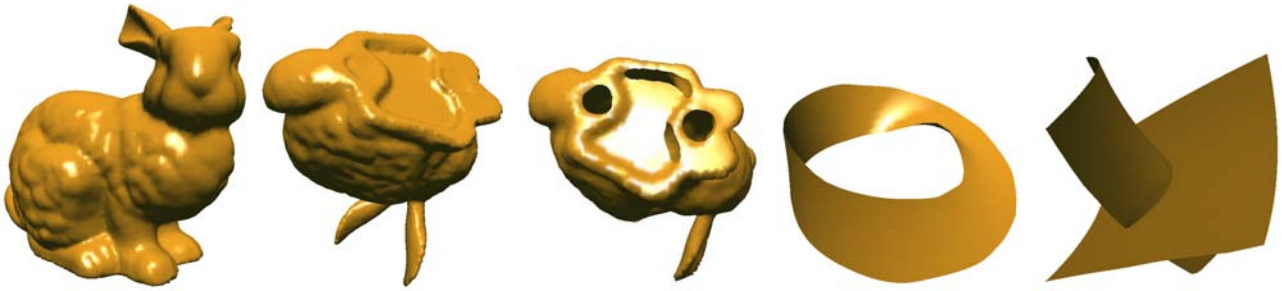
Please see supplementary material on conference DVD.

Reconstructing Manifold and Non-Manifold Surfaces from Point Clouds

Jianning Wang*
CVC,Computer Science
Stony Brook University

Manuel M. Oliveira†
Instituto de Informática
UFRGS, Brazil

Arie E. Kaufman*
CVC,Computer Science
Stony Brook University



(a) (b) (c) (d) (e)

Figure 1: Examples of surfaces reconstructed with our algorithm. A single framework handles all cases. (a) A view of the Stanford bunny. (b) Surface reconstruction with hole filling. (c) Surface reconstruction with boundary preservation. (d) Non-orientable surface. (e) Non-manifold surface.

ABSTRACT

This paper presents a novel approach for surface reconstruction from point clouds. The proposed technique is general in the sense that it naturally handles both manifold and non-manifold surfaces, providing a consistent way for reconstructing closed surfaces as well as surfaces with boundaries. It is also robust in the presence of noise, irregular sampling and surface gaps. Furthermore, it is fast, parallelizable and easy to implement because it is based on simple local operations. In this approach, surface reconstruction consists of three major steps: first, the space containing the point cloud is subdivided, creating a voxel representation. Then, a voxel surface is computed using gap filling and topological thinning operations. Finally, the resulting voxel surface is converted into a polygonal mesh. We demonstrate the effectiveness of our approach by reconstructing polygonal models from range scans of real objects as well as from synthetic data.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

Keywords: surface reconstruction, non-manifold surfaces, topological thinning

1 INTRODUCTION

Surface reconstruction from unorganized points has many practical applications ranging from reverse engineering [24], entertainment, and analysis of forensic records, to digitization of cultural heritage [15] and creation of virtual museums. This subject has gotten considerable attention in recent years due to the increasing availability of 3D scanning devices, which are capable of sampling complex geometric objects at very high resolutions. Surface reconstruction from unorganized point clouds is, however, a challenging and ill-posed problem and although much progress has been made in the last few years [3, 9, 13, 18], currently no single approach can consistently handle all possible surface configurations. Moreover, essentially all approaches seem to assume that point clouds represent the surfaces of solids, and very little attention has been given to reconstructing surfaces with boundaries or non-orientable surfaces [1].

In this paper, we present a novel approach for surface reconstruction from unorganized point clouds. Our approach is general in the sense that it naturally handles manifold and non-manifold surfaces, surfaces with boundaries, as well as non-orientable surfaces, all in a consistent way. It does not require any extra geometric information other than the positions of the samples. Depending on no extra information makes this algorithm suitable for a broader range of applications. Our method is robust to irregular sampling and surface gaps, and relatively robust to the presence of noise. Furthermore, it is fast, parallelizable and easy to implement because it is based on simple local operations.

In addition to its central contribution, the new reconstruction algorithm, this paper also introduces three other original results:

- An extension to Tsao and Fu's algorithm [22] to perform thinning while preserving surface boundaries (Section 3.2);
- An extension to Azernikov's meshing algorithm [4] to support the creation of polygonal meshes for non-manifold surfaces represented as point clouds (Section 3.3);
- A new algorithm for smoothing surface boundaries that significantly improves the quality of the meshes reconstructed for surfaces with boundaries (Section 3.3.1).

In our approach, surface reconstruction is performed using a three-step process. First, the space containing the point cloud is subdivided, creating a voxel representation. Gap filling is performed at this stage (Section 3.1). Then, a voxel surface with the same topology of the sampled surface is computed using topological thinning operations based on Tsao and Fu's algorithm [22] (Section 3.2). Finally, a polygonal mesh is extracted using a modified version of Azernikov's algorithm (Section 3.3). Figure 1 illustrates

*e-mail: {jianning,ari}@cs.sunysb.edu
†e-mail:oliveira@inf.ufrgs.br

IEEE Visualization 2005
October 23-28, Minneapolis, MN, USA
0-7803-9462-3/05/$20.00 ©2005 IEEE.

the versatility of our technique to reconstruct surfaces with different topologies.

2 Previous and Related Work

There are many algorithms for surface reconstruction from unorganized points. They can be generally classified as *computational geometry*, *algebraic* and *implicit methods*. Computational geometry methods use mechanisms such as Delaunay triangulation [3, 10] and region growing [5, 11] to build a set of simplices whose vertices are the original points. These methods tend to perform well on dense and clean datasets and are capable of reconstructing surfaces with boundaries. They are, however, sensitive to noise and tend to leave holes in undersampled regions. Algebraic methods (*e.g.*, [19, 21]) fit a smooth function to the set of input points thus recovering smooth surfaces. Most of these methods cannot handle arbitrary topology, nor reconstruct complex geometry.

