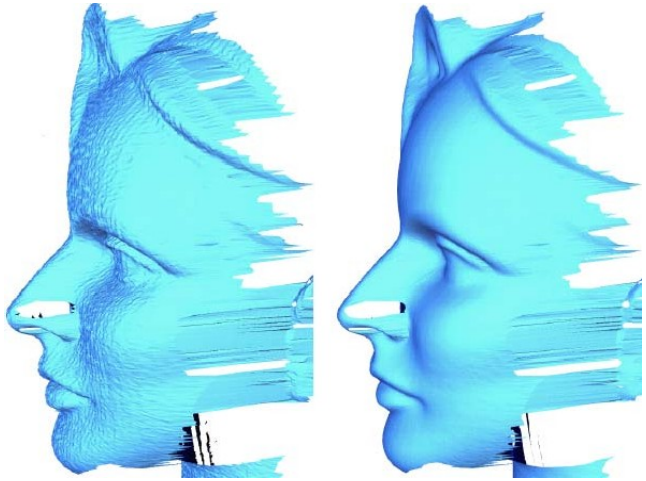
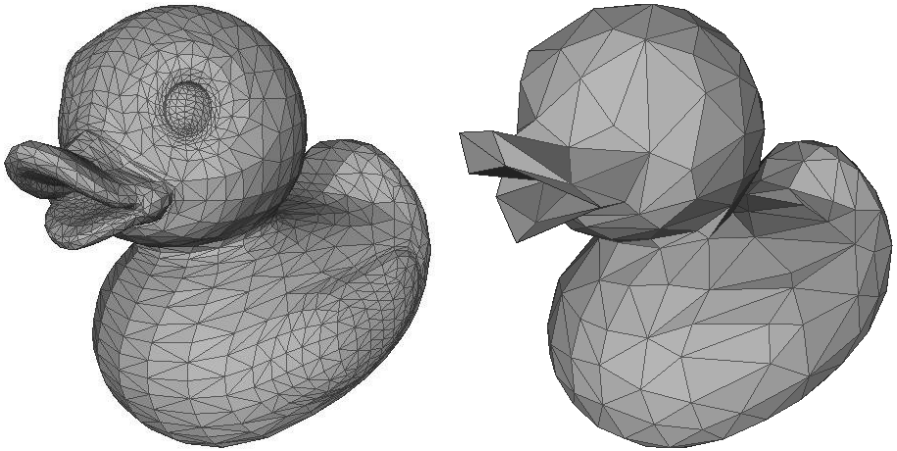
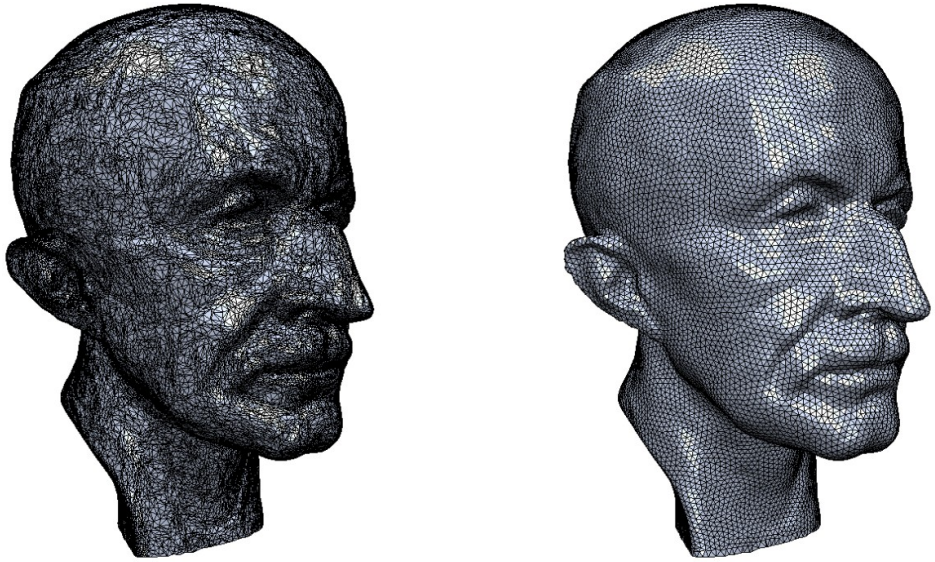
Mesh Filtering, Remeshing, and Simplification





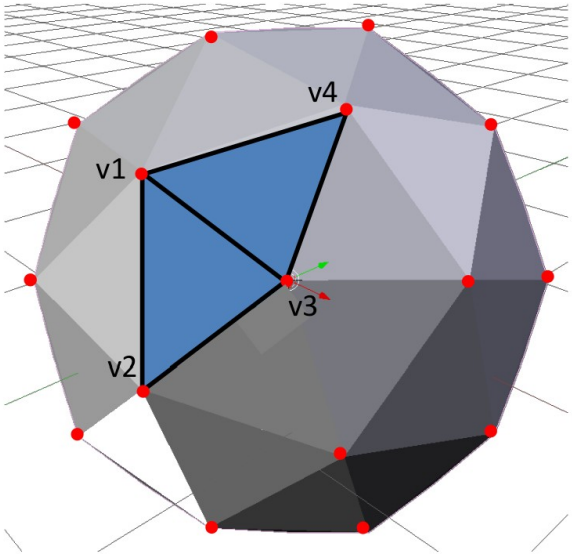


Today

- Consider (*triangle*) meshes
- Basic operations using notions that you’ve already seen
- Three important applications :
  - Filtering (changing the vertex positions to denoise a mesh)
  - Remeshing (resample in a « smart » way)
  - Simplification (downsample in a « smart » way)
- We’ll see the last one (upsample in a « smart » way) next.

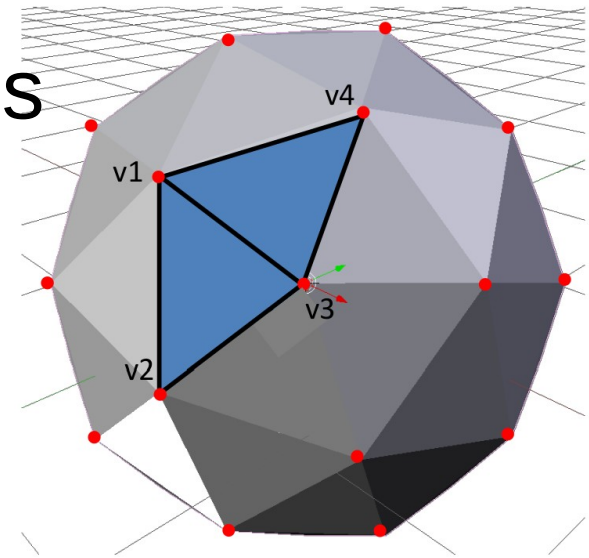
Triangle meshes : Reminder

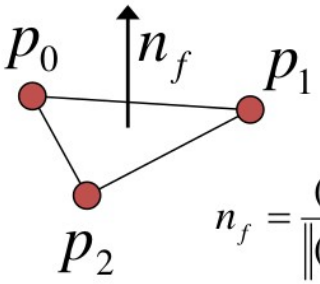
- An array of vertex positions
  xyzxyzxyzxyzxyz…
  size 3 * nb of vertices
- An array of triangle indices
  ijkijkijkijk…
  size 3 * nb of triangles
  (ex in the picture : 1,2,3,1,3,4,...)



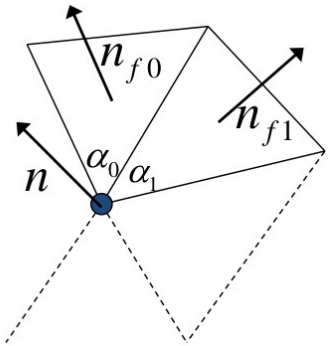
Normals : Reminder

- Compute per-triangle normals
- Average normals of neighboring triangles to obtain vertex normals



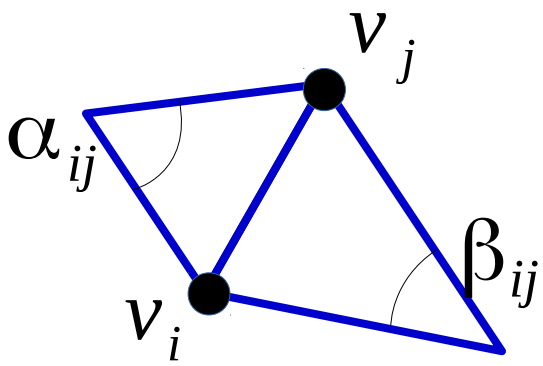


$$n_f = \frac{(p_1 - p_0) \wedge (p_2 - p_0)}{\|(p_1 - p_0) \wedge (p_2 - p_0)\|}$$



Laplacian operator : Reminder

Takes scalars defined on vertices, computes the Laplacian at each vertex



$$\nabla^2 f(v_i) = \frac{1}{|v_i|} \sum_{e_{ij}} \frac{\cot(\alpha_{ij}) + \cot(\beta_{ij})}{2} \left(f(v_j) - f(v_i) \right)$$

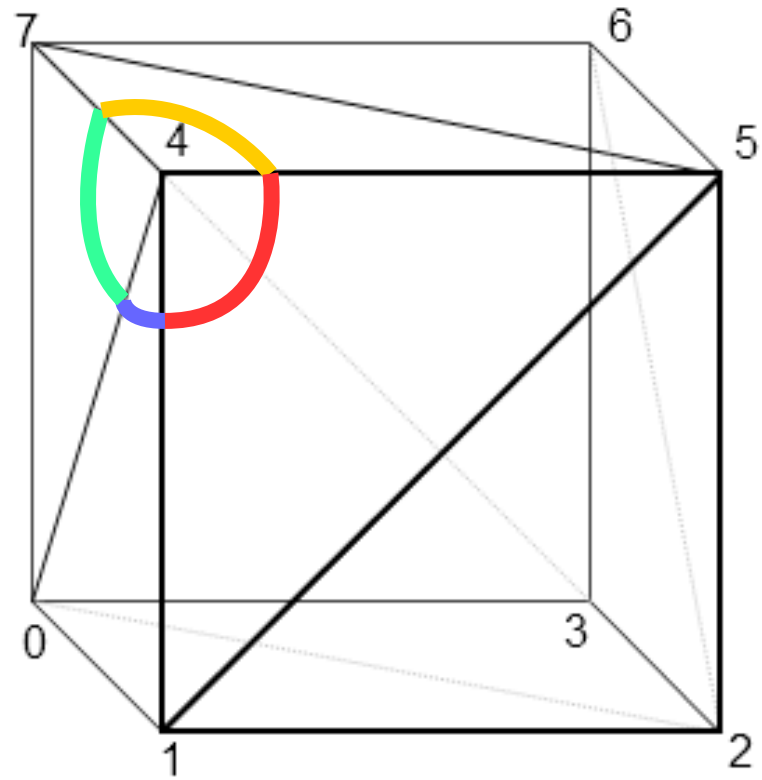
Note : Once again, you may see the version without 1/|v_i| here and there. Once again, the version without is the **integrated** operator (integrated over the area around vertex v_i), and the version with is the point-wise operator.

