Architectural Modeling from Sparsely Scanned Range Data*

Jie Chen† Baoquan Chen‡

Revision: September 30, 2007

Abstract

We present a pipeline to reconstruct complete geometry of architectural buildings from point clouds obtained by sparse range laser scanning. Due to limited accessibility of outdoor environments, complete and sufficient scanning of every face of an architectural building is often impossible. Our pipeline deals with architectures that are made of planar faces and faithfully constructs a polyhedron of low complexity based on the incomplete scans. The pipeline first recognizes planar regions based on point clouds, then proceeds to compute plane intersections and corners[1], and finally produces a complete polyhedron. Within the pipeline, several algorithms based on the polyhedron geometry assumption are designed to perform data clustering, boundary detection, and face extraction. Our system offers a convenient user interface but minimizes the necessity of user intervention. We demonstrate the capability and advantage of our system by modeling real-life buildings.

Keywords 3D scanning; range image; geometry reconstruction

1 Introduction

Acquiring 3D models of real-world objects has been an interesting and challenging problem in the computer vision and graphics communities, and is beneficial to many applications such as urban planning, architectural design, surveillance, and entertainment, to name just a few. Image-based techniques [2, 8, 17] can only achieve simple 3D geometry and generally are not robust or require significant human input. In the last decade researchers have started to employ laser scanning technology to directly perform 3D measurement of real objects; examples include the Michelangelo project [1] and the IBM Pietà project [3]. The objects that are of

*This work is dedicated to the Minnesota 3D scanning project conducted at the University of Minnesota, Twin Cities. For more information, please refer to the project website `http://www.cs.umn.edu/~baoquan/scan.html`.

†Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. `jchen@cs.umn.edu`.

‡Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. `baoquan@cs.umn.edu`.

[1]In this paper, we use the informal terms *corner* or *vertex corner* to stand for a polyhedron vertex. See the Overview section for notation declarations.

interest in these projects are usually of small to medium size (up to several meters tall); scans can be carefully set up to ensure a fairly complete and dense sampling of the entire object. Constructing 3D geometry from such scans is performed by triangulating the dense point clouds [13, 6]. Strategies have been developed to patch holes where data is missing in the scans. More recent work has explored using context [16], atomic volumes [15], or example models [14] in achieving geometry completion. However, there still does not exist a general technique that can be applied to objects that come in a wide variety of shapes.

For large outdoor architectural object scanning, it is intrinsically difficult to always obtain complete and sufficient sampling of all the surfaces due to the physical constraint of positioning the scanner. Scans obtained under these conditions can partially or completely miss entire faces (such as roof tops). Moreover, reflective surfaces such as glass windows and walls often return invalid signals to the scanner and hence are often missed. The sampling rate of a surface is sensitive to its distance and relative orientation to the scanner location. Given these additional challenges, we strive to generate complete geometry from sparse point clouds. Our strategy is to take a top down approach to geometry construction, instead of the conventional bottom up scheme of direct triangulation. At the current stage, we handle only buildings that fulfill the following condition:

Basic Assumption. *Surfaces of a building exhibit planarity and it can be represented by a (possibly non-convex) bounded polyhedron.*

A majority of architectures in existence nowadays satisfy this assumption. The planarity property allows faithfully fitting a polyhedron to the scanned data. Our polyhedral models are of significantly low polygon count compared to those with millions of triangles obtained by conventional triangulation methods. Fitting in polyhedron rather than triangulation makes the geometry construction process more immune to the usual deficiency of point cloud data in outdoor scanning. Moreover, our modeling process is mostly automatic and requires user's assistance only when certain ambiguities cannot be resolved computationally; in such cases, the user input is extremely straightforward and simple, i.e., merely selecting planes, lines or corners.

There has been research on employing certain knowledge or priors to improve modeling accuracy. For example, certain properties such as near perpendicularity between walls and floors can be leveraged when performing data fitting and shape parameter estimation [9]. Our approach can generally benefit further from such assumptions or constraints.

2 Overview

For consistency, throughout this paper, our polyhedral model representation is defined combinatorially as a collection of *faces*, *edges* and *corners*. The term *vertex* is reserved for the discussion of graphs. Each bounded face lies on an infinite *plane*, which is fitted to a set of scanned points. An edge of the polyhedron resides on a *plane intersection (line)*, i.e., it is finite and its end points are polyhedral corners.

Before introducing the modeling pipeline, we first quote the following observation from our experience of outdoor environment scanning:

Data Deficiency. *Data obtained from outdoor long range scanning suffers from noise, self and inter-object occlusion, and uncontrollable physical conditions (e.g., light and wind[2]). A laser scanner emitting lights that pass through glassed surfaces (such as windows) does not obtain valid data representing these regions.*

The scans we usually work with are missing large portions of data, and the high level of noise makes the traditional approach of triangulating point clouds inappropriate. This calls for an alternative method for modeling objects.

Our modeling process begins with identifying planar regions of the scanned data and computing their plane representations. This is accomplished by defining reliability measure for the data points, and performing clustering according to their confidence. Then neighboring information for the resulting clusters can be easily confirmed and adjacency between planes is computed. To deal with building faces that are completely missing, we devise a boundary detection algorithm to compute the piecewise linear boundaries of the identified clusters. These boundary line segments are used to guide the recovery of all the missing planes and intersections through an efficient and simple user interface.

Now that we have all the planes and intersection lines, we extract the faces of the target polyhedron. Each face is represented by one (or more) bounded polygon(s). An elegant algorithm based on *dual polyhedron* can be used to facilitate this operation with the condition that each face falls on a distinct plane and no two edges rest on the same line. We relax this restriction and solve for the face boundaries by introducing a new concept—the *cluster graph*, which shares a similar spirit with dual polyhedron but is more accommodating in practice. For certain ambiguous cases, the user provides cues or selections to carry forward the extraction. The final polyhedron consists of a collection of oriented faces that are defined as ordered lists of corners.

Figure 1 illustrates individual steps of the whole pipeline.

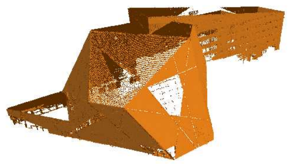
3 Planar Regions and Their Intersections

This basic step is to detect all the planar surfaces captured in the scanned data. Choosing a maximal subset of points that can be fitted by a plane within an error threshold can be done via progressive regression. However, such fitting is vulnerable to the presence of outliers.

