A Fast and Efficient Projection-Based Approach for Surface Reconstruction

M. GOPI[1], SHANKAR KRISHNAN[2]

[1]Univ. of California, Irvine
gopi@ics.uci.edu

[2]AT&T Labs - Research
krishnas@research.att.com

Abstract.

We present a fast and memory efficient algorithm that generates a manifold triangular mesh S passing through a set of unorganized points $P \subset \mathcal{R}^3$. Nothing is assumed about the geometry, topology or presence of boundaries in the data set except that P is sampled from a real manifold surface. The speed of our algorithm is derived from a projection-based approach we use to determine the incident faces on a point. Our algorithm has successfully reconstructed the surfaces of unorganized point clouds of sizes varying from 10,000 to 100,000 in about 3–30 seconds on a 250 MHz, R10000 SGI Onyx2. Our technique can be specialized for different kinds of input and applications. For example, our algorithm can be specialized to handle data from height fields like terrain and range scan, even in the presence of noise. We have successfully generated meshes for range scan data of size 900,000 points in less than 40 seconds.

1 Introduction

The surface reconstruction problem can be loosely stated as follows: *Given a set of points P which are sampled from a surface in $\mathcal{R}^3$, construct a surface S so that the points of P lie on S*. A variation of this *interpolatory* definition is when *S approximates* the set of points P.

Surface reconstruction has wide ranging applications including scanning complex 3D shapes like objects, rooms and landscapes with tactile, optical or ultrasonic sensors are a rich source of data for a number of analysis and exploratory problems. Surface representations are a natural choice because of their applicability in rendering applications and surface-based visualizations (like information coded textures on surfaces). The challenge for surface reconstruction algorithms is to find methods which cover a wide variety of shapes. We briefly discuss some of the issues involved in surface reconstruction.

We assume in this paper that the inputs to the surface reconstruction algorithm are sampled from an actual surface (or groups of surfaces). A proper reconstruction of these surfaces is possible only if they are "sufficiently" sampled. However, sufficiency conditions like sampling theorems are fairly difficult to formulate and as a result, most of the existing reconstruction algorithms ignore this aspect of the problem. Exceptions include the work of [Att97, ABK98].

If the surface is improperly sampled, the reconstruction algorithm can produce artifacts. A common artifact is the presence of spurious surface boundaries in the model. Manual intervention or additional information about the sampled surface (for instance, that the surface is manifold without boundaries) are possible ways to eliminate these artifacts. The other extreme in the sampling problem is that the surface is sampled unnecessarily dense. This case occurs when a uniformly sampled object with a few fine details can cause too many data points in areas of low curvature variation.

Sometimes the input data might contain additional information for easier reconstruction. For example, in data from laser scanners that generate samples uniformly on a sphere (or a cylinder, depending on its degrees of freedom), adjacent data points have a very high probability of being adjacent to each other in the final mesh. We refer to these data sets as *organized point clouds*. This information can be exploited by some algorithms, including ours, to give quick results.

Another issue in surface reconstruction is the presence of noise and outliers in the original data. The mode of data acquisition has a direct impact on this. For example, range scan data can be very noisy when the surface is not oriented transverse to the scanning beam. Noisy data introduce high frequency artifacts in the reconstructed surface (like microfacets) and this is a cause of concern for many algorithms.

Finally, the recent thrust in research to build augmented reality and telepresence applications has introduced an interesting variation of the surface reconstruction problem. Consider an application where multiple cameras or camera-projector pairs are used to extract the geometry of dynamic scenes at interactive rates[RWC+98]. In this scenario, the surface reconstruction algorithm should be able to handle

extremely large data sets (order of many millions of points) and provide a suitable surface representation without significant latency. One of the main motivations for this work is to develop an approach that handles both *organized* and *unorganized* point clouds very efficiently in time and memory requirements.

1.1 Main Contributions

In this paper, we present a fast and efficient projection-based algorithm for surface reconstruction from unorganized point clouds. Our algorithm incrementally develops an interpolatory surface using the local estimated surface orientation of the given data points. The main contributions of this paper include:

- **Asymptotic Performance:** Each iteration of our algorithm advances the reconstructed surface boundary by choosing one point on it and completes all the faces incident on it in constant time. Even though the worst case theoretical run-time complexity is $O(n \log n)$, in practice it exhibits linear time performance with a very small constant of proportionality.

- **Speed:** We have tested our algorithm on a number of data sets ranging from 10,000 to 100,000 unorganized points. It takes about 3–30 seconds to reconstruct the mesh. We have also tested our algorithm on an organized point cloud of size 6.5 million. After simplifying this data to around 900,000 points, it took us about 40 seconds to generate the mesh on a 250 MHz, R10000 SGI Onyx2 with 16 GB of main memory.

- **Memory efficiency:** Our algorithm has minimal memory overhead because it goes through a single pass of all data points to generate the mesh. We do not maintain the computed triangles in our data structure because our method does not revisit them. Only the input data has to be stored.

- **Robustness:** In the special case of terrain data or data from common center-of-projection scanning devices, our algorithm can tolerate high noise levels. The error introduced by noise has to be bounded, however.

2 Previous Work

The problem of surface reconstruction has received significant attention from researchers in computational geometry and computer graphics. In this section, we give a brief survey of existing reconstruction algorithms. We use the classification scheme of Mencl et. al. [MM98] to categorize the various methods. The main classes of reconstruction algorithms are based on *spatial subdivision*, *distance functions*,and *incremental surface growing*.

The common theme in spatial subdivision techniques is that a bounding volume around the input data set is subdivided into disjoint cells. The goal of these algorithms is to find cells related to the shape of the point set. The cell selection scheme can be surface-based or volume-based.

The surface-based scheme proceeds by decomposing the space into cells, finding the cells that are traversed by the surface and finding the surface from the selected cells. The approaches of [HDD$^+$92, EM94, BBX97, Att97] fall under this category. The differences in their methods lie in the cell selection strategy.