Curvature : Reminder

Gaussian curvature :

$$K_i = \underbrace{\left(2\pi - \sum_{t \in T_1(i)} \gamma_i(t)\right)}_{\text{Angle deficit around vertex i}} / A_i$$

Area around vertex i



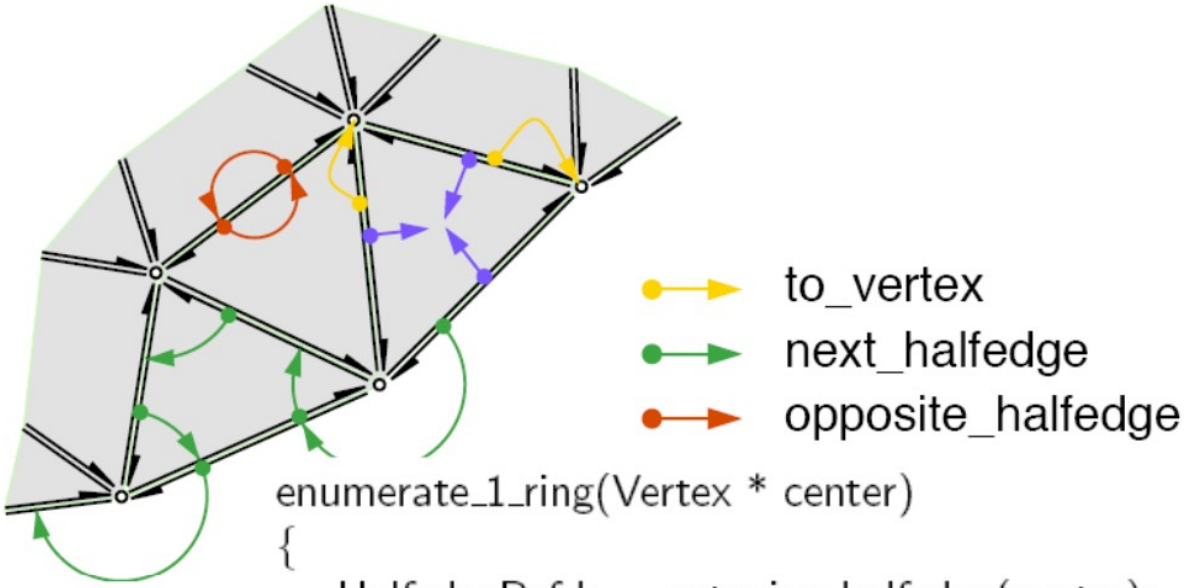
Mean curvature :

$$H_i = \frac{-1}{4 A_i} \sum_{i \in V_1(i)} \left(\cot(\alpha_{ij}) + \cot(\beta_{ij})\right)(v_j - v_i) . N_i$$

[Meyer et al. 2002] Discrete Differential-Geometry Operators for Triangulated 2-Manifolds

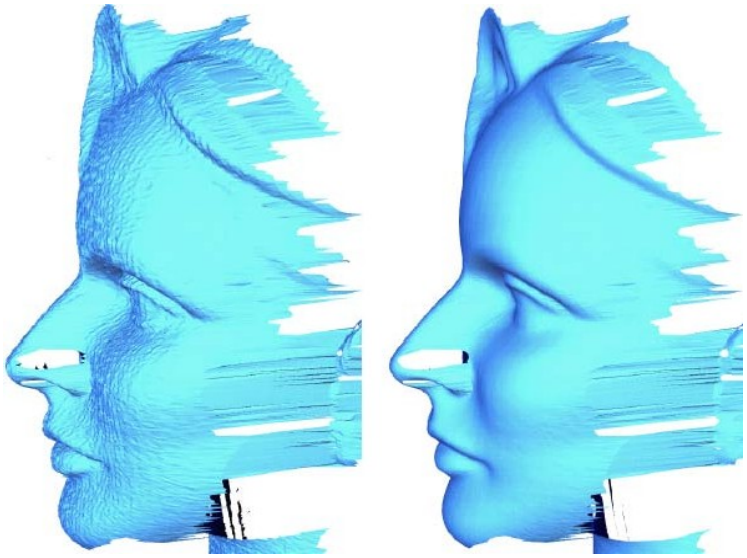
Half-edge data structure

```
struct Halfedge {
   HalfedgeRef      next_halfedge;
   HalfedgeRef      opposite_halfedge;
   FaceRef          face;
   VertexRef        to_vertex;
};

struct Face {
   HalfedgeRef      halfedge;
};

struct Vertex {
   HalfedgeRef      outgoing_halfedge;
};
```

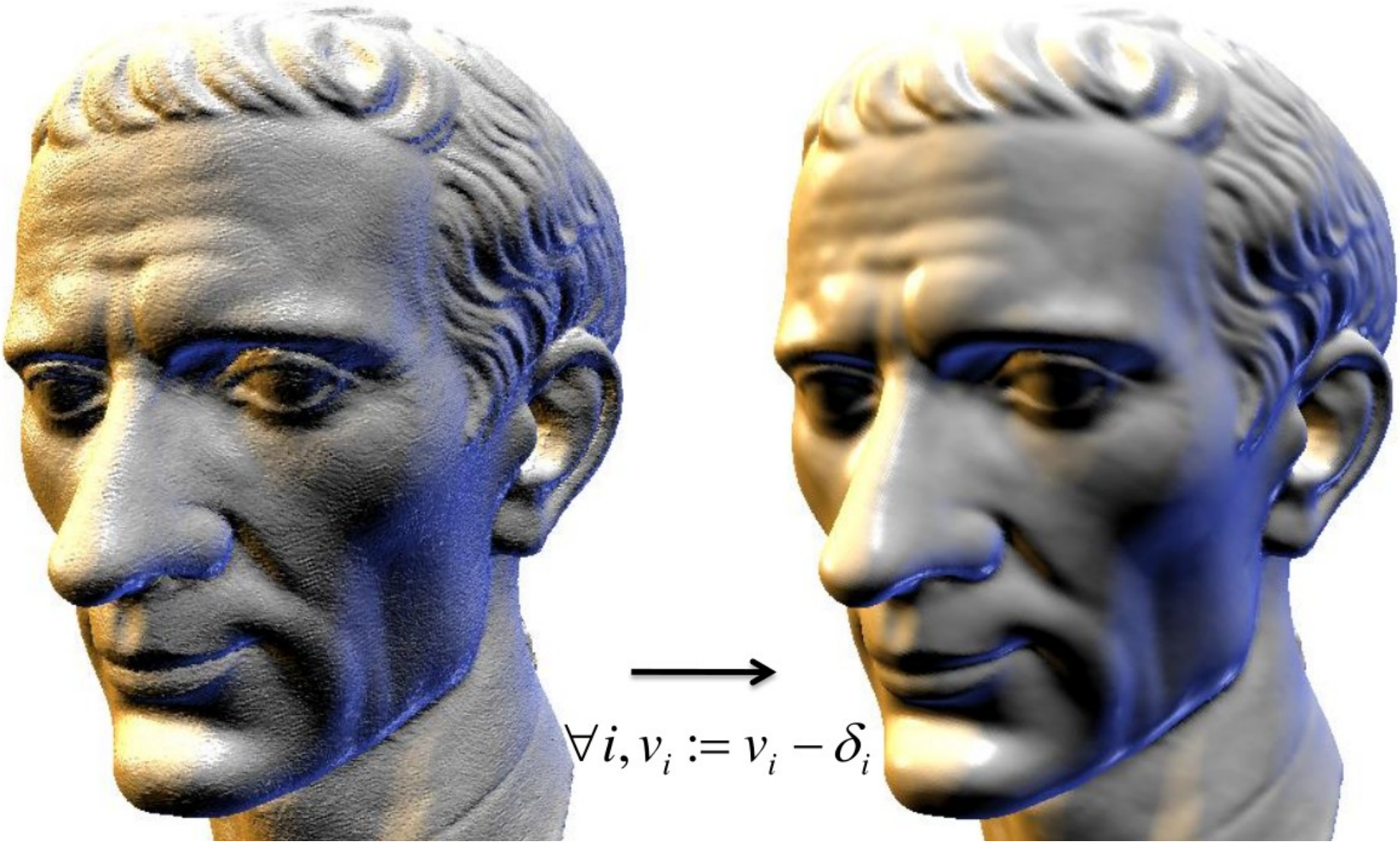



```
enumerate_1_ring(Vertex * center)
{
    HalfedgeRef h = outgoing_halfedge(center);
    HalfedgeRef hstop = h;
    do {
        VertexRef v = to_vertex(h);
        // do something with v
        h = next_halfedge(opposite_halfedge(h));
    } while ( h != hstop );
}
```