The use of implicit functions is the most popular approach for surface reconstruction from sets of unorganized points [9, 13, 17, 18, 23]. These methods are based on the use of a signed distance function, which allows the representation of different topologies. Surface reconstruction is quite straightforward, usually performed using a variation of the Marching Cubes algorithm [16]. The primary limitation of these methods is their inability to reconstruct non-manifold surfaces. According to Bloomenthal et al. [7], boundaries are usually specified via additional functions that are required to have all the same sign. Bloomenthal and Ferguson [8] describe an algorithm for polygonizing non-manifold implicit surfaces defined by multiple regions of space. The algorithm can handle surfaces with boundaries and intersections, but the existence of multiple regions significantly adds to the complexity of the polygonizer.

Ohtake and collaborators [18] have demonstrated a very interesting and flexible hierarchical surface reconstruction method based on the blending of local implicit functions. Their approach handles very large datasets, can reconstruct sharp features and provides support for CSG and morphing operations. However, it cannot be used to reconstruct non-manifold surfaces.

Adamson and Alexa [1] presented a point-based algorithm for rendering (not reconstructing) surfaces with boundaries and non-orientable surfaces. In their approach, a dense set of points defines an implicit surface, which is identical to a moving least squares (MLS) surface approximation [2]. The technique can only be used for rendering manifold surfaces and is based on ray casting. A surface with boundary is then locally defined as the set of points x (on the implicit surface), whose distances to a weighted average position in a neighborhood Ω that contains x is less than a user-specified threshold. Thus, sparsely sampled regions are rendered as holes. Unlike [1], our technique actually reconstructs a polygonal representation for these surfaces.

Azernikov's approach [4] constructs a connectivity graph and then creates facets by traversing the graph and finding minimal loops. In the resulting mesh, each voxel yields one vertex, computed as the centroid of input points inside the associated voxel. The algorithm cannot guarantee that the voxel surface has a well-defined local topology. Therefore, a number of heuristics are used to assure the correctness of the resulting mesh. It also assumes that its input consists of a densely sampled point cloud and does not fill gaps. There are three major differences between our modified meshing strategy and Azernikov's algorithm: (1) we only find a connectivity graph for *surface* and *border voxels* (see Section 3.2 for a definition of these terms); (2) our approach handles non-manifold surfaces; and (3) because some empty octree voxels corresponding to holes need to be incorporated in the voxel surface (in order to automatically fill them), we employ a new way to identify such voxels and assign vertices to them.

3 The Surface Reconstruction Algorithm

Our surface reconstruction algorithm constructs a polygonal mesh from a point cloud using voxels as an intermediate representation. The steps of the algorithm are shown in Algorithm 1. First, the point cloud is turned into a voxel representation. At this stage, gaps are filled using a new gap-filling algorithm. Then, topological thinning is used to create a voxel surface representation. Finally, a polygonal mesh is extracted from the voxel surface.

1. voxelization and gap filling;
2. topological thinning;
3. meshing.

Algorithm 1: Surface reconstruction algorithm

3.1 Voxelization and Gap Filling

The algorithm starts by computing a tight bounding box for the point cloud, which is then subdivided into voxels. The voxel size should be small enough to avoid merging distinct surface features, but large enough to avoid unnecessary computations. Currently, the voxelization process is performed based on a user-specified voxel size. Voxels containing original points are called *p-voxels*. Some empty voxels (*i.e.*, not containing points) falling in between p-voxels and representing non-sampled surface patches are called *g-voxels*. g-voxels will later be used for gap filling (Section 3.1.2 explains how to identify g-voxels). We call *foreground voxels* the set defined by the union of p-voxels and g-voxels. Foreground voxels are used for topological thinning (Section 3.2). Because they only occupy a small portion of the space, we use a dexel data structure [12] to store them. The complement of the foreground voxels with respect to the bounding box is called *background voxels*.

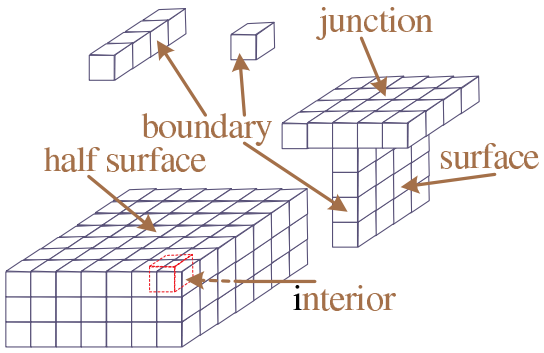


Figure 2: Topological classification of voxels.

3.1.1 Topological Classification of Voxels

We are interested in reconstructing surfaces and surface junctions, while preserving surface boundaries. The *local topological type* of a voxel is determined by the number of foreground and background connected components in its neighborhood. A voxel is called a *junction voxel* if it is at the junction of two or more surfaces. Likewise, a voxel is called a *boundary voxel* if it is at the boundary of some surface, along some curve, or just an isolated voxel. *Half surface voxels* are on one side of a thick voxel surface and are the only ones to be deleted during topological thinning. The existence of thick voxel surfaces is usually associated with the presence of noise in the dataset. Figure 2 illustrates the different topological types of a voxel.

Proceedings of the IEEE Visualization 2005 October 23-28, Minneapolis, MN, USA (VIS'05)
0-7803-9462-3/05 $20.00 © 2005 IEEE

Table 1: Voxel topological types

Topological type	NFC_3	NBC_3	$NBC_{>3}$
interior	1	0	any
boundary	>1 or 0	any	any
boundary	1	1	1
surface	1	2	2
junction	1	≥ 2	> 2
half surface	1	1	2

To compute the local topological type of a voxel v, one needs to compute NFC and NBC, the number of connected components formed exclusively by foreground (18-connected) and background (6-connected) voxels, respectively, in the neighborhood of v. Before computing NFC, v should be deleted from its neighborhood. The appropriate size for the neighborhood depends on the level of noise in the point cloud and on the surface features to be reconstructed. According to our experience, a maximum neighborhood size of $5 \times 5 \times 5$ voxels seems to work well in practice for most situations. All examples shown in the paper were reconstructed using a $5 \times 5 \times 5$ voxel neighborhood.

