Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

OLIVER SANDER, DANIEL RUNGE

Fast Surface Reconstruction Using a Probe Sphere

ZIB–Report 00-50 (December 2000)

Fast Surface Reconstruction Using a Probe Sphere

Oliver Sander, Daniel Runge

Abstract

We introduce a new method for reconstructing a triangular surface from an unorganized set of points in space. It is based on placing a probe sphere on the point set and rolling it around, connecting all triples of points that the sphere comes to rest on with a triangle. Therefore, the algorithm interpolates, rather than approximates, the input points. The method needs considerably less running time than previous algorithms and yields good results on point sets that are reasonably well-behaved.

1 Introduction

Turning a real-life object into a computer model is a frequently encountered problem. In such different areas as technical design and film-making, physical objects are created as a first step towards a computer model.

Digitizing these objects usually involves, as an intermediate step, a representation of the object as a set of points in space, possibly with no further information about their mutual relationship than their respective 3D positions. Such point sets can come from a wide variety of devices, such as laser range scanners, radar or seismic surveys, or contact probe digitizers, but also from purely mathematical models such as implicit functions.

The task of recreating a polygonal surface of the original object from the points alone is a nontrivial one. Two important criteria have to be met:

The first one is *robustness*. A reconstruction algorithm should provide good results under non-ideal conditions, i.e., on noisy or sparse data. Humans, when looking at a point set, usually have an immediate, intuitive idea of what the represented object looks like. One would wish an algorithm to come as close as possible to this human intuition, a task which obviously touches the field of artificial intelligence. However, this does not mean that a reconstruction algorithm has to produce a sensible output all the time, since there are point sets that do not represent any object surface, e.g., cloud-like distributions.

The second important factor is *speed*. With a perfectly robust algorithm, execution time would not matter much, since, in order to create a lot of models in a short time, a human operator would be able to start many processes at once, being assured that they will all produce good output without further assistance. In that case, high individual reconstruction times would not prevent a high productivity. However, all algorithms we have encountered so far need a certain amount of human assistance and postprocessing, which might even involve letting the algorithm run several times on slightly modified point sets. For this, interactive execution times would be desirable.

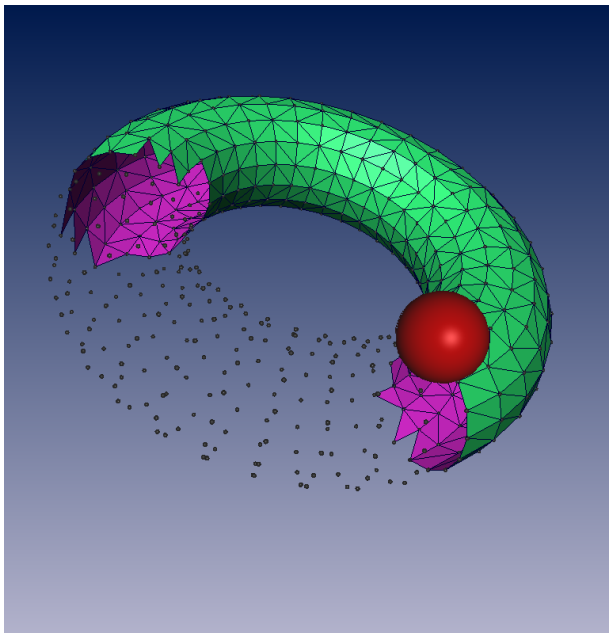

Figure 1: Reconstruction of a torus.

Existing algorithms for surface reconstruction usually belong to one of several families. Ours is a member of the family of *advancing-front algorithms*. Advancing front algorithms construct surfaces by trying to iteratively extend a partially constructed surface across the current boundary. It is one of the lesser used paradigms for surface reconstruction (see [5]). In comparison to existing methods, this paper introduces a new heuristic for extending the surface, which appears to be more robust than previous schemes.