The volume-based scheme decomposes the space into cells, removes those cells that are not in the volume bounded by the sampled surface and creates the surface from the selected cells. Most algorithms in this category [Boi84, Vel95, ABK98] are based on Delaunay triangulation of the input points.

The distance function of a surface gives the shortest distance from any point to the surface. The surface passes through the zeroes of this distance function. This approach leads to approximating instead of interpolatory surfaces [HDD$^+$92, CL96].

The basic idea behind incremental surface construction is to build-up the surface using surface-oriented properties of the input data points. The approach of Mencl and Muller [MM98] use graph-based techniques to complete the surface. Boissonnat's surface contouring algorithm [Boi84] starts with an edge and iteratively attaches further triangles at boundary edges of the emerging surface using a projection-based approach to generate manifolds without boundaries. The Spiraling-Edge triangulation technique proposed by Crossno and Angel [CA97] is similar to our algorithm. Differences include the fact that they make several limiting assumptions about the data, including normal and neighborhood information for each point. Bernardini et. al. [BMR$^+$99] describe a ball-pivoting algorithm to grow the surface locally. Gopi et al. [GKS00] use localized Delaunay triangulation to compute the final neighborhood in the triangulation.

3 Algorithm Overview

The input to our algorithm is a set of unorganized points with no additional information (like normals). The output is a triangulated mesh which interpolates the input point set. Our algorithm starts at a data point, and finds all its incident triangles. Then each of its adjacent vertices in the boundary of the triangulation is processed in a breadth-first fashion to find their other incident triangles. Thus the boundary of the completed triangulation propagates on the surface of the point cloud till it processes all the data points. In the rest of the paper, we refer to the point being processed as the *reference point*, R.

There are three assumptions we make about the data set. The sampling of the data is *locally uniform*, which means that the distance ratio of the farthest and closest neighbor of a sample in the given sampling of the object is less than a constant value. The second assumption is to distinguish points from two close layers of the object. The closest distance between a point P in one layer and another layer is at least μm, where μ is a constant and m is the shortest distance between P and another point in its layer. The third assumption is about the smoothness of the underlying object. The normal deviation between the any two triangles incident on a vertex should be less than 90°. This assumption is used in justifying our choice of tangent plane in section 4.1.

Our algorithm can be broadly divided into three stages: *bucketing*, *point pruning*, and finally the *triangulation step*.
Bucketing: In this stage, the data structure is initialized with the input data. Our data structure is a depth pixel array similar to the *dexel* structure [Hoo86]. We maintain a 2D pixel array into which all data points are orthographically projected. The points mapped on to the same pixel are sorted by their depth (*z*) values.

Point Pruning: This step is similar to clustering algorithms used by other triangulation schemes [HDD+92]. We first apply a *distance criterion* to prune down our search for candidate adjacent points in the spatial proximity of R. It is executed in two stages. In the first stage, the simpler L_∞ metric is used to define the proximity around R. Our algorithm takes an axis-aligned box of appropriate dimensions centered at R and returns all the data points inside it. The major difference in our approach compared to other approaches is the use of *dexel* like data structure for this stage. By using our data structure, this search is limited to the pixels around the pixel where R is projected. Another advantage of this data structure is explained in Section 5 where the information about the characteristics of the scanning device, used for collecting the data points, is used to improve the robustness of the algorithm. The second stage of pruning uses the Euclidean metric, which further rejects the points that lie outside a *sphere of influence* centered at R. The choice of the box dimensions and the radius of the sphere are described in the next section. The points chosen after the pruning using the Euclidean metric are called the *candidate points* of R, C_R.

Visibility Criterion: Next, we estimate the tangent plane at R, and project R, C_R, and the mesh boundary in their vicinity on this tangent plane. The projected points of C_R are then *ordered by angle* around R. Points in C_R that are occluded from R by the mesh boundary in the projection plane are removed.

Angle Criterion: This is an optional step, which tries to remove "skinny" triangles at R, to improve the quality of triangulation.

Triangulation: Finally, the remaining points in C_R are then connected in order around R to complete the triangulation.

4 Surface Reconstruction

In this section, we describe our approach to surface reconstruction in detail. The output of our algorithm is an interpolatory, non-self-intersecting triangular mesh of the given point cloud.

The implicit function theorem of smooth surfaces forms the basis of our approach. Without loss of generality, it states: *"Given an implicit surface $S \equiv f(x, y, z) = 0$, and a point P on it, such that the tangent plane to S at P is parallel to the (x, y) plane, then S in the neighborhood of P can be considered as a height function $f(x, y, h(x, y)) = 0$, a local parameterization on its tangent plane"*. By a suitable rigid transformation of the coordinate frame, any other point on S can be made to satisfy the above theorem.

Our algorithm is a greedy method and works with two parameters: μ, which quantifies our definition of *locally uniform sampling*, and α, which gives a lower bound on the angle between consecutive neighbors of a point on a boundary of the surface. Typically, α is a large obtuse angle. In our implementation, we have set α to be 120°. All other parameters, which are required for the implementation of the algorithm are derived from μ. In order to improve the quality of triangulation, we can optionally specify a minimum angle parameter, β. It is not necessary for the completion of our algorithm, though.
Terminology: We categorize the data points at any given stage of our algorithm as *free*, *fringe*, *boundary* and *completed* points. The *free* points are those which have no incident triangles. The *completed* points have all their incident triangles determined. Points that lie along the current surface boundary are either *fringe* or *boundary* points. *Boundary* points are those points which have been chosen as a reference point but have some missing triangles due to the maximum allowable angle parameter α. *Fringe* points have not yet been chosen as a reference point.