The topological type of a voxel v is determined by looking up Table 1 using the values of NFC and NBC. The subscripts of NFC and NBC in Table 1 represent the size of a neighborhood (*e.g.*, a subscript value k indicates a $k \times k \times k$ neighborhood). For efficiency reasons, we first search the $3 \times 3 \times 3$ neighborhood of v. If the $NBC_3 = 0$, we have an interior voxel, and the classification process stops. If, on the other hand, $NFC_3 > 1$ or $NFC_3 = 0$, v is a boundary voxel. Although Table 1 shows the necessary conditions for identifying v's topological type on a $3 \times 3 \times 3$ neighborhood, such a small neighborhood size can only be safely used with clean datasets. Noise tends to cause voxel surfaces to appear thicker. In this case, the use of a $3 \times 3 \times 3$ neighborhood is often insufficient to correctly characterize the different components in v's neighborhood, thus leading to a incorrect topological classification. If v's topological type cannot be safely identified based on the values of NFC_3 and NBC_3, we increase the neighborhood size to $5 \times 5 \times 5$ and recompute NBC. In this case, if $NFC_3 = 1$ and $NBC_3 = 1$ and $NBC_5 = 1$ then v is a boundary voxel. If $NFC_3 = 1$ and $NBC_3 = 1$ and $NBC_5 = 2$ then v is a half surface voxel. Otherwise, if $NBC_5 = 2$, v is a surface voxel. If $NBC_5 > 2$, v is a junction voxel. These conditions are summarized in Table 1 and in Algorithm 2.

```
compute NFC_3 and NBC_3 for v;
if (NBC_3 = 0)
    return interior voxel;
if (NFC_3 > 1 or NFC_3 = 0)
    return boundary voxel;
else
    compute NBC_5 for v;
    if (NBC_3 = 1)
        if (NBC_5 = 1)
            return boundary voxel;
        else
            return half surface voxel;
    else
        if (NBC_5 > 2)
            return junction voxel;
        else
            return surface voxel;
```

Algorithm 2: Topological classification (v: voxel)

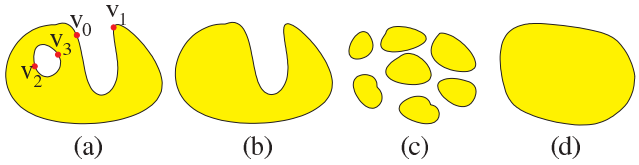


Figure 3: Possible situations in gap filling: (a) a surface with an inner hole. Here $< v_2, v_3 >$ is a valid voxel line segment, but $< v_0, v_1 >$ is not. (b) the surface reconstructed from (a) with hole filled. (c) several surface patches with in-between gaps, and (d) the surface reconstructed from (c).

3.1.2 Identifying g-voxels and Filling Gaps

The voxel surface defined by the set of p-voxels may contain holes and/or gaps due to insufficient sampling, as illustrated in Figure 3(a) and (c). We will not make a distinction between a hole (Figure 3a) and a gap (Figure 3c). For simplicity, we use the term "gap" to refer to hole and gap. Because we adopt a more general definition of "gap", the classical hole-filling schemes (*i.e.*, finding hole boundary before filling the hole [14]) cannot be applied here. Algorithm 3 fills gaps before a polygonal mesh is extracted from the voxel surface. The algorithm uses two dexel data structures D_{pg} and D_b, which are both initialized with all p-voxels. At the end of the procedure, D_{pg} will contain the set of foreground voxels (p-voxels and g-voxels) and will be used as input for the topological thinning procedure; D_b, on the other hand, will contain the set of boundary voxels, and will be used to prevent boundary voxels in D_{pg} from being eroded during topological thinning (Section 3.2).

The idea behind Algorithm 3 is straightforward: after initializing D_{pg} and D_b with p-voxels, the algorithm tries to fill gaps by filling voxel line segments with increasing lengths ranging from 1 to L (a maximum user-specified value). Each segment should connect a pair $< v_0, v_1 >$ of boundary voxels (classified using Algorithm 2). Before a voxel line segment is filled, the segment has to pass a validation test.

```
store all p-voxels in both D_pg and D_b;
for d=1 to hole size L;
    for each voxel v in D_b;
        check the topological type of v in D_pg;
        if v is not a boundary voxel;
            delete v from D_b;
    for each voxel v_0 in D_b;
        for each v_1 in D_b with ||v_1 - v_0|| ≤ d;
            if voxel line segment < v_0, v_1 > is valid;
                create g-voxels connecting v_0 to v_1;
                add these g-voxels to D_pg and D_b;
```

Algorithm 3: Gap filling in voxel space

Gap refers to the background voxels between adjacent patches of the same surface component. And we call the background voxels separating two different surface components *separation*. In Algorithm 3, we first fill gaps with small width. When the length of voxel line segments become larger, we need to test whether they are connecting already connected patches.

The validity tests of voxel line segment $< v_0, v_1 >$ include the following: (1) except the two end *boundary* voxels, the voxel line segment should be solely composed of background voxels; (2) v_0 and v_1 do not come from the same outmost boundary (*e.g.*, v_0 and v_1 in Figure 3(a)). If the voxel line segment passes the tests, all (empty) voxels along the segment are turned into g-voxels, and used to fill gaps. g-voxels are automatically incorporated into the set of

foreground voxels.