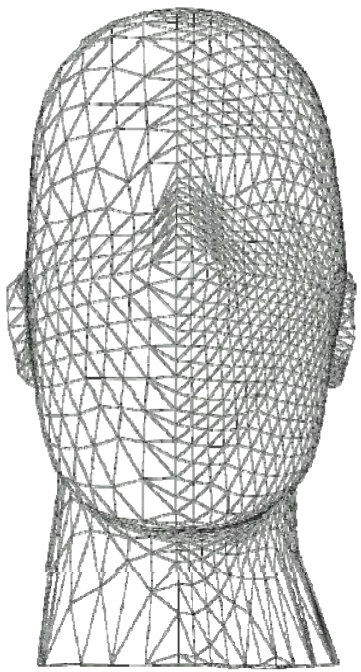
Mesh Filtering



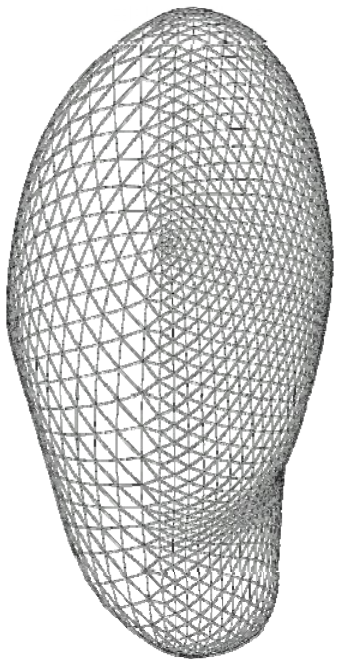
Laplacian filtering



Laplacian filtering

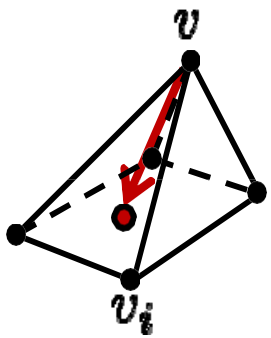


original

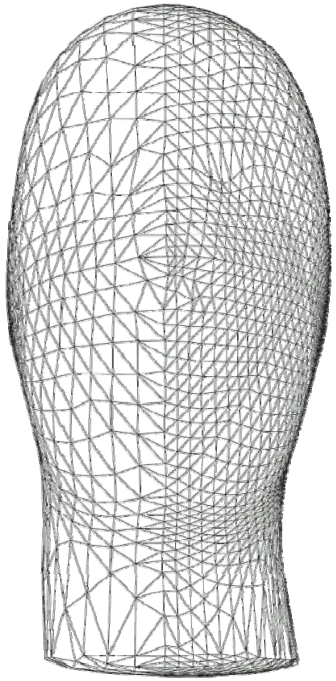


Uniform weights

$$w_{ij} = 1$$

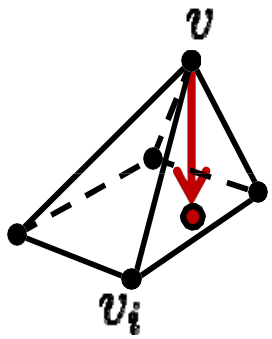


normal
and
tangential
movement

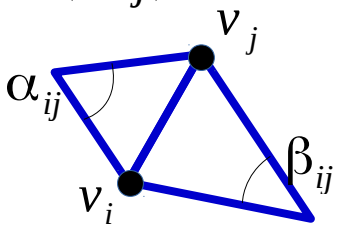


Cotangent weights

$$w_{ij} = \cot(\alpha_{ij}) + \cot(\beta_{ij})$$

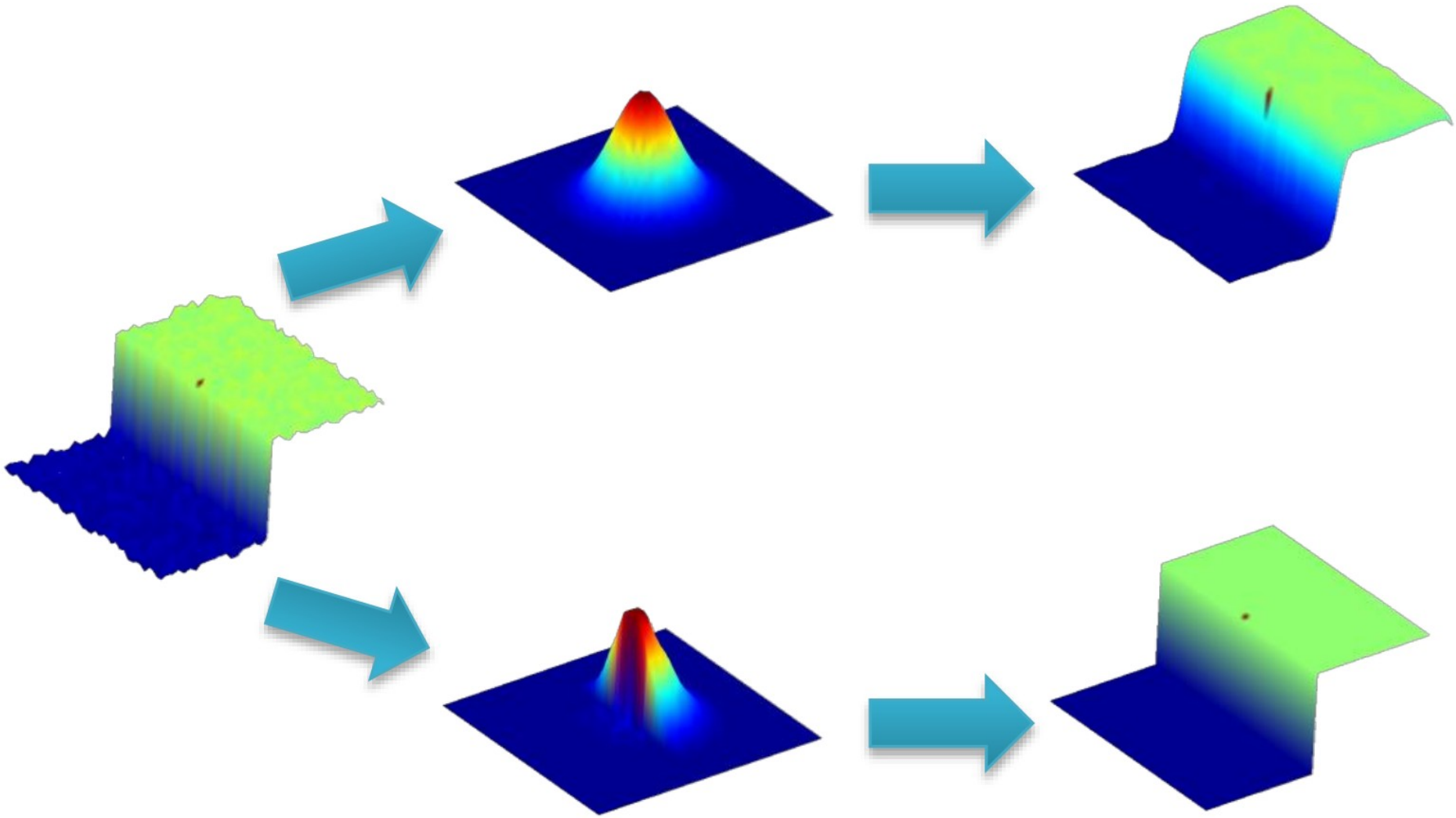


normal
movement

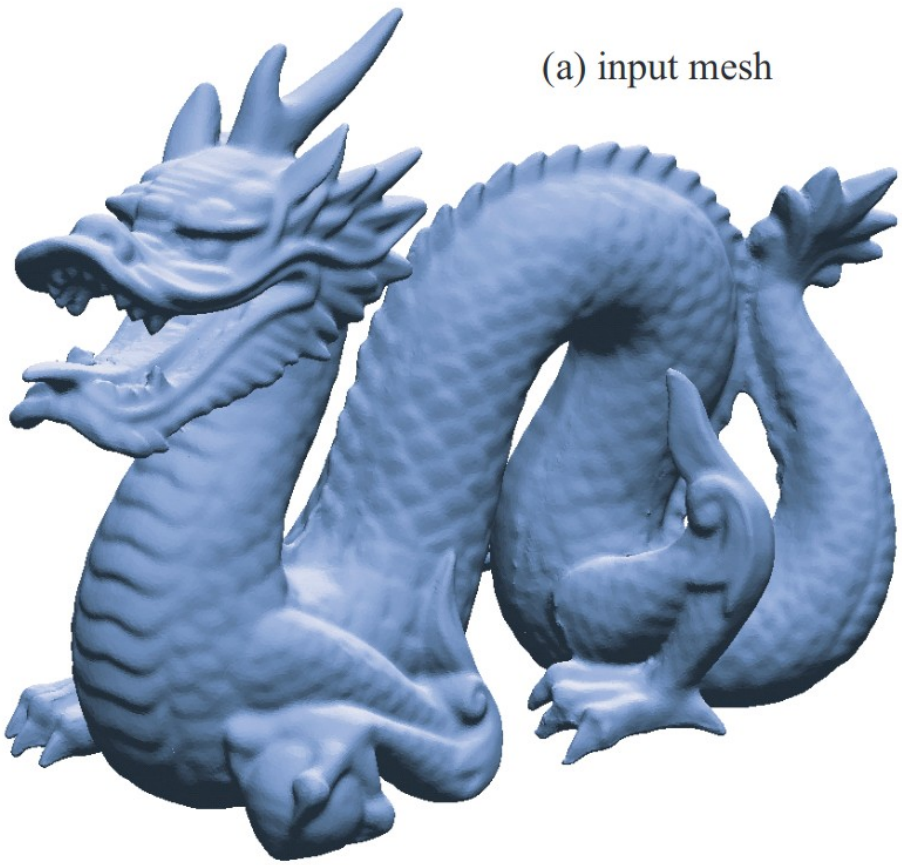


$$v_i' = \sum_{j \in V_1(i)} w_{ij} v_j \Big/ \sum_{j \in V_1(i)} w_{ij}$$

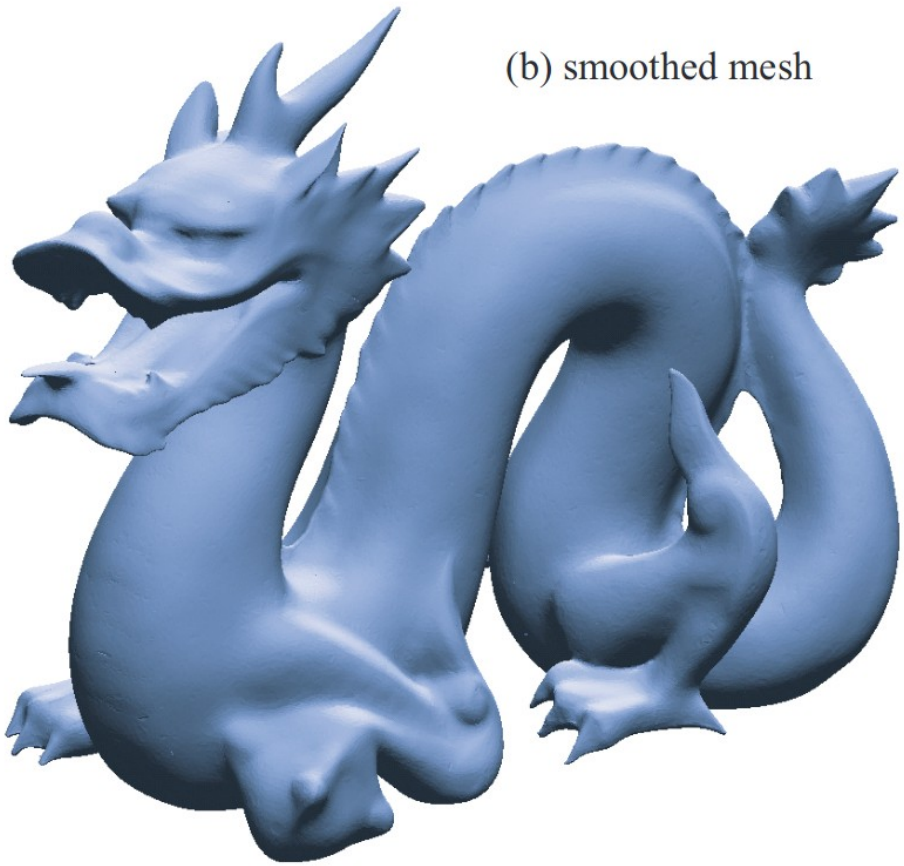
Bilateral filtering



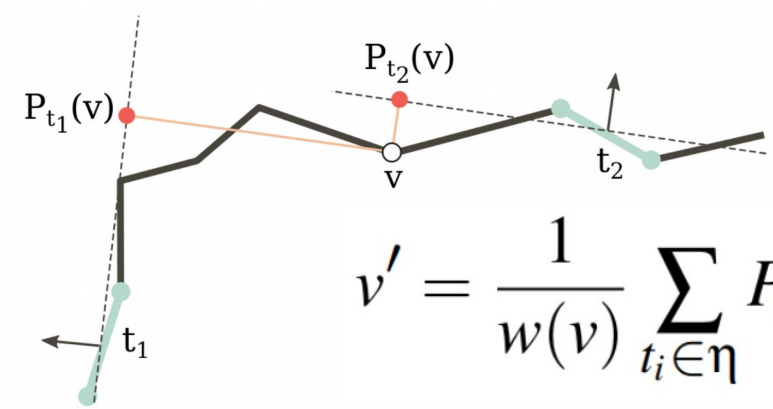
Bilateral filtering



(a) input mesh

