To be published in Computers & Graphics, vol. 28, no. 6, 2004.

Point Cloud Surfaces using Geometric Proximity Graphs

Jan Klein [a] and Gabriel Zachmann [b]

[a] *Heinz Nixdorf Institute and Institute of Computer Science, University of Paderborn, Germany*
[b] *Department of Computer Science II, University of Bonn, Germany*

Abstract

We present a new definition of an implicit surface over a noisy point cloud, based on the weighted least squares approach. It can be evaluated very fast, but artifacts are significantly reduced.

We propose to use a different kernel function that approximates geodesic distances on the surface by utilizing a geometric proximity graph. From a variety of possibilities, we have examined the Delaunay graph and the sphere-of-influence graph (SIG), for which we propose several extensions.

The proximity graph also allows us to estimate the local sampling density, which we utilize to automatically adapt the bandwidth of the kernel and to detect boundaries. Consequently, our method is able to handle point clouds of varying sampling density without manual tuning.

Our method can be integrated into other surface definitions, such as moving least squares, so that these benefits carry over.

Key words: Weighted least squares, moving least squares, proximity graphs, surface approximation, implicit surfaces, local polynomial regression.

1. Introduction

In the past few years, point clouds have had a renaissance caused by the wide-spread availability of 3D scanning technology. In order to render [1, 2, 3, 4] and interact [5] with objects thus represented, one must define an appropriate surface (even if it is not explicitly reconstructed).

This definition should produce a surface as close to the original surface as possible while being robust against noise (introduced by the scanning process). At the same time, it should allow to render and interact with the object as fast as possible.

Email addresses: janklein@uni-paderborn.de (Jan Klein), zach@cs.uni-bonn.de (Gabriel Zachmann).

In this paper, we present a new definition of a surface over a given point cloud. It builds on an implicit function defined using weighted least squares (WLS) regression. Our techniques can also be applied to other surface definitions, such as Levin's popular projection operator (which is based on moving least squares approximation).

The simple WLS definition of point cloud surfaces is quite attractive and can be evaluated very fast. However, it suffers from artifacts in the surface. They are caused by a distance function that is not adapted to the topology of the surface: the Euclidean distance makes points "close" to $\mathbf{x}$ that are really topologically far away. They may be also be caused by an inappropriate kernel bandwidth,

which should be based on the sampling density.

The idea of our method is to utilize (conceptually) a Voronoi diagram to find the nearest neighbor of a query point $\mathbf{x}$, and then traverse the Voronoi diagram breadth-first to compute approximate geodesic distances between the query point and the cloud points. Since the Voronoi diagram, in this context, basically provides just an adjacency relation based on some notion of proximity, we can also use other *proximity graphs*. Here, we investigate also the *sphere-of-influence* graph with several extensions, which provides a natural notion of proximity in our context. [1]

In order to evaluate the quality of our surfaces, we generate noisy point clouds from a given "exact" surface. For these, we compute the deviation of the zero sets of the different definitions from the original exact surface. The results show that our new definition produces much better surfaces. In addition, our experiments show that our method can be evaluated very fast.

Note that in this paper we are not concerned with actually rendering the implicit surface. This can be done with ray tracing [6], sphere tracing [7, 8], or tesselation [9].

2. Related Work

The representation of objects by point clouds is based on some notion of *surface* that describes the surface in-between the points, which are samples taken from an original surface, usually with error.

One way is to extend the points to so-called surfels yielding a piece-wise linear surface [1, 2]. Our work does not deal with this kind of surface representation.

Another way is to consider the problem of *reconstruction*, where a continuous surface is explicitly constructed from the set of points, usually in the form of a polygonal mesh. Several methods can be distinguished; particularly attractive are *combinatorial methods* because they can guarantee the reconstructed mesh to be homeomorphic to the original surface under some reasonable assumptions [10, 11]. Other methods are more cluster- or graph-based [12, 13]. We are not concerned with this kind of approach, because it does not stay within the framework of point clouds.

An attractive way of handling point clouds is to define the surface as the zero set of an *implicit function* that is constructed from the point cloud. Usually, this function is not analytically but "algorithmically" given. This is a general method that can be used for reconstruction as well as ray-tracing or collision detection.

An interesting method pursuing this approach is the use of natural coordinates (which are based on Voronoi diagrams) [14]. They are used to turn Hoppe's discontinuous definition [13] into a continuous one ($\mathcal{C}^\infty$ almost everywhere). However, computing the natural coordinates is very expensive.

A very popular class of methods is to define the surface as the set of fixed points of a projection map based on local polynomial regression [15, 16, 17, 18]. For each evaluation of the function, an approximating polynomial needs to be computed over a suitable plane, both of which are found using moving least squares. A simpler, and faster, method is to define the surface as the zero set of a function, which is algorithmically constructed by local linear regression (weighted least squares) [6, 19, 20]. These methods are fairly easy to implement but difficult to make robust. In particular, non-uniform point clouds are difficult to handle generally, and there can be extra zero sets.

Recent publications have, therefore, proposed to partition the point set by an octree and fit quadratic functions only to leaves that are occupied by points [7].

As mentioned above, our method is based on proximity graphs, which have been studied extensively in the past decade. There is a broad spectrum of them, including the Delaunay graph, nearest-neighbor graph, γ-graph, α-shape, and the spheres-of-influence graph, to name but a few; see [21] for a good survey. They have been used for OCR [22, 23], reconstruction [24], and many other applications.

In [25], a Euclidean minimum spanning tree is used to perform thinning on a set of unorganized points sampled from a curve.

[1] Another way to encode approximate geodesic distances is a triangle mesh. However, here we would have a "bootstrapping" problem, because the mesh would have to be created by some kind of tesselation, which would need to evaluate the implicit function at many points in space.

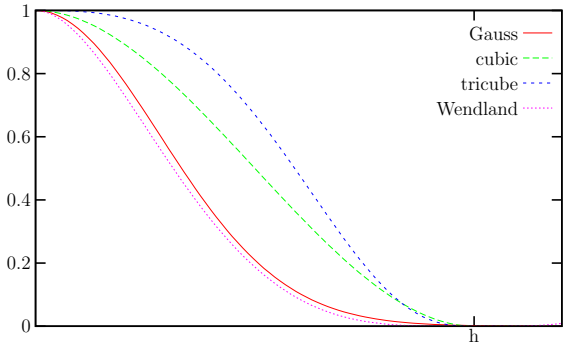


Figure 1. Our method is independent of the mapping from distances to weights, so different weight functions can be used.

3. Implicit Surface Model

In this section, we will first give a quick recap and then explain the problem of the conventional WLS method. For sake of clarity, all illustrations are in 2D, but the methods work, of course, in any dimension.

3.1. *Surface Definition*

Let a point cloud $\mathcal{P}$ with N points $\mathbf{p}_i \in \mathbb{R}^3$ be given. Then, an appealing definition of the surface from $\mathcal{P}$ is the zero set $S = \{\mathbf{x}|f(\mathbf{x}) = 0\}$ of an implicit function [19]

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x}) \tag{1}$$

where $\mathbf{a}(\mathbf{x})$ is the weighted average of all points $\mathcal{P}$

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{p}_i}{\sum_{i=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_i\|)}. \tag{2}$$

Usually, a Gaussian kernel (weight function)

$$\theta(d) = e^{-d^2/h^2}, \quad d = \|\mathbf{x} - \mathbf{p}\|, \tag{3}$$

is used, but other kernels work as well (see below).

The bandwidth of the kernel, h, allows us to tune the decay of the influence of the points. It should be chosen such that no holes appear [5].