A boundary is a closed voxel curve. We start from an arbitrary voxel and propagate it via adjacency along the curve until all voxels along the curve have been reached. The number of propagation becomes then the length of the curve. If we propagate a boundary towards adjacent non-boundary regions, we obtain a new voxel curve. If the boundary is longer than the new curve, it is considered to be an outmost one. In practice, however, this criterion could fail and users might want to preserve some hole boundaries (*e.g.*, the holes in the bottom of the bunny in Figure 1(b)). For these reasons, the user is allowed to mark any boundary as outmost later.

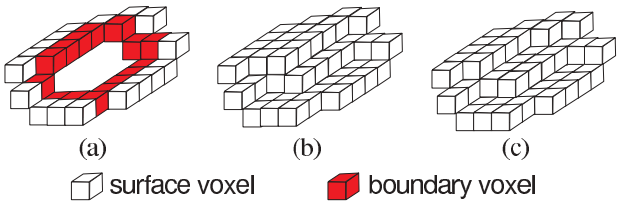


Figure 4: Gap filling and topological thinning. (a) Fragment of a voxel surface with a boundary highlighted in red. (b) Result of the gap filling procedure. (c) Resulting voxel surface after topological thinning.

3.2 Topological Thinning

Before converting the voxel representation into a polygonal mesh, we should remove all voxels that cause the "surface" to be unnecessarily thicker than one voxel deep. Such voxels are called half-surface voxels (see Figure 2) and usually result from the existence of noise in the point cloud or inappropriately large voxel size. In order to create a thin voxel shell, we perform a topological thinning operation. By iteratively removing boundary voxels and reclassifying the remaining voxels, Tsao and Fu [22] find the skeleton (medial axis or medial surface) from voxel representations. However, their algorithm erodes surface boundaries as well, which we want to preserve. Therefore, we have extended Tsao's and Fu's original algorithm to preserve boundaries. In the extended algorithm, thinning is applied to the set of foreground voxels D_{pg}, while D_b (the set of boundary voxels) is checked to avoid improper removal of boundary voxels. Both D_{pg} and D_b are computed using Algorithm 3. At the end of the thinning process, D_{pg} contains a *voxel surface* (more precisely, D_{pg} will contain a set of connected components representing the surface).

Figure 4 illustrates the process of gap filling and topological thinning. The red voxels represent a surface boundary and the result of gap filling is shown in Figure 4(b). Note the existence of some extra voxels. Figure 4(c) shows the output produced by our topological thinning algorithm, which will be used as input for the meshing stage (last step of our surface-reconstruction algorithm).

3.3 Meshing

Meshing is performed using a variation of the algorithm presented by Azernikov et al. [4]. Since the local topological type of all voxels in D_{pg} is known, the step in Azernikov's algorithm required to eliminate "false facets" (*i.e.*, facets that might appear in the connectivity graph but do not belong to the original surface) can be safely skipped. For the case of manifold surfaces, this simpler version of the algorithm can be used. Since Azernikov's algorithm does not handle non-manifold surfaces, we modified the original algorithm by allowing one surface to be composed of several *sub-surfaces* connected at junction voxels.

Given two surface patches P_i and P_j, their intersection defines a set of junction voxels. Figure 5 illustrates this situation in 2D showing a slice (side view) through two interpenetrating surface patches. In Figure 5, blue squares represent junction voxels. Red squares represent surface voxels. Surface voxels touching junction voxels are called *border voxels*. A set of border voxels of a given patch defines a *border curve* for the patch. We call *intersection curve* the curve defined by the set of junction voxels.

Given an intersection between two patches P_i and P_j, there are three possible types of border curves for one participating patch, as illustrated in Figure 6 for the case of patch I: two disconnected border curves (Figure 6a), one open border curve (Figure 6b), and one closed border curve ((Figure 6c). For the cases (b) and (c), each patch still consists of a single connected component. Meshing can then be performed after junction voxels are connected to voxels of the border curve, and the resulting set of voxels is submitted to a thinning operation.

If the intersection causes a patch to be split into two disjoint connected components (Figure 6a), the algorithm needs to regroup all components into a single mesh. Figure 5 illustrates this situation for a pair of intersecting patches P_i and P_j, resulting in four disjoint connected components S_0, S_1, S_2 and S_3, shown in red in Figure 5(a).

The process of regrouping starts by identifying all components and, for each one, creating a mesh using the simplified version of Azernikov's algorithm. After these meshes have been created, a normal vector is computed for each border voxel by averaging the normals of all faces sharing that particular voxel (Figure 5a). We start with a voxel in the intersection curve and walk along this curve. Let v_j be the current junction voxel in the intersection curve. Also, let v_a and v_b be any two border voxels adjacent to v_j and belonging to different border curves C_p and C_q. We compute $m_{a,b} = |n_a \cdot n_b|$, where n_a and n_b are the normals of border voxels v_a and v_b, respectively. We use a triangular matrix M to store in element $M_{p,q}$ the average of all values $m_{a,b}$ such that v_a is in border curve C_p and v_b is in border curve C_q. In this case, averaging is used to compensate for the fact that different border curves may have different numbers of border voxels.