Many situations exist where additional data other than the mere point positions is available. Laser scanners usually take several views of a model, hence the viewing directions associated to the different views constitute extra information that can be used by reconstruction algorithms. Some scanners even provide surface normals for each point. Another example is data taken from segmented CT images. There, sample points are arranged in slices and they form contours within each slice. Some algorithms focus specifically on this case. Nevertheless, since additional data does not always exist and is different depending on the data source, we focus on the general case where no information other than the geometric point positions is known.

2 Previous Work

In the past, a wide range of different approaches has been used to reconstruct surfaces.

One of the most commonly known methods are the α-*shapes* of Edelsbrunner and Mücke [8]. They start with the Delaunay tetrahedrization of the input point set. From the resulting set of tetrahedra, which fills the convex hull of the point set, they remove any tetrahedron, triangle, and edge whose circumsphere does

not fit into what they call an α-*ball*, i.e., a sphere with radius α. The remaining simplicial complex is termed α-*shape*. In order to extract a surface, a triangle is kept if an α-ball touching its three vertices does not contain any other point.

The algorithmic complexity of the α-shape construction is bounded from below by the construction of the Delaunay tetrahedrization, whose complexity is $O(n^2)$.

Despite its rather slow asymptotic behaviour, the Delaunay tetrahedrization is the starting point for a number of other reconstruction algorithms. Attali, for example, introduces the so-called *Normalized Meshes* [1], which are subgraphs of the Delaunay graph fullfilling a certain regularity condition. The approach of Isselhard et. al. [11] starts with the Delaunay triangulation and removes different types of tetrahedra until the final surface emerges. The γ-*indicator* of Veltkamp [16, 15] follows a similar idea, but sorts the removable tetrahedra according to a constantly updated weight function (the name-giving γ-indicator). Bajaj's α-*solids* [2, 3, 4], finally, are a variation on the standard α-shape approach. While α-shapes use eraser spheres at every point in space, the spheres are now applied from outside the convex hull.

All these algorithms have in common, that they reconstruct surfaces by removing elements from the full Delaunay triangulation according to different heuristics.

Another class of algorithms uses signed distance functions in order to extract a surface. Examples are the works of Hoppe [10, 9] and Curless and Levoy [7]. The idea is to create a function that for each point in space gives the distance to the surface. This function can be constructed by approximating the surface around a point by least-squares fitting a tangent plane in its k-neighborhood. The set of these planes, properly oriented, is combined to give the distance function. A standard marching cubes algorithm is then used to extract the 0-isosurface of this function. Curless and Levoy have more recently tuned their algorithm for laser-range data, using error and tangent plane information. They also add a hole-filling step, which makes their program even more robust.

Boissonnat's algorithm [5] is one of the few incremental surface-based constructions, and hence in the same family as our own method. It starts with a first heuristically picked edge. The neighborhood of that edge is then approximated by a tangent plane, and all neighboring points are projected into that plane. The point whose projection sees the edge under the largest angle is chosen as the next point to be inserted.

A more extensive description of surface reconstruction algorithms can be found in the survey article of Mencl and Müller [13].

3 The Algorithm

The basic idea of our algorithm stems from computer chemistry. One standard way to visualize large molecules is the *solvent excluded surface (SES)*. It is constructed by assuming the atoms in the molecule to be spheres of possibly different sizes. A solvent, also approximated by a sphere, is then moved around the molecule. The SES is the boundary surface between the region in space that is accessible to the probe sphere and the rest. (For more on the construction of the SES, see e.g. [14].) Shrinking all atoms to zero radius leads directly to our algorithm.

