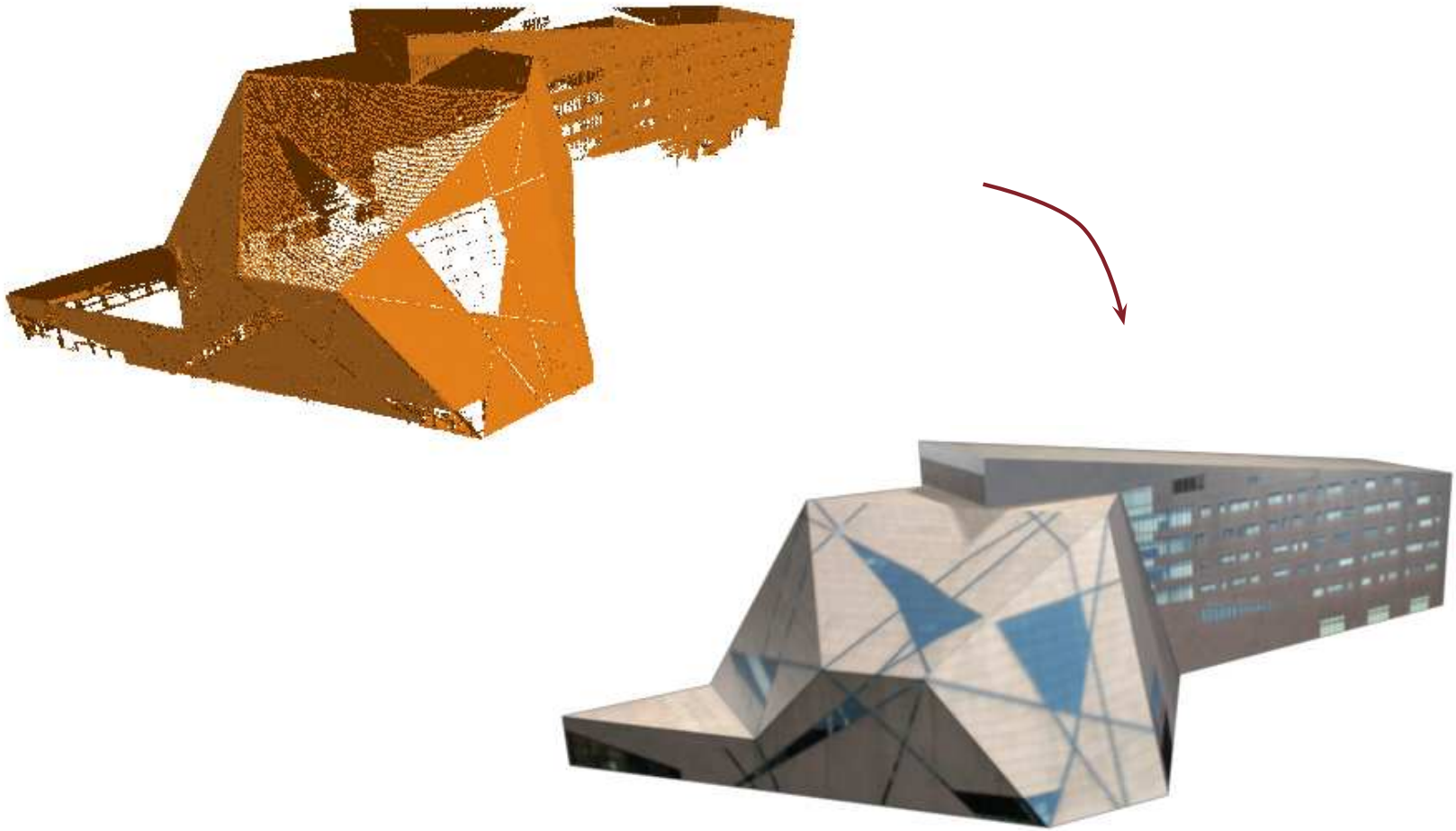
Architectural Modeling from Sparsely Scanned Range Data

Jie Chen

Department of Computer Science and Engineering
University of Minnesota

(Joint work with Baoquan Chen)