After the walk through all junction voxels in the intersection curve is over, let $M_{p,q}$ be the element of matrix M with the biggest value. Then, we connect the components associated with border curves C_p and C_q and set $M_{p,q}$ to zero. The rationale here is that the orientation of border voxels belonging to these two components correlate the most. This is interpreted as a strong indication that the two components belong to the same surface. Then, we identify the element in M with the next biggest value, connect their corresponding components and set the matrix cell to zero. This process is repeated until M contains no non-zero elements. The entire process is illustrated in Figure 5. First, components S_1 and S_2 are connected (Figure 5b), then S_0 and S_3 are also connected (Figure 5c). Connecting two components amounts to computing the union between the voxels belonging to the components themselves and the associated junction voxels, performing a thinning operation, and finally using the simplified version of Azernikov's algorithm to create a mesh. Figure 5(c) shows the final result after the algorithm is applied to the four components shown in Figure 5(a). The position of the vertex associated with a junction voxel is given by the average position of all points falling inside the corresponding voxel. On the other hand, the vertex position inside a g-voxel is initially at the center of the voxel. In order to produce smoother meshes, these positions can be relaxed using any active contour techniques [6]. Currently, we minimize the following energy with a simplistic mass-spring system:

$$E = \sum_{i \in G} (\| \sum_{k \in M(v_i)} (v_k - v_i) \|)^2 \qquad (1)$$

Proceedings of the IEEE Visualization 2005 October 23-28, Minneapolis, MN, USA (VIS'05)
0-7803-9462-3/05 $20.00 © 2005 IEEE

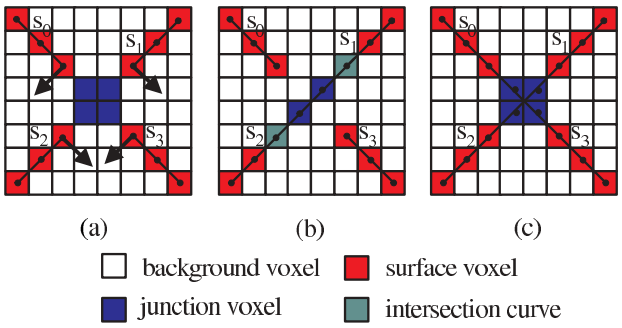


Figure 5: Creating non-manifold surfaces. (a) Individual surfaces are reconstructed. s_1 and s_2 are part of the same surface intersected by the surface defined by s_0 and s_3. (b) Surface (s_1, s_2) is reconnected. (c) Surface (s_0, s_3) is reconnected, thus reconstructing the non-manifold surface.

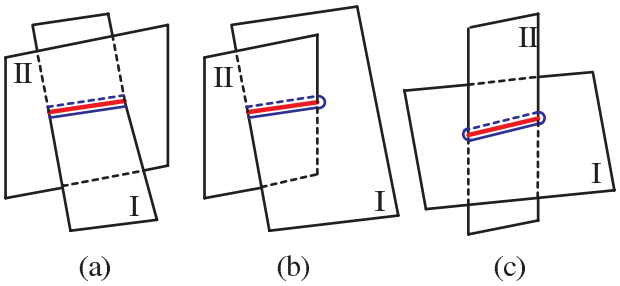


Figure 6: Three types of border curves in surface intersections (considered with respect to patch I): (a) two border curves; (b) a single non-closed border curve (c) a single closed border curve.

where G is the set of junction voxels and g-voxels, and $M(v)$ is the set of voxels which are directly connected to v in the resulting mesh. In order to accelerate the convergence, we solve the system iteratively until E becomes negligible.

3.3.1 Smoothing Boundaries

The representation of boundaries of surfaces is usually affected by under-sampling and noise. Also, the spatial discretization imposed by the voxelization process tends to worsen the problem. As a result, the meshing procedure described in section 3.3 tends to reconstruct ragged boundaries. This situation is illustrated in Figure 7(a) for the case of a mesh fragment showing a surface boundary. In order to improve the appearance of boundaries, we take groups of three consecutive boundary vertices and set the position of the middle one as a weighted average of the other two. Thus, let c be the vertex position associated with a voxel v at the boundary of a given surface S, and let c_l and c_r be the positions of the vertices associated with v_l and v_r, the left and right neighbors, respectively, of v at the same boundary. We then make c be collinear with c_l and c_r. As we proceed to the next group of three neighbor boundary voxels, the collinearity among c_l, c and c_r is lost, but the boundary smoothing effect is preserved. Using this simple procedure, the resulting boundary (Figure 7b) becomes much smoother, leading to more pleasing results. Figure 10 compares the results produced by mesh extraction (a) without and (b) with boundary smoothing and illustrates the significant improvement obtained with this technique.

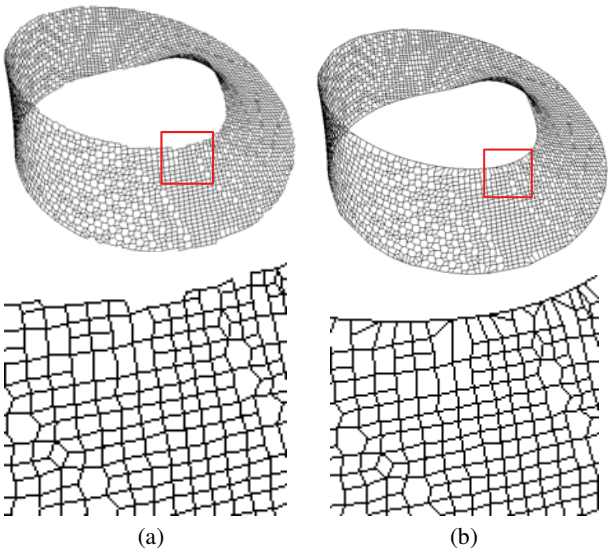


Figure 7: (a) Low sampling, noise, and spatial discretization imposed by the voxels can lead to the reconstruction of ragged boundaries. (b) New mesh obtained after boundary smoothing.

3.4 Cost of the Algorithm

Let n be the number of samples in the point cloud and let m be the number of voxels used to discretize its bounding box. The cost of assigning a given input point to a voxel is $O(1)$, adding up to $O(n)$ for the entire input data. The topological classification of one voxel is done by analyzing a finite neighborhood and, thus is $O(1)$. Therefore, the classification of all voxels is performed in $O(m)$. During gap filling, we have at most $(2L+1)^3$ voxel line segments and may have up to $L(2L+1)^4 m$ voxel operations, where L the maximum length allowed for a segment. Gap filling is then performed in $O(m)$. Thus, the first step of the surface reconstruction algorithm has cost $O(n+m)$. Topological thinning is performed in $O(m)$, since each voxel is visited and either deleted, if it is a half surface voxel, or kept, otherwise. Finally, for meshing, we need to process all foreground voxels (*i.e.*, p- and g-voxels) in order to compute the average position of the sets of points inside each voxel. Since the cost of creating the mesh itself is $O(m)$, the cost of the entire meshing stage is $O(n+m)$. Therefore, the total cost of our surface reconstruction algorithm is $O(m+n)$.