We maintain two invariants during our algorithm's execution:
Invariant 1: No *free*, *fringe* or *boundary* point can be in the interior of a triangle (because of our distance criterion).
Invariant 2: At the end of each iteration, the point chosen as the reference point becomes a *completed* or a *boundary* point. This is used later to prove claims about occluded points (for visibility criterion).

Our algorithm starts with the bucketing step by orthographically projecting the data points onto the *dexel* data structure. The following steps are used to choose the right set of points to be connected to the reference point R.

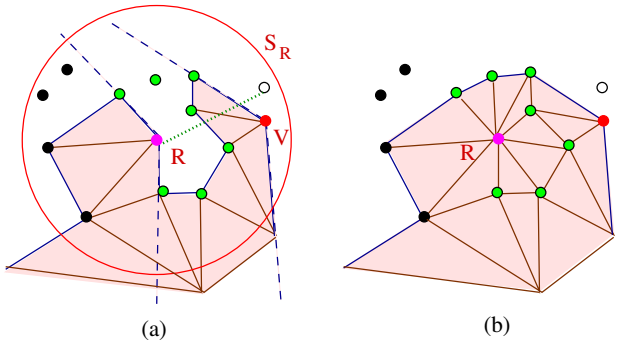


Figure 1: (a) Visibility test around R. The black points are behind R's boundary edges, the white points are occluded by other edges, and the point V is eliminated as R is behind its boundary edges. (b) Completed mesh at R

4.1 Point Pruning

Pruning by Distance Criterion: Points far away from the reference point R are not likely to be adjacent to it. We eliminate them by applying the distance criterion in two stages. Initially, we employ the cheaper L_∞ metric to narrow down our search. It is performed by constructing an axis-aligned box of suitable dimension around R and choosing all the *free*, *fringe* and *boundary* points inside the box. By using our *dexel* array, this is a logarithmic time operation with small constant.

The dimension of the box is derived from μ as follows. In a general case, R (a *fringe* point) already has a few incident triangles. Let m be the minimum distance from R to its existing adjacent vertices. From our definition of *locally uniform sampling*, the farthest neighbor of R can be at most μm away. This gives an estimate on the dimension of the box. When R has no incident triangles (for example, at the very beginning), we find the closest point to R using the *dexel* array representation and find m. The minimum distance between the points in the above box and R, refines the previous estimate of m. Using this new m, the next step further prunes the chosen set of points in the proximity of R.

We call a sphere of radius μm centered at R as the *sphere of influence* (S_R) around R. The second stage of pruning uses a stricter L_2 metric and returns all points inside S_R. These points are the *candidate points* (C_R) of R. We would like to make an observation about the candidate point set. The radius of S_R is dependent on m, which changes from one vertex to another. Therefore, it is possible that a vertex p might be in the *sphere of influence* of R, but not vice-versa. But this asymmetry does not affect the topology of the reconstructed mesh.

Choice of Projection Plane: The triangulation around R implicitly defines an ordering of its adjacent vertices around R on a projection plane. We find this ordering directly by projecting C_R on a plane. The choice of the projection plane is an important issue, and dictates the robustness of our algorithm. According to the implicit function theorem, the best projection plane would be the tangent plane at R. One can adopt more robust algorithms like the one described in Hoppe et.al [HDD+92] or Amenta et.al. [ABK98]. An alternate cheaper approach to compute the projection plane normal is by averaging normals of existing triangles incident on R. Since we are interested only in the relative ordering of points around R, we use this latter approach in our implementation. The ordering of the *candidate* points (C_R) around R in this plane will be incorrect only if there is a triangle incident on R with its normal deviating by more than 90° from the projection plane normal. Our choice of projection plane is justified by our assumption about the smoothness of the underlying object. We assume that the object from which the input is sampled is smooth enough so that the variation in the tangent planes for proximate points is very small.

Angle Ordering: A main step in our algorithm is to order points in C_R projected on P_R by angle around R. We now describe a fast and inexpensive method to perform this ordering.

We define a new local coordinate system where the reference point R is the origin and P_R is the xy-plane. Initially, each candidate point of R is projected on P_R in this coordinate system. Let this set of projected candidate points be C_R^p. The ordering around R is based on the angle (θ) between the x-axis of the local coordinate system and the vector from origin to the projected candidate point.

The set C_R^p is partitioned by the quadrants in which they lie. In each of these quadrants we order the points based on $sin^2(\theta)$. We use $sin^2(\theta)$ because it is almost linear within a quadrant and is inexpensive to compute. The actual angle θ in the projection plane is computed using a look-up table and a simple linear interpolation. We now order the points within each quadrant and finally merge these four ordered sets. We use the actual angle to identify holes, boundaries, and skinny triangles in the model.

Pruning by Visibility: We use the angle ordering of C_R^p to efficiently perform the next stage of pruning based on visibility in the plane P_R. It eliminates the points which potentially form a self-intersecting mesh. We define the *boundary edges* of a point as the set of edges incident on that point that lie on the current surface boundary. Any edge with no triangle formed on one of its side, is a *boundary edge*. All *boundary edges* connect *fringe* and/or *boundary points*. On the other hand, *internal edges* are the edges which connect *completed* points with any other point. We project R, C_R, and their *boundary edges* on the plane P_R. If the line of

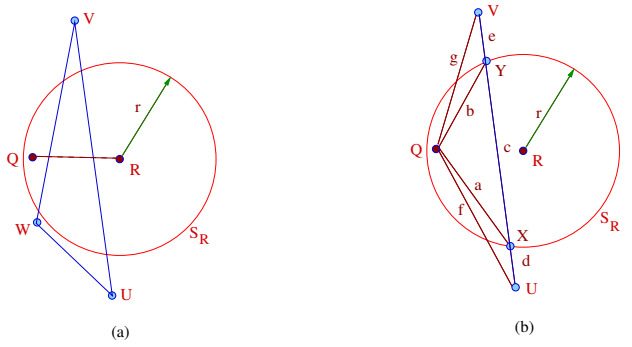


Figure 2: (a) Determining occluding edges (b) Angle XQY is obtuse, so $a < c$. Further, $f < d+a < d+c < d+c+e$. Therefore, $|QU| < |UV|$.

sight from R to a projected candidate vertex is obstructed by any edge, then that point is an occluded point. The existence of visibility between these points in the plane is a sufficient but not a necessary condition for the visibility between them in the object space. In the limit, when the local surface approaches the tangent plane in a densely sampled point cloud, it becomes a necessary condition as well. We take a conservative approach and prune all the points in C_R which are occluded from R on P_R.

