the feature classification method of Hubeli and Gross [24] to our local reconstruction. While the original method is used to classify the feature strength of *edges*, we instead apply their ESOD operator to point samples. For each edge pq in p's umbrella, we use the umbrellas of q's neighbours in p to evaluate $\cos\left(|\langle n_i, n_j\rangle|\right)^{-1}$, then divide by $\pi/2$. The absolute value term is required as our computed normals are not oriented, and thus not guaranteed to be consistent between neighbouring umbrellas. To determine the weight of each point we simply take the maximum computed weight among its edges.

Rather than implement hysteresis and patch skeletonization, we find it sufficient for visualization simply to treat the normalized weight of each point as its alpha value during rendering. This conveys curvature information in a view-independent way without competing with the silhouette for emphasis; see Figures 1 and 12.

6. Results

Some results of our point set silhouette and feature extraction algorithm for quick point cloud visualization are shown in Figures 1, 2, and 12. In all the cases, we chose $Q_k(p)$ to be the 16 nearest neighbours, our sampling density parameter $\omega = \pi/6$, and our local uniformity parameters $\varphi = \pi - \omega$ and $\gamma = 2$. Small changes to these parameters tend to produce small changes in the results, and the values chosen here reflect the fact that all of our raw point cloud data were processed with WLOP [23]. Inputs with different sampling characteristics will require changes to the parameters that reflect those differences in sampling.

Our SGS-building algorithm is a preprocessing step while silhouette and feature extractions are interactive. We performed our experiments on a workstation running Linux 2.6.18 with two Intel Xeon 3.2GHz processors, 4.0GB of RAM, and an NVidia GeForce 9800 GX2 card. The preprocessing step took between 5 seconds (hand model, 6,191 points) and 40 seconds (oil pump model, 54,220 points). Framerates for incremental silhouette updates varied with silhouette size, but never dropped below 190 frames per second.

Qualitatively, comparisons to the display of only visible points [26], as shown in Figure 1, reveal the ability of silhouettes and detected feature points to emphasize underlying shapes especially near surface features and fine-scale details. It is also worth pointing out that as the point cloud becomes more sparse, pure point rendering (left of Figure 1) becomes less effective in revealing geometric details while the usefulness of the characteristic curves is increased.

While normal estimation is not a key contribution of our work, our method's ability to reconstruct correct umbrellas in the presence of close-by surface sheets is demonstrated by Figure 11. Here, we show the robustness of normal estimation using our local umbrella construction and patch filtering algorithm (Appendix A), in comparison to PCA. While our umbrella construction starts with kNN, as in PCA, the optimization can identify the correct local neighborhood which does not straddle between the nearby sheets; this results in more accurate normal estimates.

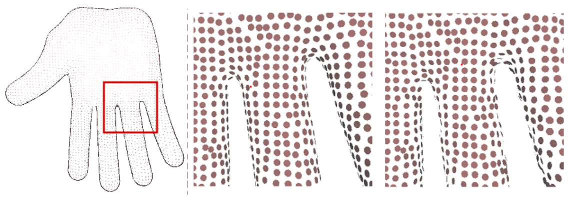


Figure 11: Point rendering using oriented splats on the hand mesh (left) to compare normal estimation: (centre) using our local reconstruction and (right) using PCA, where the same k for initial kNN is used. The PCA-estimated normals between adjacent fingers are inconsistent with their neighbours, while ours are coherent.

In Figures 2 and 12, we compare to the use of normal thresholding for point set silhouette extraction, where features are included for a better depiction of the shapes. Focusing on just the silhouettes, it is quite evident that using our local reconstruction and the SGS-based extraction scheme effectively avoids both under- and over-detection of silhouettes, which occur simultaneously under normal thresholding even on models derived from triangle meshes. Specifically, in regions with low curvature (either due to dense sampling or the geometry of the underlying surface), we produce a set of silhouette points with far more consistent thickness than the thresholding method. This is particularly evident on the fandisk and oil-pump models. In regions with high curvature, we are able to identify silhouette points where thresholding fails, both on sharp edges such as on the fandisk and over smooth regions of high curvature such as the fingers of the hand. Even on the fertility model, which is best suited to normal thresholding, we are able to identify more silhouette points on the higher-curvature arms and avoid overselection on the flat base.