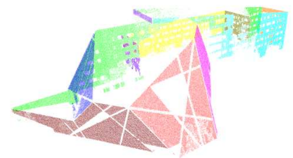
Several statistical models have been proposed to fit a function to a (sub)set of data points by pruning outliers. Fleishman *et al.* [10] use a forward search approach that grows a cluster of points to its maximal size and iteratively works on the remaining points such that several clusters, each of which represents a smooth part, are found. The algorithm robustly fits a piecewise smooth surface to a point set, but the search process is time consuming, with quadratic complexity to the size of the clusters.

Our approach is more economic and is similar to the one introduced in Stamos

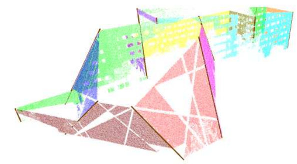
[2]Wind will affect unstable objects. For example, tree branches and leaves shake because of wind, which is a typical problem for outdoor tree scanning. However, for architectural objects, it seems that the influence of wind is minimal. Still, for reliability, a long range laser scanner is advised to be operated under mild weather conditions.



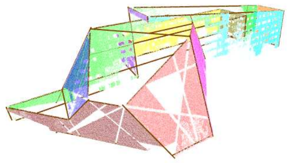
(a) Scanned data.



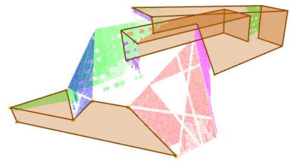
(b) Points are clustered and representative planes are fitted. (Section 3.1 and 3.2)



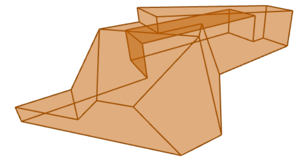
(c) Some of the plane intersections are computed. (Section 3.3)



(d) All the planes and intersections are recovered. (Section 4)



(e) Some of the faces of the target polyhedron are extracted. (Section 5)



(f) User intervention incorporated, the final model is reconstructed.

Figure 1: Pipeline of the modeling process.

et al. [18], which progressively merges points that are coplanar[3]. Note that since points on a plane share the same normal orientation, the *Gauss map* maps a polyhedron to a discrete set of points on the unit sphere. Considering noise, the normals of the data points form clusters whose centers best approximate the normals of the polyhedron's faces. This inspires us to cluster points according to their normals, where a cluster is defined as a collection of points who are approximately coplanar. Figure 2 shows an example of the Gauss map of a building. Several clusters are clearly found. Note the 'bands' in between the clusters. They represent points whose normals deviate from two clusters. Most likely they are close to the intersection of two planes, where estimation of normal rely on points from two different clusters.

3.1 Normal Estimation

Normal of a point p can be computed by least-squaredly fitting a plane to the set of points within its neighborhood. The neighborhood is defined as a 7×7 pixels region centered at p in the range image. A complete set of neighbors of p are formed by taking the union of the results from each image. Further cleaning-up, such as using segmentation techniques, distance thresholding, etc, can be performed to exclude points belonging to surfaces different from on which p lies [19].

Let $\{p_i\}_{i=1:N}$ denote the set of neighboring points of p. The eigenvectors v_1, v_2, v_3 of the covariance matrix

$$M = \frac{1}{N}\sum_{i=1}^{N}(p_i - \bar{p})(p_i - \bar{p})^T, \tag{1}$$

[3]Their method performs clustering on 2D range images and uses labeling algorithm to find connected components, while ours does clustering in 3D and no 'connected' concept is involved.

(a) Gauss map of the scanned data. Color opacity indicates confidence rate, the lighter in color the lower in confidence.

(b) Clustering result of the data with confidence higher than 0.9. Note that the two clusters, yellow and green, overlap, which indicates that they share the same normal orientation.

Figure 2: Gauss map of the scanned data corresponding to Figure 1(a).

where centroid $\bar{p} = \sum_{i=1}^{N} p_i/N$, form a local coordinate system originating at $\bar{p}$. Let the corresponding eigenvalues λ_1, λ_2, λ_3 be ordered as $0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3$. The plane being fitted to $\{p_i\}$ has a normal in the same direction[4] as the least eigenvector, i.e., v_1. Oriented towards the scanner, it is assigned to be the normal of p.

The eigenvalues of M indicate the principal components variances. The smaller λ_1 is relative to λ_2 and λ_3, the flatter the distribution of $\{p_i\}$ is. We define the *confidence rate* of p, denoted κ_p, as

$$\kappa_p = 1 - \frac{3\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \in [0, 1]. \tag{2}$$

When κ_p approaches 1, the neighborhood of p can be safely approximated by a plane, and the noise on the points $\{p_i\}$ is relatively small. Thus κ_p is a reliability estimate of p.

3.2 Scanned Data Clustering

We now cluster data points based on the computed normals n_p of each point p. The objective is that all points belonging to the same cluster are captured from the same planar surface of the building. Thereafter principal component analysis (PCA, as in (1)) can be performed on each cluster to derive plane representations which will make the faces of the target polyhedron.

In order to do fast clustering, we design an efficient algorithm that utilizes the confidence rates of the data points. They have two impacts:

- Low κ occurs on points whose neighborhood is not flat or is noisy, which means that these points occur either at the discontinuity of surfaces or where noise is high. A threshold κ_T can be introduced to filter points with low reliability.
- Points with high κ are reliable and can serve as seeds when growing clusters.

[4]We use the term *direction* in representing both of the opposite directions of a normal vector without differentiation. Only the term *orientation* exactly represents the vector direction.

Our algorithm first prunes out points with low confidence rate. For the remaining set, we pick the point p with the highest confidence (called the *seed*), search points that potentially lie on the same plane as p, and form a cluster. We set a threshold N_T to supervise the size of the cluster. In the event that the cluster is too small, it is suspected to be highly influenced by noise and we conservatively ignore point p. After a cluster is found, the process proceeds recursively with the remaining points and finds more clusters.

We set two criteria to check whether a pair of points, p and q, lie on the same plane:

1. n_p and n_q are roughly parallel, i.e., $n_p \cdot n_q$ is close to 1.
2. $p-q$ is roughly orthogonal to both n_p and n_q, i.e., $\max\{n_p \cdot (p-q), n_q \cdot (p-q)\}$ is close to 0. This criterion excludes the possibility that the two points lie on two different planes that are parallel.

Again, we can set two thresholds, p_T and o_T, to screen the parallelism of the normals of p and q, and their orthogonality to the vector $p - q$.

In order to obtain more accurate clustering, we grow a cluster in multiple passes, within each the centroid of the cluster and the normal of the approximated plane is used as the seed to grow the cluster in the next pass. We stop iterating these passes once we confirm that the normals of the planes computed in successive passes are close enough. In practice the convergence is very fast and two passes is sufficient to obtain accurate clusters.

Algorithm 1 summarizes these details. Figure 2(b) shows the clustering result of an example building.

