Architectural Modeling from Sparsely Scanned Range Data

Jie Chen Baoquan Chen

University of Minnesota

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 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

[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.

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, rather than the conventional bottom up approach 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 polyhedra 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 *vertices* will be used in 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)*.

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

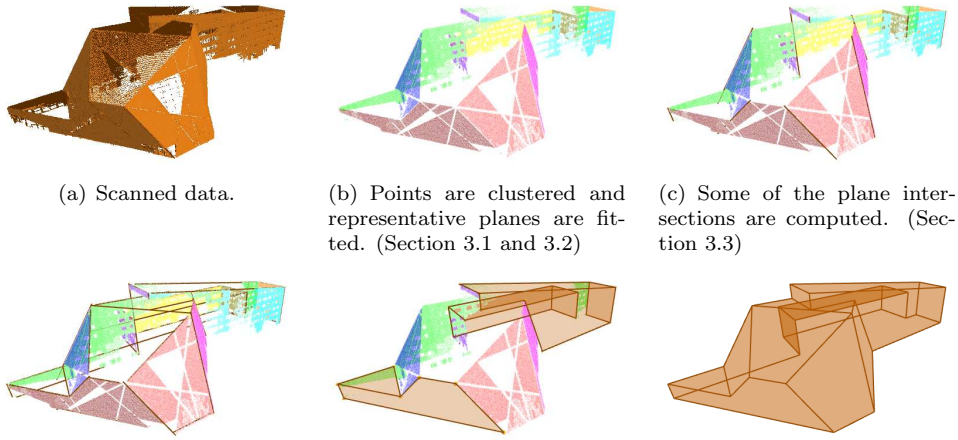
Data Deficiency. *Data obtained from outdoor long range scanning suffers from noise, self and inter-object occlusion, and uncontrollable physical conditions (e.g., light and wind). A laser scanner emitting lights that pass through glassed surfaces (such as windows) does not obtain valid data representing 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. A reliability measure for the data points is defined. We estimate normals of points and utilize their confidence to perform clustering of coplanar points. 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 where the target polyhedron faces lie as well as their intersection lines, we extract the bounded polygons of each face. 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 intersection 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 shows the pipeline of the whole process.



(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) The final reconstructed model.

Figure 1: Pipeline of the modeling process.

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.

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. Hence normals of the scanned data can be estimated, and considering noise, they form clusters whose centers best approximate the normals of the polyhedron's faces. We thus use the assistance of these normals to cluster the data points. Figure 2 shows an example of the Gauss map of a building whose surfaces are mostly planar.

(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 larger than 0.9. Note that the two clusters, yellow and green, share the same normal orientation.

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

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 neighboring points can be efficiently located in each 2D range image. A complete set of neighbors 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 [18].

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

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

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 $\{p_i\}$ passes through $\bar{p}$, and has a normal in the same direction [2] 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, as

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

When κ 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 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 of clustering 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.

The algorithm first prunes out points with confidence rate lower than κ_T. For the remaining set, it picks the point p with the highest κ (called the *seed*), searches points that potentially lie on the same plane as the seed, and forms 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. The clustering process proceeds recursively with the remaining points after a cluster is found.

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] 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.

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.

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 $p - q$.

By taking the parallelism criterion $n_p \cdot n_q \leq p_T$, the algorithm essentially considers only points having normals within the cone that has a central axis n_p and open angle $2\arccos(p_T)$. In case n_p is far from the normal of the plane that it resides on, many potential points are omitted. We do clustering in multiple passes, within each the centroid of the cluster and the normal of the approximated plane is used as the seed to find the cluster in the next pass. In practice the convergence is very fast and a 2-pass clustering is sufficient to include all points on the same plane.