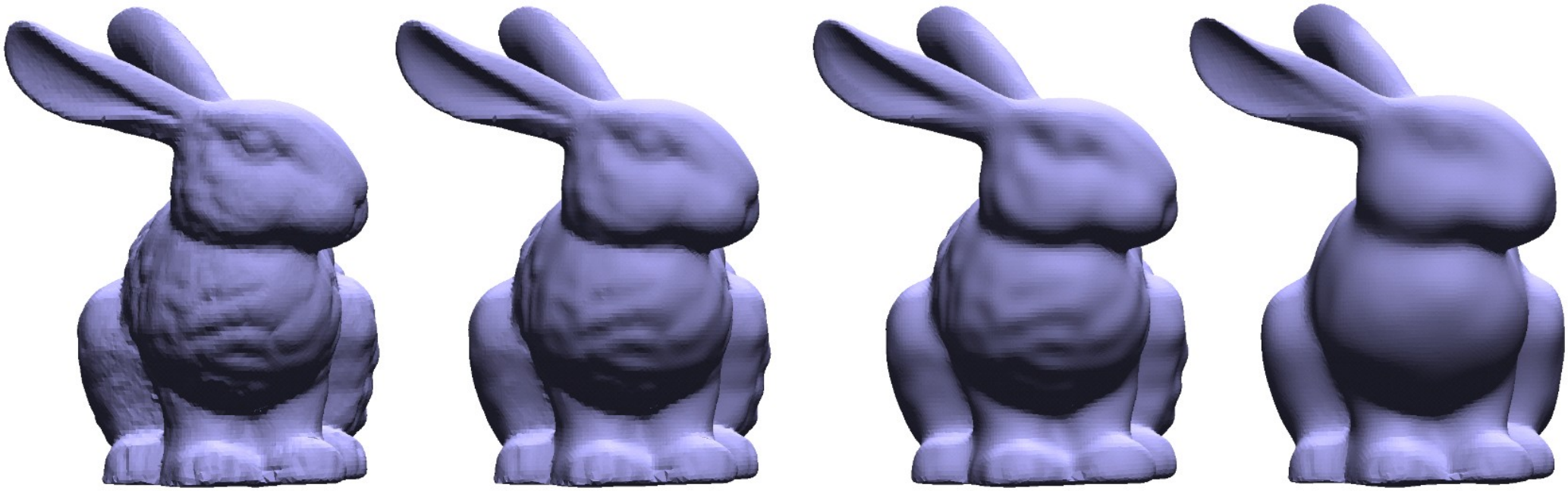


(b) smoothed mesh



$$v' = \frac{1}{w(v)} \sum_{t_i \in \eta} P_{t_i}(v) a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|)$$

Bilateral filtering



original

noise removal
narrow spatial, narrow influence
($\sigma_f = 2$, $\sigma_g = 0.2$)

smooth small features
narrow spatial, wide influence
($\sigma_f = 2$, $\sigma_g = 4$)

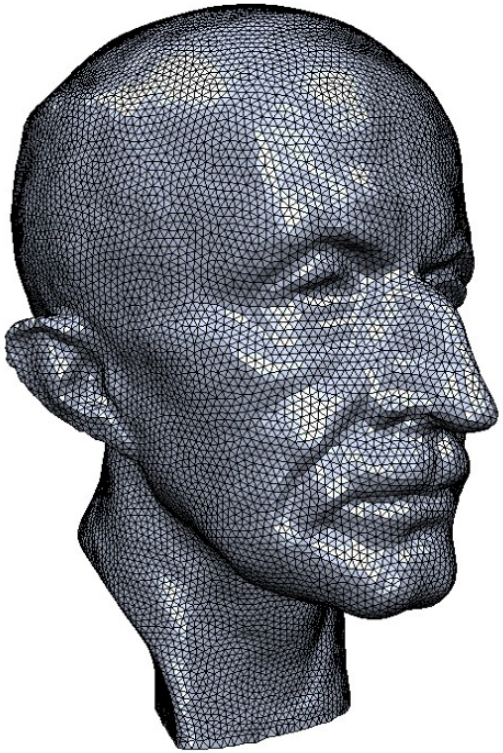
smooth large features
wide spatial, wide influence
($\sigma_f = 4$, $\sigma_g = 4$)

Laplacian vs Bilateral

- Laplacian : one-ring vertex neighborhood → fast
- Bilateral : large neighborhoods → slow
- Laplacian : derived from « sound maths »
  - Mean curvature flow
- Bilateral : preservation of rich geometric frequencies
  - Ad-hoc inspiration from image filtering

?