This algorithm runs $O(N)$ time to prune out points with low confidence. Let N_r denote the number of surviving points. To grow one cluster, the cost is linear to the size of the remaining set of points. To one extreme, if each time the point selected as seed does not successfully grow a cluster, or the grown cluster is of too small size, then the time to grow all the clusters is $O(N_r^2)$, or $O(N_r)$ with a very large coefficient. To the other extreme, if most of the points each belongs to some cluster, and there are only a limited number of clusters found, then the running time is only $O(N_r)$. In summary, the time cost for Algorithm 1 depends on two parts: the total number of points in the dataset, and the number of points with high confidence. The asymptotic cost falls somewhere between $O(N + N_r)$ and $O(N + N_r^2)$.

3.3 Plane Intersections

Given two non-parallel planes P_1 and P_2 fitted from clusters C_1 and C_2, their plane intersection line $l_{P_1 P_2}$ is easy to compute. If in each of the clusters, there exist data points close to $l_{P_1 P_2}$, then these two planes are confirmed as neighboring faces of the building. We use this criterion to conservatively find the adjacent plane pairs. Nevertheless, due to deficiency of the scanned data, some pairs may not be detected this way, e.g., one plane has perfect sampling but the other one has missing data near their intersection. In such cases, the user can intervene to provide further guidance as discussed in Section 4.2.

Algorithm 1 Clustering Scanned Data

$C \leftarrow \{p \mid \kappa_p \geq \kappa_T\}$
while $C \neq \varnothing$ **do**
    $p^* \leftarrow \arg\max_{p \in C}\{\kappa_p\}$
    Set seed $s \leftarrow p^*$, $n_s \leftarrow n_{p^*}$
    repeat
        $C' \leftarrow \varnothing$
        for all $p \in C$ **do**
            if $n_p \cdot n_s \geq p_T$ **and** $|\max\{n_p \cdot (p - s), n_s \cdot (p - s)\}| \leq o_T$ **then**
                $C' \leftarrow C' \cup \{p\}$
            end if
        end for
        if $|C'| \geq N_T$ **then**
            Fit a plane to C'. Compute cluster centroid and plane normal
            Set seed $s \leftarrow$ centroid, $n_s \leftarrow$ normal
        end if
    until convergence of plane normal **or** $|C'| < N_T$
    if $|C'| < N_T$ **then**
        $C \leftarrow C \backslash \{p^*\}$
    else
        A new cluster C' is thus formed. $C \leftarrow C \backslash C'$.
    end if
end while

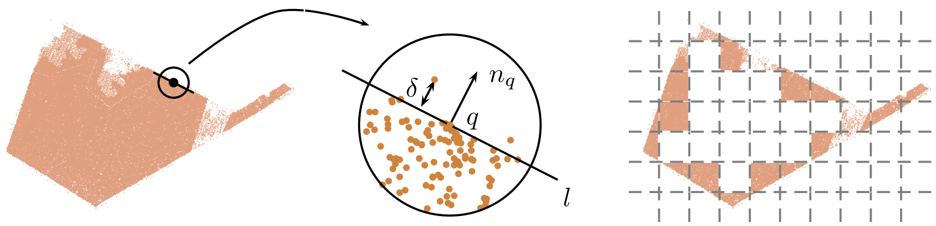
4 Boundary Detection

As a more significant situation of data deficiency, a face of a building can be entirely missed during scanning, such as roof top. The intersection of such a plane with neighboring planes can only be inferred from the boundaries of the captured data. For this purpose, we design an algorithm to compute the boundary of each identified cluster in 2D space.

4.1 Cluster Boundary

2D edge detection is a topic in image processing that has prevailed for a long time in computer vision and graphics. One can either use feature detecting filters [20], or apply the Hough Transform [4] on a particular shape, to detect edges presented in an image. Several 3D edge detection techniques have also been developed, mainly to solve the problem of range image segmentation [12]. We propose a novel method that computes the piecewise linear boundary indicated by a set of 3D points.

By projecting a cluster onto its representative plane, we obtain a discrete set of points, denoted $Q \subset \mathbb{R}^2$. See Figure 3(a). Locally if a point q sits exactly on the boundary, all its neighboring points lie on one side of the local boundary line passing through q (Figure 3(b)), unless q is close to a concave corner. In order to find this line, let the set $U = \{u_i\} \subset Q \backslash \{q\}$ denote all points within r distance to q, and let t of unit length represent the direction of the line. We setup the problem



(a) A cluster of points.

(b) The local neighborhood of a point q.

(c) Fast filtering out points far way from the boundary.

Figure 3: Analysis on a cluster of points projected onto plane.

as finding t such that

$$\max \sum_i t \times \bar{u}_i, \quad \text{where } \bar{u}_i = (u_i - q)/\|u_i - q\|. \tag{3}$$

Since $t \times \bar{u}_i$ points towards the same direction for all i (perpendicular to the plane), and the sign of $t \times \bar{u}_i$ indicates on which side of the line u_i lies, maximizing $\sum t \times \bar{u}_i$ gives a t such that as many u_i's are on the positive side of t as possible.

In the simplest computation, t is a direction orthogonal to $\sum \bar{u}_i$. In other words, $\sum \bar{u}_i$ is the normal direction of the line passing through q. Orienting it outward, we assign $-\sum \bar{u}_i$ to be the normal of q.

Let the signed distance of a point u_i to the line l be negative when the angle between $u_i - q$ and n_q is acute. We set a threshold $\delta_T < 0$ to screen the smallest signed distance (denoted δ) between all the u_i's and l. $|\delta|/r \in [0, 1]$ indicates how close q is to the boundary. When $\delta/r \leq \delta_T$, q can hardly be considered close to the boundary. When $\delta > 0$, all the points $\{u_i\}$ are on the inner side of the line. Hence using δ_T, we can identify all the boundary points.

Then we cluster the boundary points and compute the local boundary line segments. This process is simply the 2D version of Algorithm 1. $1 - |\delta|/r$ serves as the confidence rate. Points are equipped with normals, and the plane equation of the cluster becomes line equation. The fitted line segments are also computed from PCA.

Considering efficiency, we need a fast way to collect points within r-distance to point q. kd-tree [5] is the most appropriate data structure that properly supports distance queries. We use a simpler way to quickly identify and ignore the points that are not close to the boundary. See Figure 3(c). We first discretize the plane into cells of size $r \times r$. Points lying inside a cell whose 8-neighbors all contain data points are pruned out. For each surviving point q, we collect its neighboring points from only its neighboring 3×3 cells. These points go through a further check to see if they are within r-distance to q.

