Lecture 14 - Remeshing

Philip Caplan

CSCI 0422 - Geometric Modeling (Spring 2022)
Learning objectives

By the end of this lecture you will be able to:

- implement a remeshing algorithm to achieve a target element size and quality by using splits, collapses, swaps and smoothing,
- practice implementing a local mesh modification operator using a half-edge mesh representation.
Getting started...

Switch Host & Client today!

$ git pull
$ make update
$ cmake .
$ make template_class14_flip
$ cmake .

Compiling and running the exercise:

$ make class14_flip

Compiling and running the solution (after class):

$ make class14_flip_sol
The goals of remeshing.

make a surface mesh look better.

get a more accurate simulation.
Ideally, we would like all vertices to have the same valency (regularity).

\[ \text{valency} = \# \text{ neighboring vertices} \]
Remeshing: starting from scratch.

**example**: Delaunay/Voronoi-based methods

![Remeshing examples](image)

Figure 6.8 from *Polygon Mesh Processing*

these are nice because they minimize the *Centroidal Voronoi Tessellation* "energy":

\[ E(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} \int_{V_i} \rho(x) ||x - p_i||^2 dx \]
Remeshing: starting from an existing mesh.

why throw away a mesh that might have been really hard to generate in the first place?

(source: Pointwise)
Local mesh modification operators can be used to achieve some target criteria.

Split → Collapse → Smoothing → Swap
How to schedule local operators?

remesh( target_edge_length )
  low = (4/5) * target_edge_length
  hi = (4/3) * target_edge_length
  for i = 0 to nb_iter do
    split_long_edges(high)
    collapse_short_edges(low, high)
    equalize_valences()
    tangential_relaxation()
    project_to_surface()
Splitting long edges.

```plaintext
split_long_edges(high)
    while exists edge e with length(e) > high do
        split e at midpoint(e)
```

Split
Collapsing short edges.

collapse_short_edges(low, high)

while exists edge e with length(e) < low do

    // we will collapse p onto q
    e = (p, q)

    // retrieve the one-ring of p
    ring = one_ring(p)

    collapse_ok = true
    for v in ring do // loop through the one-ring of p
        if length(p, v) > high
            collapse_ok = false

    if collapse_ok
        collapse p onto q along e
Equalizing valences.

```c
equalize_valences()
    for each edge e do
        let a, b, c, d be the vertices of the two triangles adjacent to e

        deviation_pre = abs( valence(a) - target_val(a) )
                        + abs( valence(b) - target_val(b) )
                        + abs( valence(c) - target_val(c) )
                        + abs( valence(d) - target_val(d) )

        // attempt the flip
        flip(e)

        deviation_post = abs( valence(a) - target_val(a) )
                         + abs( valence(b) - target_val(b) )
                         + abs( valence(c) - target_val(c) )
                         + abs( valence(d) - target_val(d) )

        if deviation_pre <= deviation_post
            flip(e) // flip back since the deviation did not improve
```
Tangential relaxation: a.k.a. "smoothing"

For each vertex:

\[
q = \frac{1}{\mathcal{N}(p)} \sum_{p_j \in \mathcal{N}(p)} p_j
\]

where \(\mathcal{N}(p)\) is the one-ring of \(p\).

The new position \(p'\) can be computed by projecting \(q\) onto the tangent plane at \(p\):

\[
p' = q + (n \cdot (p - q))n.
\]

where \(n\) is the normal vector at \(p\) (you can compute the average of the face normals surrounding this vertex).
Alternative remeshing strategies: isotropic sizing field.

- prescribe desired sizing field $h(x)$ throughout domain,
- try to have all normalized edge lengths within target range:

$$\frac{4}{5} \leq \frac{\ell}{h(x)} \leq \frac{4}{3}$$
Alternative remeshing strategies: anisotropic sizing field.

Goals for a mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$ of $\Omega \subset \mathbb{R}^n$

- **Edge lengths** are 1:
  \[ \ell_m(e) = 1, \quad \forall e \in \mathcal{E}(\mathcal{T}) \]

- **Quality** is that of equilateral simplex:
  \[ q_m(\kappa) = \frac{1}{q_\Delta} \frac{v_m^{2/n}(\kappa)}{\sum_{e \in \mathcal{E}(\kappa)} \ell_m^2(e)} = 1, \quad \forall \kappa \in \mathcal{T} \]

- **# simplices** matches metric field complexity:
  \[ n_s v_\Delta = \int_{\mathcal{M}} \sqrt{\det m} \, dx \]
Alternative remeshing strategies: anisotropic sizing field.