Theoretically, θ's support is unbounded. However, it can be safely limited to the extent where it falls below the machine's precision, or some other, suitably small threshold θ_ε. Alternatively, one could use the cubic polynomial [25]

$$\theta(d) = 2\Big(\frac{d}{h}\Big)^3 - 3\Big(\frac{d}{h}\Big)^2 + 1,$$

or the tricube weight function [26]

$$\theta(d) = \Big(1 - |\frac{d}{h}|^3\Big)^3,$$

or the Wendland function [27]

$$\theta(d) = \Big(1 - \frac{d}{h}\Big)^4 (4\frac{d}{h} + 1),$$

all of which are set to 0 for $d > h$ and, thus, have compact support (see Figure 1 for a comparison). However, the choice of kernel function is not critical [28].

The normal $\mathbf{n}(\mathbf{x})$ is determined by weighted least squares. It is defined as the direction of smallest weighted covariance, i.e., it minimizes

$$\sum_{i=1}^{N} \big(\mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{p}_i)\big)^2 \theta(\|\mathbf{x} - \mathbf{p}_i\|) \tag{4}$$

for fixed $\mathbf{x}$ and under the constraint $\|\mathbf{n}(\mathbf{x})\| = 1$.

Note that, unlike [19], we use $\mathbf{a}(\mathbf{x})$ as the center of the PCA, which seems to make $f(\mathbf{x})$ much more well-behaved (see Figure 2). Also, we do not solve a minimization problem like [15, 16], because we are aiming at an extremely fast method.

The normal $\mathbf{n}(\mathbf{x})$ defined by (4) is the smallest eigenvector of the centered covariance matrix $\mathbf{B} = (b_{ij})$ with

$$b_{ij} = \sum_{k=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_k\|)(p_{k_i} - a(\mathbf{x})_i)(p_{k_j} - a(\mathbf{x})_j). \tag{5}$$

There are several variations of this simple definition, but for sake of clarity, we will stay with this basic one. Our new method can be applied to more elaborated ones as well.

3.2. *Euclidean Kernel*

The above definition can produce artifacts in the surface S (see Figure 2); two typical cases are as follows. First, assume $\mathbf{x}$ is halfway between two (possibly unconnected) components of the point cloud; then it is still influenced by *both* parts of the point cloud, which have similar weights in Equ. 2 and 4. This can lead to an *artificial* zero subset $\subset S$

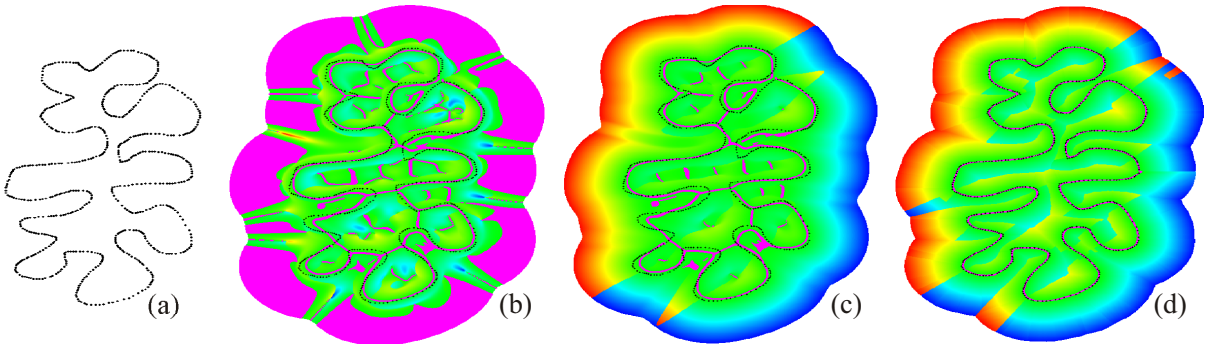


Figure 2. Visualization of the implicit function $f(\mathbf{x})$ over a 2D point cloud. Points $\mathbf{x} \in \mathbb{R}^2$ with $f(\mathbf{x}) \approx 0$, i.e., points on or close to the surface, are shown magenta. Red denotes $f(\mathbf{x}) \gg 0$ and blue denotes $f(\mathbf{x}) \ll 0$. (a) point cloud; (b) reconstructed surface using the definition of [19]; (c) utilizing the centered covariance matrix produces a better surface, but it still has several artifacts; (d) surface and function $f(\mathbf{x})$ based on our more geodesic kernel using the sphere-of-influence graph.

where there are no points from $\mathcal{P}$ at all. Second, let us assume that $\mathbf{x}$ is inside a cavity of the point cloud. Then, $\mathbf{a}(\mathbf{x})$ gets "drawn" closer to $\mathbf{x}$ than if the point cloud was flat. This makes the zero set *biased* towards the "outside" of the cavity, away from the true surface. In the extreme, this can lead to cancellation near the center of a spherical point cloud, where all points on the sphere have a similar weight.

This thwarts algorithms based solely on the point cloud representation, such as collision detection [5] or ray-tracing [6].

In all of these cases, the problem is caused by the following deficiency in the kernel (3). The Euclidean distance $\|\mathbf{x} - \mathbf{p}\|$, $\mathbf{p} \in \mathcal{P}$, can be small, while the distance from $\mathbf{x}$ to the closest point on S and then along the shortest path to $\mathbf{p}$ on S (the geodesic) is quite large.

The problems mentioned above could be alleviated somewhat by restricting the surface to the region $\{\mathbf{x} : ||\mathbf{x} - \mathbf{a}(\mathbf{x})|| < c\}$ (since $\mathbf{a}(\mathbf{x})$ must stay within the convex hull of $\mathcal{P}$). However, this does not help in many cases involving cavities.

4. Geodesic Distance Approximation

As mentioned above, the main problems are caused by a distance function that does not take the topology of S into account. We propose to use a different distance function that is based on geodesic distances on the surface S. Unfortunately, we do not have an explicit reconstruction of S, and in many applications, we do not even want to construct one.

Therefore, we propose to utilize a geometric proximity graph where the nodes are points $\in \mathcal{P}$. In such proximity graphs, nodes $\mathbf{p}$ and $\mathbf{q}$ are connected by an edge if some geometric proximity predicate holds. So, it is obvious that geodesic distances between the points can be approximated by shortest paths on the edges of the graph.

There is a whole spectrum of different proximity graphs over a set $\mathcal{P}$, for instance the Delaunay graph $DG(\mathcal{P})$, the Gabriel graph, the relative nearest neighbor graph, and the nearest neighbor graph [21]. These are all subgraphs of the DG, with different densities, so we choose to investigate the DG. Another interesting proximity graphs seems to be the sphere-of-influence graph $SIG(\mathcal{P})$, because it is not a subgraph of the DG, and because it seems to capture the notion of *sampling density* fairly well (see below).

In the following, the length of an edge is the Euclidean distance $\|\mathbf{p} - \mathbf{q}\|$ (or any other metric).