Algorithm 2 summarizes the details. An advantage of this algorithm is its simplicity to implement, meanwhile it is efficient. Let N_c denote the total number of points in a cluster. It takes $O(N_c)$ time to discretize the plane into cells and prune out non-boundary cells. If N_{cr} is the number of points in boundary cells, and N_g

Algorithm 2 Boundary Detection for A Cluster

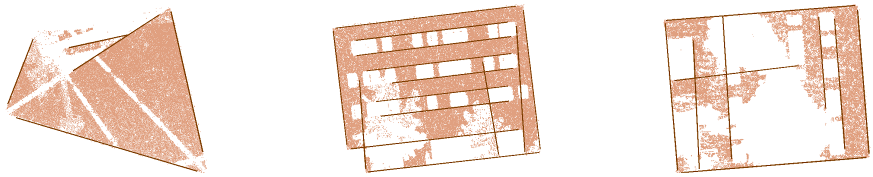
Project the cluster C onto its representative plane P. Denote the new set of points Q.
Discretize the plane into cells of size $r \times r$.
Find all cells whose 8-neighbors all contain points in Q. Denote the union of these cells $\mathfrak{C}$.
Initialize $Q' \leftarrow \varnothing$
for all $q \in Q$ **and** $q \notin \mathfrak{C}$ **do**
  Collect points from the cell that q lies in and its 8-neighbors.
  Prune out points whose distance to q is greater than r. Denote the surviving set U_q.
  for all $u_i \in U_q$ **do**
    $\bar{u}_i \leftarrow u_i - q$. $\bar{u}_i \leftarrow \bar{u}_i / \|\bar{u}_i\|$.
  end for
  $n_q \leftarrow -\sum \bar{u}_i$. $n_q \leftarrow n_q / \|n_q\|$.
  $\delta_q \leftarrow \min_i\{-(u_i - q) \cdot n_q\}$
  $\kappa_q \leftarrow 1 - |\delta_q|/r$
  if $\delta_q \geq \delta_T$ **then**
    $Q' \leftarrow Q' \cup \{q\}$
  end if
end for
Cluster points in Q' in a way similar to Algorithm 1. Use normal n_q and confidence rate κ_q for each $q \in Q'$. Fit a line segment to each cluster. Return all the computed line segments.

is the expected number of points inside a cell, then to compute normals and confidence rates for all such points takes $O(N_{cr} \times N_g)$ time. Hence the total cost is $O(N_c + N_{cr} \times N_g)$. This approach is empirically cheaper than using kd-tree, who takes $O(N_c \log N_c)$ construction time, and costs $O(\log N_c)$ in each query for a single neighbor. The cost for the remaining clustering procedure to find boundary line segments is the same as in Algorithm 1. Due to the limited number of boundary lines, this is done very efficiently.

By virtue of occlusion and scanning quality, not every cluster has a clear boundary. Some boundary lines are hard to infer. Moreover, special structures of each face, such as glass windows and doors, present fake boundaries. See Figure 4. Adding more carefully planned scans can potentially give a better outer boundary inference, but inner boundaries are largely unavoidable. It is not safe to assume that all lines computed from Algorithm 2 serve as real boundaries of the cluster, otherwise missing planes could all be automatically recovered. This necessitates the next section that discusses manual repair of missing planes and intersections.

4.2 Recovering All Planes and Intersections

We utilize the boundary line segments output from Algorithm 2 to infer faces that have not been captured by the scanner (such as roof top), and specify all the non-detected intersections. For each missing face, the user picks at least two boundary line segments and fits a plane to them. She can further indicate the missing intersec-



(a) Incomplete data that misses partial boundary.

(b) Fake boundary lines caused by windows.

(c) A cluster ambiguously split into two parts.

Figure 4: Boundary detection results for different clusters.

tion lines by choosing pairs of planes. A complete set of planes and their neighboring information is necessary before we proceed to construct the target polyhedron.

5 Constructing the Polyhedron

In this section, we describe an algorithm that reconstructs a polyhedron given all faces and edges, by way of its dual. Then we relax the restriction and introduce another way of solving the polyhedron, if only the planes and lines that its faces and edges reside on are given. Iterative user interaction is needed to resolve ambiguous situations, but such effort is minimal. We begin with introducing the concept of *dual polyhedron* in combinatorics and computational geometry in the following subsection.

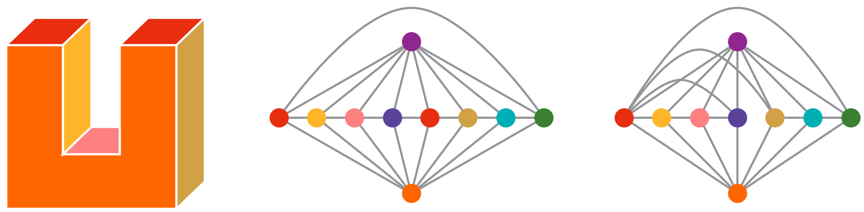
5.1 Dual Polyhedron

Every polyhedron G is associated with its dual G^*, where each vertex corresponds to a face of the other. There's an edge connecting two vertices in G^* if and only if the two corresponding faces share an edge in G. Steinitz's Theorem (1922) reveals the isomorphism between a polyhedron and a 3-connected planar graph [11].

The nice duality produces a neat algorithm to reconstruct the polyhedral model from its dual. Given the faces and edges, we form a planar graph by using vertices to represent the faces and connect a pair of vertices if the two faces have an intersection edge. Each region of the planar graph corresponds to a vertex corner in the polyhedral model. By tracing all the regions in the graph, all the corners of the target polyhedron are computed and hence the model is clear.

This appealing algorithm fails for the case that several separate faces fall on the the same fitted plane, and/or different edges rest on the same computed intersection line. See Figure 5 for an example. As can be seen from the previous Figure 4(c), due to scanning quality, it is very difficult to tell whether a cluster represents a single face or several. We state the following fact that prompts other more robust approaches to constructing the final shape.

Building Structure. *Some separate faces of a building may be (nearly) coplanar, and more than one edge may lie on the same line. By observation, most corners of a building are incident to exactly three faces.*



(a) A u-shape polyhedron.

(b) Its dual polyhedron embedded to plane.

(c) Two red vertices in (b) are collapsed into one.

Figure 5: A u-shape polyhedron where two of its faces (red) are coplanar. Its dual has a planar graph embedding. But if the two faces are considered as one whole plane, its 'dual' graph as shown in (c) cannot be planar.

5.2 Cluster Graph

We introduce the term *cluster graph* in a similar sense to the *dual polyhedron*. Each vertex in the cluster graph G^+ represents a cluster of the scanned data. Since each cluster is fitted by a plane, we also say that each vertex represents a plane of the model. There's an edge in G^+ connecting a pair of vertices if the two representative planes share an intersection line.