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

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'. Plane equation $n \cdot (x - c) = 0$.
      Set seed $s \leftarrow c$, $n_s \leftarrow n$
    end if
  until convergence of n **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

3.3 Plane Intersections

Given two non-parallel planes P_1 and P_2 computed from two clusters C_1 and C_2, and their intersection line $l_{P_1P_2}$, the evidence that the two are neighboring faces of the building is that in each of C_1 and C_2, there exist data points close to $l_{P_1P_2}$.

Consider two neighboring faces of a real building. If both faces can be captured by the scanner, it is highly likely that there are scanned points near the intersection edge, unless the edge is obscured. We use this criterion to conservatively find the

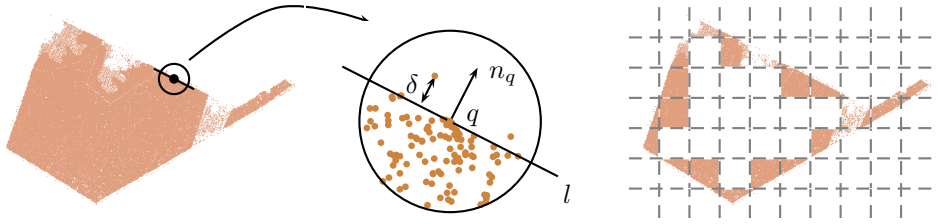
adjacent plane pairs. Nevertheless, due to deficiency of the scanned data, some pairs may not be detected this way. In such cases, the user can intervene to provide further guidance as discussed in Section 4.2.

4 Boundary Detection

As a more significant situation of data deficiency, a face of a building can be entirely missed during scanning, such as rooftops. The intersection of the plane on which such a face resides with neighboring planes can only be inferred from the piecewise linear boundaries of the captured data. On the other hand, the missing planes can be approximated from these boundary line segments. For this purpose, we design an algorithm to compute the boundary of each identified cluster.

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 [19], 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.



(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.

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, unless q is close to a concave corner. Let the set $U = \{u_i\} \subset Q \backslash \{q\}$ denote all points within the radius-r circle centered at q. Any line passing through q partitions U into two parts: U_1 and U_2. The local boundary line maximizes $||U_1| - |U_2||$.

Regardless of whether q is a boundary point or not, the line l maximizing $||U_1| - |U_2||$ indicates how close q is to the boundary. When it indeed is a boundary point, this line best approximates the local boundary. Let t represent the direction of the

line, where $\|t\| = 1$, the problem is equivalent to

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

Since $t \times \bar{u}_i$ points towards the same direction for all i, and the sign of $t \times \bar{u}_i$ indicates on which side of t 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. When $\delta \leq \delta_T$, 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. $|\delta|/r \in [0, 1]$ indicates how close q is to the boundary.

Once boundary points are identified, they can be clustered and local boundary lines are approximated. The whole clustering process is simply the 2D version of Algorithm 1. $1 - |\delta|/r$ serves as the confidence rate, and δ_T is used to prune out points that are not close to the boundary. 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. A further improvement is 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 points from only the cell that q lies in and its 8-neighbors. These points go through a further check to see if they are within r-distance to q.

Algorithm 2 summarizes the details.

Unfortunately due to 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 be automatically recovered by selecting coplanar lines. 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, and specify all the non-detected intersections. For each missing plane, the user picks at least two boundary line segments and fits a plane to the end points of them. She can further indicate the missing intersection lines by choosing pairs of planes. A complete set of planes and their

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.

neighboring information is necessary before we proceed to construct the target polyhedron.

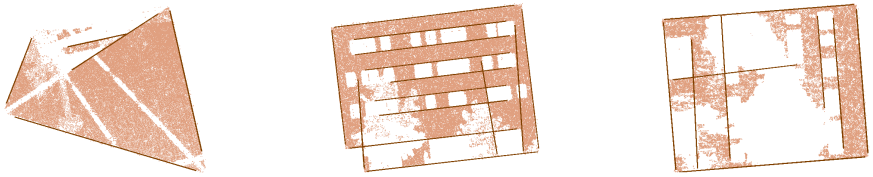
5 Constructing the Polyhedron

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.