Finally, in Table 1 and Figure 13 we show the results of our boundary estimation on a number of datasets with boundaries. In order to evaluate our results we chose input data from triangle meshes with connectivity removed, and used the mesh boundaries as ground truth for the statistics in Table 1; while this may not be an ideal metric, it is at least an objective external standard. Each input surface contains one or more boundaries and includes sampling features that make boundary estimation nontrivial. Note that our method identifies more spurious boundary vertices than it fails to detect. Mesh connectivity is not restricted by vertex position or density, and in areas with extremely acute or obtuse triangles our method is likely to find small boundary loops.

We also compare our results to those of the excellent PEEL algorithm described in [12] by Dey *et al.*.

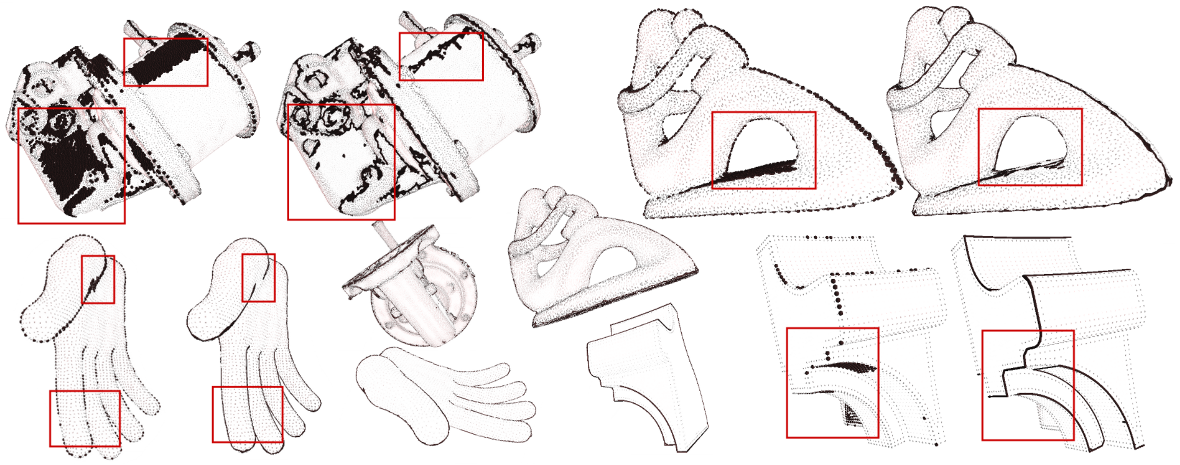


Figure 12: Comparison between normal thresholding (left figure of each pair) and our method (right). Insets show the models from the silhouette viewpoint. Red boxes highlight details discussed in the text.

Model	Bdry	Vertices		Percentage	
		Missed	Extra	Missed	Extra
Saddle	256	0	0	0	0
Pig	544	16	38	2.9	6.9
Face-HY	338	13	23	3.8	6.8

Table 1: Quantitative comparison of our method base mesh as ground truth. For each model in Figure 13, we show the number of boundary vertices on the mesh, and the number and percentage of missed and extra (spurious) boundary vertices.

This sophisticated method produces a provably-isotopic and globally consistent Delaunay mesh, even on non-orientable point clouds with boundaries. Note that only default parameters were used with PEEL. All three models generally satisfy PEEL's assumption of sufficiently uniform sampling, but its output mesh organization makes quantitative comparison difficult.

The *saddle* model is a simple synthetic rendering of the neighbourhood of a saddle point. Of note is the fact that this model was computed as a single octant and stitched together with significant point overlap at octant boundaries; our method finds spurious features on some of these boundaries, while PEEL detects spurious potential boundaries. The inset shows our local umbrellas along an octant boundary near the centre of the model; note the abrupt changes in sample density.

Similarly, the *pig* model exhibits sharp linear features, particularly at the ears (shown in the inset). These features are inherently undersampled, and neither our local reconstruction nor PEEL's more global method can perfectly reconstruct them. Note also that our method finds boundaries at the eyes, while PEEL triangulates them; this is due to our assumption that the local surface can be characterized by the Gabriel triangle radius.