Remeshing

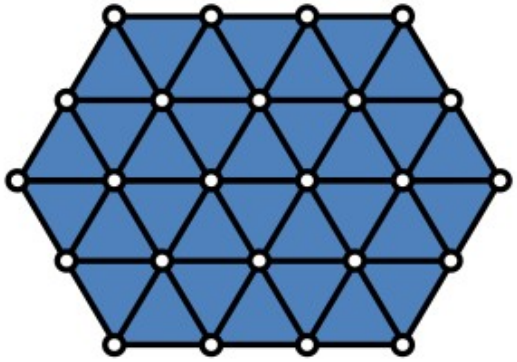


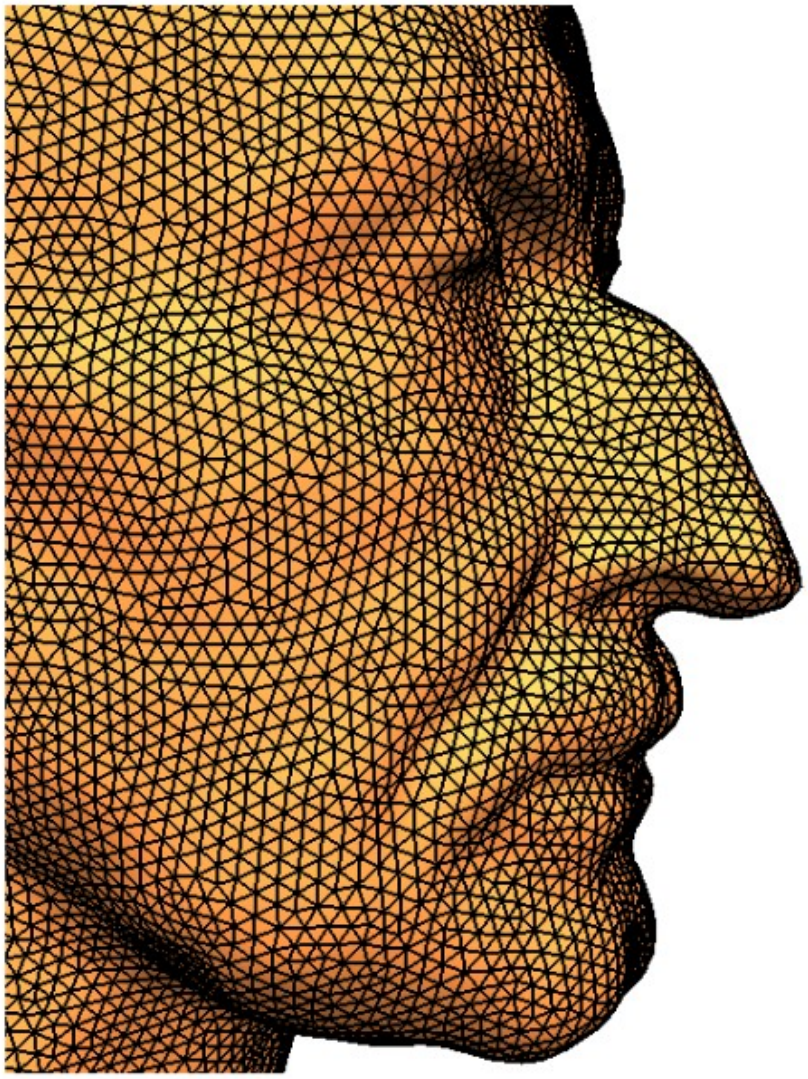
Good meshes

Equal edge lengths

Equilateral triangles

Valence close to 6





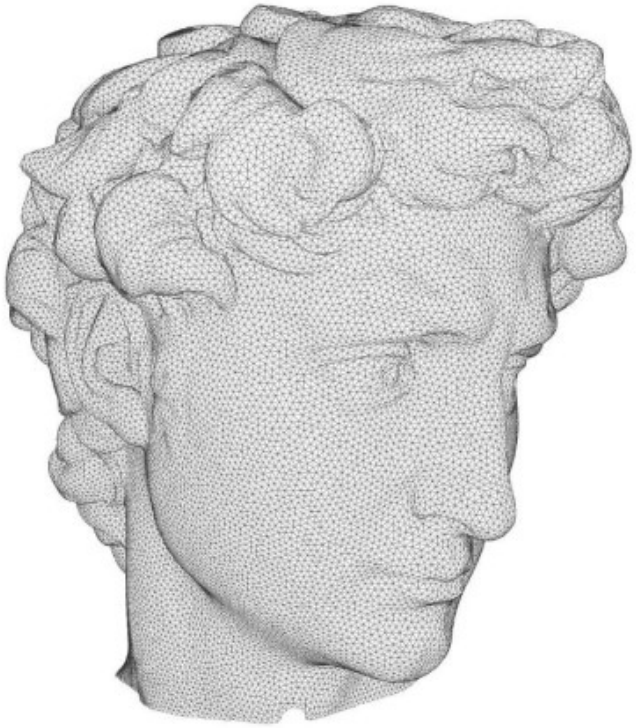
Good meshes

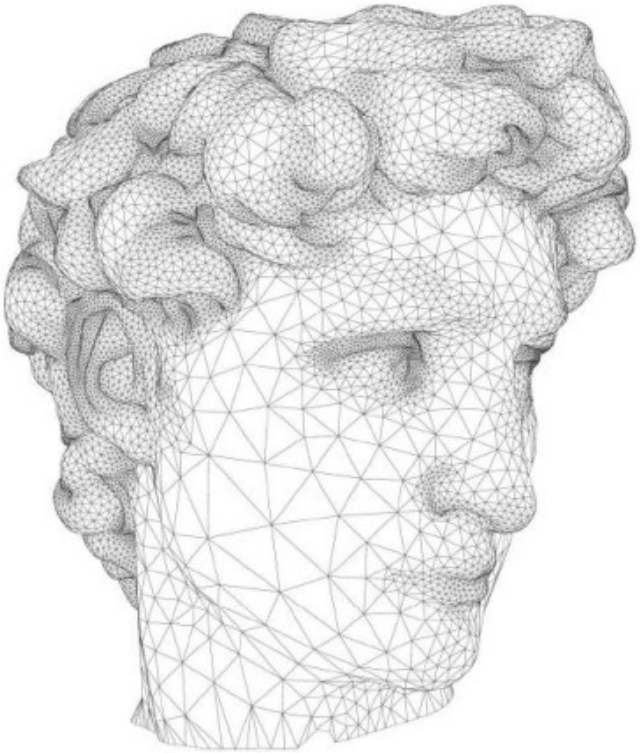
Equal edge lengths

Equilateral triangles

Valence close to 6

Uniform vs. adaptive sampling





Good meshes

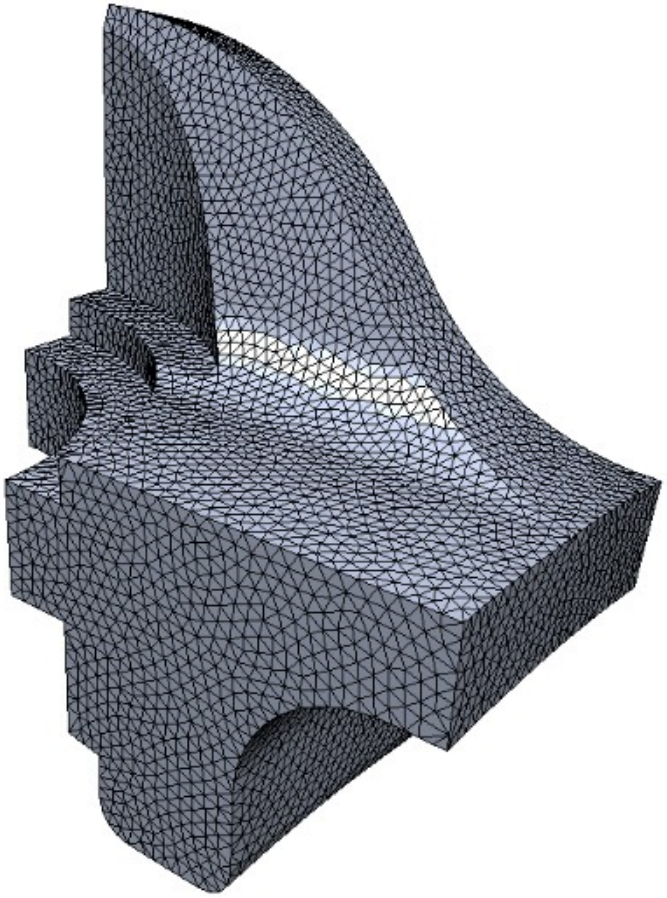
Equal edge lengths

Equilateral triangles

Valence close to 6

Uniform vs. adaptive sampling

