CS749 Midsem, Spring 2017

Many of these questions are open-ended, and may have many correct answers. This does NOT mean you need to write pages and pages of answers. Please write short, to-the-point answers as far as possible. You may discuss among yourselves but **your answers must be your own**. This is a small class so rest assured that copied answers can be easily detected. You may consult any resource you want, but you **must cite** these resources in your answers.

(1) **Mesh reconstruction from points (40):**

You have been hired by Google's Street View team and told that your job is to convert an entire LIDAR-scanned city – a single unannotated point cloud with 1 trillion points (with normals) covering 100 km^2 – to a polygon mesh.

(a) Why is Poisson surface reconstruction, as presented in class, not a great approach to solve this problem? Be as specific as possible.

(b) Either suggest how you would modify/adapt Poisson surface reconstruction to work in this situation, or devise a different algorithm to solve the problem scalably. Provide as much detail as possible and explain how to do the non-trivial steps so that the reader is convinced your method will actually work and has no infeasible bottlenecks. (But don't go overboard, I don't need a full research paper on the topic, a page of explanation is more than sufficient.)

(2) Marching squares (40):

(a) Is there a universal "marching n-gons" algorithm? In other words, assume you're given *any* finite tiling of a bounded 2D planar region into convex polygons (not necessarily identical, and of arbitrarily large finite degree). For instance, the Voronoi diagram of a set of points is such a tiling. There are scalar values at the polygon vertices indicating an underlying continuous scalar field. Can you easily adapt the ideas of marching squares to trace a continuous, piecewise-linear isocontour of the field? If not, argue why not. If yes, sketch out the algorithm (ambiguities can be resolved arbitrarily, and time complexity is not important). Please note that your algorithm cannot make assumptions about the degrees of the polygons in the tiling.

(b) Recall that regular hexagons can tile the plane. For "marching hexagons", draw all the different cell configurations (after factoring out symmetries) and the line segments that are generated by each configuration, just as we did for marching squares. Mark every ambiguous configuration as "ambiguous" and show all the different ways in which the ambiguity might be resolved.

(3) **Polygon meshes (20):**

You are given a polygon mesh in the stored in memory as a `Mesh` object with the following specification:

```
struct R3
{
  double x, y, z;
};

struct Mesh
{
  vector<R3> vertices;
  vector< vector<int> > faces;
};
```

`vertices[i]` is the position of the i'th vertex, and `faces[i]` is the list of vertex indices (starting from 0) of the i'th face.

Write the body of the following function `findEdges` as compactly (but readably) as possible:

```
struct Edge
{
  int v[2];  // indices of two endpoints (in arbitrary order)

  Edge() { v[0] = -1; v[1] = -1; }
  Edge(int i, int j) { v[0] = i; v[1] = j; }

  bool operator==(Edge const & b) const
  { return v[0] == b.v[0] && v[1] == b.v[1]; }

  bool operator<(Edge const & b) const
  { return v[0] < b.v[0] || (v[0] == b.v[0] && v[1] < b.v[1]); }
};

// Populates "edges" with the set of undirected edges of mesh
// "m", without repetition.
void findEdges(Mesh const & m, set<Edge> & edges)
{
  // FILL IN THIS BLANK
}
```

Your syntax doesn't need to be perfect and it's ok to use shorthand and pseudocode, but keep it reasonably close to actual C++. Add comments so we know what you're doing.