Points occluded from R are determined as follows.

1. All the points between consecutive *boundary edges* of R (shown by the dotted-line wedge at R in Figure 1(a)) are removed as they cannot be visible from R. They are said to be in the *invisible* region of R. The black points in the figure are examples.

2. Similarly, points are removed which have R in their invisible region (for example, point V in the same figure). We denote the set of points from C_R remaining after this step as C_R^v.

3. Finally, we eliminate points that are occluded from R because of an existing edge in the mesh (for example, the white point in Figure 1(a)).

A straightforward approach of checking all possible occluding edges is very expensive. We state the following theorem which limits our search to very few edges. This theorem is true under the assumption about the smoothness of the underlying object.

Theorem 4.1 *Only the* boundary edges *of the points in the set C_R can be possible occluding edges between R and C_R^v.*

Proof: From Invariant 1, it is easy to show that if an internal edge is occluding, there must be at least one boundary edge which is also occluding. This eliminates all the internal edges from our consideration. Figure 2(a) shows an example where boundary edges (like UV or/and VW) occlude the point Q from R, but its endpoints are not in C_R. Given that at least one of UV or VW is a *boundary* edge, it must be part of some existing triangle (like UVW in the figure). This implies that one of U, V or W (let us assume U) must have already been chosen as a reference point. We have to prove that if any one of these points was an earlier reference point, then it should have chosen either Q or R (or both) as its neighbor(s). If it had chosen Q (resp. R) as its neighbor, then R (resp. Q) will lie on Q's (resp. R's) invisible region, and Q would be eliminated from C_R^v.

In the Figure 2(b), let us choose one of the edges, say UV, to complete our proof. Let X and Y be the intersection points of this edge with the projection of S_R. $\angle XQY$ is obtuse, because angle subtended by the diameter on the circumference is a right angle, and $\angle XQY$ is clearly greater than that. Hence the distance $c = |XY|$ is the longest edge of the $\triangle XQY$, which means that $a < c$ and $b < c$. Hence $(d+a) < (d+c) < (d+c)+e$, and by triangle inequality, $|UQ| = f < (d+a) < (d+c)+e = |UV|$. Similarly we can prove that $|VQ| < |VU|$. This argument extends to any edge that is placed similar to UV.

From our distance criterion, we claim that vertex Q must be adjacent to U. With Q as its neighbor, U completes its triangulation by adding edges QV and QW. This implies that R lies in the invisible region of Q, and hence cannot belong to C_R^v as it will be eliminated by condition 2 above. Therefore, UV cannot be an occluding edge. □

The rest of the points which are ordered by angle around R can be triangulated as shown in Figure 1(b).

Pruning by Angle Criterion: The triangulation we get from the previous step is a valid one. However, to improve the quality of triangulation, this pruning step removes points that could potentially form triangles with very small angles ("skinny" triangles). This is not a necessary component for the working of our algorithm. Since our algorithm does not introduce additional (*Steiner*) points, it cannot always achieve the desired quality. It is a greedy approach, which would eliminate sliver triangles whenever possible.

We explain the working of this step using an example. In Figure 3, consider the points N_1 and N_2. Let us assume that the angle at R of $\triangle RN_1N_2$ is less than β (the minimum angle parameter). One of these points can be removed to improve the triangulation. The choice of the removable vertex is not arbitrary. For example, if N_2 is rejected, it gets trapped inside the triangle (in the projection plane) formed by R, N_1, and any one of N_3, N_4, or N_5. This violates Invariant 1.

The following algorithm describes a way to avoid such scenarios and to form a good triangulation whenever possible. Assume that we have to complete the triangulation

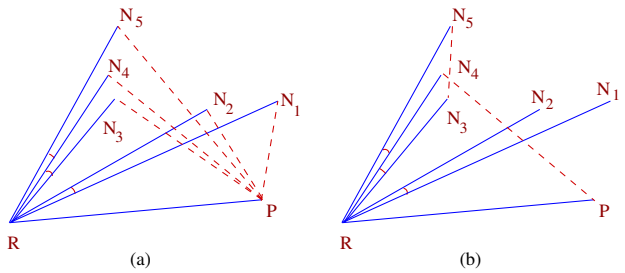


Figure 3: Pruning by Angle Criterion: (a) Ordering around R and P; angles between N_1N_2, N_3N_4, and N_4N_5 are less than β. (b) N_3 trapped in $\triangle RPN_4$, and N_4 trapped inside $\triangle RN_3N_5$

from P to N_5 around R in Figure 3, where RP and RN_5 are consecutive *boundary edges* of R. We start our processing by ordering the points around P. In our example, this ordering would be $P_s = (N_1, N_2, N_5, N_4, N_3)$, and the ordering around R is $R_s = (N_1, N_2, N_3, N_4, N_5)$. Let $P_s[i]$ ($R_s[i]$, respectively) be the i^{th} element in P_s (R_s, respectively). Without loss of generality, we assume $R_s[i] = N_i$. The following pseudo-code finds all possible adjacent points to P around R, without trapping any other point inside the triangle.