4.1. *Geodesic Kernel*

We define our new distance function $d_{\mathsf{geo}}(\mathbf{x}, \mathbf{p})$ as follows. Given some location $\mathbf{x}$, we compute its nearest neighbor $\mathbf{p}_1^* \in \mathcal{P}$. Then, we compute the

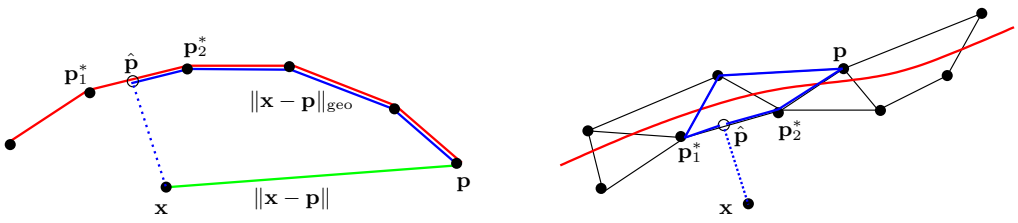


Figure 3. Instead of the Euclidean distance, we use an approximate geodesic distance based on the close-pairs shortest-paths matrix over a proximity graph.

closest point $\hat{\mathbf{p}}$ to $\mathbf{x}$ that lies on an edge adjacent to $\mathbf{p}_1^*$. Now, conceptually, the distance from $\mathbf{x}$ to any $\mathbf{p} \in \mathcal{P}$ could be defined as

$$d_{\text{geo}}(\mathbf{x},\mathbf{p}) = \min_{\mathbf{n}\in\{\mathbf{p}_1^*,\mathbf{p}_2^*\}} \big\{ d(\mathbf{n},\mathbf{p}) + \|\hat{\mathbf{p}}-\mathbf{n}\| \big\},$$

where $d(\mathbf{p}^*,\mathbf{p})$ for any $\mathbf{p} \in \mathcal{P}$ is the accumulated length of the shortest path from $\mathbf{p}^*$ to $\mathbf{p}$, multiplied by the number of "hops" along the path (see Figure 3 left). However, it is not obvious that this is always the desired distance. In addition, it is desirable to define the distance function with as few discontinuities as possible. Therefore, we just take the weighted average (see Figure 3 right)

$$\begin{aligned} d_{\text{geo}}(\mathbf{x},\mathbf{p}) = \quad & (1-a)\big(d(\mathbf{p}_1^*,p) + \|\hat{\mathbf{p}}-\mathbf{p}_1^*\|\big) \\ & + a\big(d(\mathbf{p}_2^*,p) + \|\hat{\mathbf{p}}-\mathbf{p}_2^*\|\big) \end{aligned} \tag{6}$$

with the interpolation parameter $a = \|\hat{\mathbf{p}}-\mathbf{p}_1^*\|$.

Note that we do *not* add $\|\mathbf{x}-\hat{\mathbf{p}}\|$. The effect is that $f(\mathbf{x})$ is non-zero everywhere far away from the point cloud.

Of course, there are still discontinuities in d_{geo} and thus in function f. These can occur at the borders of the Voronoi regions of the cloud points, in particular at borders where the Voronoi sites are far apart from each other, such as the medial axis.

The rationale for multiplying the path length by the number of hops is the following: if an (indirect) neighbor $\mathbf{p}$ is reached by a shortest path with many hops, then there are many points in $\mathcal{P}$ that should be weighted much more than $\mathbf{p}$, even if the Euclidean distance $\|\mathbf{p}^*-\mathbf{p}\|$ is small. This is independent of the concrete proximity graph used for computing the shortest paths.

Overall, when computing f by (1)–(5), we use d_{geo} in (3). We call this modified kernel a *geodesic kernel*.

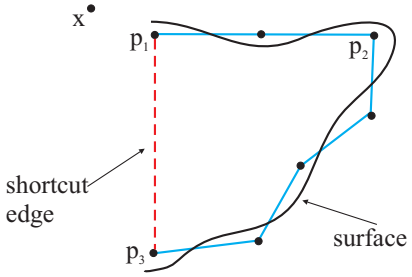


Figure 4. Shortcuts across cavities lead to imprecise approximate geodesic distances, especially if the path length is multiplied by the number of hops along the path.

Note that the proximity graph should not contain edges which significantly "short-circuit" cavities. In that case, the shortcut would lead to much higher weights for farther points than for some points in-between. This problem is illustrated in Figure 4. Assume, point $\mathbf{p}_1$ is the closest point to some space point $\mathbf{x}$. Then, $\mathbf{p}_2$ can have a higher approximate geodesic distance from $\mathbf{p}_1$ than $\mathbf{p}_3$ from $\mathbf{p}_1$.

4.2. *Proximity by Delaunay Graph*

It is very intuitive to use the Delaunay graph $DG(\mathcal{P})$ for our problem, because [10] described an intriguing algorithm for reconstructing a polygonal surface over a point cloud without noise from its Voronoi diagram (which is the dual of the Delaunay graph). Later, [29] extended this to provable reconstruction from noisy models.

So we investigated the possible use of the $DG(\mathcal{P})$ as a proximity graph. Since it induces a neighborhood relation that also includes "long distance" neighborhoods, some shortest paths can "tunnel"

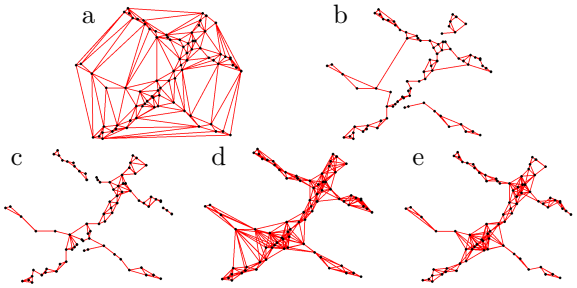


Figure 5. Different proximity graphs. (a) $DG(\mathcal{P})$, (b) $DG(\mathcal{P})$ where edges are pruned according to $Q_3 + IQR$, (c) $1 - SIG(\mathcal{P})$, (d) $3 - SIG(\mathcal{P})$, (e) $3 - SIG(\mathcal{P})$ with pruning.

through space that should really be a gap in the model (see Figure 5, left). Therefore, we prune edges from $DG(\mathcal{P})$ based on criteria that involve an estimation of the local spatial density of the point cloud (see below). However, this can make the $DG(\mathcal{P})$ too sparse, while at the same time it does not always prune all "long" edges. This will cause artifacts in the surface (see Figure 6, left).

If our point cloud is well-sampled in the sense of [10], then we could prune all edges incident to a point $\mathbf{p} \in \mathcal{P}$ that are longer than the distance of $\mathbf{p}$ from the medial axis of S — *provided* we knew that distance for each $\mathbf{p}$. This is, of course, not feasible.

Therefore, we propose to utilize a statistical outlier detection method to prune edges. This is motivated by the observation that most of the unwanted "long distance" edges are local outliers, or form a cluster of outliers. In the following, we describe a simple outlier detection algorithm that seems to perform well in our case, but, of course, other outlier detection algorithms [30] should work as well.

In statistics, an outlier is a single observation which is far away from the rest of the data. One definition of "far away" in this context is "greater than $Q_3 + 1.5 \cdot IQR$" where Q_3 is the third quartile, and IQR is the interquartile range $Q_3 - Q_1$. Our experiments showed that best results are achieved by pruning edges with length of at least $Q_3 + IQR$.

4.3. *Proximity by Sphere-of-Influence Graph*

The sphere-of-influence graph (SIG) is a fairly little known proximity graph [22, 23]. The idea is

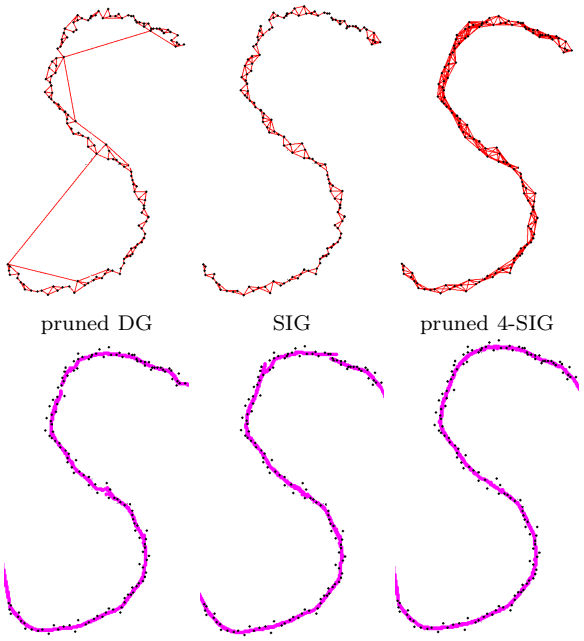


Figure 6. Surface induced by different proximity graphs. Clearly, the pruned DG and the plain SIG incur artifacts, due to sparsity and disconnected components. In our experience, the 4- or 5-SIG, with pruning, seems to work well in all cases.

to connect points if their "spheres of influence" intersect. More precisely, for each point $\mathbf{p}_i$ the distance d_i to its nearest neighbor is determined and two points $\mathbf{p}_i$ and $\mathbf{p}_j$ are connected by an edge if $\|\mathbf{p}_i - \mathbf{p}_j\| \leq d_i + d_j$.

