Dynamic Mesh Refinement
Benoît Hudson

Joint work with Umut A. Acar,
Gary Miller, Todd Phillips
Why Mesh?

- Geometry is continuous
- Computers are discrete
- Mesh for any geometric processing:
  - Graphics
  - Scientific Computing
  - Vision, AI, ...
Scientific computing: historically

- CAD
- Mesh
- Ax = b
- Numbers
- Weak form

Once per project
Once per career
Once per timestep

\[ \rho \left( \frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \nabla \cdot T + f. \]
Scientific computing: historically

Once per project

CAD → Mesh

Ax = b → Numbers

Once per career

Weak form

\[ \rho \left( \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}. \]

Once per timestep
Scientific computing: modern

$\rho \left( \frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \nabla \cdot T + f.$

- **CAD** → **Mesh** (Once per timestep)
- **Ax = b** (Once per timestep)
- **Weak form** (Once per career)
- **Numbers** (Once per timestep)
Claims of the Thesis

- The fastest static meshing code
- A framework with lots of explicit freedom for point placement
- The first dynamic meshing algorithm
Outline in four parts

Goal of the talk:
Enough pictures to understand main points
Enough main points to understand thesis

• What makes a good mesh?
• Experimental results
• General conceptual algorithm
• Sub-linear dynamic mesh refinement
Input Description
Quality measure: radius/edge

\[ \frac{r}{e} = \frac{1}{2} \sin(\phi) \]

\[ \phi > \alpha \]
\[ \frac{r}{e} < \rho \]
Voronoi aspect ratio

if $r/\text{NN}(v) < \rho$
then $r/e < \rho$
3d: Slivers
3d: Slivers

$0^\circ \text{ angles}$

$r/e = 1/\sqrt{2}$

Edelsbrunner et al 00
Cheng et al 00
Chew 97 / Li-Teng 01
Labelle, Shewchuk 06/07
Simulation Runtime

- Runtime: \(O(# \text{ triangles}) \) or \(O(# \text{ triangles}^{3/2})\)
  - Don’t create too many elements
- Timestep length: \(O(\text{shortest edge})\)
  - Don’t create tiny elements
local feature size

\[ \text{lfs}(x) \]

\[ x \]
lfs and mesh size

- \( \text{NN}(v) \in \Omega(\text{lfs}(v)) \) at every vertex \( v \):
  - Size-conforming

- Then \( \#\text{vertices} \in O(\int \text{lfs}^{-d}(x)dx) \)

- Any no-small-angle mesh is \( \Omega(\int \text{lfs}^{-d}(x)dx) \)
Requirements

- **Quality**: no bad radius/edge output
- **Respect**: the input appears in the output
- **Size-conforming**: spacing in output ~ in input
Outline in four parts

- What makes a good mesh?
- Experimental results
- General conceptual algorithm
- Sub-linear dynamic mesh refinement
Ruppert’s algorithm
Pathology: Line & Circle

n/2 points on circle

n/2 points on line

n^2 Delaunay triangles!
SVR

Provably $O(n \log (L/s) + m)$
Comparative Results: Line & Circle

seconds

#input points (x1000)

#output points (x1000)

Pyramid
SVR

Pyramid
SVR

#input points (x1000)
Comparative Results: 27 Stanford Bunnies

seconds

#output points (x1000)

Pyramid
SVR

#input points (x1000)
How much to shrink?

seconds

#output points (x1000)

shrink to

shrink to
Query Structure

By simplex:
+ easy, traditional
+ cheap query
- expensive update
- non-robust

By Voronoi:
+ easy
+ fast update
+ robust
- bigger query
Speeding up queries
Outline in four parts

• What makes a good mesh?
• Experimental results
• General conceptual algorithm
• Sub-linear dynamic mesh refinement
What makes a good Steiner point?

