Refine elements

- if error is too large
  \[ \| u - \Pi h u \| \leq C h^2 \| u \| \]
- estimate 2nd derivative
- mark all elements to be refined
Adaptive Meshing

Adapting a triangulation
- 1D: bisect edges which are too long according to an error criterion
- binary refinement
  - restriction criterion

Binary Refinement

Irregular refinement
- new point doesn’t have to be in the middle
- naturally managed as a binary tree
- un-refinement?
UN-REFINEMENT

Remove excessively refined elements
■ particularly important in dynamic problems
■ in a binary tree, just prune

2D MESH REFINEMENT

Different options
■ quadrisection
■ conforming edges
2D Mesh Refinement

Quadrisection
- manage as a quad tree
- conforming edges are only temporary
- restriction criterion now essential
- un-refinement as before
- but: doesn’t fix bad aspect ratios

Bisection
- for a marked triangle bisect its longest edge
- angles degenerate
- not good...
Maintaining a Delaunay triangulation
- this only works in the planar setting
- insert new points in worst triangle
- then re-establish Delaunay property through edge flips

Bisection
- works both in 2D and 3D
- recursively bisect the longest edge
- decreases the largest angle
- LSPP: longest side propagation path
**LSPP**

**Definition**
- the LSPP of a triangle $t_0$ is a sequence of triangles $\{t_0, t_1, t_2, \ldots\}$ such that $t_i$ is the neighbor of $t_{i-1}$ across the longest edge of $t_{i-1}$
- always finite for bounded geometry
- ends on boundary or in pair of terminal triangles

**Example**
- backward refinement

```c
Refine(t) while( !bisected(t) ){ l=LSPP(t) if( boundary(l.tail) ) bisect(l.tail) else bisect(l.tail) bisect(prev(l.tail)) }
```
**Backward Refinement**

Properties
- linear time for bounded geometries
- results in quasi-equilateral triangulations
- smallest angle created is larger than half the smallest angle in $t_0$

---

**Delaunay Version**

Backward longest edge refinement of Delaunay triangulations

```java
DelaunayRefine(t)
while( !bisected(t) ){
    l=LSPF(t).tail
    e=l.longest
    DelInsert(e.midpoint)
}
```
**UN-REFINEMENT**

For arbitrary triangulations
- vertex removal
- followed by retriangulation
- edge collapse
- triangle collapse
- instance of 2 successive edge collapses

---

**PROGRESSIVE MESHES**

Hoppe 96
- originally designed for geometric mesh simplification
- basic operation: edge collapse

![Edge collapse and half-edge collapse](image-url)
**Edge Collapse**

When is this topologically legal?
- in planar setting iff no triangle reverses or gets zero area
- measure signed areas before/after
- manifold case
  - vertex rings of edge intersect in exactly 2 vertices

**Legality condition**
- topology
- for manifolds also
  - geometry
  - much harder
  - use projection plane
    - linear programming problem

CS257c 1999
ERROR CRITERIA

Geometry approximation
- least squares distance
- like putting in springs!
- conserved volume
- amazingly effective
- error quadrics
- weights for tension and bending

ERROR QUADRICS

Distance of point from plane
- each vertex gets one
\[ \sum_{i=1}^{k} \|v^T n_i\|^2 = v^T (\sum_{i=1}^{k} n_i n_i^T) v = v^T Q v \]
- error of edge collapse is value of quadric for best collapsed point position
- can be extended to other properties, e.g., temperature
Take quality of mesh into account
- distance is 0th order
- 1st and 2nd order also interesting
  - 1st order is distortion of triangle
  - 2nd order is curvature
    - e.g., excess angle at vertex