Finally, the *face-HY* model is constructed from a single raw point scan, and exhibits typical scanner errors near the eyelashes, nostril, and lips. Both our method and PEEL misidentify internal points as boundaries where this occurs, but our ability to construct multiple partial umbrellas where sharp features occur produces fewer spurious boundaries at the eyes and mouth.

7. Conclusion

We present a silhouette computation scheme for point clouds based on the notion of *silhouette-generating sets*, which allows us to extract point set silhouettes with higher accuracy than normal thresholding. The method utilizes a local umbrella reconstruction algorithm that efficiently captures shape geometry and is able to reconstruct neighbourhoods near sharp features, boundaries, and close-by surface sheets. The locally-reconstructed geometry also allows for feature extraction, leading to intuitive depiction of complex models. Our local method is suitable for partial or incremental application to a data set, is well-suited to GPU-based or out-of-core implementation, and produces results approaching those of mesh silhouette algorithms, supporting our argument that fully-consistent surface reconstruction is overkill for silhouette extraction on point clouds.

Limitations. The primary theoretical limitation of our method is the lack of an intrinsic scheme to handle noise or highly non-uniform point sampling. For practical purposes a denoising step [23] applied as pre-processing is sufficient to produce satisfactory results. While our method is able to recover sharp features well in general, it may fail to reconstruct sharp corners with negative

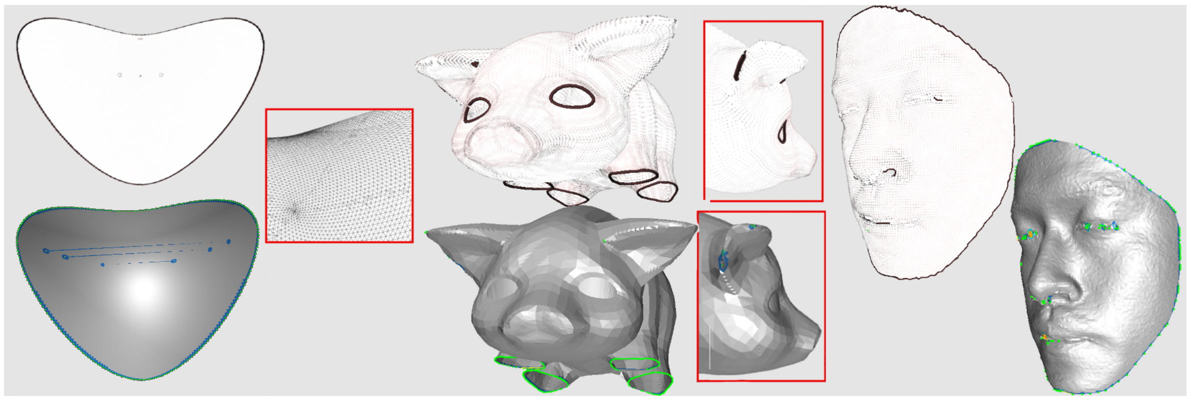


Figure 13: Comparison of our boundary detection results (white/brown) with those of PEEL (grey/green) on the *saddle*, *pig*, and *face-HY* models. Insets show features described in the text. Despite its purely local support, our method produces results comparable to *Peel*.

curvature since such a configuration does not respect well our Gaussian curvature criterion. Expanding our method for propagating plane estimates between points may provide a remedy.

Future work. In our current implementation, we only produce local arcs from the point umbrellas to approximate the characteristic curves of a shape. We may be able to exploit the fact that boundaries and silhouettes form closed loops and utilize existing edge linking and thinning algorithms [24] to produce much cleaner curves. However, producing topologically-correct silhouette loops on a point cloud is challenging, particularly without globally consistent connectivity. We would like to either extract such silhouette loops or show that finding them is as hard as constructing a globally-consistent triangulation. Also, our local method's reliance on a small neighbour set and ability to cope gracefully with missing data opens the door to a number of specialized and efficient implementations; we would like to explore these possibilities. Finally, many point processing tasks, such as computing the Laplace operator [5] and photon mapping [37], rely on accurate local neighbourhood computations. We would like to evaluate the benefits offered to those algorithms by our local umbrella construction.