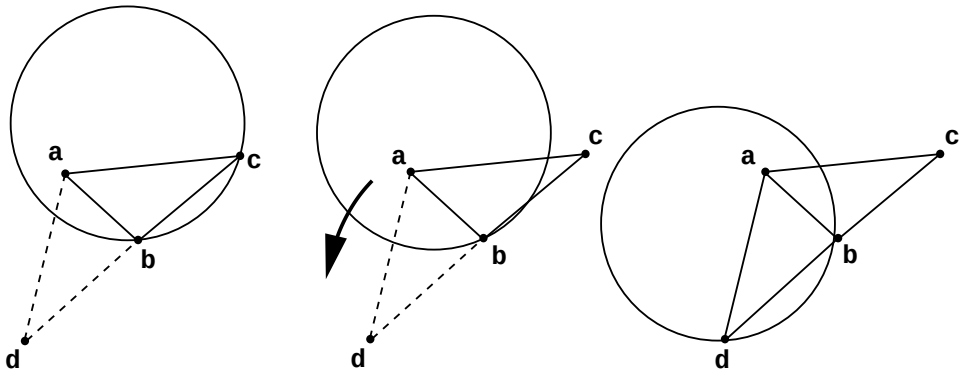

Figure 2: *Finding a new Triangle: i) the sphere rests on points a, b, c; ii) it rolls over edge (a, b); iii) it is stopped by point d. Points a, b, d are added as a new triangle.*

3.1 Outline

Picture the points as little balls (with zero diameter), hovering in space on their respective, fixed positions. Picture now a sphere of a fixed radius, approaching the cloud of points. Eventually, it will touch (at least) three points at once. We assume now that these three points are adjacent to each other on the surface to be constructed. The assumption rests on the fact that a sphere approaching a solid object will eventually be stopped by points on the surface of the object.

Imagine now that the sphere starts rolling (see Fig. 2). It keeps in touch with two of the three points it rests on, and turns away from the third. It will eventually touch a new point. (This new point might be the old point, touched from the other side. In this case we assume that we have found a boundary and continue our search somewhere else.) We again add the three resting points as a triangle to the output surface and repeat the process recursively.

Several observations follow from the description above:

- The algorithm is *interpolating*, which means that the input points actually become vertices of triangles in the output triangle mesh. For an example of a reconstruction algorithm that is instead *approximating*, see Hoppe [10, 9].
- The output surface is connected. Therefore, the algorithm cannot correctly reconstruct surfaces that consist of several different connected components. In order to find all 'parts' of a surface, one might consider restarting the algorithm on the points left over from a successfull reconstruction.
- Also, it will not find cavities within solids. This, however, doesn't seem to be a serious drawback at the moment, since a lot of input data is provided by laser scanners, which don't detect cavities either.
- By definition, the output surface is a subset of the α- shape of the point set for the given radius. However, the running time is asymptotically better than the $O(n^2)$ needed for the 3D Delaunay triangulation involved in α-shape construction. (See Section 4.1 for more details on the algorithmic complexity of our scheme.)

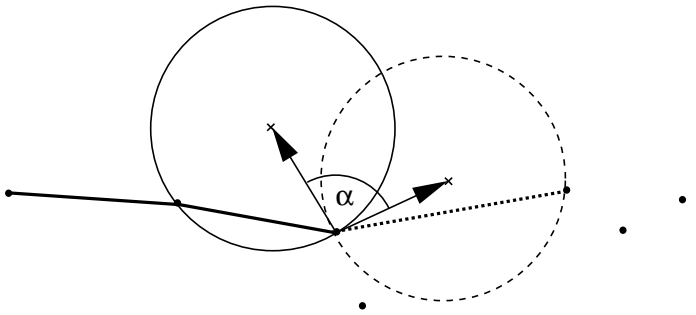

Figure 3: *When rotating the probe sphere around an edge, the point that minimizes α is chosen.*

The basic algorithm is as follows:

1. *Find a good starting triangle (see below) and add it to the output surface.*
2. *Place the three edges of that triangle in a job queue.*
3. *For each edge (v1, v2) in the queue do:*
   - *If the edge is already connected to another triangle, return.*
   - *Compute the position of the center of the sphere sitting on the triangle.*
   - *For each point within reach do:*
     - *Compute the position of the center of the sphere sitting on v1, v2 and the new point.*
   - *Choose the new point such that the angle at the midpoint of (v1,v2) between the old and the new probe center position becomes minimal. (See Fig. 3.)*
   - *Add v1 and v2 together with the new point as a new triangle.*
   - *Place the two new edges in the job queue.*