**Goals for a mesh** \( \mathcal{M} = (\mathcal{V}, \mathcal{T}) \) of \( \Omega \subset \mathbb{R}^n \)

- **Edge lengths** are close to 1:
  \[
  \frac{1}{\sqrt{2}} \leq \ell_m(e) \leq \sqrt{2}, \quad \forall e \in \mathcal{E}(\mathcal{T})
  \]

- **Quality** is close to that of an equilateral simplex:
  \[
  q_m(\kappa) = \frac{1}{q_\Delta} \frac{\nu_m^{2/n}(\kappa)}{\sum_{e \in \mathcal{E}(\kappa)} \ell_m^2(e)} \in [0.8, 1], \quad \forall \kappa \in \mathcal{T}
  \]

- **# simplices** matches metric field complexity:
  \[
  n_s v_\Delta \approx \int_\mathcal{M} \sqrt{\det m} \, dx
  \]
Making it work in higher dimensions.

\[ T^{k+1} = T^k \setminus C(f) \cup B(p, \partial C^k) \]

- Cavity
- Insertion

Diagram showing a complex network with labeled nodes and edges.
Making it work in higher dimensions.

\[ \mathcal{T}^{k+1} = \mathcal{T}^k \setminus \{ C(f) \} \cup \{ B(p, \partial C^k) \} \]

cavity
insertion

\[ \{ v, p, q \} \]

\[ \{ u, t, r, s \} \]
Making it work in higher dimensions.

\[ \mathcal{T}^{k+1} = \mathcal{T}^k \setminus \{C(f) \cup \mathcal{B}(p, \partial C^k)\} \]

- cavity
- insertion
Making it work in higher dimensions.

\[ \mathcal{T}^{k+1} = \mathcal{T}^k \setminus C(f) \cup \{B(p, \partial C^k)\} \]

- cavity
- insertion
avro in 3d.

Benchmarks of the Unstructured Grid Adaptation Working Group (UGAWG)

Cube Linear

Cube-Cylinder Linear

Cube-Cylinder Polar 1

Cube-Cylinder Polar 2

feflo.a  EPIC-ICSM  Omega_h  avro
Expect 39k tetrahedra for the Cube Linear case.

<table>
<thead>
<tr>
<th></th>
<th>%\ell_{\text{unit}}</th>
<th>%q_{\text{unit}}</th>
<th># simplices</th>
</tr>
</thead>
<tbody>
<tr>
<td>avro</td>
<td>99.10 %</td>
<td>92.15 %</td>
<td>38.30k</td>
</tr>
<tr>
<td>feflo.a</td>
<td>98.28 %</td>
<td>74.28 %</td>
<td>45.16k</td>
</tr>
<tr>
<td>EPIC-ICSM</td>
<td>93.04 %</td>
<td>59.52 %</td>
<td>47.55k</td>
</tr>
<tr>
<td>Omega_h</td>
<td>93.00 %</td>
<td>47.15 %</td>
<td>51.67k</td>
</tr>
</tbody>
</table>
Expect 36.4k tetrahedra for the Cube-Cylinder Polar 2 case.

% edges

<table>
<thead>
<tr>
<th>% edges</th>
<th>avro</th>
<th>feflo.a</th>
<th>EPIC-ICSM</th>
<th>Omega_h</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10^6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>10^5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10^4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>10^3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10^2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>10^1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% simplices

<table>
<thead>
<tr>
<th>% simplices</th>
<th>avro</th>
<th>feflo.a</th>
<th>EPIC-ICSM</th>
<th>Omega_h</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10^6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>10^5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>10^4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>10^3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>10^2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>10^1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>% elongation unit</th>
<th>% quality unit</th>
<th># simplices</th>
</tr>
</thead>
<tbody>
<tr>
<td>avro</td>
<td>95.76 %</td>
<td>34.20k</td>
</tr>
<tr>
<td>feflo.a</td>
<td>93.83 %</td>
<td>53.12k</td>
</tr>
<tr>
<td>EPIC-ICSM</td>
<td>91.77 %</td>
<td>44.28k</td>
</tr>
<tr>
<td>Omega_h</td>
<td>92.19 %</td>
<td>49.15k</td>
</tr>
</tbody>
</table>
Exercise: implementing a flip (swap) operator.

for the edge between vertices 5 & 10.

before

after
Your TODO list . . .

- add tests for HalfEdgeMesh::flip function (in halfedges.h, cpp),
- work on Project 3.