4 Results

We have implemented the described algorithm and used it to reconstruct manifold and non-manifold surfaces with different topologies. The point clouds used as input for the reconstruction process were obtained from actual range scans as well as from synthetic datasets. Figure 1 illustrates the versatility of our approach showing reconstruction results for different kinds of topologies. Our algorithm can be used to reconstruct closed surfaces (Figure 1b), surfaces with boundaries (Figure 1c), non-orientable surfaces (Figure 1d) and non-manifold surfaces (Figure 1e), all using a single framework. For rendering, the meshes were obtained using the procedure described in section 3.3 and were then triangulated. Vertex normals were approximated by averaging the normals of all faces sharing the given vertex.

Relying on the voxelization, topological classification and gap filling procedures, our algorithm is robust to the presence of noise and irregular sampling. Figures 8 and 9 illustrate the results of the proposed algorithm applied to actual range scans. Figure 8 shows

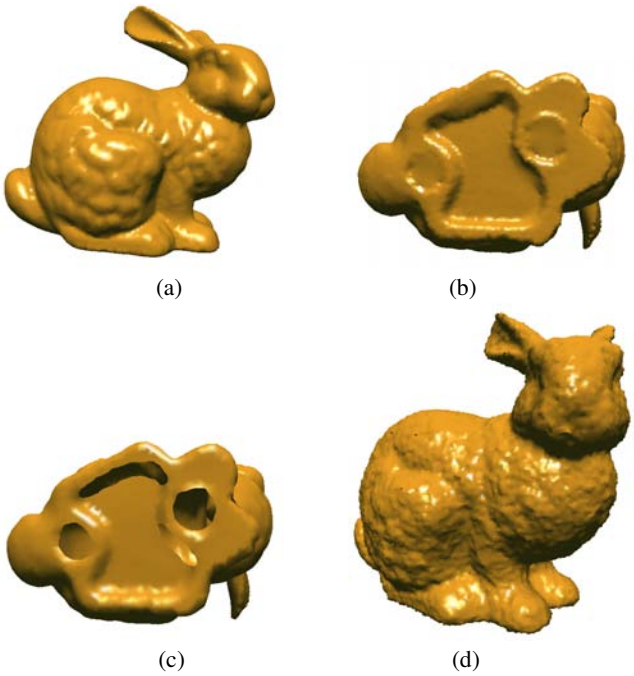


Figure 8: (a) Stanford bunny reconstructed from 10 range scans using the proposed algorithm. (b) Reconstruction of the bunny as a closed surface. (c) Reconstruction of the bunny preserving the holes (boundaries). (d) Noisy bunny.

several reconstructions of the Stanford bunny obtained from a point cloud created after merging 10 range scans (containing noise and registration errors), with a total of 362,272 points. The bunny model (Figure 8a) is well known to have holes at the bottom. Figure 8(b) shows the reconstructed bunny with the holes filled, whereas in Figure 8(c) the boundaries were preserved. We also added Gaussian noise with magnitude of 1% of the size of the original bounding box to the point cloud. The resulting reconstructed model is shown in Figure 8(d). The large number of g-voxels in the noisy bunny model (see Table 2) results from the existence of many small holes due to the added noise.

Figure 9 shows the Stanford dragon reconstructed with our algorithm after merging 60 range scans containing noise and registration errors. The merged dataset for this genus-2 model has about 1.7 million points and includes a number of noticeable outliers. Using our approach, the global topology is correctly reconstructed and delicate surface details are nicely preserved. No pre- or post-processing step is required for removing the outliers.

Figure 10 shows the reconstruction of a Möbius strip model (a non-manifold) and illustrates the ability of our approach to handle non-orientable surfaces with boundaries. On the left, we show the reconstructed model before boundary smoothing. Notice the ragged edges. On the right, one sees the result after our smoothing boundary algorithm has been used. The new boundaries are significantly smoother, resulting in a more pleasant model. Figure 7 shows close-up views of the two meshes for comparison. One should note that the ability to reconstruct non-manifold and non-orientable surfaces has more than just theoretical importance. For example, these surfaces have been represented using points [1], which one may want to visualize using polygonal models.

Figure 11 shows another view of the non-manifold model shown in Figure 1(e), which consists of two intersecting surfaces. This is a particularly challenging test and has been perfectly reconstructed by our approach. To the best of our knowledge, no other contemporary surface-reconstruction algorithm is capable of reconstructing such a model.

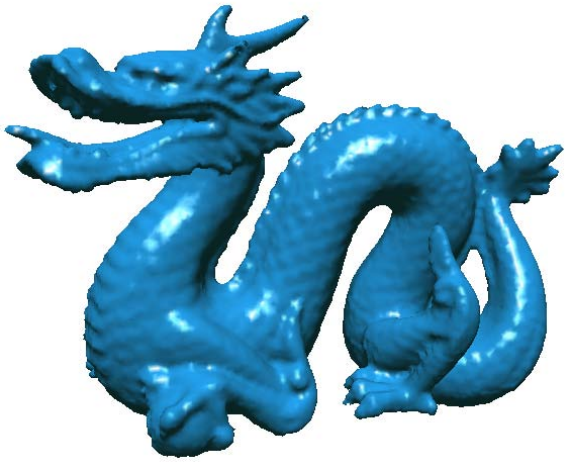


Figure 9: The reconstruction of the dragon model obtained from 60 range scans (containing registration errors, noise and outliers) using our algorithm.