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



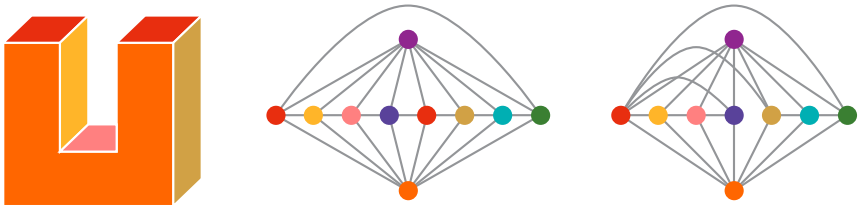
(a) Incomplete data that misses partial boundary.

(b) Fake boundary lines caused by windows.

(c) A cluster ambiguously split into two parts.

Figure 4: Boundary detection results for different clusters.

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.



(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.

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.

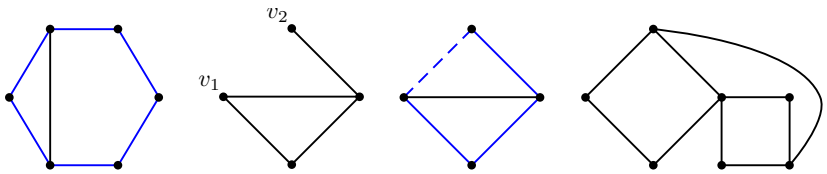
Building Structure. *Some separate faces of a building may be (nearly) coplanar, and more than one edge may lie on the same line. By observation, most corners of a building are incident to exactly three faces.*

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(a) shows an example case that is common.



(a) G^+_α (b) G^+_β (c) (d) G^+_γ

Figure 6: The cluster graphs G^+_P for different planes P. For plane labels, see Figure 7(a). (a) is the cluster graph for the plane α. Hamiltonian circuit (colored in blue) is found. (b) is the cluster graph for the plane β. After a pseudo edge is added (as in (c)), the Hamiltonian circuit is also found. (d) is the cluster graph for the plane γ. It does not have a Hamiltonian circuit.

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.

A corner need not be the intersection of only three planes. Figure 6(b) shows an example case of what G^+_P looks like if there's a corner being the intersection of four planes. The two planes represented by v_1 and v_2 are intercepted by a fourth plane that does not share a line with P. All these four planes intersect to form a corner. It is sufficient to add a pseudo edge connecting v_1 and v_2 as in Figure 6(c), and traversing the corners of the face is equivalent to traversing the blue circuit.

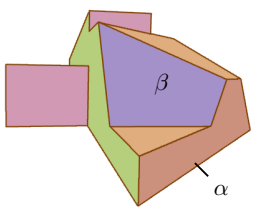
The above observation gives an algorithm to tracing out most of the faces of the polyhedral model. For a plane P, a cluster graph G^+_P related to all P's neighbors is formed. In case two vertices in G^+_P do not represent two planes that intersect on a line, but they are part of the set of planes (including P) that intersect at a corner, we add a pseudo edge connecting these two vertices. (How this corner is known beforehand will be discussed later in Section 5.4.) 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 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.

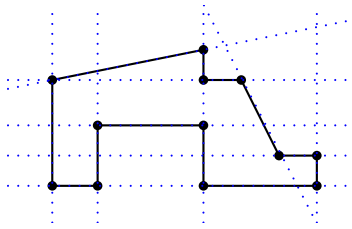
5.3 Polygonal Faces

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.

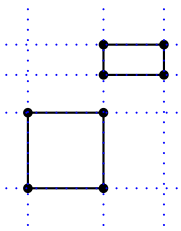


(a) The target polyhedron.