Feature preservation



Good meshes

Equal edge lengths

Equilateral triangles

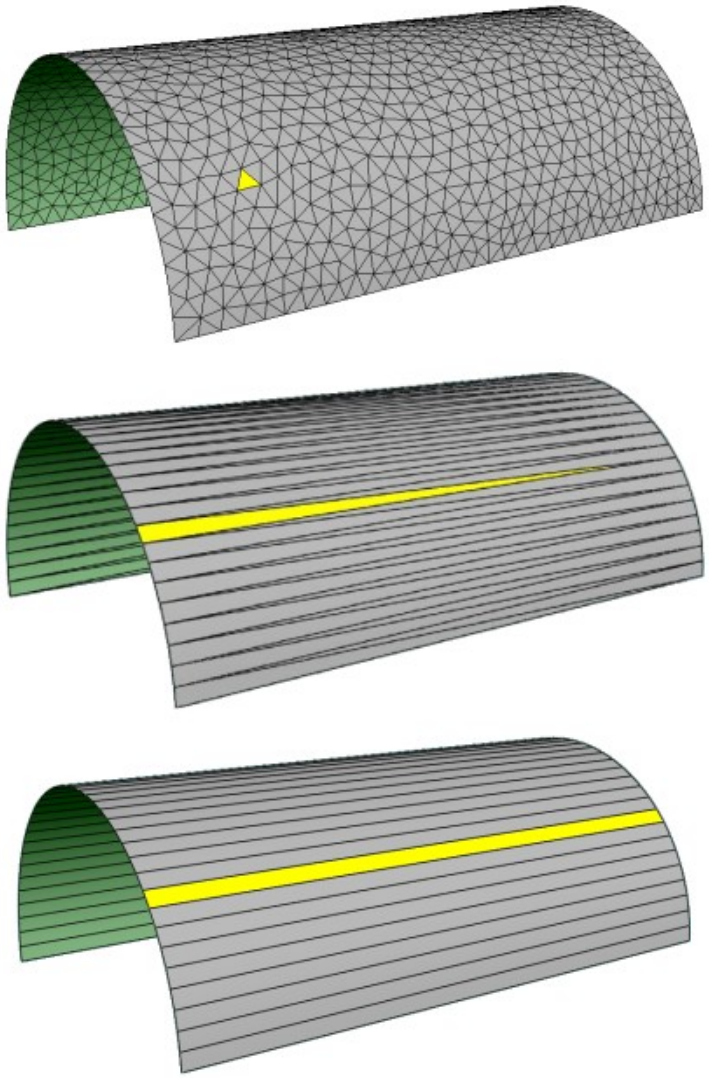
Valence close to 6

Uniform vs. adaptive sampling

Feature preservation

Alignment to curvature lines

Isotropic vs. anisotropic



Good meshes

Equal edge lengths

Equilateral triangles

Valence close to 6

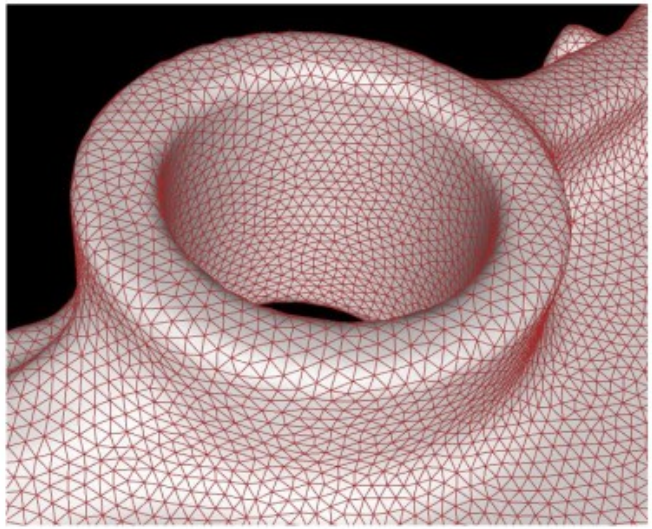
Uniform vs. adaptive sampling

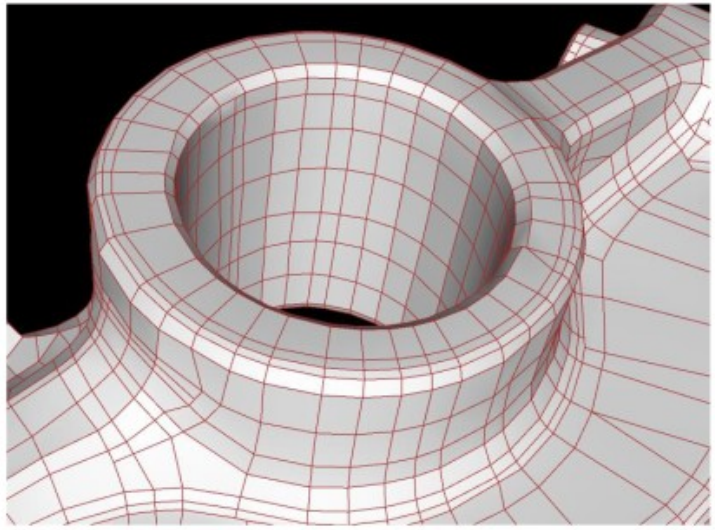
Feature preservation

Alignment to curvature lines

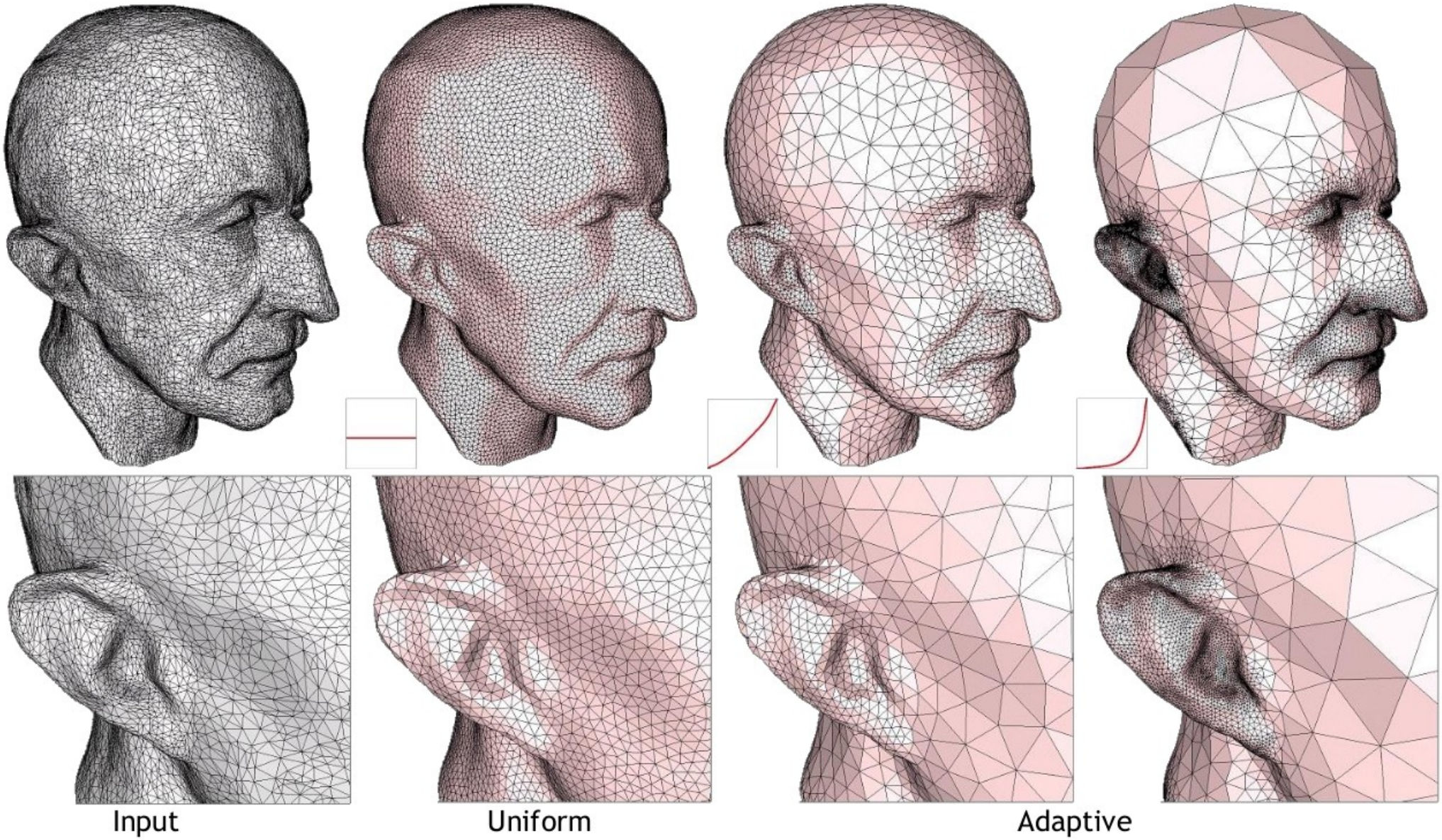