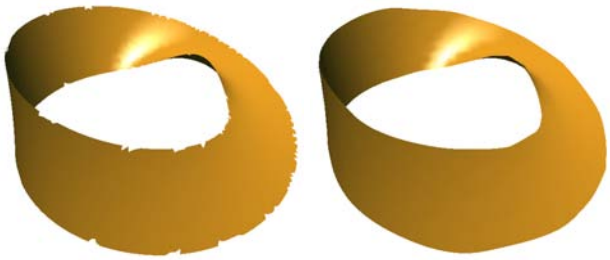


Figure 10: Reconstruction results of a non-orientable surface (Möbius strip) (a) before and (b) after smoothing of boundaries.

Figure 12 shows reconstruction results for a vase model with varying sampling rates and illustrates the ability of our approach to work with sparse datasets. Figure 12(a) shows a point cloud obtained by randomly selecting only 20% of the points (13,619 points) from a digitized vase model. The small square highlights the irregular sampling in the resulting point cloud. Undersampling can also be observed by comparing the number of g-voxels in Table 2. The number of such voxels for the reduced model is considerably bigger, thus implying severe undersampling. Figure 12(b) shows the reconstructed model obtained from the point cloud shown in (a). For comparison, Figure 12(c) shows the reconstructed vase using the full dataset (68,097 points). Although it is possible to observe differences on the vase's relief and at the boundary of its base, all major features were faithfully reconstructed from a much smaller dataset.

Table 2 presents some statistics associated with the models shown in the paper. Measurements were performed on a Pentium 4, 2 GHz PC with 1GB of memory. The last three columns of the table show the running times, in seconds, of the three steps of our surface reconstruction algorithm.

4.1 Discussion and Limitations

For models with dense and regular samples, we obtain voxel surfaces with correct local topological types for each voxel. In the

Proceedings of the IEEE Visualization 2005 October 23-28, Minneapolis, MN, USA (VIS'05)
0-7803-9462-3/05 $20.00 © 2005 IEEE

Table 2: Statistics of some reconstruction results (time in seconds)

Model	# of input points	voxel grid	# of p-voxels	# of g-voxels	# of facets	time of gap filling	time of thinning	time of meshing
Merged Bunny	362,272	128^3	25,751	419	27,953	5.64	11.00	0.56
Merged Dragon	1,769,513	256^3	41,243	662	101,997	20.80	28.60	3.79
Möbius Strip	7,560	64^3	2,912	0	2,838	0.89	1.23	0.11
Intersected surfaces	20,402	32^3	2,171	0	2,021	0.43	0.00	0.11
Noisy bunny (with 1% noise)	34,834	128^3	19,122	13,917	35,531	75.61	29.23	1.32
Vase	68,097	64^3	11,830	459	13,654	1.40	2.78	0.35
Undersampled Vase	13,619	64^3	6,413	6,092	13,855	7.50	4.20	0.37

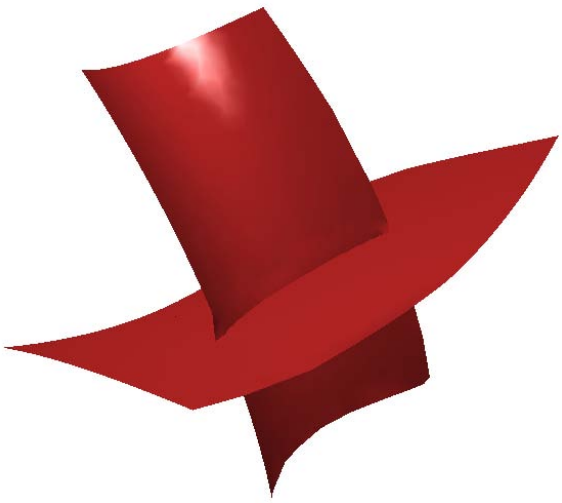
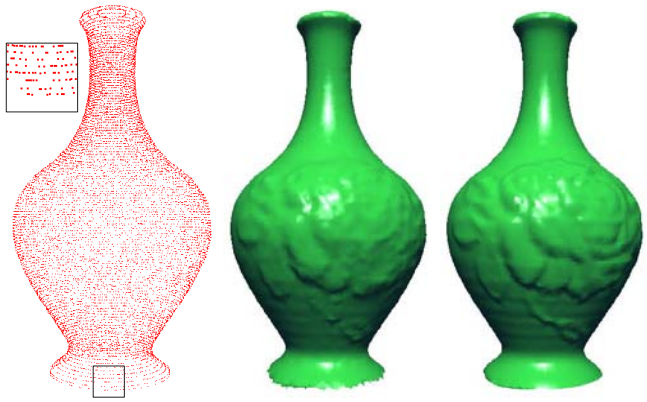


Figure 11: Non-manifold surface reconstructed with our algorithm. The surface consists of two intersecting patches.



(a) (b) (c)

Figure 12: Reconstruction of sparsely and unevenly sampled objects: (a) A point cloud obtained by randomly selecting 20% of the points (13,619 points) of a digitized vase model. The small square highlights the irregular sampling. (b) Vase model reconstructed from the point cloud in (a). (c) Vase reconstructed from the full point cloud (68,097 points). Notice that all important features were appropriately reconstructed in (b).

presence of noise, sample positions tend to be shifted, creating "inflated" voxel surfaces and possibly introducing small gaps in the surface. In most cases, the level of noise in the dataset is smaller than the voxel size. Thus the introduced gaps are at most one voxel wide and can be filled with ease using Algorithm 3 with $L = 2$.

Our approach subdivides the bounding box of the point cloud into voxels with uniform sizes. While the use of an adaptive data structure, such as an octree, has advantages in terms of memory requirements, the topological thinning algorithm requires a constant voxel size. Note that using small voxels impacts the running time of the algorithm. Thus, the most suitable voxel size can be understood as the biggest one that still allows the reconstruction of the intended surface details. Such a value can be obtained based on the user experience or using a binary-search-based trial and error process.