As a consequence, the SIG tends to connect points that are "close" to each other relative to the local point density. In contrast to the $DG(\mathcal{P})$, no "long distance" neighbor relations are created (see Figure 5 c), except for some pathological cases when the surface is very irregularly sampled.

4.4. *Extensions of the SIG*

In the following, we propose several extensions to the plain SIG.

4.4.1. *r-SIG*

In noisy or irregularly sampled point clouds, there can be several pairs of points that are placed much farther apart from each other than

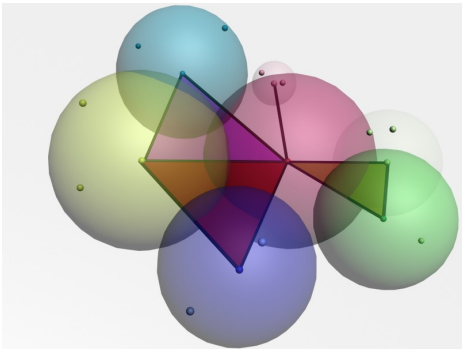


Figure 7. The sphere-of-influence graph can be extended to a sphere-of-influence complex in 3D, which allows better geodesic approximations than the sphere-of-influence graph (in 3D). Note that only a few of all simplices of the complex for this point set are shown.

their inter-pair separation. In such situations, the $SIG(\mathcal{P})$ would consist of a lot of isolated "mini-clusters", even though there are no holes in the original surface (see Figure 6, middle). Consequently, the corresponding surface could not be reconstructed correctly, because the approximated geodesic distances are too imprecise: on the one hand, they are too large because points close together can only indirectly be accessed through the graph by visiting other nodes; on the other hand — in the case of unconnected components — for some points in space, too few cloud points are considered for the reconstruction.

To overcome this problem, we propose the *r-th order SIG*: instead of computing the distance to the nearest neighbor for each node, we compute the distance to the r-nearest neighbor and then proceed as in the case of $r = 1$. It is obvious that the larger r, the more nodes are directly connected by an edge, and that too large r can result in "long distance" edges as in the case of the $DG(\mathcal{P})$ (see Figure 5 d).

In our experience, it seems best to choose $r = 4$ or $r = 5$, and then prune away all "long" edges by the method described above (see Figure 5 e), which yields almost always a nice surface (see Figure 6, right).

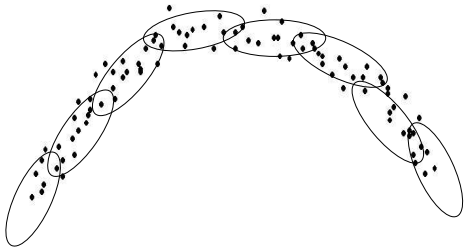


Figure 8. We propose the anisotropic SIG to adapt the "spheres"-of-influence better to the neighborhood of the points, thus yielding a better proximity graph for our purposes.

4.4.2. *Sphere-of-Influence Complex*

In 3D, we can either stay with the SIG presented so far, or we can extend the definition analogously to contain triangles as well. Thus, we obtain a complex that consists of the vertices, the edges as defined before, and triangles. The latter are formed among all triples of points $(\mathbf{p}, \mathbf{q}, \mathbf{r})$ that are close to each other: $\mathbf{p}, \mathbf{q}, \mathbf{r}$ are considered close iff $B_p \cap B_q \cap B_r \neq \varnothing$, where B_x is the sphere of influence around $\mathbf{x}$ with radius r_x.[2] Figure 7 shows a few triangles and points from such a complex.

The advantage of such a complex is that we can compute much better approximations to geodesic paths than if the paths were restricted to edges only [31, 32].

4.4.3. *Anisotropic SIG*

Sometimes, the "post-processing" of the r-SIG, as described above, can prune away too many or too few edges. In order to reduce the susceptibility of the proximity graph to the pruning threshold, and in order to adapt it better to our problem, we propose the *anisotropic SIG*. The idea is use ellipsoids instead of spheres around the points, where the axes are the principal components of a suitable number of neighboring points (see Figure 8).

For each point $\mathbf{p} \in \mathcal{P}$, we construct its "sphere"-of-influence as follows. We start with its k nearest

[2] This is somewhat related to the Czech complex. However, in the Czech complex all spheres have the same radius.

neighbors, k being small (3 or 4). [3] Then, we compute the principal axes of this set and determine radii so that the ellipsoid contains the set. [4] Then we (conceptually) scale the ellipsoid until it contains one more point. [5] With this increased neighborhood point set, we compute a new ellipsoid, as before. We repeat this procedure, until the ellipsoid contains r points, or until $\sigma(\mathbf{p}) = \frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$ exceeds a predefined threshold. Here, λ_1 is the smallest eigenvector, so that $\sigma(\mathbf{p})$ captures the surface variation [18].

The advantage of this is, that in areas with small variance, we can automatically choose large spheres-of-influence, while in areas with high curvature or close sheets these spheres are kept small. Thus, the edges of the anisotropic SIG are less prone to "short-circuit" sharp features (as depicted in Figure 4).

Finally, for determining the edges of the anisotropic SIG, we need to check intersections of ellipsoid. This could be computed exactly by [34, Ch. 3.3], for instance, but we have opted for a simpler, approximate check: we just sample each ellipsoid by a number of points and check whether any of them is contained in the other one.

4.5. *Automatic bandwidth computation*

A critical parameter is the bandwidth h in (3). On the one hand, if this is chosen too small, then variance may be too large, i.e., noise, holes, or other artifacts may appear in the surface. On the other hand, if it is chosen too large, then bias may be too large, i.e., small features in the surface will be smoothed out. To overcome this problem, [18] proposed to scale the parameter h adaptively.

Here, we can use the proximity graph (pruned Delaunay, or r-SIG) to estimate the local sampling density, $r(\mathbf{x})$, and then determine h accordingly. Thus, h itself is a function $h = h(\mathbf{x})$.

[3] Should these, including $\mathbf{p}$, be (almost) coplanar, we take more until we get a non-coplanar set.

[4] Alternatively, we could have computed a smallest enclosing ellipsoid by [33], but the benefit seemed questionable.

[5] This can be done efficiently by transforming the points in the coordinate system of the ellipsoid and then choosing the closest one.

Assuming the terminology from Section 4.1 (see Figure 3), let r_1 and r_2 be the lengths of the longest edges incident to $\mathbf{p}_1^*$ and $\mathbf{p}_2^*$, resp. Then, we set

$$r(\mathbf{x}) = \frac{1}{r} \cdot \frac{\|\hat{\mathbf{p}} - \mathbf{p}_2^*\| \cdot r_1 + \|\hat{\mathbf{p}} - \mathbf{p}_1^*\| \cdot r_2}{\|\mathbf{p}_2^* - \mathbf{p}_1^*\|} \tag{7}$$

$$h(\mathbf{x}) = \frac{\eta\, r(\mathbf{x})}{\sqrt{-\log \theta_\varepsilon}} \tag{8}$$

where θ_ε is a suitably small value (see Section 3.1), and r is the number of nearest neighbors that determine the radius of each sphere-of-influence. (Note that $\log \theta_\varepsilon < 0$ for realistic values of θ_ε.) Thus, p_i with distance $\eta\, r(\mathbf{x})$ from $\hat{\mathbf{p}}$ will be assigned a weight of θ_ε (see Equ. 3).