for $1 \leq i \leq |P_s|$
  Let N_j be the vertex in $P_s[i]$
  Mark N_j as **considered**
  $T(N_j)$ = $\{N_k \mid k < j$, N_k is not marked **considered**$\}$
  if ($T(N_j) = \emptyset$)
    then N_j **can be an adjacent point** to P around R
    else N_j **cannot be an adjacent point** to P around R

In our example, we first select $P_s[1] = N_1$. Since there is no N_k such that $k < 1$ and N_k is not marked **considered**, N_1 is a possible adjacent point to P around R. This is also true for $P_s[2] = N_2$. Now consider $P_s[3] = N_5$. Here, we have two points N_3 and N_4 which are not marked **considered** and hence belong to the set $T(N_5)$. We can see that N_3 and N_4 are inside the $\triangle RPN_5$. Therefore N_5 cannot be an adjacent point to P around R. In the general case, the set $T(N_j)$ consists of precisely those vertices that will be trapped if N_j were chosen as the adjacent point to P around R. If we complete the above algorithm N_3 will also be chosen as a possible adjacent point.

From the set of possible adjacent points $\{N_1, N_2, N_3\}$, we can choose any vertex. For the sake of argument, let us choose N_3 as the adjacent point to P and form the triangle RPN_3. Now the same algorithm is applied at N_3, and the points N_4 and N_5 are ordered around it. It can be seen that the point N_4 cannot be removed, as it will get trapped inside the triangle RN_3N_5. Hence, we cannot eliminate the skinny triangle RN_3N_4. It is important to note that even if we had chosen N_1 or N_2 from the original possible adjacent point set, we would have ended up in the same situation.

Model	No. of points	No. of Tris	Init. Time (in secs)	Rec. Time (in secs)
Club	16864	33660	0.2758	3.9644
Bunny	34834	69497	0.5961	9.1809
Foot	20021	39919	0.3802	5.2725
Skidoo	37974	75461	0.6680	8.536
Mannequin	12772	25349	0.2405	3.9289
Phone	83034	165730	1.5634	26.597

Table 1: Performance of our algorithm: See Color Plate 1

4.2 Triangulation

The remaining points from C_R after the various pruning steps are the final adjacent points and are connected in order around R to complete the triangulation in the object space. If consecutive adjacent points subtend more than α (maximum allowable angle parameter) at R in the object space, then they are not connected to form a triangle. This maximum angle describes the characteristics of the holes in the model, and R is considered as a *boundary* point. All the *free* points in the adjacent point list are labeled as *fringe* points and are appended in order at the end of the queue. The algorithm chooses the next point from the queue as the new reference point R, and continues with the triangulation around R.

5 Triangulating Terrain Data

Simple solutions are available to triangulate smooth terrain data. In this section, we show that these algorithms can be made as a special case to our 3D triangulation algorithm by fixing the projection plane.

Typically, devices used for data acquisition generate sample data in some order. In our algorithm, we can make use of this order and the characteristics of the device to handle noise. We have applied our method on a massive data set of a room (Color Plate 2 - top row) acquired by a laser range scanner. This is a common Center-Of-Projection (COP) device which returns a very dense sampling as a spherical depth map ((θ, ϕ) map) of the environment around itself. With adjacent samples of this high density sampling being less than an inch apart, the noise in the samples is nearly two inches. If we apply our original algorithm to this noisy data set, the selected projection plane would be completely altered by innumerable micro-facets formed in the vicinity of a point.

We make use of the fact that the data set is in the spherical coordinate system to solve this problem. In such a co-

ordinate system, this surface can be considered as a monotonic surface with respect to a unique projection plane, namely, the (θ, ϕ) plane, similar to height field or terrain data. Since the perturbations in the data set due to noise are usually orthogonal to the projection plane, our algorithm is not affected by it.

Traditional 3D reconstruction algorithms are not well-suited to handle terrain or range data. As a result, specialized algorithms [GH95] have been developed to exploit the simplicity of the input. However, in our algorithm, terrain models are a special case where we fix the projection plane to remain constant. Further, since estimating the projection plane at each point is avoided, our algorithm runs significantly faster as well.

5.1 Specializing our Algorithm

The underlying two dimensional *dexel array* is considered as the (θ, ϕ) projection plane with only one data point at each *dexel*. The neighbors of a point in the final triangulation can only be from its adjacent *dexels*. Hence the first step of pruning (by L_∞ metric), can be made to choose only the points from the adjacent *dexels* of R. The radius of S_R is set to a slightly higher value than the noise in the system. As all the points in C_R are visible from R, visibility and angle checks can also be skipped. By setting the parameters and removing these tests, it takes less than seven seconds to reconstruct a data set of size around 900,000 points. Essentially, we can think of our approach as a parameterized algorithm where fixing certain parameters results in highly specialized and efficient algorithms for different classes of inputs.

In practice, we fix the dimensions of the *dexel* array. We retain one representative point if multiple points get mapped onto the same *dexel*. This thins the high sampling density, and forms a level of simplification. The dimensions of the *dexel* array controls the amount of simplification and the run time of the algorithm. Since all the processing time in our algorithm is dependent on the number of *candidate points*, bounding this number is a major source of speed-up in handling terrain data.

The image in Color Plate 2 (top row) shows the textured reconstructed room model. The texture is created from the intensity values returned by the laser device. The image on the right shows the micro-facets in the floor of the room, in spite of point simplification.

6 Performance and Results

The complexity of our algorithm is input sensitive, *i.e.* time spent is proportional to the model complexity. This can be seen from the results shown in Table 1. The bunny model, which has fewer points than the skidoo model, takes more time for reconstruction, due to its complexity. Similarly, the mannequin model takes almost the same time as that of the club model, because of high curvature variations and non-uniform sampling in the regions near the nose, eyes and ears. For the same reason, we are able to handle massive data sets of size in the order of millions of points in a few seconds, as we are making use of height field data properties.