References

[1] Adamy, U., Giesen, J., John, M., 2002. Surface reconstruction using umbrella filters. Comput. Geom. Theory Appl. 21 (1), 63–86.

[2] Akenine-Möller, T., Assarsson, U., 2003. On the degree of vertices in a shadow volume silhouette. Journal of Graphics Tools 8 (4), 21–24.

[3] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C. T., 2001. Point set surfaces. In: IEEE Visualization. pp. 21–28.

[4] Amenta, N., Choi, S., Dey, T. K., Leekha, N., 2000. A Simple Algorithm for Homeomorphic Surface Reconstruction. In: Symp. Comp. Geom. pp. 213–222.

[5] Belkin, M., Sun, J., Wang, Y., 2009. Constructing laplace operator from point clouds in rd. In: SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1031–1040.

[6] Boissonnat, J.-D., Flötotto, J., 2004. A coordinate system associated with points scattered on a surface. Computer-Aided Design 36 (2), 161 – 174.

[7] Bouma, W., Vanecek, G., 1995. Velocity-based collision detection. In: Paeth, A. (Ed.), Graphics Gems V. Academic Press, pp. 380–385.

[8] Buchanan, J. W., Sousa, M. C., 2000. The edge buffer: A data structure for easy silhouette rendering. In: Proc. of the 1st Int. Symp. on Non-Photorealistic Animation and Rendering (NPAR). pp. 39–42.

[9] Crow, F. C., 1977. Shadow algorithms for computer graphics. In: ACM SIGGRAPH. pp. 242–248.

[10] Deussen, O., Strothotte, T., 2000. Computer-generated pen-and-ink illustration of trees. In: Proc. SIGGRAPH 2000. pp. 13–18.

[11] Dey, T., 2007. Curve and Surface Reconstruction; Algorithms with Mathematical Analysis. Cambridge University Press.

[12] Dey, T. K., Li., K., Ramos, E. A., , Wenger, R., 2009. Isotopic reconstruction of surfaces with boundaries. In: ACM Symposium on Geometry Processing. pp. 1371–1382.

[13] Dyer, R., 2010. Self-Delaunay meshes for surfaces. Ph.D. thesis, Simon Fraser University, Burnaby, Canada.

[14] Dyer, R., Zhang, H., Möller, T., 2007. Delaunay mesh construction. In: ACM Symposium on Geometry Processing. pp. 271–282.

[15] Fleishman, S., Cohen-Or, D., Silva, C. T., 2005. Robust moving least-squares fitting with sharp features. In: ACM SIGGRAPH 2005 Papers. SIGGRAPH '05. ACM, New York, NY, USA, pp. 544–552.

[16] Glisse, M., Lazard, S., 2008. An upper bound on the average size of silhouettes. Discrete Comput. Geom. 40 (2), 241–257.

[17] Gooch, B., Sloan, P. J., Gooch, A., Shirley, P., Risenfeld, R., 1999. Interactive technical illustration. In: Proc. ACM Symp. on 3D Interactive Graphics. pp. 31–38.

[18] Gopi, M., Krishnan, S., 2002. A fast and efficient projection-based approach for surface reconstruction. In: SIBGRAPI. pp. 179–186.

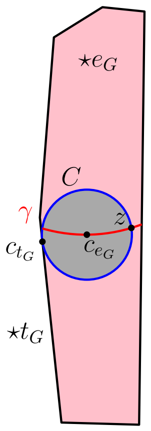
[19] Gopi, M., Krishnan, S., Silva, C. T., 2000. Surface recon-