We have now replaced the scale- and sampling-dependent parameter h by another one, η, that is *independent* of scale and sampling density. In our experience, this can just be set to 1, or it can be used to adjust the amount of "smoothing". Note that this automatic bandwidth detection works similarly for many other kernels as well (see Section 3.1).

Depending on the application, it might be desirable to involve more and more points in (1), so that $\mathbf{n}(\mathbf{x})$ becomes a least squares plane over the complete point set $\mathcal{P}$ as $\mathbf{x}$ approaches infinity. In that case, we can just add $\|\mathbf{x} - \hat{\mathbf{p}}\|$ to (7).

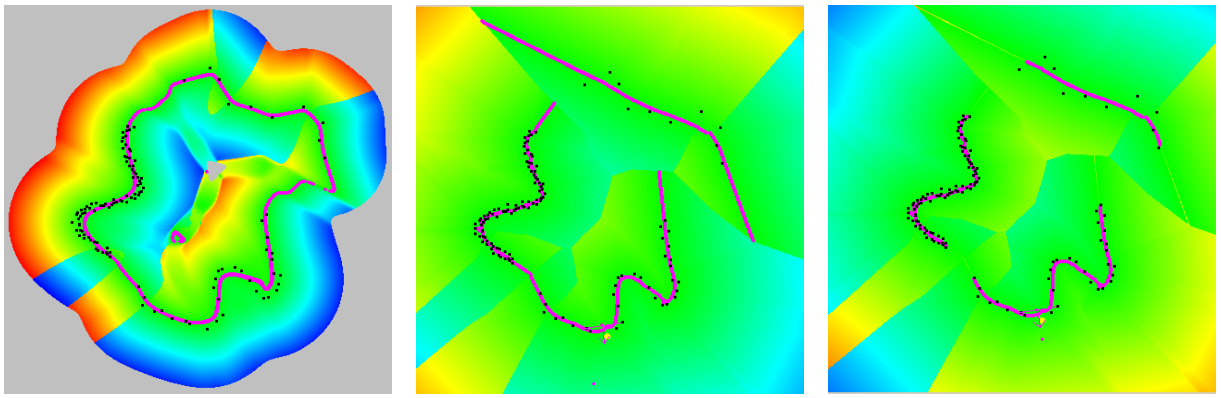
Figure 9 shows that the automatic bandwidth determination allows the WLS approach to handle point clouds with varying sampling densities without any manual tuning. Notice that the smoothing of the different sampling densities is very similar, compared to the "scale" (i.e., density).

4.6. *Automatic boundary detection*

Another benefit of the automatic sampling density estimation is a very simple boundary detection method. Our method builds on the one proposed by [6].

The idea is simply to discard points $\mathbf{x}$ with $f(\mathbf{x}) = 0$, if they are "too far away" from $\mathbf{a}(\mathbf{x})$ *relative* to the sampling density in the vicinity of $\mathbf{a}(\mathbf{x})$. More precisely, we define a new implicit function

$$\hat{f}(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } |f(\mathbf{x})| > \varepsilon \vee \|\mathbf{x} - \mathbf{a}(\mathbf{x})\| < 2r(\mathbf{x}) \\ \|\mathbf{x} - \mathbf{a}(\mathbf{x})\|, & \text{else} \end{cases} \tag{9}$$



plain WLS, $h = 22$ | 2-SIG with pruning, $\eta = 1.7$ | plus boundary detection

Figure 9. Our method offers automatic sampling density estimation individually for each point, which allows to determine the bandwidth automatically and independently of scale and sampling-density (middle), and to detect boundaries automatically (right).

Figure 9, right, shows that this simple method is able to handle different sampling densities fairly well.

5. Runtime and Complexity

This section gives an overview about the complexity of our auxiliary data structures and the time for evaluating $f(\mathbf{x})$ in the 3D case.

5.1. *Close-Pairs Shortest-Paths*

Computing shortest paths on-the-fly during the computations of our geodesic distances would be, of course, prohibitively expensive, so we precompute them. However, computing and storing an *all pairs shortest paths* (APSP) matrix would also be infeasible for larger point clouds. Since the Gaussian (3) decays fairly quickly (for reasonable choices of h), and other weight functions have bounded support, we need to store only paths up to some length; the contribution of nodes in Equations 2 and 5 that are farther away can be neglected. In Section 5.1, we show that the resulting matrix can be computed and stored in $O(N)$ time and space using a simple lookup table. Therefore, we denote it just as CPSP (close-pairs shortest-paths) table.

In the following, we show that this table can be computed and stored in $O(N)$ with $N = |\mathcal{P}|$.

Definition 1 (Sampling radius) *Consider a set of spheres, centered at points $p_i \in \mathcal{P}$, that cover the surface defined by $\mathcal{P}$, where all spheres have equal radius. We define the sampling radius $r(\mathcal{P})$ as the minimal radius of such a sphere covering.*

The bandwidth h should be chosen such that points up to a distance of about $m \cdot r(\mathcal{P})$ around a point $\mathbf{p}_i \in P$ have an influence in Equ. 1 ($m \approx 5$) [5]. That means, a point $\mathbf{p}_j$ is only used, if $\theta(\|\mathbf{p}_i - \mathbf{p}_j\|) \geq \theta(m \cdot r(\mathcal{P}))$.

As a consequence, for each point $\mathbf{p}_i \in \mathcal{P}$ we have to run a SSSP algorithm for the source $\mathbf{p}_i$ to points $\mathbf{p}_j$ where $\theta(d_{\mathsf{geo}}(\mathbf{p}_i, \mathbf{p}_j)) < \theta(m \cdot r(\mathcal{P}))$. Such points $\mathbf{p}_j$ are obviously contained in a sphere S_i with radius $m \cdot r(\mathcal{P})$ centered at $\mathbf{p}_i$, and they can easily be determined by a depth-first or breadth-first search starting at $\mathbf{p}_i$.

The following lemma shows that only a constant number of points is inside S_i, if $\mathcal{P}$ is a uniform (possibly noisy) sampling of a surface. As a consequence, we have to start N times a SSSP algorithm for a point set of constant size. Overall, our CPSP table can be computed in time $O(N)$.

Lemma 2 *Let a point cloud $\mathcal{P}$ with uniformly distributed points $\mathbf{p}_i \in \mathbb{R}^d$ ($d \in \{2, 3\}$) and sampling radius $r(\mathcal{P})$ be given. Then, at most $\lceil \sqrt{d} \cdot m \rceil^d$ points $\in \mathcal{P}$ lie in a sphere with radius $m \cdot r(\mathcal{P})$.*

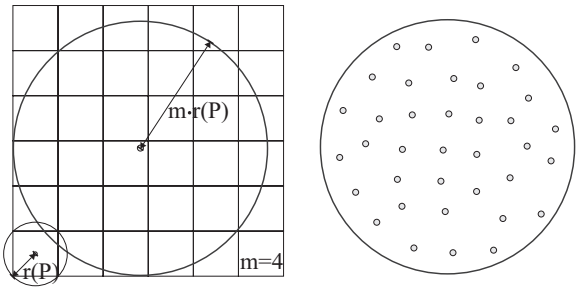


Figure 10. Under reasonable assumptions, the close-pairs shortest-paths matrix has size $O(N)$. Left: a sphere with radius $m \cdot r(\mathcal{P})$ can be covered by $O(m^3)$ spheres with radius $r(\mathcal{P})$. Right: $\lceil\sqrt{2}\cdot m\rceil^2$ uniformly distributed points inside.