No.of points	No. of Triangles	Init. Time* (in secs)	Rec. Time 1 (in secs)	Rec. Time 2 (in secs)
143858	267131	82.508	5.998	1.020
883577	1707468	88.554	38.782	6.913

Table 2: Performance of the system for the Room range data set: See Color Plate 2. Reconstruction Time 2: without visibility and angle criteria check. (*: Includes the reading time of the original data set – 6479713 points)

Our algorithm is a single pass algorithm, and does not need to revisit the triangles once they are formed. We do not produce any higher dimensional simplices (like tetrahedra [Boi84]) that require their removal to make the model a valid manifold. We also do not change the triangulation once they are completed.

Reduced memory requirement is another feature of our algorithm. Our algorithm does not store the triangles formed during reconstruction in the main memory. Only those triangles which are incident on *fringe* and *boundary* points are retained, as they are used for visibility pruning. Hence we are able to handle massive models with millions of data points.

Table 1 shows the time taken by our algorithm on various point clouds. The initialization time in the table includes the time taken to read in the model and initializing the data structure. All the timing measurements in this paper were made on a 250 MHz, R10000 SGI Onyx2 with 16 GB of main memory. Table 2 shows the timing of our algorithm on the laser data. The initialization time includes the time to read in the original model of around 6.5 million points, filling up the data structure and eliminating the points. The two entries in the table show the timings for two different sizes of the *dexel array*: 400 $\times$ 600 and 1000 $\times$ 1500.

6.1 Robustness of our algorithm

We avoid most of the robustness problems faced by purely geometric methods (like noise and degenerate situations) by our partially combinatorial approach. In our algorithm, we face robustness problems in the projection plane evaluation. For example, sharp curvature variations in the object might lead to incorrect estimates of the projection plane.

To test the robustness of our approach to perturbations in the estimated tangent plane at R, we used one of just three projection planes – (X, Y), (Y, Z), and (Z, X),

whichever was close to the actual estimate. We were able to triangulate many models including the bunny model satisfactorily. The execution time with this approach is much less than the times listed in the Table 1 because we do not need to explicitly transform the vicinity of R to its tangent plane. But the disadvantage of this approach is that it has a few favorable orientations of the model in the coordinate frame, and different orientations gave different results.

6.2 Limitations of Our Approach

Any projection-based approach gives different triangulation for different starting points. Our approach also suffers from the same limitation. But once the seed point is fixed, the triangulation is same for any transformation of the model. The second limitation is also common to most surface reconstruction algorithms – sharp curvature variations. If the faces incident on a vertex do not satisfy our criterion of surface smoothness, then our algorithm might produce incorrect triangulations. For under-sampled and extremely non-uniformed models, our algorithm produces spurious model boundaries, as shown in the Color Plate 2 (bottom row).

7 Conclusion

We have presented a new projection-based surface reconstruction algorithm from unorganized point clouds. The key features of our method are speed and memory efficiency. Further, it is a single pass algorithm and can make use of the characteristics of the data acquisition phase to handle noisy data. We have demonstrated the application of our algorithm on various data sets, including a massive, noisy range scan model of a room. We have successfully generated valid, non-self-intersecting, orientable manifold surface meshes for point clouds of size a few hundred thousand in a matter of tens of seconds. By fixing certain parameters in our algorithm, we obtain highly specialized and efficient methods for various input classes. We believe that such a versatility and performance without any manual intervention is a big win for our algorithm.

References

[ABK98] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *ACM Siggraph*, pages 415–421, 1998.

[Att97] D. Attali. r-regular shape reconstruction from unorganized points. In *ACM Computationl Geometry*, pages 248–253, 1997.

[BBX97] C. Bajaj, F. Bernardini, and G. Xu. Reconstructing surfaces and functions on surfaces from unorganized 3d data. *Algorithmica*, 19:243–261, 1997.

[BMR+99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 1999.

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

[CA97] P. Crossno and E. Angel. Isosurface extraction using particle systems. *IEEE Visualization '97*, pages 495–498, November 1997.

[CL96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *ACM Siggraph*, pages 303–312, 1996.

[EM94] H. Edelsbrunner and E. Mucke. Three dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[GH95] Michael Garland and Paul S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical report, CS Dept., Carnegie Mellon U., Sept. 1995.

[GKS00] M. Gopi, S. Krishnan, and C. T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum, Eurographics*, 19(3):C467–C478, 2000.

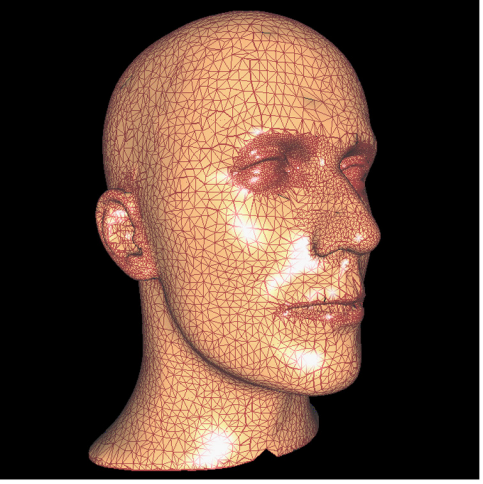
[HDD+92] H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized point clouds. In *ACM Siggraph*, pages 71–78, 1992.

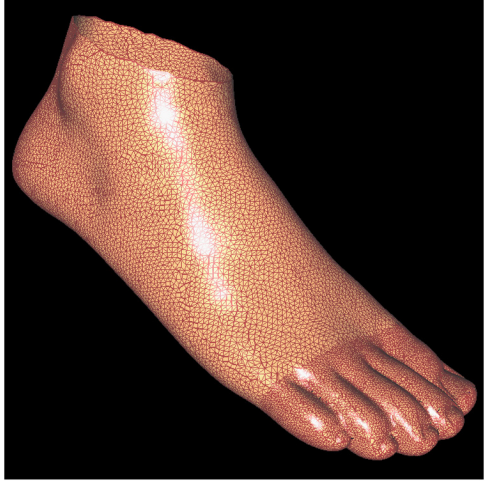
[Hoo86] T. Van Hook. Real-time shaded NC milling display. In *ACM Siggraph*, pages 15–20, 1986.