struction based on lower dimensional localized Delaunay triangulation. In: Gross, M., Hopgood, F. R. A. (Eds.), Computer Graphics Forum (Eurographics). Vol. 19(3). pp. 467–478. URL citeseer.ist.psu.edu/gopi00surface.html
[20] Gross, M., Pfister, H., 2007. Point-Based Graphics. Morgan Kaufman.
[21] Hertzmann, A., Zorin, D., 2000. Illustrating smooth surfaces. In: In ACM SIGGRAPH. pp. 517–526.
[22] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., 1992. Surface reconstruction from unorganized points. In: ACM SIGGRAPH. pp. 71–78.
[23] Huang, H., Li, D., Zhang, H., Ascher, U., Cohen-Or, D., 2009. Consolidation of unorganized point clouds for surface reconstruction. ACM Transactions on Graphics, (Proceedings SIGGRAPH Asia 2009) 28 (5), 176.
[24] Hubeli, A., Gross, M., 2001. Multiresolution feature extraction for unstructured meshes. In: Proc. of IEEE Visualization. pp. 287–294.
[25] Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., Strothotte, T., 2003. A developer's guide to silhouette algorithms for polygonal models. IEEE Comput. Graph. Appl. 23 (4), 28–37.
[26] Katz, S., Tal, A., Basri, R., 2007. Direct visibility of point sets. In: International Conference on Computer Graphics and Interactive Techniques. ACM New York, NY, USA, p. Article 24.
[27] Kil, Y. J., Amenta, N., 2008. GPU-assisted surface reconstruction on locally-uniform samples. In: Meshing Roundtable. pp. 369–385.
[28] Koenderink, J., 1984. What does the occluding contour tell us about solid shape? Perception 13, 321–330.
[29] Lipman, Y., Cohen-Or, D., Levin, D., Tal-Ezer, H., 2007. Parameterization-free projection for geometry reconstruction. ACM Trans. Graph. 26 (3), 22.
[30] McGuire, M., 2004. Observations on silhouette sizes. jgt 9 (1), 1–12.
[31] Mederos, B., Velho, L., De Figueiredo, L., 2003. Moving least squares multiresolution surface approximation. In: SIBGRAPI. pp. 19–26.
[32] Mehra, R., Tripathi, P., Sheffer, A., Mitra, N., 2010. Visibility of noisy point cloud data. Computers & Graphics 34 (3), 219–230.
[33] Olson, M., Zhang, H., 2006. Silhouette extraction in hough space. Computer Graphics Forum (Special Issue on Eurographics 2006) 25 (3), 273–282.
[34] Pauly, M., Gross, M., 2002. Efficient simplification of point-sampled surfaces. In: IEEE Visualization. pp. 163–170.
[35] Plaenkers, R., Fua, P., 2002. Model-based silhouette extraction for accurate people tracking. In: European Conf. on Computer Vision. pp. 325–339.
[36] project, A., ???? Smooth-feature. URL http://shapes.aim-at-shape.net/view.php?id=42
[37] Spencer, B., Jones, M. W., 2009. Into the blue: Better caustics through photon relaxation. Computer Graphics Forum (Special Issue on Eurographics 2009) 28 (2).
[38] Xu, H., Nguyen, M., Yuan, X., Chen, B., 2004. Interactive silhouette rendering for point-based models. In: Eurographics Symposium on Point-Based Graphics. pp. 13–18.
[39] Zakaria, N., Seidel, H.-P., 2004. Interactive stylized silhouette for point-sampled geometry. In: Computer Graphics and Interactive Techniques in Australasia and South East Asia. Association for Computing Machinery, pp. 242–249.
[40] Zwicker, M., Pfister, H., van Baar, J., Gross, M., 2001. Surface splatting. In: Proc. of SIGGRAPH. pp. 371–378.

Appendix A. The canonical Gabriel triangle is a cocone triangle

As noted in Section 4.1, the normal error of a triangle with vertices on a smooth surface is bounded by the circumradius. The Cocone surface reconstruction algorithm [4, 11] exploits this observation by filtering triangles with small circumradii from the Delaunay tetrahedralization of the samples. The *cocone* of p consists of those points $y \in V(p)$ for which the acute angle between $\overline{py}$ and the line generated by (an estimate of) n_p is greater than $\frac{3\pi}{8}$. Given a sampling radius ρ, a point in the cocone of p is $O(\rho(p))$ from p [11].

A *cocone triangle* is a triangle t in the Delaunay tetrahedralization for which the dual Voronoi edge intersects the cocones of each of its vertices. For such a triangle, r_t is bounded by $O(\rho(p))$. Thus cocone triangles lie close to the tangent plane at p. We demonstrate that t_G, the Gabriel triangle constructed in Section 4.1, is a cocone triangle. The idea is to show that the diametric ball for t_G is small. This is done by considering a sphere with this radius centred at p, and examining how it intersects the Voronoi face closest to p.