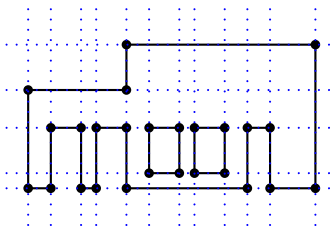


(b) The polygonal face λ. The vertices and blue dotted lines are used as input to trace the polygon.

Figure 7: The target polyhedron (same as the one in Figure 1(f)) and its polygonal faces. Labels α, β and γ are used by Figure 6. Plane γ also appears in the next Figure 8.



(a) The polygonal faces lying on plane γ, Figure 7(a).



(b) A face having holes. This is a real face from the model in Figure 9.

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

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, ..., w_{2n}, in increasing-x (or y) order. The segments $[w_1, w_2]$, $[w_3, w_4]$, ..., $[w_{2n-1}, w_{2n}]$ 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 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 previous algorithm 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.

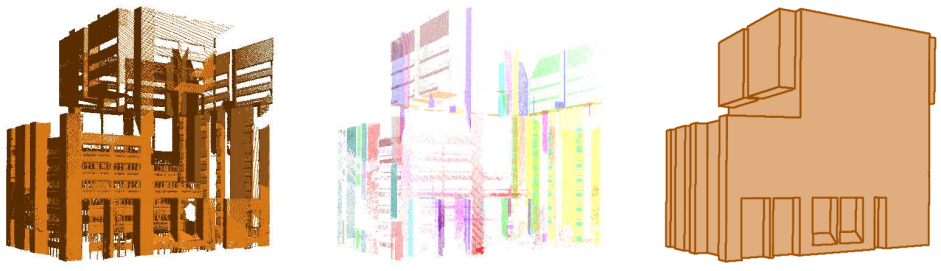
It's not difficult to see that the Hamiltonian circuit of G_P^+ (after pseudo edges are added) exists if and only if there's only one face resting on plane P and no two edges lying on the same plane intersection line. (This equivalently means that G_P^+ is connected and the circuit does not pass a vertex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

We first compute for the user a set of potential locations that actual corners may stand at, then enter the interaction loop. 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 clusters. 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 construct all remaining nonextracted 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.

6 Results

Figure 9(a) shows the model reconstruction of the Phillips Wangensteen Building, which is located at the University of Minnesota, Twin Cities Campus. The scanned data consists of about 2.5 million points registered from seven scans. Note the large portion of data missing from the top and sides. 15% of the data has confidence rate higher than 0.9. The clustering and boundary detection are handled in preprocessing and consume about two minutes. The final reconstructed polyhedron contains 82 faces, 234 edges, 156 vertices and 2 holes.



(a) Scanned data. (b) Points with at least 0.9 confidence (c) The reconstructed polyhedron.

Figure 9: A typical scanned building and its reconstructed model.

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

The scanned model in Figure 1(a) illustrates the applicability of our pipeline to reconstruct arbitrarily shaped buildings. Many of its faces/edges are not parallel to any of the three principal axes of the object coordinate system. It is also difficult to construct from parametric primitives such as cuboid, prism or tetrahedron. Figure 10(b) shows the textured model.

7 Conclusions 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.

(a) (b)

Figure 10: The texture-mapped model corresponding to the dataset in Figure 1. (a) is the photo of the McNamara Alumni Center, University of Minnesota. (b) shows two different views of the reconstructed model.

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. There are a number of implicit parameters, such as confidence rate threshold κ_T, cluster size threshold N_T, normal parallelism threshold p_T, orthogonality threshold o_T, etc, which need to be fine-tuned according to different datasets. How to automatically adjust them based on the scanning setup, e.g., scanning density, scanner distance, data quality feedback, object occlusions, etc, is an important component of improving process automation.

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.

More fundamentally, we plan to relax the basic planarity assumption to accommodate non-planar shapes, such as spherical, cylindrical and other parameterized curved patches. We will also expand the current user interface accordingly.

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] 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.

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