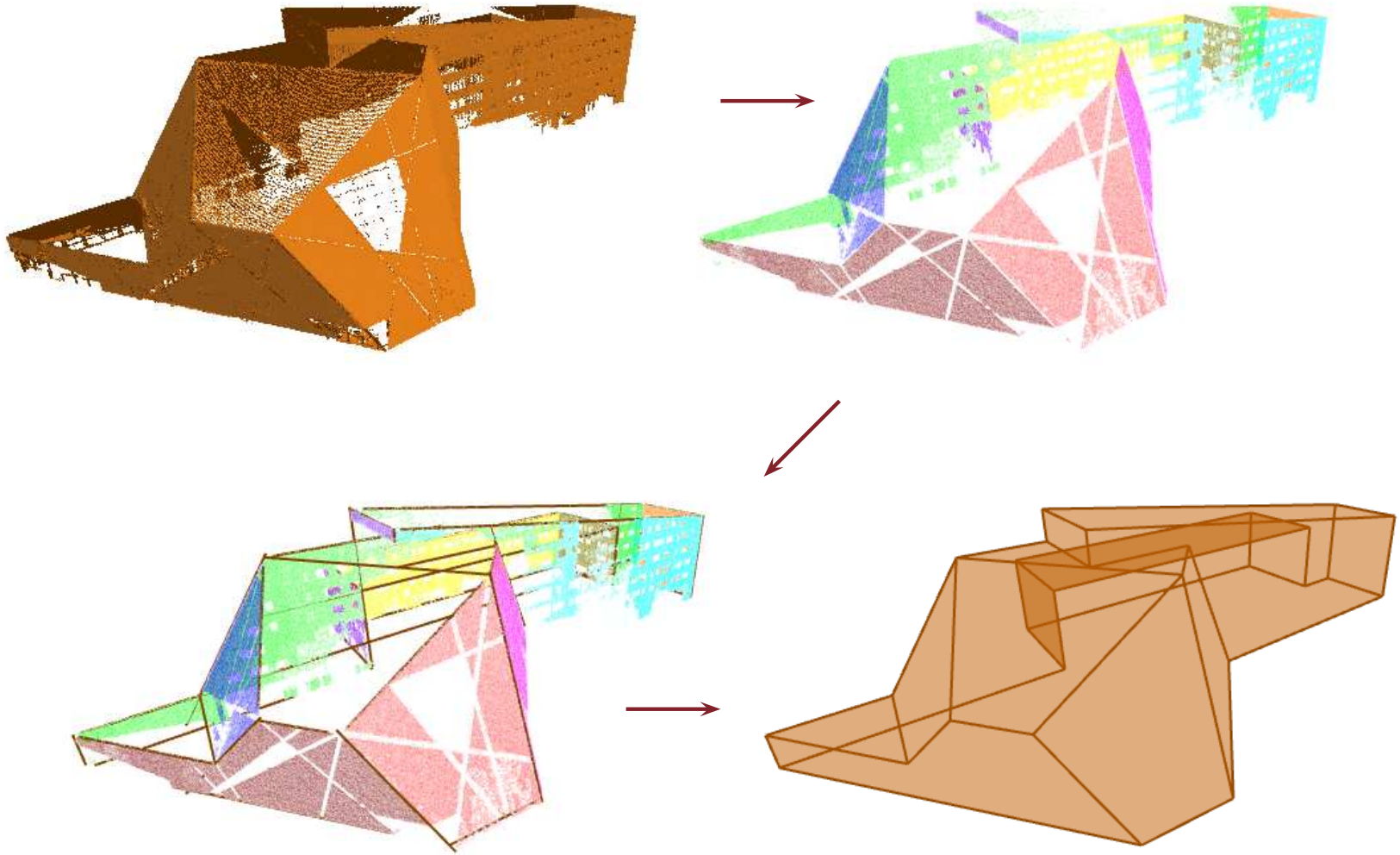
Challenges

- Sparse data; Non-uniform sampling.
- Severely missing data. (Complete and sufficient scanning impossible)
- Noisy.
- Meshing/Triangulation might not be a good idea.

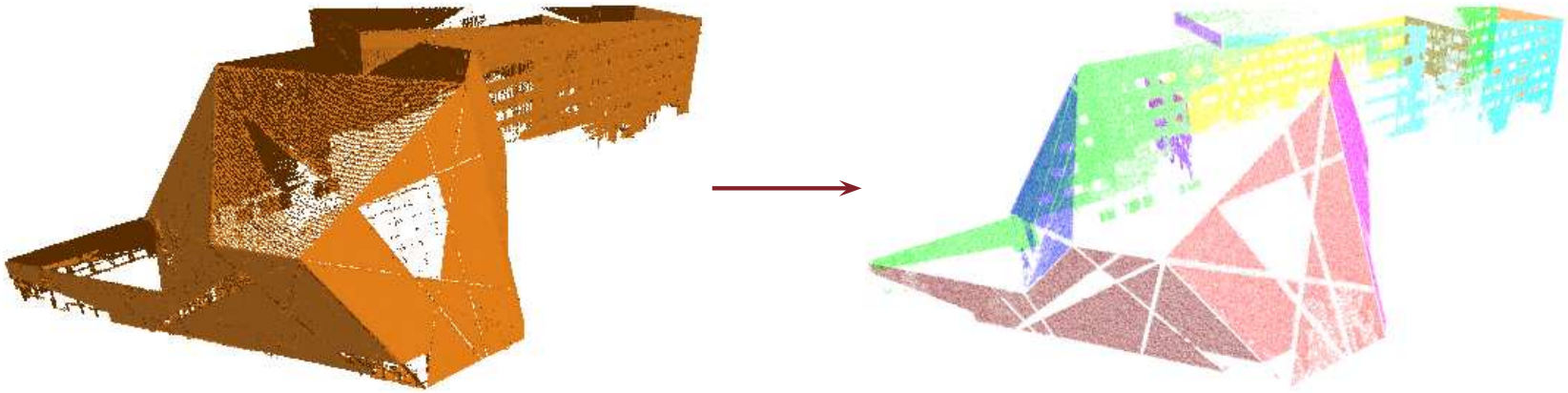
Solution: Assume planarity.

- Find planar faces for the building.
- Return a simple polyhedral model.

Pipeline



Step 1: Detect Planar Regions



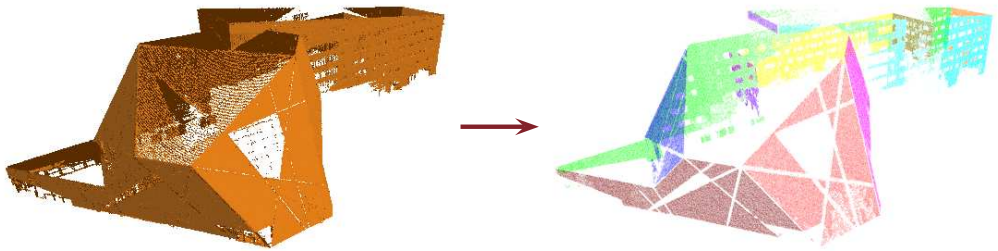
Step 1: Detect Planar Regions

General idea:

- Each scanned point is equipped with normal.
- Normals of points that are on the same planar region should roughly point to the same direction.
- Clustering the normals should yield some planar regions.

So...

- How to estimate the normal of a point?
- How to do the clustering?
- How to fit a plane given a region of points?



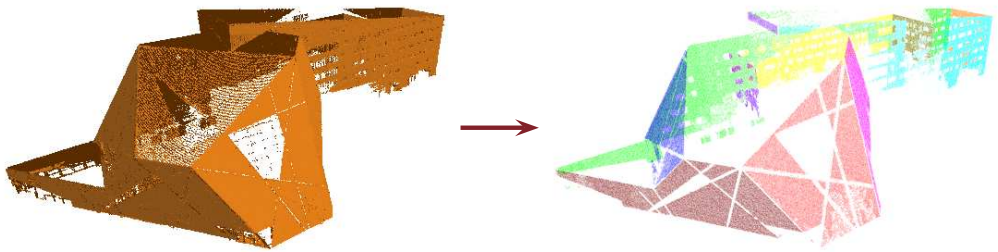
Step 1: Detect Planar Regions

Normal estimation:

- For a point p, find its neighboring points $\{p_i\}_{i=1:N}$.
- Compute the average $\bar{p} = \frac{1}{N}\sum_{i=1}^{N} p_i$.
- Compute the covariance matrix

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

- M has three unit eigenvectors v_1, v_2, and v_3, with corresponding eigenvalues $0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3$.
- This is the so-called PCA.