Let c_{t_G} be the circumcentre of t_G, and consider the sphere Σ of radius r_{t_G} centered at p. Referring to the figure, let $\star e_G$ be the Voronoi face dual to e_G, and $\star t_G$ be the Voronoi edge dual to t_G. Then c_{e_G} and c_{t_G} lie on $\star e_G$ and $\star t_G$ respectively [13, Lemma 4.3]. The sphere Σ intersects $\star e_G$ in a circle C, centered at c_{e_G} and tangent to $\star t_G$ at c_{t_G}. C cannot intersect any other of the edges bounding $\star e_G$: this would imply the existence of a Delaunay triangle on e_G with circumradius smaller than r_{t_G}.

Since e_G is the shortest edge incident to p, c_{e_G} must lie in the cocone of p [11, Lemma 3.4]. Consider the right circular cone with apex p and axis n_p, containing $[p, c_{e_G}]$ as a generatrix. The lateral surface of this cone is contained in the cocone of p. The extension of this surface intersects $\star e_G$ in a curve γ, which contains c_{e_G} and must intersect C in some point z. Since this curve is also contained in the cocone of p, it follows that z also lies in the cocone of p. Since the distance between z and p is exactly r_{t_G}, if follows that r_{t_G} is bounded by $O(\rho(p))$. Thus t_G provides a good estimate of the tangent plane at p. Since the dual Voronoi edge of a Gabriel triangle intersects the affine hull of the triangle, the bound on r_{t_G} also ensures that c_{t_G} will lie in the cocones of each of the vertices of t_G: it is a cocone triangle.

Appendix B. Responses to Reviewer 4's request for minor revisions

The method described is mainly dedicated to local (multi) umbrellas generation and the authors have greatly improved their description of the construction is section 4, except for 4.5 which should also be rewritten more clearly (as for figure 9a which is supposed to illustrate triangle removal... The authors probably mean the transition from 9a to 9b?)

We have revised Section 4.5 for improved clarity. In particular, we have revised Figure 9 to better illustrate the operations performed in the boundary-consistency pass and updated the text to make better use of the figure. It is likely that much of the confusion in Section 4.5 came from an error in its last paragraph, which referred to Figure 9(d) rather than 9(b) as it should have. This has been corrected.

Suggestions for adding references were not taken into account in the revised paper. In fact, the paper bibliography is only dealing with silhouette generation and local neighborhood in point clouds but not with umbrella generation, which is the real topic of this paper. The authors also do not mention the papers that have already dealt with filtering of k nearest neighbors near close-by surface sheets, nor the existing work on boundary detection.

More generally, the authors did not highlight why their local reconstructions should be better than the existing local umbrella reconstructions methods. They have not made the connection or comparison with similar or alternative elements that have proven to be efficient in overall reconstruction approaches but can easily be reused in local approaches.

We regret this omission and have expanded Section 2 to broaden its coverage of local reconstruction. We feel that our contribution in this area is that of a unified method, capable of boundary and sharp-edge detection in the presence of close surfaces and based on a small set of nearest neighbors, rather than an improvement in quality on these specific applications over the existing state of the art. Therefore, while existing techniques might be adapted to a local approach, we prefer to work within the framework of the Gabriel triangle and local Delaunay triangulation. Section 2 has been revised to make this more clear.

Additional remarks :

In the pseudocode of figure 5, the authors should make the tuning parameters appear in the list of the arguments of the functions which use them.

We have added these parameters.

The authors should be careful when they argue that a global reconstruction would be overkill. I agree with the idea, but in their local framework the adjacencies between points are recomputed several times (which should not be the case in a more global setting). Moreover, their algorithm seems not to be very efficient at that time.

We feel that the previous revisions qualified our claim appropriately at the end of the Introduction, where we stress that local reconstructions have significant potential when global memory access is difficult and/or costly, such as on a GPU or in an out-of-core implementation. We have revised the conclusion to further emphasize this qualification.

p1, column 2, line 53, replace angle by scalar product? p3, column2, line 45 : "the the"

We regret and have corrected these errors.

p4, column2, line 36 : what is the meaning of "lines generated by nt and np"?

This is intended to cover the case in which the orientations of nt and np are not consistent with each other; hence, we discuss a specific angle between unoriented lines rather than between vectors.