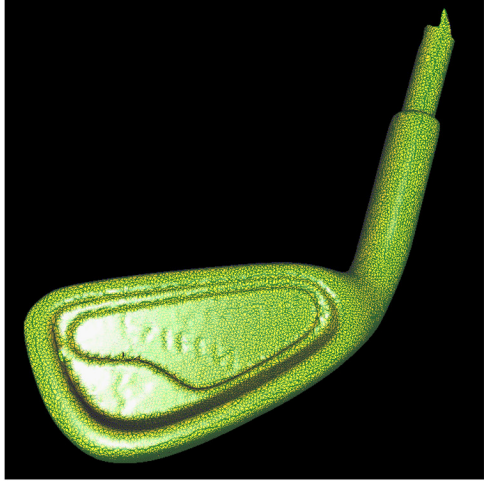
[MM98] R. Mencl and H. Muller. Interpolation and approximation of surfaces from three-dimensional scattered data points. *State of the Art Reports, Eurographics '98*, pages 51–67, 1998.

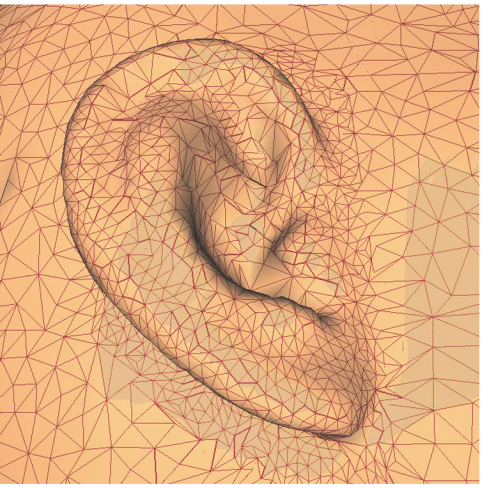
[RWC+98] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *ACM Siggraph*, pages 179–188, 1998.

[Vel95] R. C. Veltkamp. Boundaries through scattered points of unknown density. *Graphical Models and Image Processing*, 57(6):441–452, 1995.

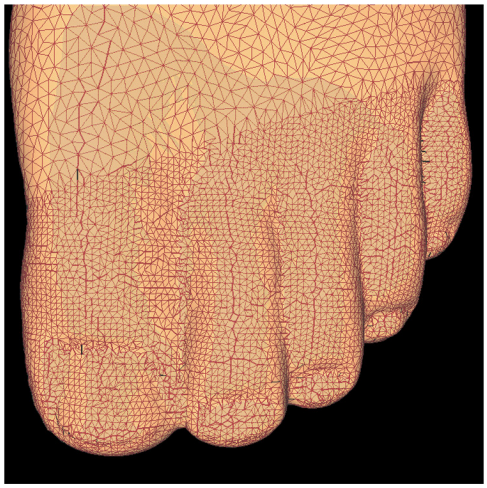




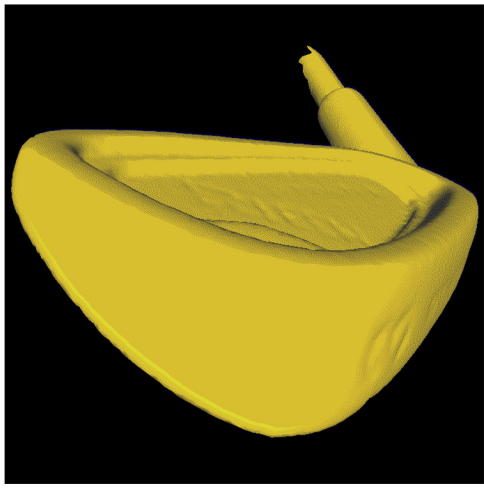




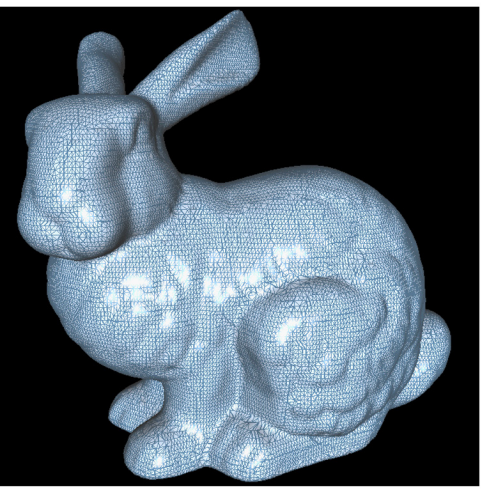
Mannequin Model
TOP: 12772 points, 25349 triangles, 4.16 seconds
BOTTOM: Triangulation in the high curvature regions of the model with non-uniform sampling



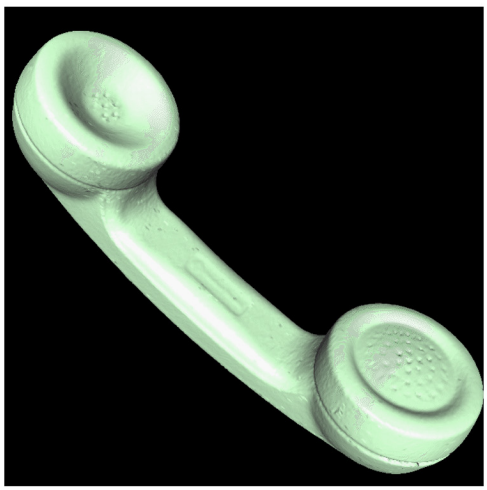
Foot Model
TOP: 20021 points, 39919 triangles, 5.65 seconds
BOTTOM: Triangulation in the regions of non-uniform sampling