Proof: In the following, we consider only the 3D case ($d = 3$), the 2D case can be shown analogously.

A sphere S_1 with radius $m \cdot r(\mathcal{P})$ can be covered with at most $c := \lceil\sqrt{3}\cdot m\rceil^3$ smaller spheres of radius $r(\mathcal{P})$. This has already been shown by Rogers [35]: the sphere S_1 can be covered by a cube with side length $2mr(\mathcal{P})$ and the smaller spheres with radius $r(\mathcal{P})$ cover cubes with side length $\sqrt{4/3}\,r(\mathcal{P})$ (see Fig. 10 left). As a consequence, the larger cube can be covered by $\lceil 2mr(\mathcal{P})/\sqrt{4/3}\,r(\mathcal{P})\rceil^3 = c$ smaller cubes and therefore by the same number of spheres with radius $r(\mathcal{P})$.

That means, c uniformly distributed spheres of radius $r(\mathcal{P})$ with centers in S_1 cover S_1. Only if the spheres are not uniformly distributed, more than c spheres with sampling radius $r(\mathcal{P})$ are necessary to cover S_1. ∎

Note that in practice often much fewer points than c lie inside S_1, in most cases $k\cdot\lceil\sqrt{d-1}\cdot m\rceil^{d-1}$ are realistic (k is a small constant of about 2 or 3).

For memory efficiency reasons, we store the CPSP matrix in a table of size $O(N)$ instead of using a N^2 matrix.

5.2. *Pre-computations of Proximity Graphs*

Under reasonable assumptions, Attali and Boissonnat showed recently that the complexity of the Delaunay graph is linear in the number of points [36]. Chan et al. proposed an output sensitive algorithms to compute the Delaunay triangulation [37]. Their result says that one can construct the Delaunay triangulation in 3D in time $O((n+F)\log^2 F)$ where F is the number of faces in the Delaunay triangulation. Applied with $F = O(N)$ (which is the result of Attali and Boissonnat) gives an upper bound of $O(N\log^2 N)$.

The r-th order SIG can be determined in time $O(N)$ on average for uniform point sampled models with size N in any fixed dimension which is a direct result from Dwyer [38]. Moreover, it consumes only linear space for well-sampled surfaces.

5.3. *Function Evaluation*

Our geodesic kernel needs to determine the nearest neighbor $\mathbf{p}^*$ of a point $\mathbf{x}$ in space. Under mild conditions, this can be done in $O(\log N)$ time by utilizing a Delaunay hierarchy [39], but this may not always be practical. Using a simple k-d tree, an approximate nearest neighbor can be found in $O(\log^3 N)$ [40].

As shown in Section 5.1, all points $\mathbf{p}_i$ influencing $\mathbf{x}$ can be determined in constant time by a depth-first or breadth-first search.

Overall, $f(\mathbf{x})$ can be determined in $O(\log N)$ (or $O(\log^3 N)$ if using the simple k-d tree), which is the same time as in the case of the Euclidean kernel if we would also restrict the influence of points there.

In order to achieve also a fast practical function evaluation, we have implemented the following algorithm for computing the smallest eigenvector [5]. First, we compute the smallest eigenvalue λ_1 by determining the three roots of the cubic characteristic polynomial of the covariance matrix $\mathbf{B}$. Then, we compute the associated eigenvector using the Cholesky decomposition of $\mathbf{B} - \lambda\mathbf{I}$.

In our experience, this method is faster than the Jacobi method by a factor of 4, and it is faster than singular value decomposition by a factor 8.

5.4. *Dynamic Point Clouds*

Because of their lack of inherent connectivity, point-based models seem to be very suitable in dynamic settings.[6] In this section, we discuss the

[6] Note that this advantage vanishes if, for instance, hierarchical data structures are used to accelerate the rendering.

computational overhead for point insertion, deletion, and moving.

In the worst case, inserting, moving, or deleting a point p_i in the general Delaunay graph or SIG can change $\Theta(N)$ edges. However, in practical cases, we can bound that number by $O(1)$ as we are only interested in paths up to some length $mr(P)$ (see Section 5.1). As a consequence, the neighbor relations have to be updated only for a constant number of points, namely for the points inside the sphere S_i centered at p_i and with radius $mr(P)$. Thereby, not only edges inside the sphere have to be added or removed, also edges to points outside the sphere can change. However, the number of edges which have to modified remains constant.

In the case of inserting or moving a point, the computation of the point set inside S_i causes an additional factor of $O(\log N)$ or $O(\log^3 N)$ using a Delaunay hierarchy or a k-d tree, respectively, as we have to perform a nearest neighbor search in the graph to find the starting point.

After updating the proximity graph, all affected entries in our CPSP table have to be recomputed. If inserting or deleting a point p_i from the graph, only the paths between points in the sphere S_i with radius $mr(P)$ around p_i have to be updated (more precisely, only the paths between points p_i and p_j where $\|p_i - p_j\| \leq mr(P)$). In the case of moving a point p_i, the radius of S_i has to be increased by the distance between the new and the old position of p_i.

As a consequence, the paths in S_i can easily be updated by an APSP algorithm on the set $\mathcal{P} \cap S_i$. As already shown in Section 5.1, $\mathcal{P} \cap S_i$ (which has constant size) can be determined by a simple depth-first or breadth-first search in time $O(1)$, and therefore, our CPSP table can be updated in constant time.

Overall, inserting or moving a point can be done in at most $O(\log^3 N)$, while deleting can be performed in constant time.

6. Results

We have implemented our new point cloud surface definition in C++. It is easy to implement and can be evaluated very fast: once the graphs are built, we can evaluate $f(\mathbf{x})$ simply by finding a nearest neighbor, traversing the graph, computing a number of weights from the CPSP table, and finally one eigenvector by Cholesky decomposition.

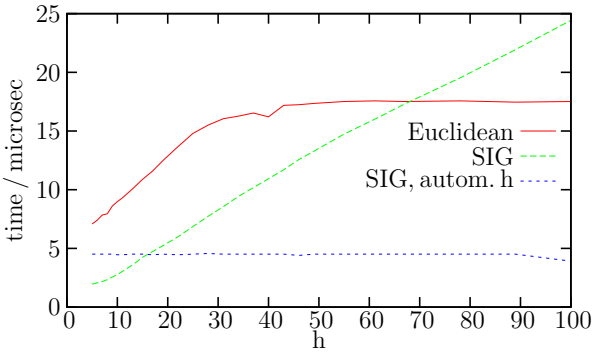


Figure 11. Average evaluation time of $f(\mathbf{x})$ depending on the kernel bandwidth h. The timings for $SIG(\mathcal{P})$ and $DG(\mathcal{P})$ are nearly identical (therefore, we omit one of them). Please note that our implementation is not yet fully optimized. Timing was done on a 1 GHz Pentium 3.

First of all, Figure 11 shows the performance that can be achieved using our new surface definition for a reasonable choice of h. Although our implementation is not fully optimized, the performance is of the same order as that of the Euclidean kernel.

Figure 12 and 13 illustrate the quality depending on the Euclidean kernel and our new geodesic one, respectively. Moreover, in order to give a numerical hint for the quality, we determined the root mean square error (RMSE) for the deviation (i.e., distance) of the reconstructed surface from the original surface. Obviously, our geodesic kernel approximates the surface very well, while the Euclidean kernel produces several artifacts. Even when the bandwidth h (see Equation 3) is chosen optimally with respect to the RMSE, the Euclidean kernel produces severe artifacts (see Figure 13).