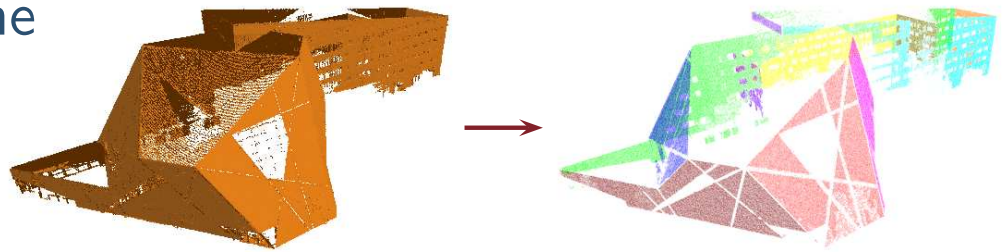
Step 1: Detect Planar Regions

Normal estimation (cont.):

- v_2 and v_3 span a plane that best fits the set of neighboring points in least-square sense.
- $\pm v_1$ is the normal direction of the plane.
- Hence $\pm v_1$ is considered the normal of point p.
- We define the reliability measure of point p:

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

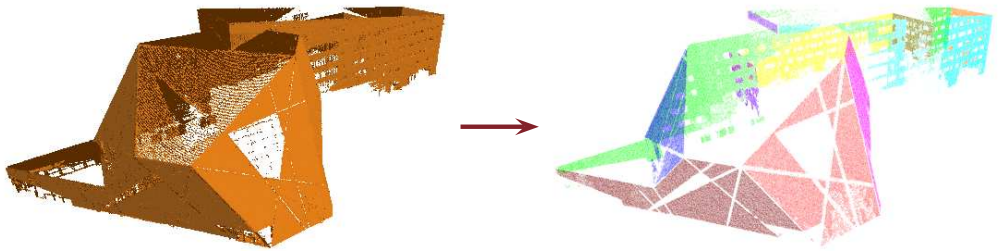
- The larger κ_p is, the flatter the distribution of $\{p_i\}$ is, hence the more confidently p lies on a planar region.



Step 1: Detect Planar Regions

Putting all the normals together, it looks like this (Gauss map):

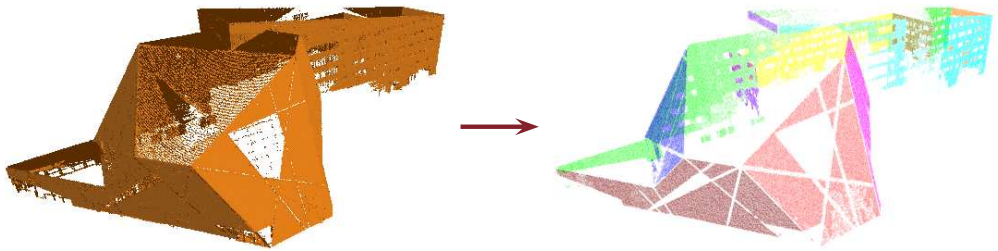
Hence a partial clustering should work! But what algorithm?
We propose a simple one that empirically works well.



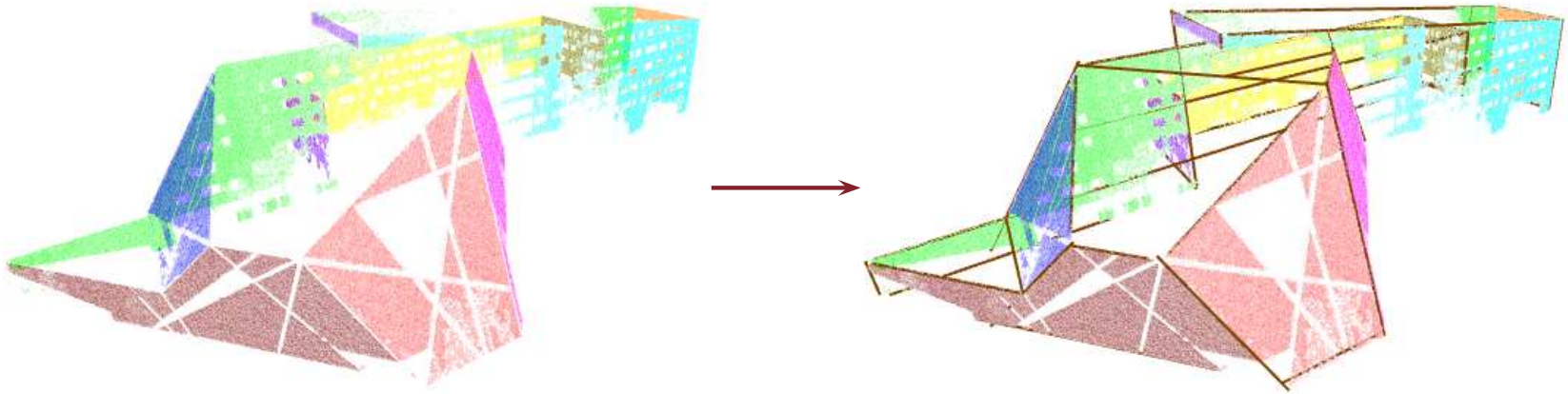
Step 1: Detect Planar Regions

ALGORITHM: Scan Data Clustering

1. Prune out points with low reliability.
2. Pick a point with highest reliability as seed.
3. Find points whose normals are close to that of the seed.
4. Prune points that cannot be on the same plane as the seed.
5. Fit a plane to the resultant points. Get plane normal.
6. Compute the centroid of the points.
7. Make the centroid as a pseudopoint, equipped with normal.
8. Use the pseudopoint as seed, goto 3 and iterate till converge.
9. A cluster is found. Delete the points. Goto 2 to find more clusters.