3.2 Finding Candidate Points

One of the most important problems in this algorithms is how to find, for a given edge, a list of *candidate points*, i.e., a list of points that are close enough to be touched by the rotating sphere. This problem is an instance of what are called *range-searching problems*. The general form of a range-searching problem is: given a set of points in $\mathbb{R}^d$ and a subset Ω of $\mathbb{R}^d$, find all those points that are in Ω [12]. Solutions for this problem obviously depend on the shape of the query set Ω, in our case a degenerated torus. Even though many sophisticated algorithms exist, we found a very simple one to work very well in practice.

We subdivide the bounding box of the point set into a uniform array of cubes. The axes of the cubes are parallel to the coordinate axes, and the edges of the cubes have the same lengths as the probe sphere radius. Each cube holds a list of the points it contains. Given a query position and a query radius,

the data structure returns a list of pointers to all cubes that have nonempty intersection with the query sphere.

In general, given the nature of the original problem, many of those cubes will be empty, and maintaining a full three-dimensional array is a waste of memory. If memory is scarce, a hash table should be used. However, we hardly ever found ourselves working with more than 15.000 cubes in total, making the memory waste negligeable. Therefore, we preferred the speed of index computation and the ease of implementation of a simple array to the more complex hash table.

3.3 Finding a First Triangle

A problem of special importance is the task of finding an initial triangle. If the algorithm errs on the first triangle, it is highly unlikely that the subsequent steps will lead to a decent result. However, the direct implementation of our physical paradigm (the probe sphere) doesn't quite work. The problem is that we don't know from where to let our sphere drop. It might just miss the point set completely.

Therefore we deviate a little from physical intuition and use the following approach: as a first point, we pick the point that is extreme along one coordinate axis. From the points in the vicinity of that first point, we choose the one that is extreme along the same axis. Note that this point is not necessarily the globally secondmost extreme point. Care has to be taken in case of several points having the same coordinate value. In order to find the third point, we place the probe sphere on the edge spanned by these two points, and have it rotate around it, much like in the general case. We choose the point that yields the lowest probe sphere center along the previously chosen axis.

A particularly delicate problem arises when reconstructing bounded surfaces. If, by accident, the above described method puts the probe sphere *into* the model, it will start rolling within it, returning undesired results in all but the most ideal cases. Unfortunately, since the terms *inside* and *outside* are topologically undefined when dealing with bounded manifolds, it is impossible to invent a perfect algorithm for this special case.

4 Results

4.1 Theory

The algorithm is basically a breadth-first search on the surface graph of the output surface. It is therefore linear in the number of output triangles [6]. As the number of triangles in a surface is a linear function of the number of the vertices, the algorithm is also linear in the size of the input point set. For each triangle added to the surface, a number of candidate points have to be checked. Since the main part of the work is the computation of the angle between two probe positions, the number of these canditate points is crucial for efficiency. Each time, we only consider points in the vicinity of the sphere, therefore, the number of candidate points, and therefore speed, is strongly dependent on the size of the probe sphere. It is actually cubic in the sphere radius.

In terms of algorithmic complexity, if the probe size is chosen small compared to the total dimensions of the point cloud, (as should normally be the case)

	Points	Triangles	Mode	Time (sec)
Torus	9216	18432	fixed	1.2
Aorta	37689	47547	fixed	21.7
Bunny	67038	134034	fixed	17.4
Bunny	67038	134053	adaptive	9.5
Paul	3157	6000	fixed	0.7

Table 1: *Exemplary running times*

we can assume that the number of points checked when handling an edge is constant. At the other end of the scale, if the sphere is enlarged to about the same size as the point set, each point gets checked every time, i.e., the asymptotic running time is more like $O(TN)$, N the number of output triangles and T the number of sample points. If the probe sphere is chosen a lot larger than the point set, we obtain a wrapping-type algorithm for the complex hull.