We also performed experiments to assess the sensitivity of our surface definition with respect to the kernel bandwidth h. The plots in Figure 14 (left and center) show for two different example surfaces that our new kernel is less sensitive towards the choice of h than the old one: for a large range of the bandwidth, the RMSE using our new sur-

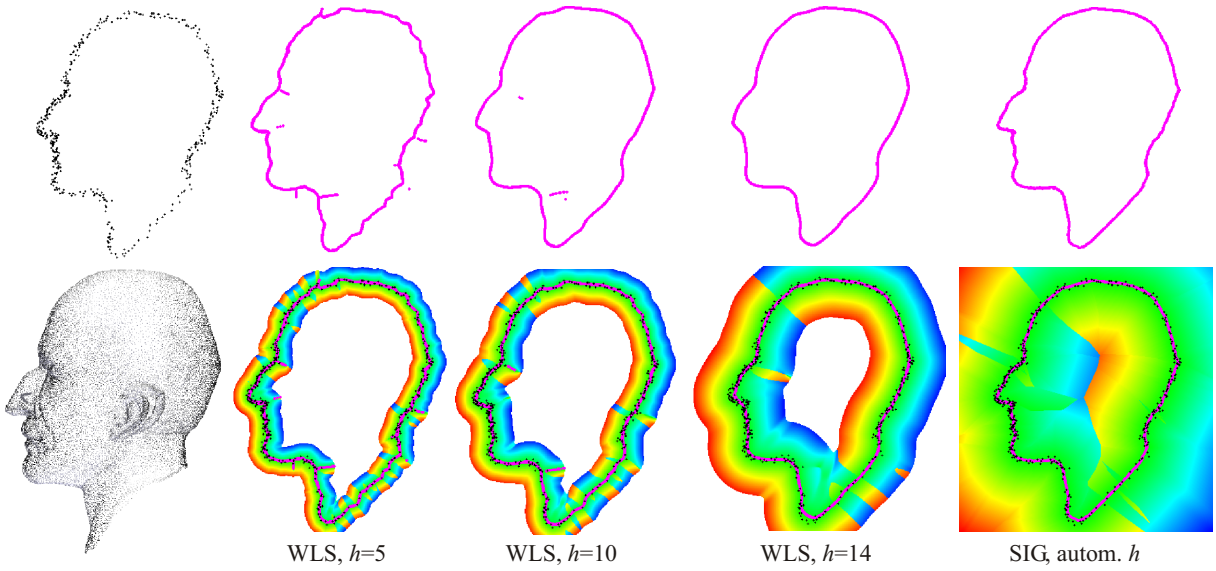


Figure 12. Reconstructed surface based on WLS and our new surface definition (rightmost) for a noisy point cloud obtained from the 3D Max Planck model (leftmost). Notice how our new definition, including automatic bandwidth detection, is able to handle fine detail as well as sparse sampling without manual tuning.

face definition is quite low. In contrast, only for a small bandwidth the Euclidean kernel yields a relatively low RMSE. Note that in almost every case the RMSE of the Euclidean kernel is larger than the RMSE of our new kernel. Note further, that the minimal RMSE of our new definition is clearly smaller than the minimal RMSE of the old one.

It might seem that there is still one parameter in our new approach, which requires fine-tuning, namely r, the radius for our modified sphere-of-influence graph r-SIG. However, numerous measurements for different point clouds suggest that $r \in [3 \dots 6]$ seems to be a good choice for all models.

7. Conclusion and Future Work

We have presented a new surface definition that utilizes proximity graphs, k-d trees, and weighted least squares to approximate geodesic distances. We have also proposed several extensions to the sphere-of-influence graph.

Overall, our new surface definition yields implicit functions over point clouds, the zero sets of which are much closer to the original surface than the previous weighted least squares approach. At the same time, our definition can be evaluated quite fast. In addition, the auxiliary data structures can be constructed efficiently and incur only little additional storage.

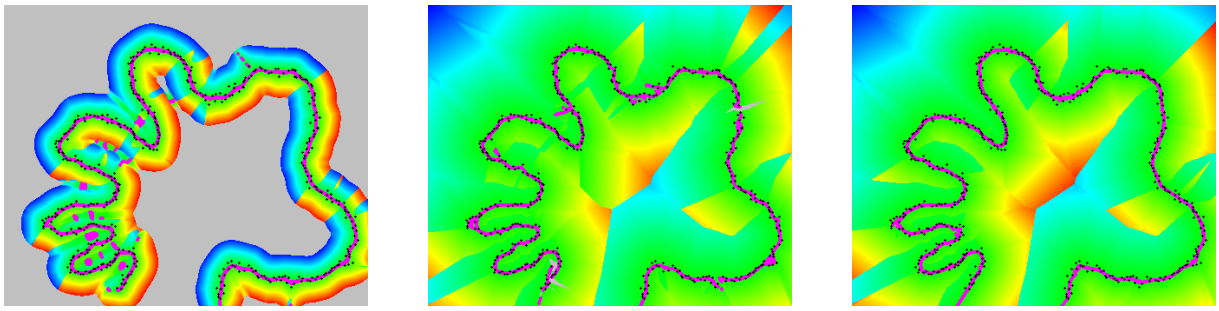
Our method can be utilized for other variants of point cloud surfaces as well, such as local polynomial approximations or the moving least squares approach [16].

We believe that this work opens up a number of further avenues for future work.

From a theoretical point of view, our implicit function $f(\mathbf{x})$ is discontinuous at more points $\mathbf{x} \in \mathbb{R}^3$ than the conventional WLS definition. Although this does not seem to be a problem from a practical point of view, it would be appealing to find a remedy that offers the same performance.

In rare cases, our r-SIG tends to "short-circuit" cavities, which can lead to inappropriate weights for (geodetically) far points. This could be alleviated by also computing kind of longest paths through the proximity graph.

Although the complexity for inserting or moving points is in $O(\log^3 N)$, it should be possible to devise more efficient algorithms when all points



plain WLS pruned DG pruned 4-SIG

Figure 13. Comparison of the Euclidean and our new geodesic kernel for a noisy point cloud. In each example, the bandwidth was chosen so that the RMSE is minimal.

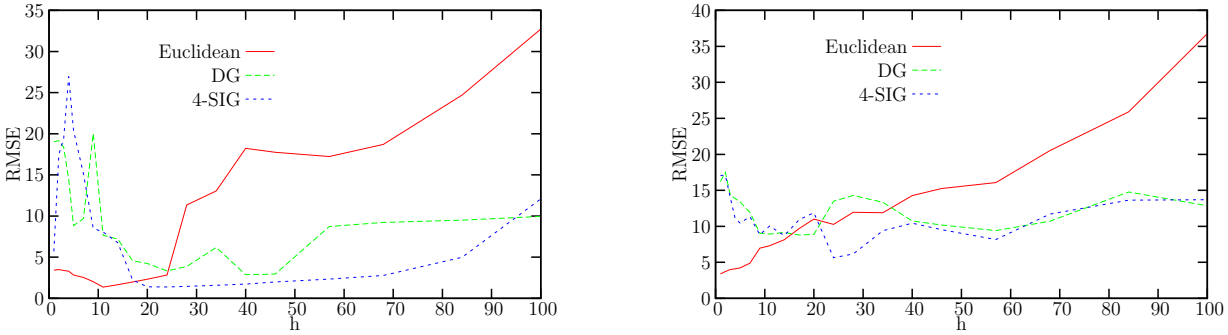


Figure 14. RMSE depending on the bandwidth, h, of the kernel. Our new kernel is less sensitive towards the choice of h than the old one. Refer to Figure 12 and Figure 13 for the corresponding models.

are moving at once. We believe that we can exploit temporal coherence in order to update the proximity graph. Possibly, a kinetic data structure can be devised to accelerate this in the average case.