Step 2: Find Edges and Recover Missing Planes



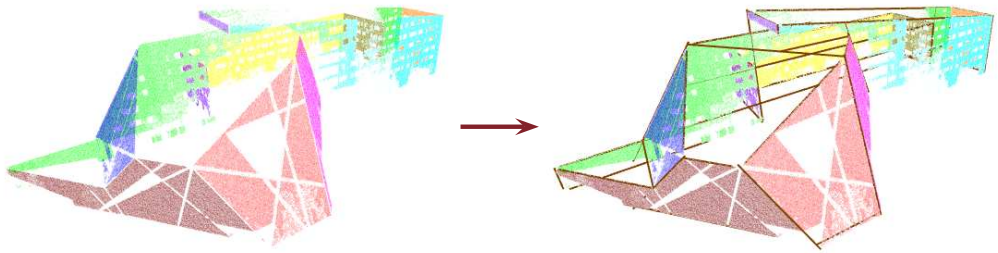
Step 2: Find Edges and Recover Missing Planes

After step 1,

- Some faces of the building are missing, due to low reliability of points or even no points.
- Sharp features (edges) at the intersection of two planes are obvious.

Three cases:

- An edge is easily known if its two adjacent planes are known and the intersection is clear.
- If two adjacent planes are known but their intersection is obscured...
- If one plane is missing...



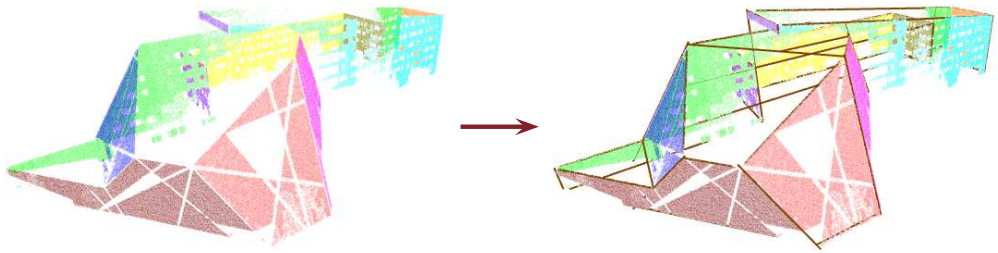
Step 2: Find Edges and Recover Missing Planes

So, a little bit user interaction can help a lot...

- Click on two planes and tell the program that they intersect to form an edge.
- Detect some sharp features on each planar region. Then click on those features to indicate a missing plane.

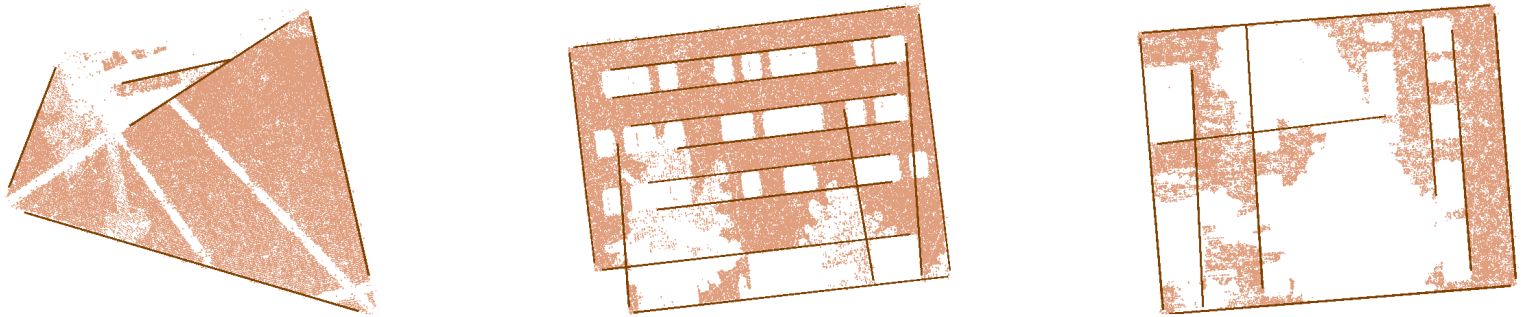
Then all planes and edges can be known.

Step 2 mostly involves user interactions, except that some sharp features can be automatically detected by the computer...

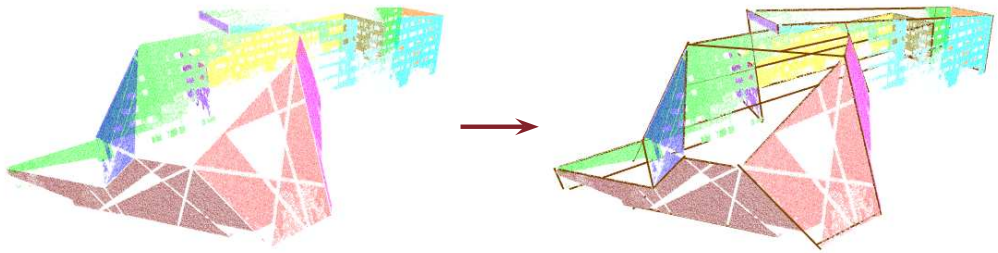


Step 2: Find Edges and Recover Missing Planes

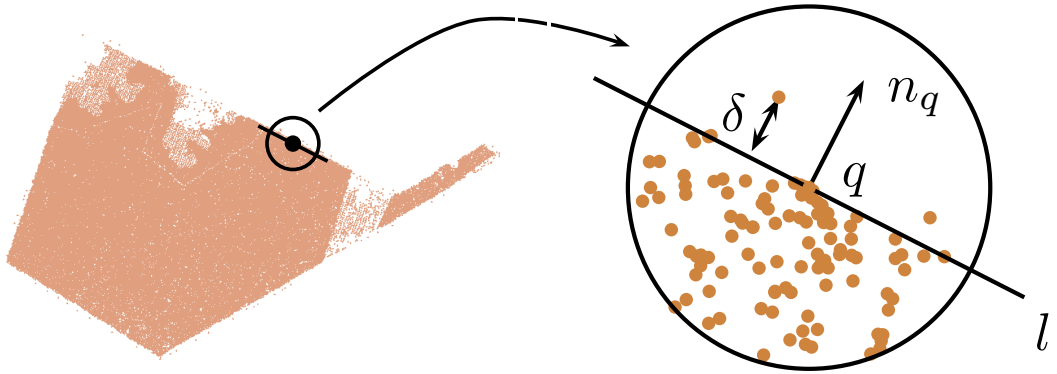
Boundary detection / feature detection:

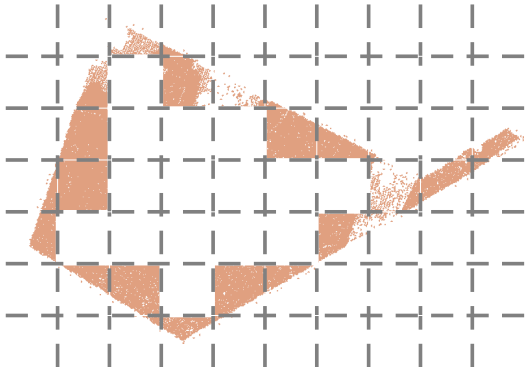


How to detect these lines (piecewise linear boundaries)?



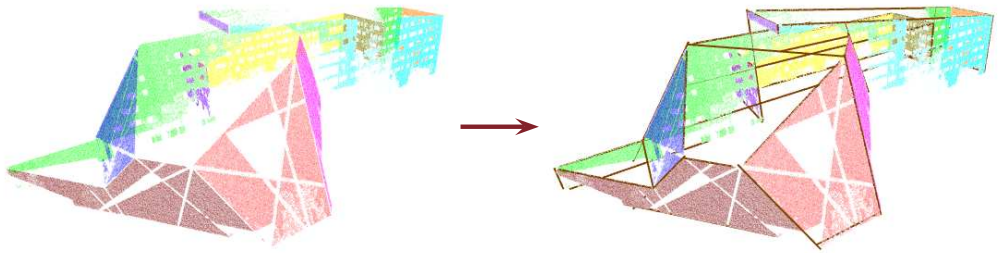
Step 2: Find Edges and Recover Missing Planes





General idea:

- For each (2D) point q, compute normal n_q.
- Compute its reliability measure (how confident on the boundary).
- Use a regular grid to prune obviously non-boundary points.
- Feed this information into the clustering algorithm previously described.



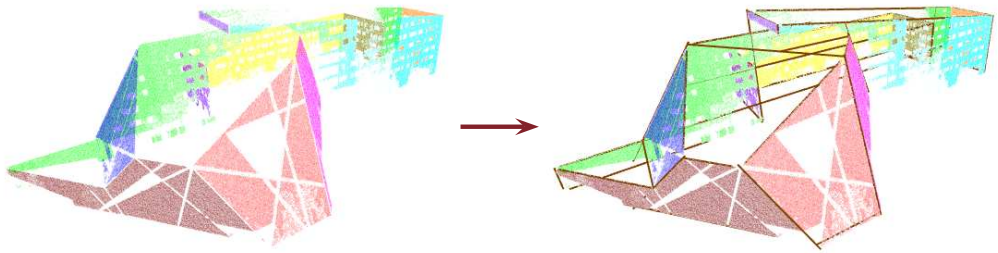
Step 2: Find Edges and Recover Missing Planes

Normal estimation:

- For a point q, find neighboring points $\{u_i\}$ within r-radius.
- Shift and normalize the neighbors $\bar{u}_i = (u_i - q)/\left\|u_i - q\right\|$.
- The normal of point q is simply $n_q = -(\sum_i \bar{u}_i)/\left\|\sum_i \bar{u}_i\right\|$.
- δ results from the farthest point away from the potential boundary: $\delta = \min_i\{-(u_i - q)\cdot n_q\}$.
- Reliability of q is $\kappa_q = 1 - |\delta|/r$.

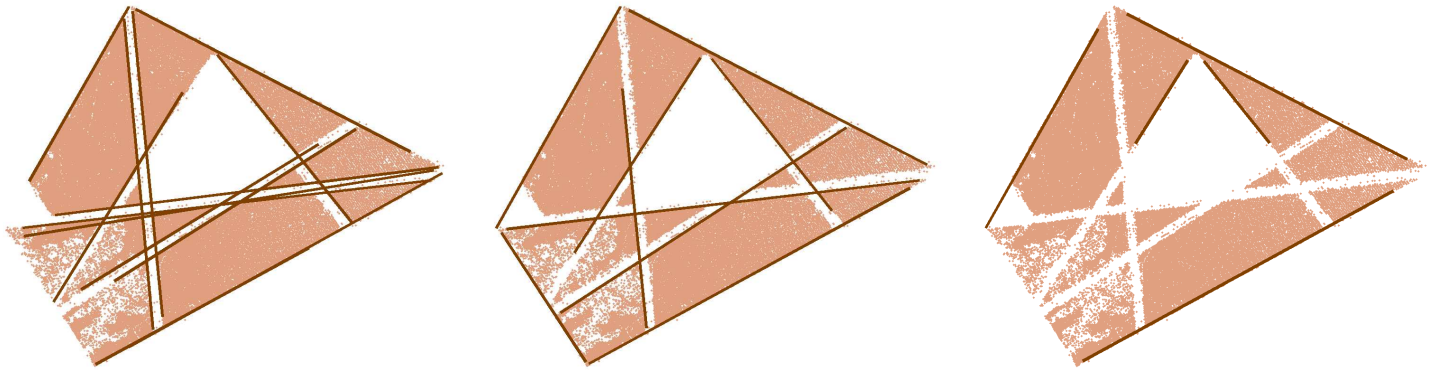
Otherwise, things are the same as the scan data clustering algorithm for detecting planar regions.

Here we detect points on the linear boundary.