The cluster graph for a plane P, denoted G_P^+, is a subgraph of G^+. G_P^+ consists of all the vertices representing the neighboring planes of P and all original edges in G^+ that connect these vertices. Figure 6 shows a polyhedral model whose planes are labeled, followed by several example cluster graphs regarding to different planes.

For the simplest case, a corner in the polyhedral model is the intersection of three planes: P and two of P's neighbors. By traversing the corners on one face of the target polyhedron, in G_P^+ it equivalently means that we are walking a cycle passing through all the vertices where each pair of consecutive vertices represents two of P's neighbors that together with P form a corner in the polyhedron. See Figure 6(b) that shows the cluster graph G_α^+ for plane α, and the cycle formed.

However, a corner need not be the intersection of only three planes. Figure 6(c) shows an example case of what G_P^+ looks like if there's a corner being the intersection of four planes. Around plane β, two planes δ and λ are intercepted by a fourth plane (not shown) that does not share a line with β. All these four planes intersect to form the corner. It is sufficient to add a pseudo edge connecting the two involved vertices of G_β^+ as in Figure 6(d), and traversing the corners of the face β is equivalent to traversing the yellow circuit.

The above observation gives an algorithm to tracing out most of the faces of the polyhedral model. For a plane P, a cluster graph G_P^+ related to all P's neighbors is formed. In case two vertices in G_P^+ do not represent two planes that intersect on a line, but they are part of the set of planes (including P) that intersect at a corner, we add a pseudo edge connecting these two vertices. If there exists a Hamiltonian circuit (HC) on G_P^+, then for every pair of consecutive vertices on the circuit, the planes they represent together with P intersect at a corner. By traversing the circuit a sequence of corners are computed, where they define the unique polygonal face that lies on plane P.

Finding the Hamiltonian circuit is an NP-complete problem, but there exist

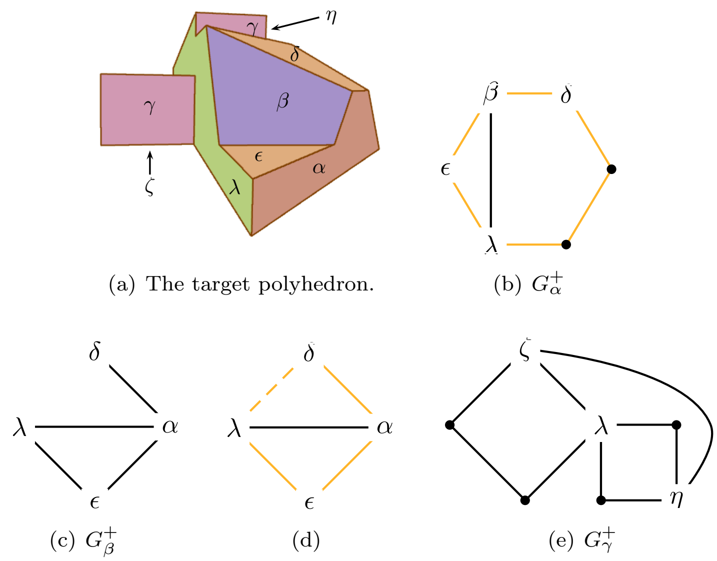


Figure 6: The target polyhedron and several example cluster graphs G^+_P. The polyhedron will be repeated in Figure 7 for ease of illustration. In the cluster graphs, whenever possible, we label the vertices by the planes they represent. (b) is the cluster graph for the plane α. Hamiltonian circuit (colored in yellow) is found. (c) is the cluster graph for the plane β. After a pseudo edge is added (as in (d)), the Hamiltonian circuit is also found. (e) is the cluster graph for the plane γ. It does not have a Hamiltonian circuit.

many heuristic low-exponential polynomial time algorithms, e.g. [7], which meet the interactive time requirement, for graphs that are not large. In practice, many of the vertices in G^+_P are of degree 2 (a corner results from the intersection of only three faces), which accelerates the finding of the circuit.

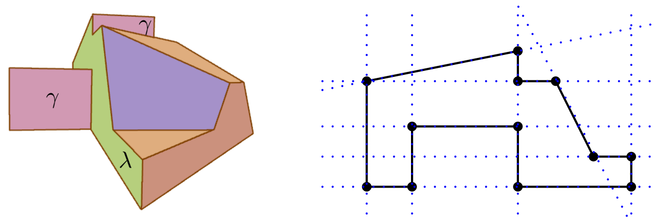
5.3 Polygonal Faces

A polygonal face can also be extracted from all its corners (order unknown) and lines passing through them, provided that the polygon is simple and no three consecutive corners are collinear, which is fulfilled in our situation. The given lines should be where the actual edges potentially lie on, and no redundant lines are presented. See Figure 7(b) for an example.

Given a line $l_{P_1P_2}$ that is the intersection of P_1 and P_2, if there are n polygon edges lying on it, then there are exactly $2n$ corners related to the plane intersection of P_1, P_2 and their common neighbors. Let the corners be w_1, w_2, w_3, $\ldots$, w_{2n}, in increasing-x (or y) order. The segments $[w_1, w_2]$, $[w_3, w_4]$, $\ldots$, $[w_{2n-1}, w_{2n}]$ define these n edges. Extracting such edges for all the intersection lines that lie on plane P, we have the exact contour of the face on P. Note that the corners $w_1 \ldots w_{2n}$ are not defined in a geometric sense, hence there may be some corners coincidentally falling on the line $l_{P_1P_2}$ but they are not caused by the intersection of P_1 and P_2.

This method has the capability of solving general cases, such as:

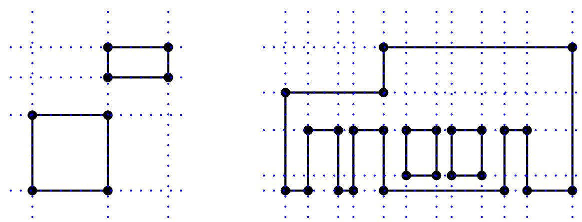
- Several mutually exclusive faces fall on the same plane. Figure 8(a) illustrates



(a) The target polyhedron.

(b) The polygonal face λ. The vertices and blue dotted lines are used as input to trace the polygon.

Figure 7: The polygonal face λ. Note that (b) is a graph different from the cluster graphs shown in Figure 6. Rather, it is the edge-vertex graph extracted from plane λ of the polyhedron. We draw the graph such that its shape is very similar to the actual shape of plane λ.



(a) The polygonal faces lying on plane γ, Figure 6(a).

(b) A face having holes. This is a real face from the model in Figure 13.

Figure 8: Illustrations of faces that consist of multiple polygons.

an example, where two polygons fall on the same plane γ.