Acknowledgements

This work was partially supported by DFG grant DA155/29-1 "Benutzerunterstützte Analyse von Materialflusssimulationen in virtuellen Umgebungen" (BAMSI), and the DFG program "Aktionsplan Informatik" by grant ZA292/1-1.

References

[1] H. Pfister, J. van Baar, M. Zwicker, M. Gross, Surfels: Surface elements as rendering primitives, in: Proc. of SIGGRAPH, 2000, pp. 335–342. 1, 2

[2] S. Rusinkiewicz, M. Levoy, QSplat: A multiresolution point rendering system for large meshes, in: Proc. of SIGGRAPH, 2000, pp. 343–352. 1, 2

[3] M. Zwicker, H. Pfister, J. van Baar, M. Gross, EWA splatting, IEEE Trans. on Visualization and Computer Graphics 8 (3) (2002) 223–238. 1

[4] K. Bala, B. Walter, D. P. Greenberg, Combining edges and points for interactive high-quality rendering, in: Proc. of SIGGRAPH, 2003, pp. 631–640. 1

[5] J. Klein, G. Zachmann, Point cloud collision detection, in: M.-P. Cani, M. Slater (Eds.), Computer Graphics forum (Proc. EUROGRAPHICS), Vol. 23, Grenoble, France, 2004, pp. 567–576. 1, 3, 4, 9, 10

[6] A. Adamson, M. Alexa, Approximating bounded, non-orientable surfaces from points, in: Shape Modeling International, 2004, pp. 243–252. 2, 4, 8

[7] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, H.-P. Seidel, Multi-level partition of unity implicits, in: Proc. of SIGGRAPH, 2003, pp. 463–470. 2

[8] J. C. Hart, Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces, The Visual Computer 12 (9) (1996) 527–545, iSSN 0178-2789. 2

[9] J. Bloomenthal, An implicit surface polygonizer, in: P. Heckbert (Ed.), Graphics Gems IV, Academic

Press, Boston, 1994, pp. 324–349. 2

[10] N. Amenta, T. K. D. S. Choi, N. Leekha, A simple algorithm for homeomorphic surface reconstruction, Intl. Journal on Computational Geometry & Applications 12 (2002) 125–141. 2, 5, 6

[11] T. K. Dey, S. Goswami, Tight cocone: A water tight surface reconstructor, in: Proc. 8th ACM Sympos. Solid Modeling Appl., 2003, pp. 127–134. 2

[12] B. Heckel, A. E. Uva, B. Hamann, K. I. Joy, Surface reconstruction using adaptive clustering methods, in: G. Brunnett, H. Bieri, G. Farin (Eds.), Computing Supplement, Vol. 14, 2001, pp. 199–218, dagstuhl 1999. 2

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, in: Proc. of SIGGRAPH, 1992, pp. 71–78. 2

[14] J.-D. Boissonnat, F. Cazals, Smooth surface reconstruction via natural neighbour interpolation of distance functions, in: Proc. 16th Annual Symp. on Computational Geometry, ACM Press, 2000, pp. 223–232. 2

[15] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. T. Silva, Computing and rendering point set surfaces, IEEE Trans. on Visualization and Computer Graphics 9 (1) (2003) 3–15. 2, 3

[16] D. Levin, Mesh-independent surface interpolation, in: H. Brunnett, Mueller (Eds.), Geometric Modeling for Scientific Visualization, Springer, 2003. 2, 3, 12

[17] N. Amenta, Y. Kil, Defining point-set surfaces, in: J. C. Hart (Ed.), ACM Trans. on Graphics (Proc. of SIGGRAPH), Vol. 23, 2004, pp. 264–270. 2

[18] M. Pauly, M. H. Gross, L. Kobbelt, Efficient simplification of point-sampled surfaces, in: IEEE Visualization 2002, 2002, pp. 163–170. 2, 8

[19] A. Adamson, M. Alexa, Approximating and intersecting surfaces from points, in: Proc. Eurographics Symp. on Geometry Processing, 2003, pp. 230–239. 2, 3, 4

[20] A. Adamson, M. Alexa, On normals and projection operators for surfaces defined by point sets, in: Eurographics Symp. on Point-Based Graphics, 2004, pp. 149–155. 2

[21] J. W. Jaromczyk, G. T. Toussaint, Relative neighborhood graphs and their relatives, in: Proc. of the IEEE, Vol. 80, 1992, pp. 1502–1571. 2, 4

[22] E. D. Boyer, L. Lister, B. Shader, Sphere-of-influence graphs using the sup-norm, Mathematical and Computer Modelling 32 (2000) 1071–1082. 2, 6

[23] T. S. Michael, T. Quint, Sphere of influence graphs and the l_∞-metric, Discrete Applied Mathematics 127 (3) (2003) 447 – 460. 2, 6

[24] R. C. Veltkamp, 3D computational morphology, Computer Graphics Forum (Proc. EUROGRAPHICS) (1993) 115–127. 2

[25] I.-K. Lee, Curve reconstruction from unorganized points, Computer Aided Geometric Design 17 (2) (2000) 161–177. 2, 3

[26] W. S. Cleveland, C. L. Loader, Smoothing by local regression: Principles and methods, in: W. Haerdle, M. G. Schimek (Eds.), Statistical Theory and Computational Aspects of Smoothing, Springer, New York, 1995, pp. 10–49. 3

[27] H. Wendland, Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree, Advances in Computational Mathematics 4 (1995) 389–396. 3

[28] W. Härdle, Applied nonparametric regression, Vol. 19 of Econometric Society Monograph, Cambridge University Press, New York, 1990. 3

[29] T. K. Dey, S. Goswami, Provable surface reconstruction from noisy samples, in: Proc. Symp. on Computational Geometry, 2004, pp. 330–339. 5

[30] T. L. V. Barnett, Outliers in Statistical Data, John Wiley and Sons, New York, 1994. 6

[31] T. Kanai, H. Suzuki, Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications, in: Proc. Geometric and Processing, 2000, pp. 241 – 250. 7

[32] J. Chen, Y. Han, Shortest paths on a polyhedron, in: Proc. 6th ACM Symp. on Computatinal Geometry, 1990, pp. 360 – 369. 7

[33] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: H. Maurer (Ed.), New Results and New Trends in Computer Science, Vol. 555 of Lecture Notes Comput. Sci., Springer-Verlag, 1991, pp. 359–370. 8

[34] S. Schönherr, Computation of smallest ellipsoids around point sets, Diploma thesis, Freie Universität Berlin, Berlin, Germany (Apr. 1994).
URL `http://page.mi.fu-berlin.de/~sven/own_work/diplom.abstract.html` 8

[35] C. Rogers, Covering a sphere with spheres, Mathematika 10 (1963) 157–164. 10

[36] D. Attali, J.-D. Boissonnat, A linear bound on the complexity of the delaunay triangulation of points on polyhedral surfaces, Discrete and Computational Geometry 31 (3) (2004) 369–384. 10

[37] T. M. Chan, J. Snoeyink, C. K. Yap, Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams, Discrete Comput. Geom. 18 (1997) 433–454. 10

[38] R. A. Dwyer, The expected size of the sphere-of-influence graph, Computational Geometry: Theory and Applications 5 (3) (1995) 155–164. 10

[39] O. Devillers, The Delaunay hierarchy, Internat. J. Found. Comput. Sci. 13 (2) (2002) 163–180. 10

[40] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, Journal of the ACM 45 (1998) 891–923. 10