Club Model
TOP: 16864 points, 33660 triangles, 4.23 seconds
BOTTOM: Notice the character '3' on the side of the club and the letters on the face

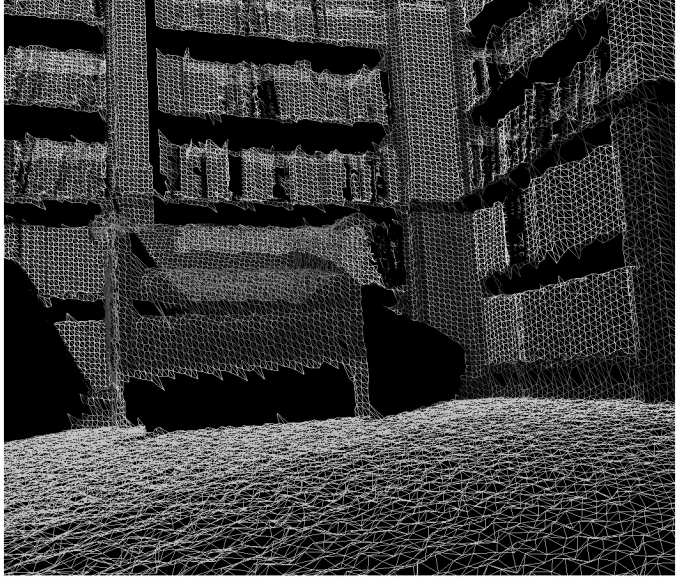


Bunny Model
34834 points, 69497 triangles, 9.77 secs

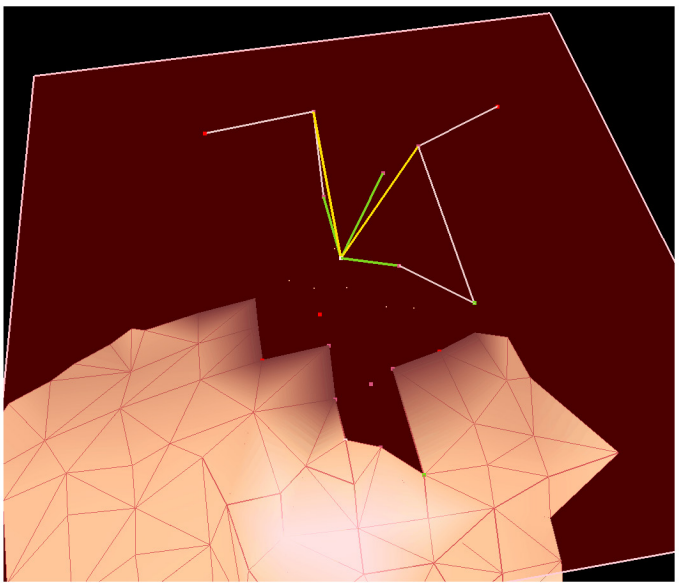


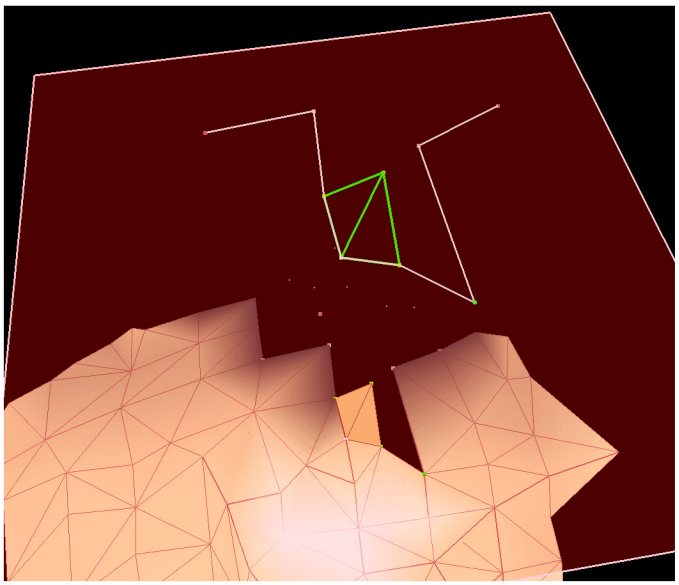
Phone Model
83034 points, 165730 triangles, 28.15 secs

Skidoo Model
37974 points, 75461 triangles, 9.19 secs

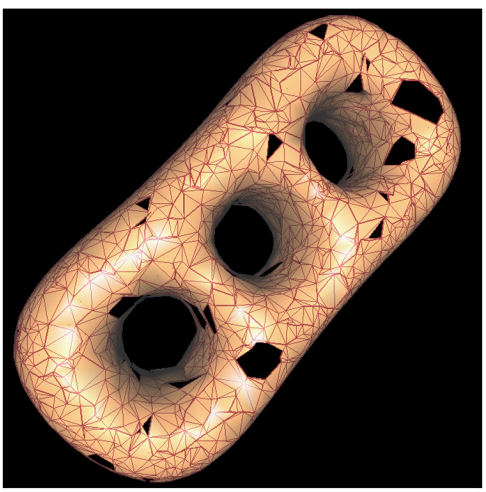


LEFT: Organized, noisy, massive range data set: 6.48 million points, 143858 points retained, 267131 triangles, 88.5 secs
RIGHT: Notice the noise and the microfacets on the floor of the room





LEFT: Ordering of the candidate points around the reference point on the projection plane. RIGHT: Final triangulation around the reference point after rejection of points based on angle and visibility criteria.



Reconstructed surface of non-uniform and under-sampled point clouds

COLOR PLATE : 2