• Helps achieve *Quality*
• Helps achieve *Respect*
• Doesn’t violate *Size-conforming*
Helps improve Quality
Helps improve *Respect*
Without violating size-conforming
Without violating size-conforming

\[ r > \rho \|uv\| \]

\[ \text{NN}(y) \ll \text{lfs}(y) \]
Without violating size-conforming
Without violating size-conforming

\[ \text{NN}(x) > \rho \|uv\|, \]
\[ \text{NN}(y) > \rho \|uv\|/2, \]
\[ \text{NN}(z) > \rho \|uv\|/4. \]
Related work

Ruppert, SVR: Biggest possible

Ungor: Just-right

\[ r \gg \rho \|vu\| \]

\[ r' = \rho \|vu\| \]

\[ r \approx \sqrt{2} \rho \|vu\| \]
Completing a mesh
A complete mesh is good
Outline in four parts

• What makes a good mesh?
• Experimental results
• General conceptual algorithm
• Sub-linear dynamic mesh refinement
Efficient Queries

How do I know this disc is empty?

Idea credit: Har-Peled, Ungor
Efficient Algorithm

1. Build the quadtree
2. Add every $p$ to $Q$, with key $NN(p)$
3. While $Q$ is not empty
   1. $v \leftarrow \text{Delete-min}(Q)$
   2. Complete($v$)
   3. Add neighbours of $v$ to $Q$

Not strictly in order: bucket by $[l, \rho l)$
Lemma 3.3.3: when processing bucket $b$, empty ball of radius $r \sim b$ intersects $O(1)$ squares.

Proof sketch: To intersect many, must intersect small squares. But small squares are in small buckets $\Rightarrow$ contain points.
Total runtime

1. Build the quadtree \( \mathcal{O}(n \log L/s + m) \)
2. Add every \( p \) to \( Q \)
3. While \( Q \) is not empty
   1. \( v \leftarrow \text{Delete-min}(Q) \) \( \mathcal{O}(1) \) [bucketing]
   2. \( \text{Complete}(v) \) \( \mathcal{O}(1) \) [fast queries]
   3. Add neighbours to \( Q \) \( \mathcal{O}(1) \)
Remeshing

- Change $\Rightarrow$ Reinterpolation $\Rightarrow$ Error

- No change $\Rightarrow$ Why recompute it?
Q: how many points blame v?
Steiner points grow

$$\text{NN}(x) = r > \rho \|uv\|$$
Dynamic stability

Q: how many points blame v?

Yellow: distance $\sim \rho$, NN $\sim \rho$
Purple: distance $\sim \rho^2$, NN $\sim \rho^2$
Cyan: distance $\sim \rho^3$, NN $\sim \rho^3$
Packing Purple Points

\[ \text{# purple points is } \frac{||v_{p_i}||^d}{\text{NN}(p_i)^d} \in O(1) \]

\[ \text{NN}(p_i) \geq k \rho^2 \]

\[ ||v_{p_i}|| \leq k' \rho^2 \]
Dynamic stability

Proves: adding or removing one point from input modifies $O(\lg L/s)$ in output

Yellow: $O(1)$
Purple: $O(1)$
Cyan: $O(1)$

... $O(\log_\rho L/s)$ total
Dynamic Algorithm

1. Build the quadtree
2. Add every $p$ to $Q$
3. While $Q$ is not empty
   1. $v \leftarrow \text{Delete-min}(Q)$
   2. Complete($v$)
   3. Add neighbours to $Q$

On update, **simulate** rerunning from scratch.

Use **Self-adjusting computation**: pay only for the changes.

**Update time**: $O(\log \frac{L}{s})$!
Outline in four parts

• What makes a good mesh?
• Experimental results
• General conceptual algorithm
• Sub-linear dynamic mesh refinement
Handling features

- Details in the document
- Static runtime is $O(n \log L/s + m)$
- Dynamic: if we need $k$ points to respect, update time is $O(k \times \log L/s)$
- Future: prove $O(k + \log L/s)$
  - Probably “just” a proof
Moving meshes

- Dynamic update: add or remove
- Kinetic update: move
- Self-adjusting framework can do kinetic
- I have no proofs ... yet
Claims of the Thesis

• The fastest static meshing code
  • And the first without pathologies
• A framework with lots of explicit freedom for point placement
• The first dynamic meshing algorithm
• A few more minor results