Isotropic vs. anisotropic

Triangles vs. quads

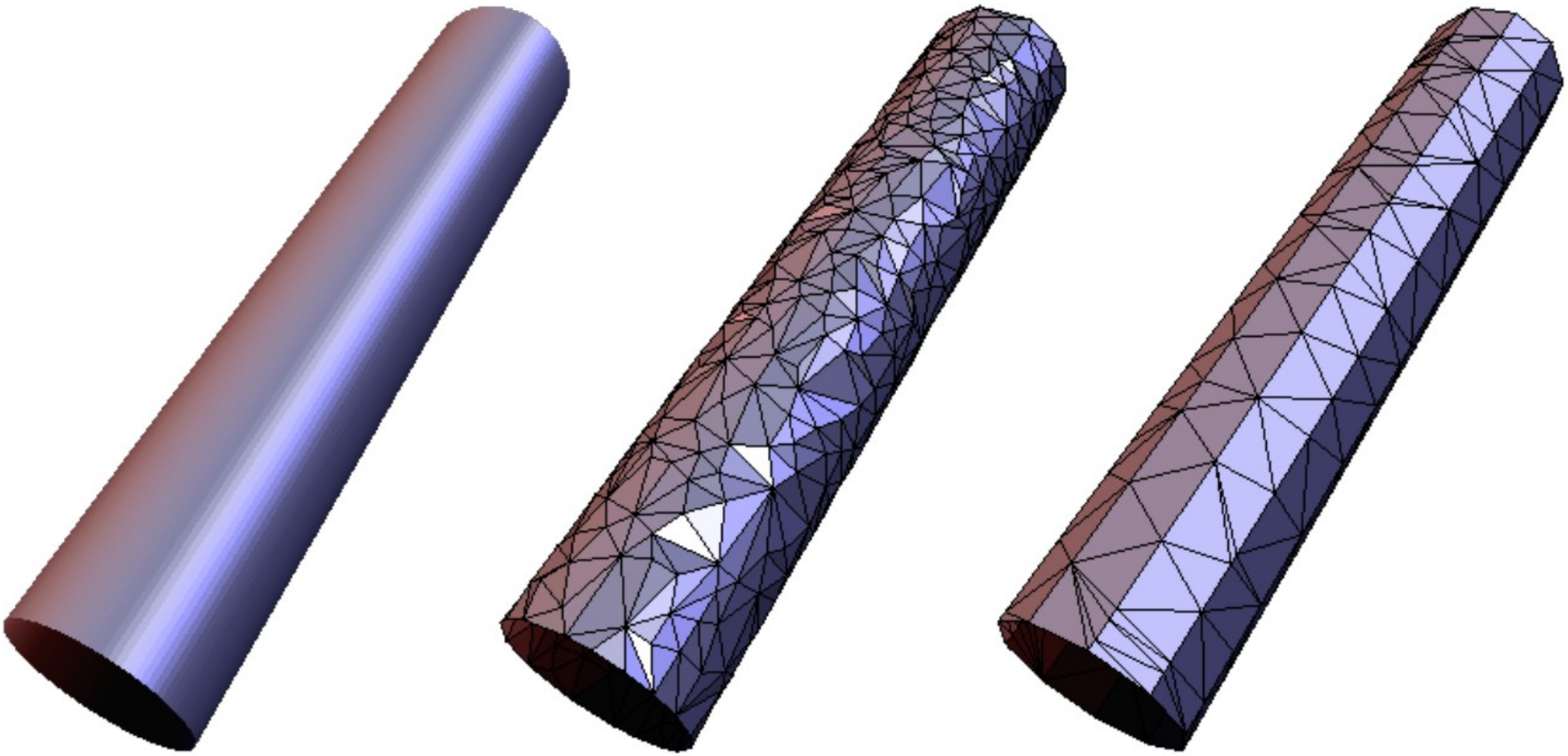




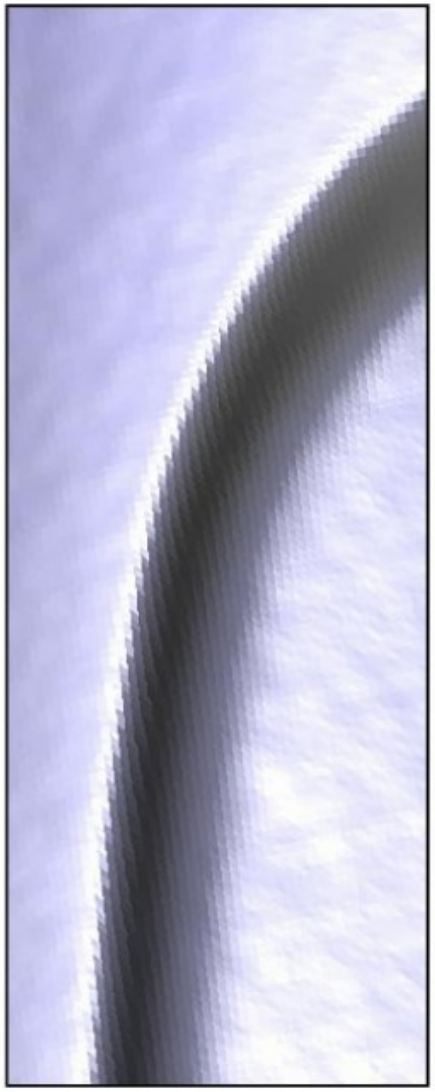
Element distribution



Element orientation



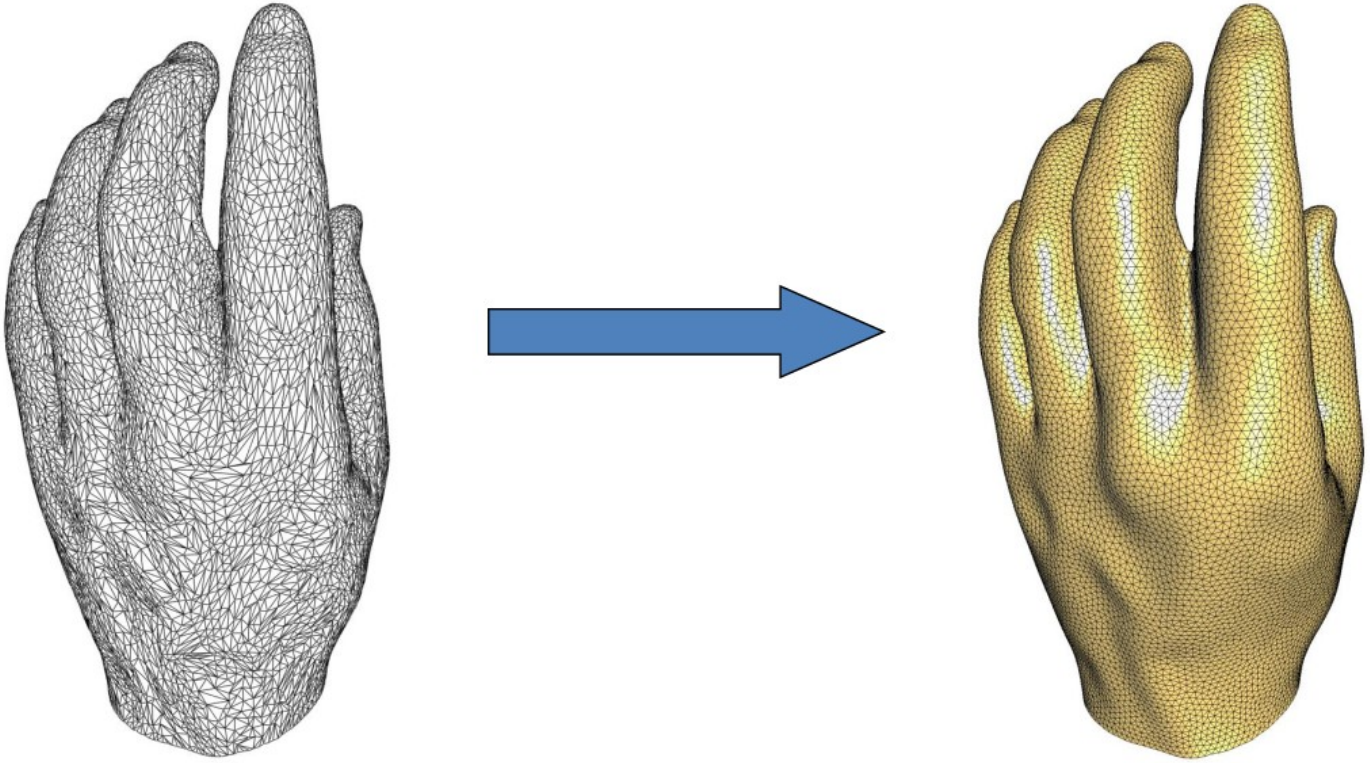
Element orientation



Applications

Well-shaped elements

for processing & simulation (numerical stability & efficiency)



A « simple » remeshing algorithm (Botsch 2004)

Avoid global parameterization

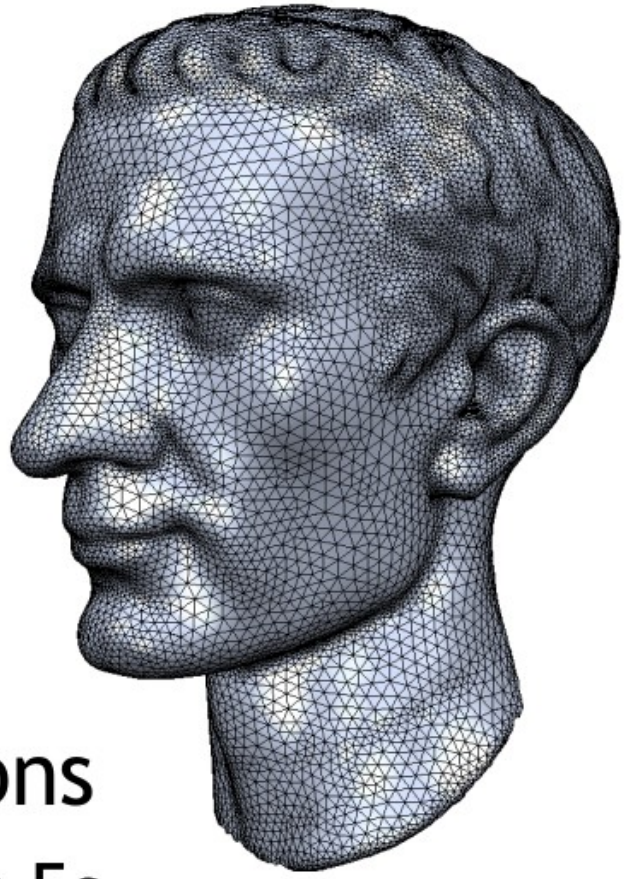
- Numerically very sensitive
- Topological restrictions

