CS 749-midsem

Uday Kusupati
140050048

1.(a)

The poisson surface reconstruction requires using the L matrix in which $L_{ij} = \left\langle \frac{\partial^2 B_i}{\partial x^2}, B_j \right\rangle + \left\langle \frac{\partial^2 B_i}{\partial y^2}, B_j \right\rangle + \left\langle \frac{\partial^2 B_i}{\partial z^2}, B_j \right\rangle$ a spars, symmetric matrix.

The size of L is $|O| \times |O|$ where $|O|$ corresponds to the size of the octree. So as the depth increases L is impractically larger and since there are a trillion points we cant store L in memory.

Also since the LIDAR data is that of a city, the input point cloud isn't that of a closed 3D shape but that of an open mesh (eg: a terrain).

As shown in Fig-6 of [poissonrecon.pdf][1], the lack of samples makes our method to fill the holes even when it is not desired. This results in incorrect construction of urban environments

1.(b)

The memory problem can be resolved through:

→ We can augment our matrix solver with a block Gauss-Seidel solver that iteratively solves for the solution: we decompose the matrix higher d^{th} dimensional space into overlapping regions and solve the matrix iteration in those different regions. Later we project them back into the d^{th} dimensional space and updating the residual for the next iteration. By splitting into a large number of regions, at any time the L matrix in memory is never large. [1]

→ we can also use a streaming implementation to maintain a small subset of the data into memory at any given time.

This is because all the computations at every node are local So we evaluate the dot products over the basis functions at every node but since almost all of them are zero except in close neighborhood, we require only the data of the neighborhood nodes. [2]

→ To resolve the other problem, we can ~~be~~ use the prior information ~~to~~ that the data is of an urban environment and model the basis functions in such a way that, they are exactly zero between buildings.

Also ~~glot~~ local fitting methods work much better for urban environments rather than the global fitting functions which approximate the surface as a sum of basis functions at the samples. ~~Since the LIDAR data contains the signed distance~~ we can form point neighborhoods by adaptively subdividing space, for example with an octree. ~~Blending is poss~~ Locally we can use the poisson method with a prior basis and blend them over an octree structure using a multilevel partition of unity, and the type of local implicit patch with each octree node can be selected heuristically based on urban priors.

Since we use LIDAR, we have ~~th~~ the ~~le~~ secondary information about the line of sight and the depth of view allowing Volumetric Range Image Processing (VRIP) [CL96] that ~~perfo~~ ~~does~~ performs the space carving necessary to disconnect non-connected regions which are connected by Poisson surface reconstruction.

[1] – Poisson Surface Reconstruction (2006). M Kazhdan, M. Bolitho, ~~H. Hoppe~~.

[2] – ~~Screened~~ Parallel Poisson Surface Reconstruction (~~2013~~ 2007) – M. Kazhdan, ~~H. Hoppe~~, Randal Burns, M. Bolitho

[CL96] – CURLESS B., LEVOY M.: A volumetric method for building complex models from range images.

Q.(a) Yes, we can easily adapt the ideas of marching squares to trace a C0 continuous, piecewise linear isocontour of the field. We can use the same marching squares algorithm.

→ Select a starting cell/n-gon

→ Calculate inside/outside state for each vertex based on the value of the scalar field at ~~each~~ the vertex.

→ Classify cell configuration for the n-gon.
Split the ~~verte~~ inside vertices of the n-gons into sets such ~~a way that~~ that each set is a connected component of inside vertices.
Now ambiguities can be resolved by doing join/split over two sets (connected components) arbitrarily.
We get which edges are intersected.

→ Calculate exact locations of edge intersections.

→ Link up the intersections to produce contour segments

→ Move (or march) into next cell and repeat
.. until all cells have been visited.

⟹ Since we look at the endpoints of the edge, the generated contour is continuous across the cells.
This is because ~~at~~ across cells the boundary is an edge on which edge intersection points are same corresponding to both cells/faces because we only see endpoints of edge
⟹ C0 continuity.