- Faces are not simply connected, i.e, having holes. Note that these holes are different from the window boundaries as shown in Figure 4(b); they are the concave or convex part of the building geometry. See Figure 8(b) for an example.

5.4 Interaction Feedback

To one extreme, we could let the user manually specify all the corners and throw them into the algorithm in Section 5.3 to compute all the polygonal faces. But to alleviate the burden on the user, we exploit the power of cluster graphs as introduced in Section 5.2 and design a user interaction loop to complete the target polyhedron in a most convenient way.

It's not difficult to see that the Hamiltonian circuit of G_P^+ (after pseudo edges are added) exists if and only if there's only one face resting on plane P and no two edges lying on the same plane intersection line. (This equivalently means that G_P^+ is connected and the circuit does not pass a vertex more than once.) This condition is the most common situation that can be utilized.

First we compute for the user a set of potential locations where actual corners may stand, then the interaction loop begins. We compute those faces that have a Hamiltonian circuit and expose computed corners. After the user specifies some additional corners, especially those resulting from the intersection of more than three planes, we attempt to compute Hamiltonian circuits for the rest of the planes. This loops until no more planes have a potential Hamiltonian circuit. Then the user has to pick out all the remaining corners and the algorithm in Section 5.3 is run to extract all remaining faces.

Note that a Hamiltonian circuit can be traversed in two opposite directions. Hence the listing order of corners for each face of the polyhedron may need to be reversed so as to conform to the orientation of the face. Algorithm 3 gives the detailed steps of this interactive procedure.

Algorithm 3 Reconstructing the Polyhedral Model

Initialize corner list $V \leftarrow \varnothing$
for all planes P_i **do**
    Mark P_i *undone*
end for
repeat
    for all planes P_i that are *undone* **do**
        Form cluster graph $G^{+}_{P_i}$
        Add pseudo edges if existing, according to corner list V
        if HC($G^{+}_{P_i}$) exists **then**
            Trace out the polygonal face lying on plane P_i
            Add new computed corners to the corner list V
            Mark P_i *done*
        end if
    end for
    Receive user input of new corners
until no new P_i is marked *done*
repeat
    Receive user input of new corners
until all corners of the target polyhedron are in corner list V
for all planes P_i that are *undone* **do**
    From V, get all corners being the intersection of P_i are other planes
    Get all intersection lines that lie on plane P_i
    Trace out all polygonal faces lying on plane P_i
end for
for all planes P_i that are are fitted from scanned data **do**
    Adjust the listing order of corners according to P_i's orientation
end for
for all planes P_i that have no scanned data attached to **do**
    Adjust the listing order of corners according to other polygons
end for

6 Experimental Results

We've experimented with the above ideas on two buildings—the McNamara Alumni Center (Figure 14) and the Phillips Wangensteen Building (Figure 13), both located at the University of Minnesota, Twin Cities campus. The alumni center has many interesting slanted faces that are not parallel to the principal axes of the object coordinate system. It is difficult to construct from parametric primitives such as cuboid, prism or tetrahedron. The other building with gigantic body effectively demonstrates the difficulty in outdoor long range scanning and the deficiency of obtained data. It consists of about 2.5 million points, while only 15% of them has confidence rate higher than 0.9. A lot of parts have missing data, but we manage to reconstruct the building by using above algorithms.

It takes several hours to obtain a reasonable amount of data for a real life building even the scanning activity is well planned. In our situation we have a few to ten scans for each building. We preprocess the dataset, including registration and normal estimation as in [19]. We also manually trim out unrelated captured data such as pedestrians, trees, lamp-posts, etc. This is easily done via a simple user interface; however, it is not essential since scanned points on such bodies usually have low confidence rates and they will be pruned out before clustering anyways.

We first analyze our proposed algorithms and give intermediate experimental results in the following subsections.

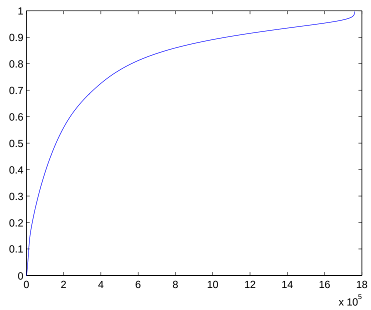
6.1 Confidence Rates and Clustering

The effectiveness of normal based clustering rely on the scanning quality. Figure 9 shows the confidence rate distributions of the two datasets. The alumni center model contains a large portion of data with high reliability. Almost 40% of points have $\kappa > 0.9$. It leaves no hesitation for us to set $\kappa_T = 0.9$ as the threshold. The clustering result is shown in Figure 2(b). For the other building, the near-linear plot indicates that confidence rates are more or less evenly distributed. As a consequence, it's difficult to set the threshold κ_T that results in good clustering. We experimented with three values. $\kappa_T = 0.9$ gives reasonable clustering that we use for demonstration throughout the paper, $\kappa_T = 0.85$ results in some extra small spurious clusters, and $\kappa_T = 0.8$ makes the program run forever which indicates that too many points do not grow a cluster and are thrown away.

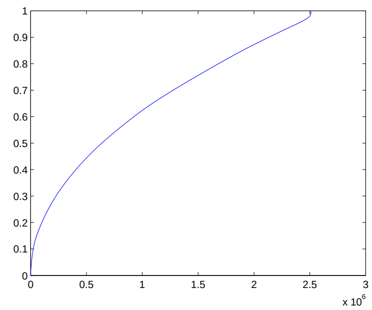
Other controlling parameters are not as critical as the confidence rate threshold. For both datasets, we dictate from experience that if two normals deviate no more than 10 degrees, and if their distance along the normal direction is less than 1, then they are considered coplanar. The minimum size of a cluster is 100. This corresponds to a 10×10 pixel area in the range image. Statistical information about the clustering will be summarized later in Section 6.4.

6.2 Boundary Detection

Our boundary detection algorithm in 2D space successfully computes real boundary line segments of a cluster of points. The effectiveness of the method rely on two parameters: the neighborhood size r and the minimum number of points in a cluster. Since r is also used as the grid size, it's easy to see that decreasing r prunes out



(a) McNamara Alumni Center.



(b) Phillips Wangensteen Building.

Figure 9: Cumulative distribution of confidence rates for two datasets. The vertical axis is the rate $\kappa \in [0, 1]$. The horizontal axis indicates the number of points with confidence rate lower than κ.

more non-boundary cells and the location of neighboring points is more efficient. However, using a smaller neighborhood tends to produce spurious boundaries, since a point is more prone to being considered on the boundary. To justify this, let's look at the the first row in Figure 10. They show the detection results of a face of a building using different parameters. The actual shape is a four-sided polygon. The large portion of missing data in the middle is due to an irregularly shaped window. Other stripes are caused by design structure. The actual boundaries are only the outer four line segments. By enlarging the neighborhood size, the 'boundary lines' in the inner part of the face are successfully ignored. But of course, it also has the risk that some actual boundary lines are omitted, as in Figure 10(c).