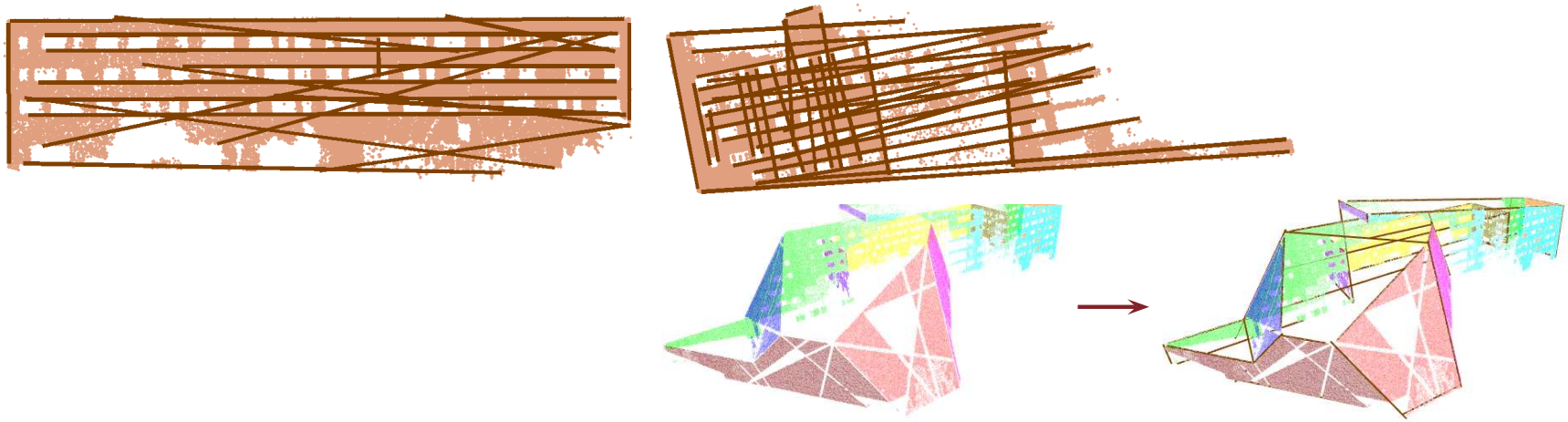


Step 2: Find Edges and Recover Missing Planes

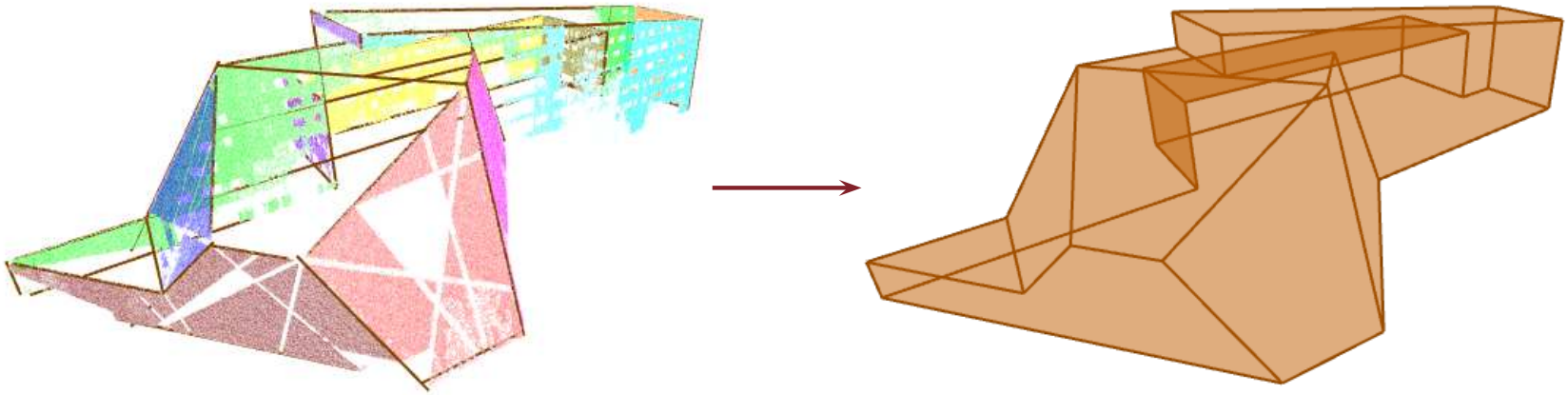
Some good/ok results:



Some wired results:

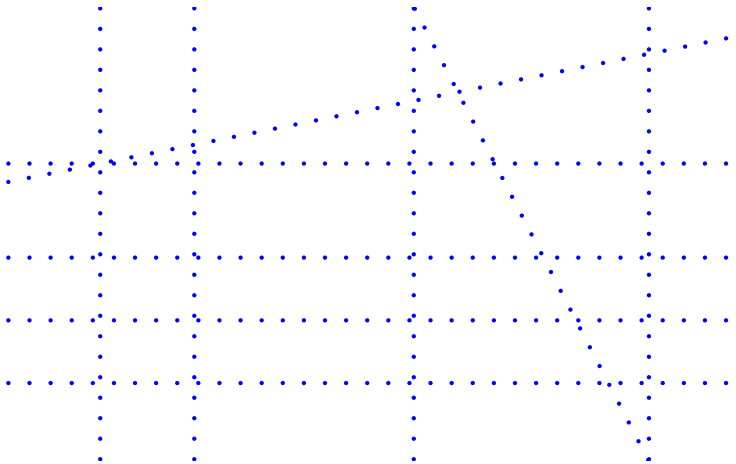


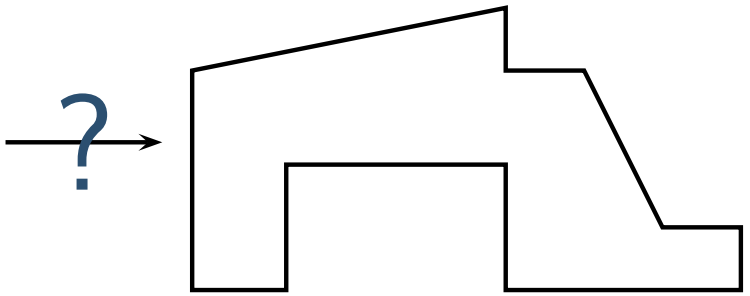
Step 3: Compute the Polyhedron



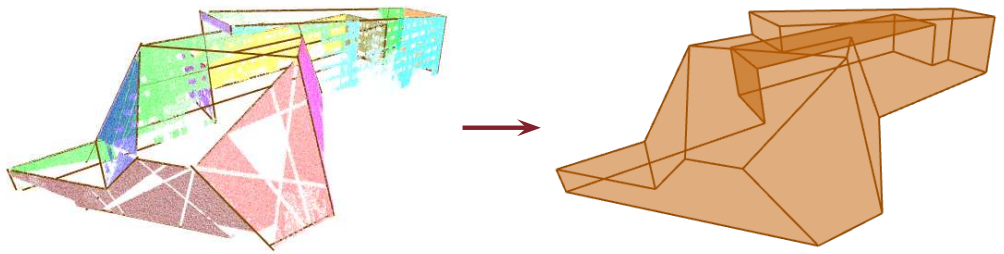
Step 3: Compute the Polyhedron

Now that we have all the planes and all the edges, and we know which plane is adjacent to which, to compute the final polyhedron is just one step further...



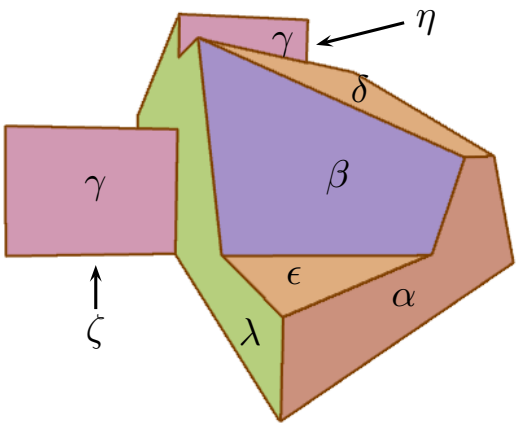


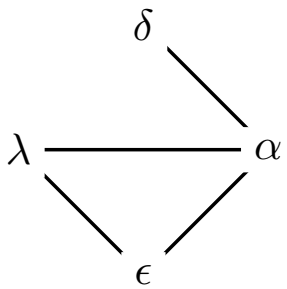
We extract the polygonal faces one by one.



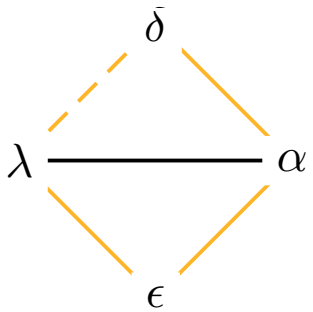
Step 3: Compute the Polyhedron

Way 1: By treating each cluster as a node, extract the Hamiltonian circuit if possible. Most of the faces can be extracted in this way.

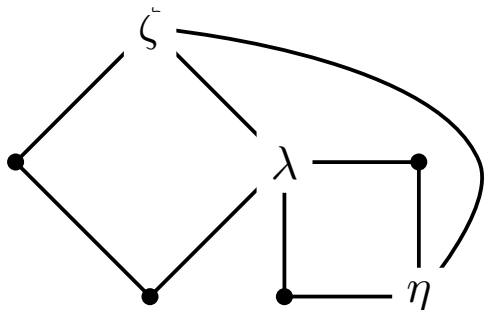




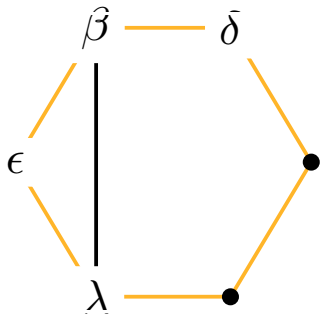
(a) G^+_β



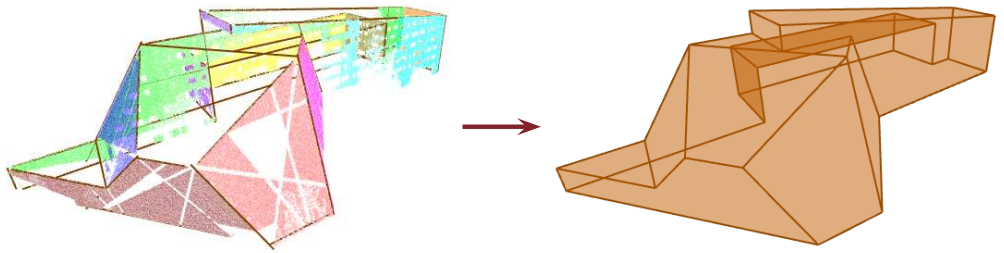
(b)



