CS 542G: Mesh Generation

Robert Bridson

November 24, 2008

This lecture isn't properly written out, but I'm summarizing the important points for now as a substitute.

1 Mesh Quality

Before jumping into algorithms for generating meshes, we should first figure out sort of mesh we want. While most applications needing meshes generally agree on some measures—we want meshes to have as few elements as possible for efficiency reasons, while still providing high quality elements—the notion of what "quality" means can vary quite a lot.

- From the standpoint of L^2 or other similar norms, we just want the maximum distance in each triangle (i.e. the longest edge) to be as small as possible. To fill a given domain with as few triangles as possible points us towards requiring the best area-to-longest-edge ratio, which is maximized for an equilateral triangle.
- However, last lecture showed Galerkin FEM really is picking the best member of the subspace with respect to the energy norm, which is based on the gradient. This gives a stronger requirement on mesh quality.
- We analyzed the quality of a triangle in the gradient norm by fitting an isotropic quadratic, i.e. one with the quadratic term proportional to $x^2 + y^2$, through the vertices of the triangle and viewing the difference between the actual gradient and the gradient of the linear interpolant.
- The difference of the functions is a paraboloid which goes through zero at the triangle vertices; the zero level set must be a circle centred on the minimum of the paraboloid, and therefore the minimum is at the circumcentre of the triangle.

- The magnitude of the gradient increases linearly away from the circumcentre, and therefore the error in the triangle is determined by how far the circumcentre is from it.
- Therefore, tall and thin acute triangles may be inefficient (small area compared to longest edge) but are basically harmless since they contain their circumcentre. However, wide and short obtuse triangles are very bad, since their circumcentre can be far away. This boils down to wanting to avoid large angles and/or large circumcircles in the mesh.

2 Delaunay Mesh Generation

- Several mesh generating approaches exist. For example, the "advancing front method" is a fairly effective engineering-style solution, where high quality elements are constructed first along the boundary and working inwards until the domain is completely filled. In 2D especially many good options are out there.
- However, dealing with 3D domains, particularly if there are sharp angles or other geometric difficulties in the domain, has proven extremely difficult for many approaches. Probably the most promising one is derived instead from computational geometry, with a strong theoretical backing: Delaunay methods.
- Delaunay arises naturally in many different guises. For example, it is the dual of the Voronoi diagram, which is the partition of space into the areas closest to each of a given set of points—this partitioning has a strong tie to numerically solving PDE's, which at some level boil down to the local interaction of one part of space with the neighbouring regions.
- Delaunay also can be defined as a triangulation over a given set of points for which no triangle contains another point in the interior of its circumcircle—clearly limiting the size of circumcircles, which is very good from the standpoint of our error analysis of FEM.
- Delaunay triangulations are also one of simplest to construct efficiently, featuring $O(n \log n)$ or better algorithms.
- One example algorithm which is very popular is Lawson's edge-flipping method. We start with an artifical giant triangle that contains all the points. Then points are inserted one at a time into the current mesh—we locate the triangle a new point is contained in, replace it with three triangles connecting up the new point, and then run edge-flipping to make the result Delaunay. With a randomized order of insertion, and some sort of tree structure (similar to Barnes-Hut) to speed up the point location, this is very efficient. The end result is the Delaunay mesh embedded inside the

initial giant triangle, which can be removed to get a mesh of the convex hull of the input points. Slightly more elaborate versions of Delaunay can handle non-convex regions as well.

- To get high quality (every edge shorter than a given Δx, no large angles) more points may need to be added—in fact often, the initial set of points is just a small set on the boundary. Adding the circumcentres of large triangles is a particularly good idea, though slightly better choices have been found.

- The mesh generation process relies on geometric tests for whether a point is inside a triangle, and whether a point is inside the circumcircle of a triangle. In some sense these can be ill-posed, since if a point is very close to the edge, a small perturbation can change the boolean answer from true to false or vice versa. In floating-point arithmetic this spells trouble! Computational geometry has developed solutions either using exact arithmetic (say with integer coordinates only) or sophisticated error analysis not for the faint of heart.

- A great freely available 2D package to use is Triangle, by Jonathan Shewchuk; in particular, it does a brilliant job of dealing with floating-point errors, and provides excellent guaranteed angle bounds etc.

- Finally, it's typical in scientific computing to take the raw mesh output of a Delaunay algorithm and further improve it by slightly adjusting the positions of the vertices to locally optimize some measure of quality. This is called "mesh smoothing".