Our approach does not require point normals. However, if the dataset contains normals, they can be used to help thinning and meshing. For instance, the sub-directions for thinning could be reduced from 6 to 2 and one could further optimize the obtained mesh by applying the constraints on surface normals.

The major limitation of the proposed approach is that the hole filling algorithm relies on the assumption that the size of gaps to be filled is smaller than the size of separation between surface components, which should not be filled. In the case of increased noise level or missing samples, the assumption might be broken and the proposed algorithm will not work.

5 Conclusions and Future Work

We have presented a new approach for surface reconstruction from unorganized points. Our approach can naturally handle both manifold and non-manifold surfaces, surfaces with boundaries and non-orientable surfaces using a single framework. Our method is robust to irregular and sparse sampling, and to surface gaps. By not relying on normals, it is relatively robust to the presence of noise. We have demonstrated the effectiveness of the proposed approach by reconstructing geometric models directly from range scans of real objects containing noise and registration errors. The user can decide whether gaps and holes should be filled or not.

The approach is based on simple local operations, resulting in a fast, parallelizable and easy to implement procedure. We have shown that the cost of the algorithm varies linearly with the size of the point cloud and with the number of voxels used to discretize the space.

Along with the surface-reconstruction method, we presented an extension to Tsao's and Fu's thinning algorithm [22] that preserves surface boundaries. This new algorithm might be useful for other applications that require non-essential geometric data to be discarded, while preserving boundaries. We have also extended Azernikov's meshing algorithm [4] to support non-manifold surfaces and introduced a new smoothing boundary algorithm that significantly improves the quality of the reconstructed surface boundaries.

The use of a fixed voxel size leads to unnecessary processing on

locally flat regions without holes. The use of an adaptive framework (octree-like) based on the local geometry and topology of the surface patch inside a voxel could be used to skip this processing. Recently, Tchon et al. [20] report an approach for thinning on octrees. Also, the topological thinning approach may smooth some sharp features. For really curved regions, the reconstructed surface may be shifted from the real one.

Currently, the voxel size is a user-specified parameter. We would like to devise a mechanism for easily detecting different surface features inside voxels. Such a mechanism should help to guide an octree subdivision process, lending to an adaptive version of our surface-reconstruction algorithm. This is conceptually similar to what has been recently described by Ohtake and collaborators [18]. We also want to use graphics hardware to accelerate our method. Developing post-processing methods for the resulting mesh to handle sharp features is another important avenue for future exploration. The use of non-manifold subdivision [25] may help to improve the quality of the resulting mesh.

ACKNOWLEDGMENTS

This work is partially supported by NSF grant CCR-0306438 and FAPERGS (Brazil) Processo No 03/51060.8. The bunny and dragon datasets are courtesy of the Stanford 3D scanning repository. The vase model was provided by Cyberware.

REFERENCES

[1] Anders Adamson and Marc Alexa. Approximating bounded, non-orientable surfaces from points. *Shape Modeling International*, pages 243–252, 2004.

[2] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. *IEEE Visualization*, pages 21–28, 2001.

[3] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. *SIGGRAPH*, pages 415–421, 1998.

[4] Sergei Azernikov, Alex Miropolsky, and Anath Fischer. Surface reconstruction of freeform objects based on multiresolution volumetric method. *the 8th ACM symposium on Solid modeling and applications*, pages 115–126, 2003.

[5] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[6] Andrew Blake and Michael Isard. *Active Contours*. Springer-Verlag, 1998.

[7] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[8] Jules Bloomenthal and Keith Ferguson. Polygonization of non-manifold implicit surfaces. *SIGGRAPH*, pages 309–316, 1995.

[9] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3D objects with radial basis functions. *SIGGRAPH*, pages 67–76, 2001.

[10] Herbert Edelsbrunner and Ernst P. Mucke. Three-dimensional alpha shapes. *ACM Trans. on Graphics*, 13:43–72, 1994.

[11] M. Gopi and S. Krishnan. A fast and efficient projection based approach for surface reconstruction. *International Journal of High Performance Computer Graphics, Multimedia and Visualisation*, 1(1):1–12, 2000.

[12] Tim Van Hook. Real-time shaded NC milling display. *SIGGRAPH*, pages 15–20, 1986.

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

[14] Tao Ju. Robust repair of polygonal models. *SIGGRAPH*, pages 888–895, 2004.

[15] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. *SIGGRAPH*, pages 131–144, 2000.

[16] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH*, pages 163–169, 1987.

[17] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modeling International*, pages 89–98, 2001.

[18] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *SIGGRAPH*, pages 463–470, 2003.

[19] Stan Sclaroff and Alex Pentland. Generalized implicit functions for computer graphics. *SIGGRAPH*, 25(4):247–250, 1991.

[20] Ko-Foa Tchon, Mohammed Khachan, Francois Guibault, and Ricardo Camarero. Constructing anisotropic geometric metrics using octrees and skeletons. In *12th International Meshing Rountable*, pages 293–304, September 2003.

[21] D. Terzopoulos and D. Metaxas. Dynamic 3d models with local and global deformations: Deformable superquadrics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13:703–714, 1991.

[22] Y.F. Tsao and K.S. Fu. A parallel thinning algorithm for 3D pictures. *Computer Graphics and Image Processing*, 17:315–331, 1981.

[23] Greg Turk and James O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1999.

[24] Tamás Várady, Ralph R. Martin, and Jordan Cox. Reverse engineering of geometric models - an introduction. *Computer Aided Design*, 29(4):255–268, 1997.

[25] Lexing Ying and Denis Zorin. Nonmanifold subdivision. *IEEE Visualization*, pages 325–332, 2001.