(c) G^+_γ

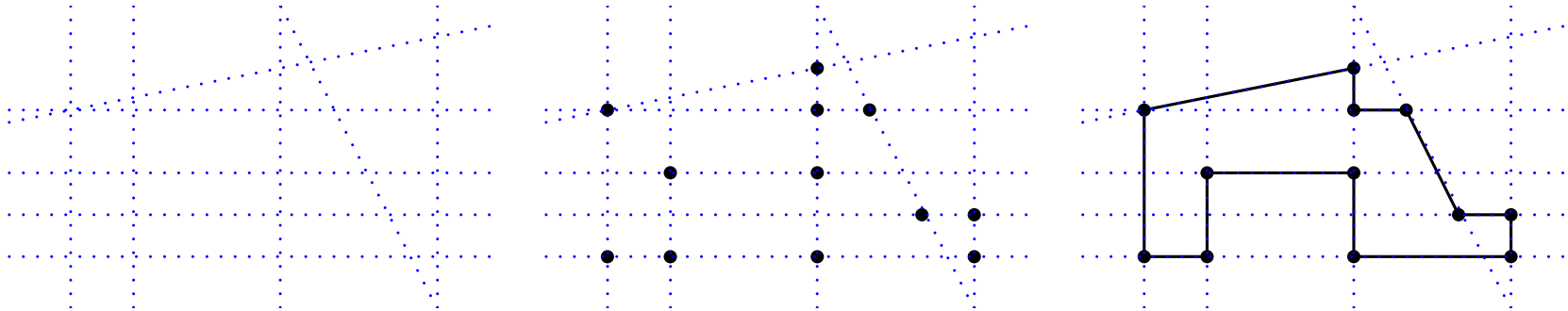


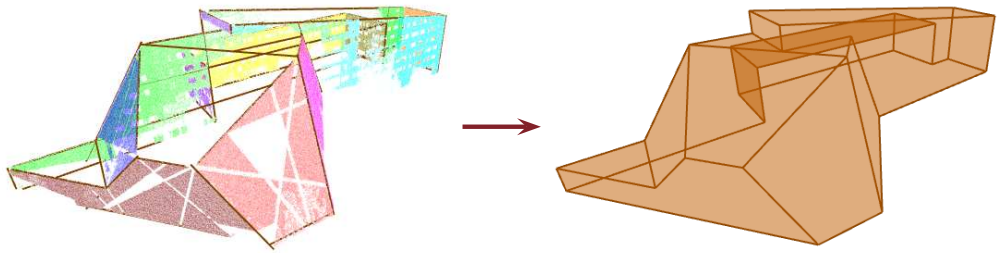
(d) G^+_α



Step 3: Compute the Polyhedron

Way 2: In case the previous method fails, we still can extract a polygonal face if we know all the corners.

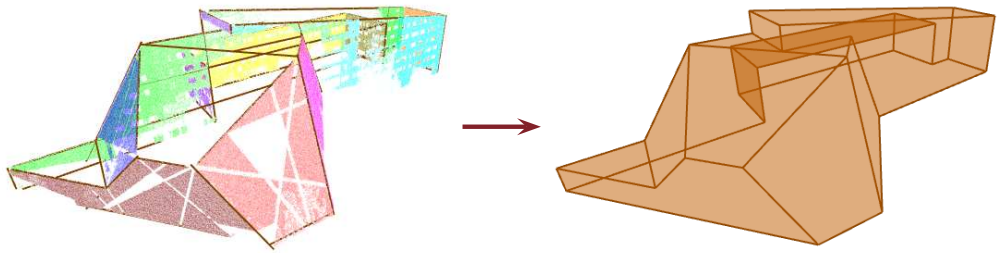




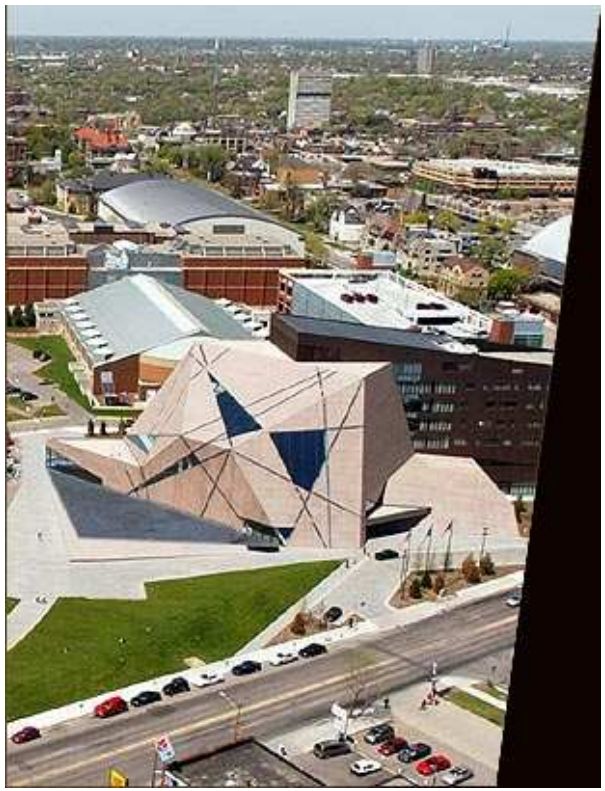
Step 3: Compute the Polyhedron

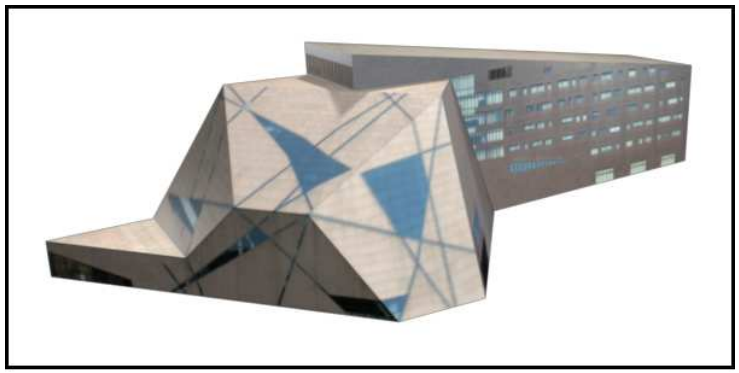
Way 2 (cont.):

Most of the corners are actually known after running the previous method, hence a user only needs to specify a small amount of unknown ones. We can pre-compute all the potential corners (since they are simply intersections of planes) and let the user click to verify.



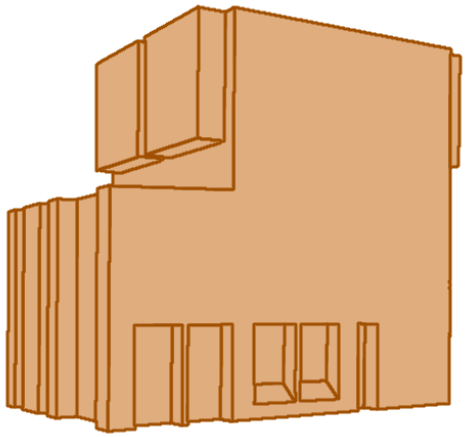
Two Real-Life Buildings





McNamara Alumni Center, University of Minnesota, USA.

Two Real-Life Buildings


Phillips Wangensteen Building, University of Minnesota, USA.

Wrap Up

- A modeling pipeline for sparse and noisy scan data.
- Three algorithms—scan data clustering, 2D boundary detection, and polygonal face extraction.
- A (hope-to-be) convenient user interface.
- For more details, please see the paper

Jie Chen and Baoquan Chen. Architectural Modeling from Sparsely Scanned Range Data. International Journal of Computer Vision, 78(2-3):223–236, 2008.

Demo Time!