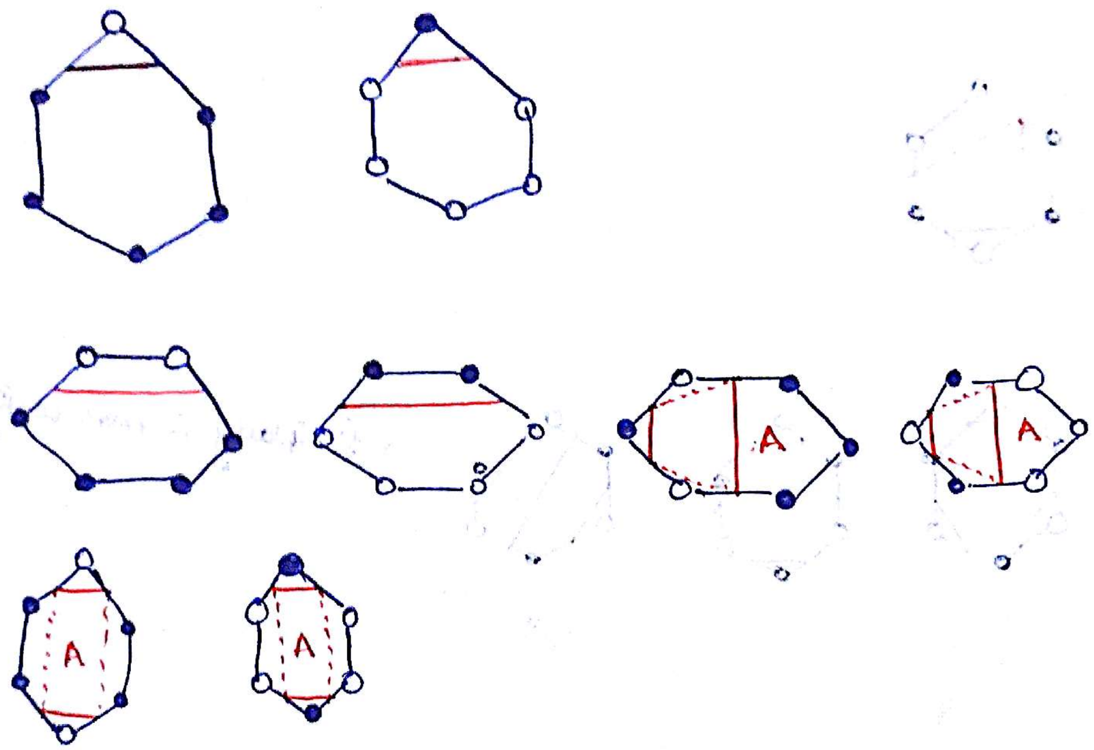
⟶ Here the degree of the polygons comes into play in only resolving ambiguities which is shown above.

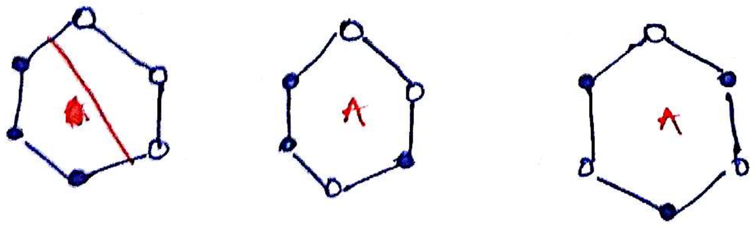
2.(b)

○ → within indicator function | green dot

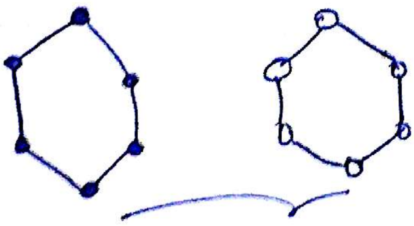
● → outside indicator function | red dot

A → Ambiguous

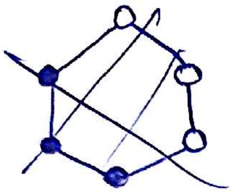


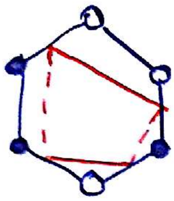


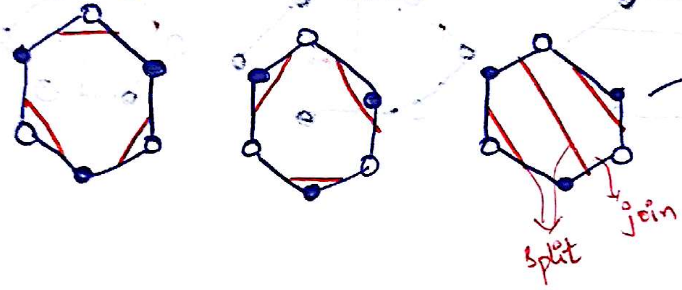
configurations listed separately



No intersections.







Similarly 2 more configurations

Q3)

Assuming set ⇒ std::set

```
void findEdges(Mesh const &m, set<Edge> &edges)
{
   for(int i=0; i < faces.size(); i++)
   {  for(int j=0 ; j<faces[i].size(); j++)
      {  int a,b;
         if(j== faces[i].size()-1)
         {  ~~a= vertices m.vertices[faces[i][j]]~~
            a=    std::min(faces[i][0], faces[i][j]);
            b=    std::max(faces[i][0], faces[i][j]);
         }
         else
         {  a= std::min(faces[i][j], faces[i][j+1]);
            b= std::max(faces[i][j], faces[i][j+1]);
         }
         edges.insert(Edge(a,b));
      }
   }
}
```