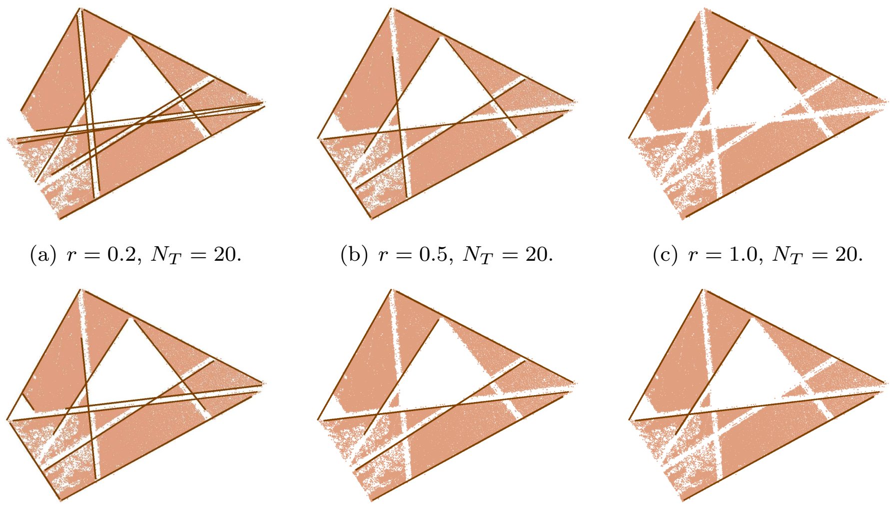
The other important parameter is the minimum size of a cluster, N_T, in grouping boundary points. Requiring larger cluster size tends to throw away 'rough' lines and keep those with good sampling. Currently the setting of this parameter is empirical, and it depends on the quality of data.

The problem of fake boundary lines exists from the nature of our algorithm and from the actual object structure. In Figure 11 we show some example building faces that exhibit bad sampling because of glass, windows, occluding trees etc. The detection algorithm outputs a lot of unwanted line segments. But a positive aspect is that the real shape and actual boundaries are able to be successfully detected.

6.3 User Interaction

The purpose of developing all the algorithms in previous Sections 3, 4 and 5 is to alleviate the user's burden. Nevertheless user input is inevitable due to the complicated datasets obtained from outdoor scanning. Different from the abstract pipeline shown in Figure 1, we show the concrete sequence of operations that build up the alumni center polyhedral model in Figure 12. It demonstrates that user intervention is very simple and never demanding. It costs an experienced user about ten minutes to finish up the model.

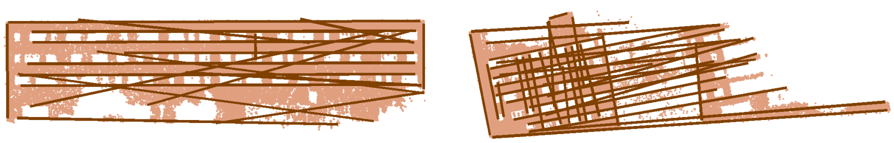
First the program processes the data, finds clusters, computes plane intersections, and detects boundaries. Most of the computations are devoted to this step,



(a) $r = 0.2$, $N_T = 20$. (b) $r = 0.5$, $N_T = 20$. (c) $r = 1.0$, $N_T = 20$.

(d) $r = 0.5$, $N_T = 10$. (e) $r = 0.5$, $N_T = 30$. (f) $r = 0.5$, $N_T = 40$.

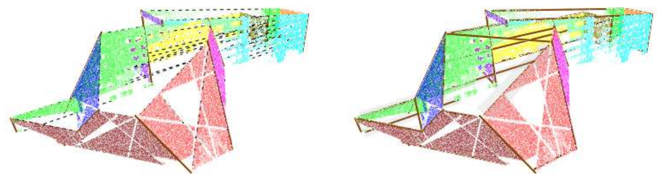
Figure 10: Boundary detection results using different neighborhood size r and cluster size threshold N_T.



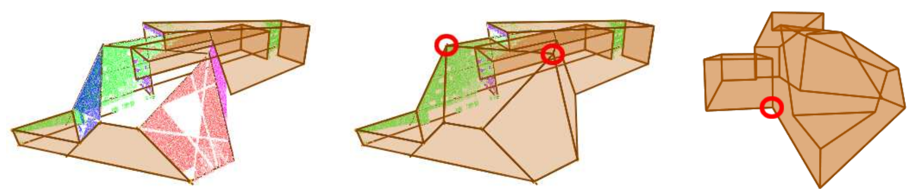
(a) (b)

Figure 11: Spurious boundary lines are obtained due to window structure and occlusion.

which takes several minutes. The result is shown in Figure 12(a). Coloring indicates different clusters, solid brown lines are confirmed plane intersections, and dotted black lines are detected boundaries. As can be seen, not all the planes are represented by the scanned data. This is where the user starts interacting with the program. See Figure 12(b). She picks a collection of boundary lines to indicate a missing plane, and selects two planes to add an uncomputed intersection line. Note that these actions are much simpler than requiring the user to manually move a plane or a line to an accurate position. With all the planes and their adjacency information available, the program automatically computes potential corners and extracts faces, as in Figure 12(c). If not all faces are computable at once, the user clicks at potential corners and let the program do the extraction again. Figure 12(c), 12(d) and 12(e) show the progression. For this typical example, the user needs only to specify three corners (emphasized by big red circles in the figures) in order to make a complete polyhedral model.



(a) Automatic. (b) Manual.

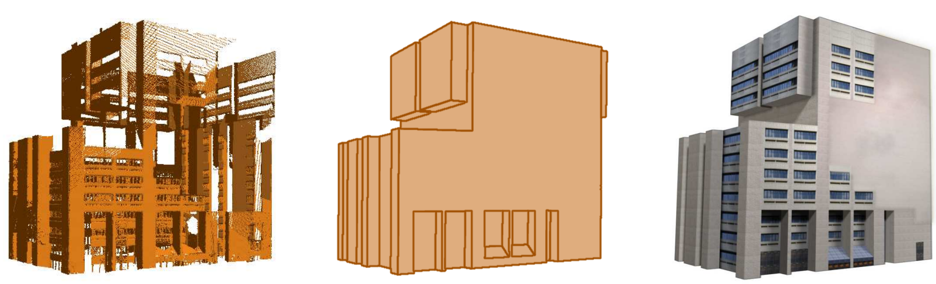


(c) Automatic. (d) Manual. (e) Manual.