The algorithm uses only constant amount of storage in addition to the input point set and the output surface.

4.2 Reality

We have tested the program on several data sets of different sizes, using an SGI O^2 equipped with a MIPS R10000 at 250 MHz and 256 Mbytes of main memory. We achieved good result on all four of them within very short time.

Our first example is the torus from Figure 4. It was constructed analytically, therefore, the vertices are evenly spaced and there's no noise at all. As was to be expected, the reconstruction was straightforward.

We chose the bunny from the CyberWare Scanners homepage as an example for a very large dataset. Its sampling quality is again very good. Using the adaptive probe size modification described below, we achieved a near-perfect reconstruction on the first try.

Another real-life example, the blood vessel shown in Figure 6 is quite noisy. In order to reconstruct the dataset, some manual preprocessing was necessary. We 'plugged' the lower end of the vein (which was originally open) with some extra points, to prevent the probe sphere from entering the model.

The last example, the front part of a face, shows that our algorithm can also recognize boundaries quite well, as long as the sampling quality is good.

All model sizes and reconstruction times can be seen in Table 1.

5 Outlook/Research Directions

5.1 Adaptive Probe Size

One major weakness of the proposed algorithm is the fixed probe size. As the amount of detail is usually not constant all over the object (consider the model of a human skull, with eyes, nose and mouth on the front, but a more or less featureless surface on the back) it makes sense to award a high sampling density to detailed regions while having fewer points in flat areas. However, in order to obtain good results from the algorithm, the probe size must always be chosen large enough to bridge the gap between the two farthest neighboring points.

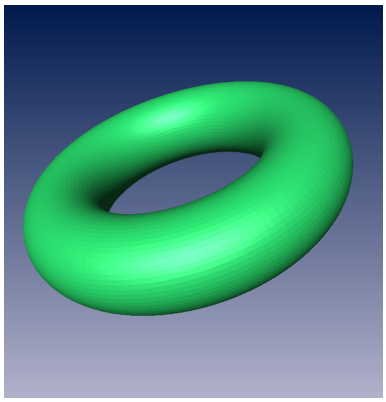

Figure 4: *The Torus*

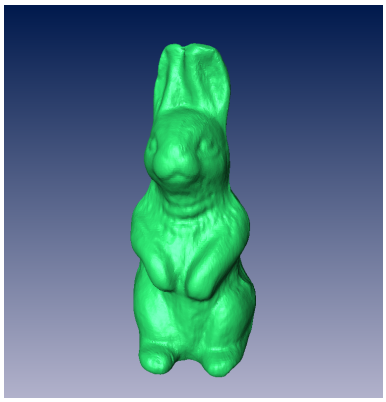

Figure 5: *The Bunny*

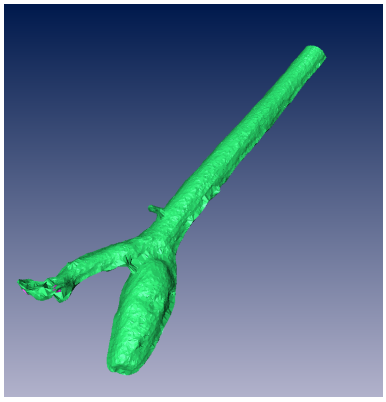

Figure 6: *The Aorta*

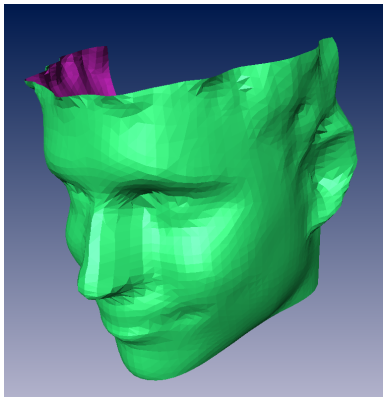

Figure 7: *Paul*

This, however, may prevent the sphere from penetrating into small cavities or depressions elsewhere, thus not revealing the full level of detail.