Figure 12: The algorithmic computations and user interaction during the whole modeling process. Most of the computation is devoted to the processing achieving (a). The cost for achieving (c) is in interactive time. The user interaction is not demanding. As can be seen in (b) (d) and (e), it requires only picking computed lines, planes or corners.

6.4 Final Constructed Models

Figure 13 shows the model reconstruction of the Phillips Wangensteen Building. Figure 14 shows the texture mapped result of McNamara Alumni Center. The ground truth image is on the left for comparison. Statistical information is listed in Table 1.



(a) Scanned data. (b) The reconstructed polyhedron. (c) Texture mapping result.

Figure 13: The reconstruction of Phillips Wangensteen Building from scanned point clouds.

As indicated previously, the Phillips Wangensteen Building has inferior scanning quality as to missing data and low confidence rate. The reconstructed model as shown in Figure 13(b) is simple yet the best we can get. The overall shape is well recovered and some structures such as the two concave parts are preserved. Windows and floors are hard to reconstruct due to data deficiency. For applications that require a high level of realism, texture mapping can be performed on the model

(a) (b)

Figure 14: McNamara Alumni Center reconstructed. (a) is a photo of the center taken from `http://www.alumnicenter.umn.edu/about/index.html`. (b) shows two different views of the reconstructed texture model.

Table 1: Statistical data about the models.

	Alumni Center	Building
# Points	1,759,804	2,510,123
Clustering	161s	63s
Boundary detection	104s	78s
# Pts w/ high κ	689,393	376,572
# Clusters	10	35
# Faces	17	82
# Edges	43	234
# Corners	28	156
# Holes	0	2

hence surface details are represented.

7 Conclusion and Future Work

We have proposed a new representation of large-scale architectures—polyhedra, as well as a pipeline achieving this model representation from deficient range scanned data. A bounded polyhedron with low complexity is capable of representing a wide range of architectures whose faces exhibit planarity. This representation is suitable for modeling from noisy range data that contains large portions of missing or invalid values due to scanning constraints and limitation of the scanner accuracy. Our approach combines high-level automatic computations and an efficient user interface, and is proven to be effective through our experiments.

Within our processing pipeline, clustering of the scanned data based on normals and point locations is first executed. We introduce the concept of *confidence rate* in guiding the process of clustering and identifying planar regions. We also propose a boundary detection algorithm so as to compute the piecewise linear boundary of a cluster of 3D points that are close to a plane. The algorithm effectively recognizes boundary points and clusters them into linear segments. Finally, we use *cluster graph*, sharing some spirit with *dual polyhedron*, to extract bounded faces of the polyhedron. This involves finding Hamiltonian circuits; because of the low complexity of the target polyhedron the circuits can be computed efficiently.

When geometry loss or ambiguity becomes unresolvable by the computer, the modeling process is facilitated by a simple user interface which simply asks the user to make a selection from computed options. For example, to specify a missing plane, the user only needs to select two computed boundary line segments; to define a missing edge, she selects two incident planes; to confirm a corner, she only clicks within its approximate location. This interface accelerates the modeling process while yielding more accuracy.

There are many avenues of future research to improve the existing modeling pipeline. The boundary detection algorithm is based on the assumption of planarity of the geometry and piecewise linearity of the boundary. Some inner boundaries (see Figure 4(b)) are not the real edges of the polyhedral model and may interfere with the modeling process. By exploiting more knowledge or designing more sophisticated algorithms we expect to improve the accuracy of boundary extraction.

The major motivation of this paper is to utilize the unique geometric properties of architectures to devise efficient algorithms and alleviate the burden of a user. So far our approach is limited by the fact that it accommodates only planar structures. Our approach can be extended to handle non-planar shapes; perhaps the easiest is to accommodate simple natural shapes such as spheres and cylinders, as they can be analytically expressed. For that, we first cluster and identify planar points; we can then fit the remaining points with parameterized shapes. The intersections of different shapes can be computed since they either have close form solutions or can be easily represented. Constructing the geometry for more general shapes is possible, but will likely require guidance from the user on the basic characteristics of the shapes.

8 Acknowledgment

This research was funded by NSF CCF-0238486 (CAREER), NSF CCF-0530673 (REU), NSF DMS-0528492 (MSPA-MCS), NSF EIA-0324864 (ITR), Ted & Linda Johnson Gift, Riegl Gift, UMN McKnight and UMN DTC Seed Grants. The authors would like to thank Michael Kaeding and Patrick Weygand for their help on laser scanning and postprocessing (the texture mapping of the constructed polyhedral models was conducted by Michael Kaeding). Thanks are also due to Nathan Gossett for his generous help on the scanning project. The final version of the manuscript has also greatly benefited from the many suggestions of the reviewers, for which we thank them.

References

[1] The digital Michelangelo project. `http://graphics.stanford.edu/papers/dmich-sig00/`.

[2] MIT city scanning project. `http://city.csail.mit.edu/`.

[3] The Pietà project. `http://www.research.ibm.com/pieta/`.

[4] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] 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*, 5(4):349–359, 1999.

[7] F. A. Brunacci. DB2 and DB2A: Two useful tools for constructing Hamiltonian circuits. *European Journal of Operational Research*, 34:231–236, 1988.

[8] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 96*, 1996.

[9] R. Fisher. Solving architectural modelling problems using knowledge. In *Proceedings of 4th International Conference on 3D Digital Imaging and Modelling*, 2003.

[10] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[11] Branko Grünbaum. *Convex Polytopes*. Wiley, London, 1967.

[12] Xiaoyi Jiang and Horst Bunke. Edge detection in range images based on scan line approximation. *Computer Vision and Image Understanding*, 73(2):183–199, 1999.

[13] 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. In *Proceedings of SIGGRAPH 00*, 2000.

[14] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-based 3d scan completion. In *Eurographics Symposium on Geometry Processing*, July 2005.

[15] Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In *Eurographics Symposium on Geometry Processing*, July 2005.

[16] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Transactions on Graphics*, 23(3):878–887, 2004.

[17] Heung-Yeung Shum, Mei Han, and Rick Szeliski. Interactive construction of 3D models from panoramic mosaic. In *Proceedings of CVPR 98*, 1998.

[18] Ioannis Stamos and Peter Allen. Geometry and texture recovery of scenes of large scale. *Computer Vision and Image Understanding*, 88(2):94–118, 2002.

[19] Hui Xu and Baoquan Chen. Stylized rendering of 3D scanned real world environments. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, 2004.

[20] D. Ziou and S. Tabbone. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.