At first glance, the solution seems to be an adaptive probe size. We have tried making the sphere size proportional to the length of the edge it turns around. However, while this approach at least partly yields the desired effect (and reduces the running time, because the sphere is smaller on average than in the fixed mode) robustness suffers, especially on noisy data. Cause for this is not so much the way the sphere size is chosen, but the fact that it changes at all, introducing inconsistencies into the system (e.g. the sphere might 'swallow' points while changing size). Different approaches, like having the size change according to some form of local point density, don't avoid this problem, either.

5.2 A Better Data Structure

A way to further reduce the algorithm's running time might be a different data structure for holding the sample points. At the moment, a three-dimensional array of 'cubes' is used, each 'cube' referencing a list of the points it contains (see Section 3.2). The main advantage here is clearly the trivial implementation, however, it's not very memory efficient.

Since the sphere revolving around two points describes a degenerate torus, it would be nice to have a data structure that effectively supports toroidal range queries. While this seems too outlandish to ever have attracted someone's attention, effective structures for spherical queries exist, and could serve as a first approximation. For more detailed information on range searching solutions, see [12].

6 Summary and Conclusion

We have presented a new algorithm for the reconstruction of surfaces from unorganized point sets. It is based on the construction of *solvent excluded surfaces* known from the visualization of molecules, namely having a probe sphere 'roll over' the point set and incorporating all point triples into the surface that the sphere gets to rest on. This method is considerably faster than previous approaches. Even very large data sets are usually processed in a matter of seconds, making the algorithm suitable for interactive use.

References

[1] Dominique Attali. r-regular shape reconstruction from unorganized points. In *ACM Symposium on Computational Geometry*, pages 248–253, 1997.

[2] Chandrajit Bajaj, Fausto Bernadini, and Guoliang Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. In *Siggraph Proceedings*, pages 109–118, 1995.

[3] Chandrajit Bajaj, Fausto Bernadini, and Guoliang Xu. Reconstructing surfaces and functions on surfaces from unorganized 3d data. *Algorithmica*, 19:243–261, 1997.

[4] Fausto Bernadini and Chandrajit Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *Proc. of the Ninth Canadian Conference on Computational Geometry*, pages 193–198, August 1997.

[5] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, October 1984.

[6] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[7] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Siggraph Proceedings*, 1996.

[8] Herbert Edelsbrunner and Hans Peter Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43–72, 1994.

[9] Hugues Hoppe. *Surface Reconstruction from Unorganized Points*. PhD thesis, University of Washington, 1994.

[10] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Siggraph Proceedings*, pages 71–78, 1992.

[11] Frank Isselhard, Guido Brunnett, and Thomas Schreiber. Polyhedral reconstruction of 3d objects by tetrahedra removal. Technical Report 288/97, Fachbereich Informatik, University of Kaiserslautern, Germany, 1997.

[12] Jirí Matoušek. Geometric range searching. *ACM Computational Survey*, 26:421–461, 1994.

[13] Robert Mencl and Heinrich Müller. Interpolation and approximation of surfaces from three-dimensional scattered data points. Technical report, Department of Computer Science, University of Dortmund, 1997.

[14] Daniel Runge. Algorithms and methods for the visualization of molecular surfaces and interfaces. Master's thesis, Humboldt-University of Berlin, 1999.

[15] Remco C. Veltkamp. Closed object boundaries from scattered points. In *Lecture Notes in Computer Science 885*. Springer Verlag, 1994.

[16] Remco C. Veltkamp. Boundaries through scattered points of unknown density. *Graphics Models and Image Processing*, 57(6):441–452, 1995.