Avoid local parameterizations

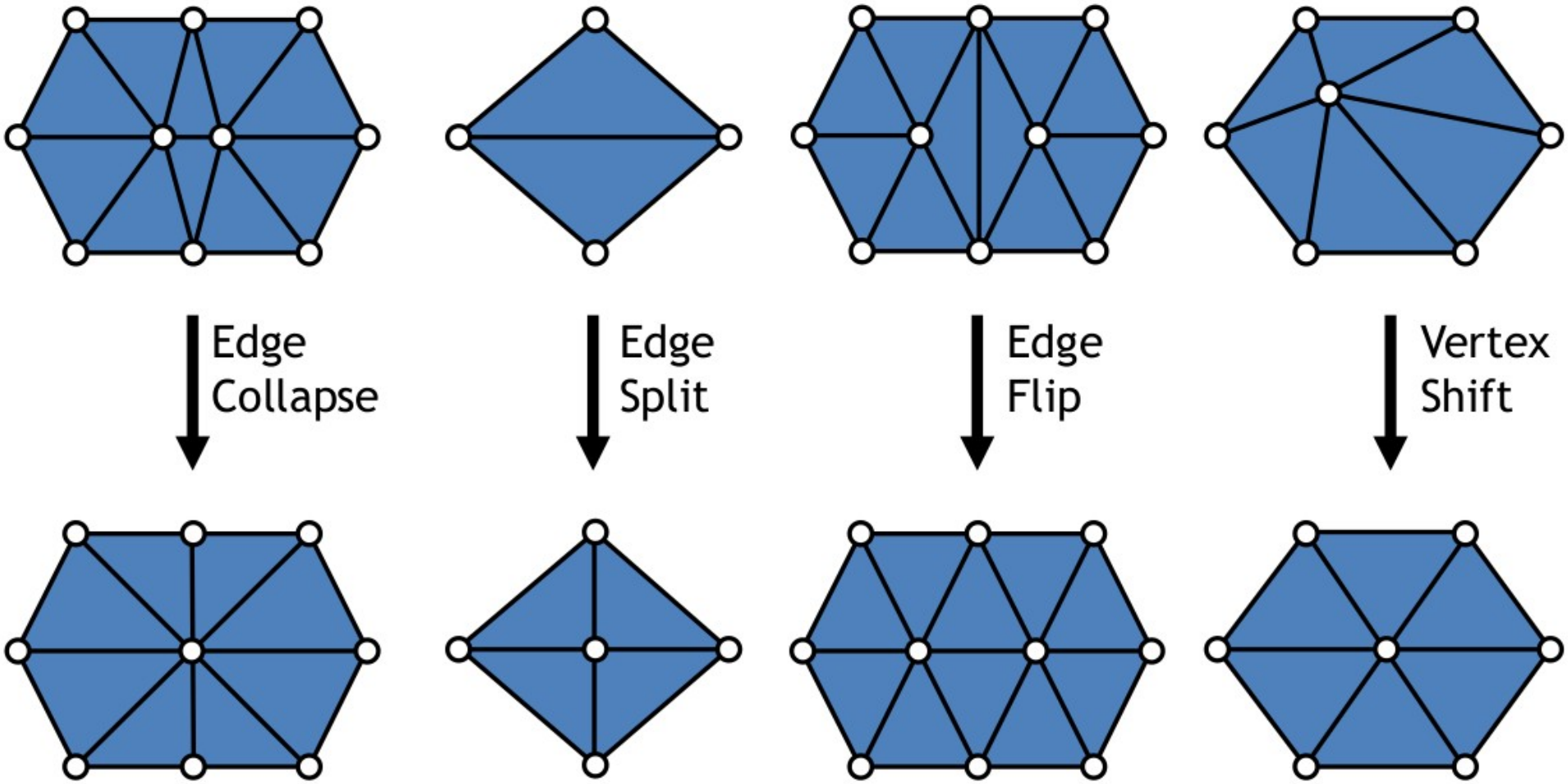
- Expensive computations

Use local operators & projections

- Resampling of 100k triangles in < 5s



Local operations



Algorithm

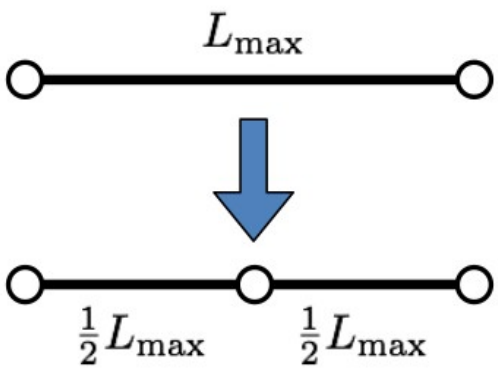
Specify target edge length L

Compute edge length range $[L_{min}, L_{max}]$

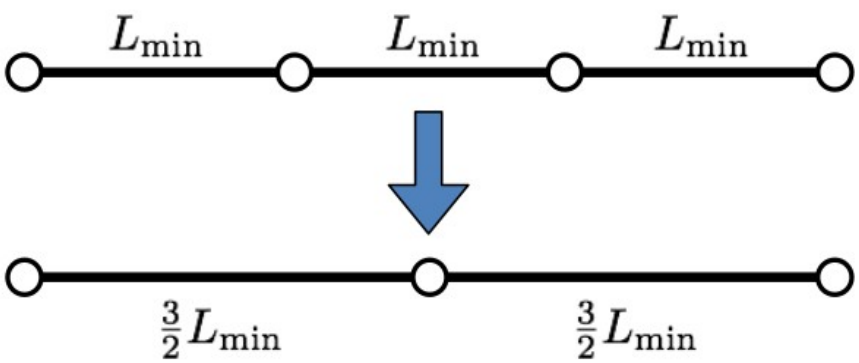
Iterate:

1. Split edges longer than L_{max}
2. Collapse edges shorter than L_{min}
3. Flip edges to get closer to valence 6
4. Vertex shift by tangential relaxation
5. Project vertices onto reference mesh

Split long edges / Collapse short edges



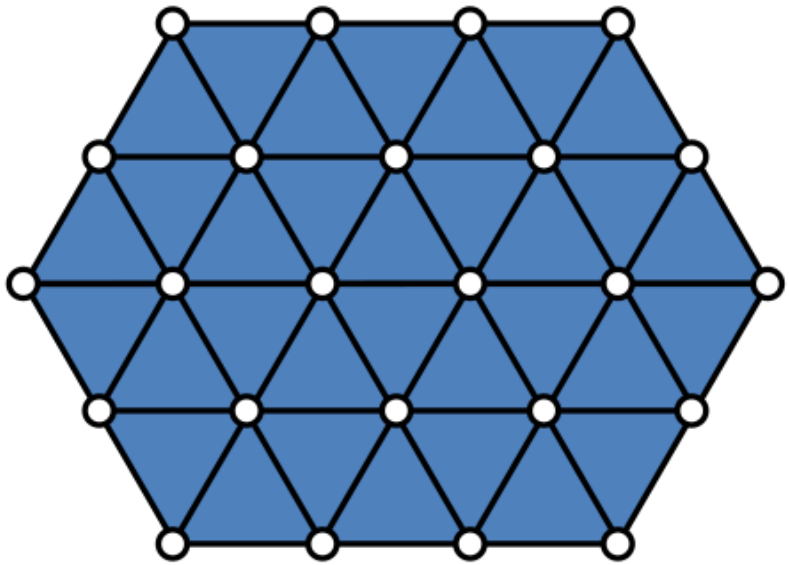
$$
\begin{aligned}
|L_{\max} - L| &= \left|\frac{1}{2}L_{\max} - L\right| \\
\Rightarrow L_{\max} &= \frac{4}{3}L
\end{aligned}
$$



$$
\begin{aligned}
|L_{\min} - L| &= \left|\frac{3}{2}L_{\min} - L\right| \\
\Rightarrow L_{\min} &= \frac{4}{5}L
\end{aligned}
$$

Flip to optimize for valence

- Improve valences
  - Avg. valence is 6 (Euler)
  - Reduce variation
- Optimal valence is
  - 6 for interior vertices
  - 4 for boundary vertices



Flip to optimize for valence

Improve valences

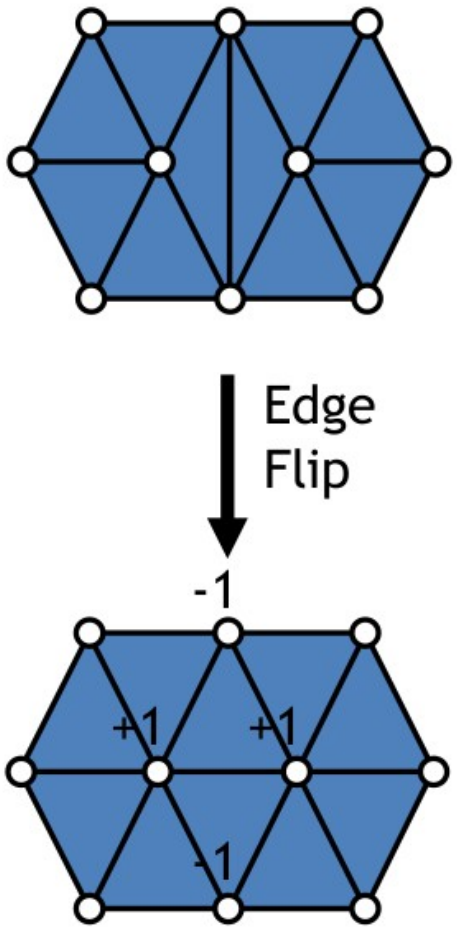
- Avg. valence is 6 (Euler)
- Reduce variation

Optimal valence is

- 6 for interior vertices
- 4 for boundary vertices

Minimize valence excess

$$\sum_{i=1}^{4} (\text{valence}(v_i) - \text{opt\_valence}(v_i))^2$$

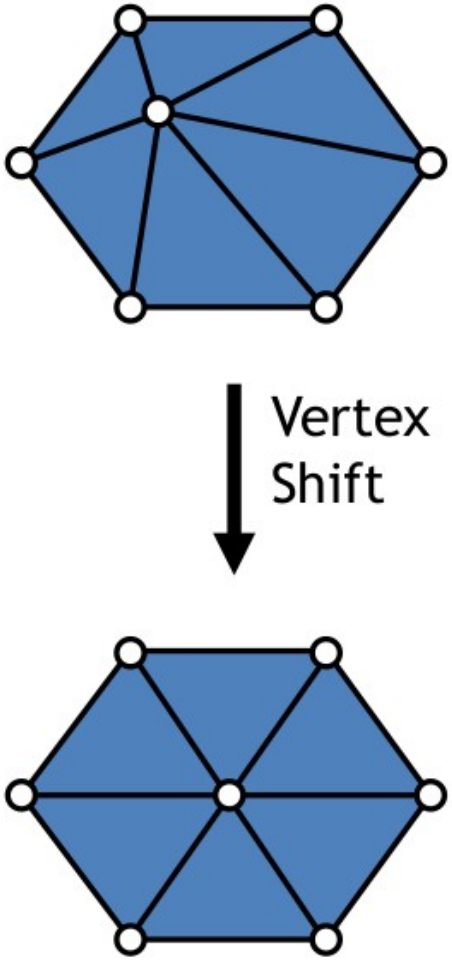


Vertex relocation

Local "spring" relaxation

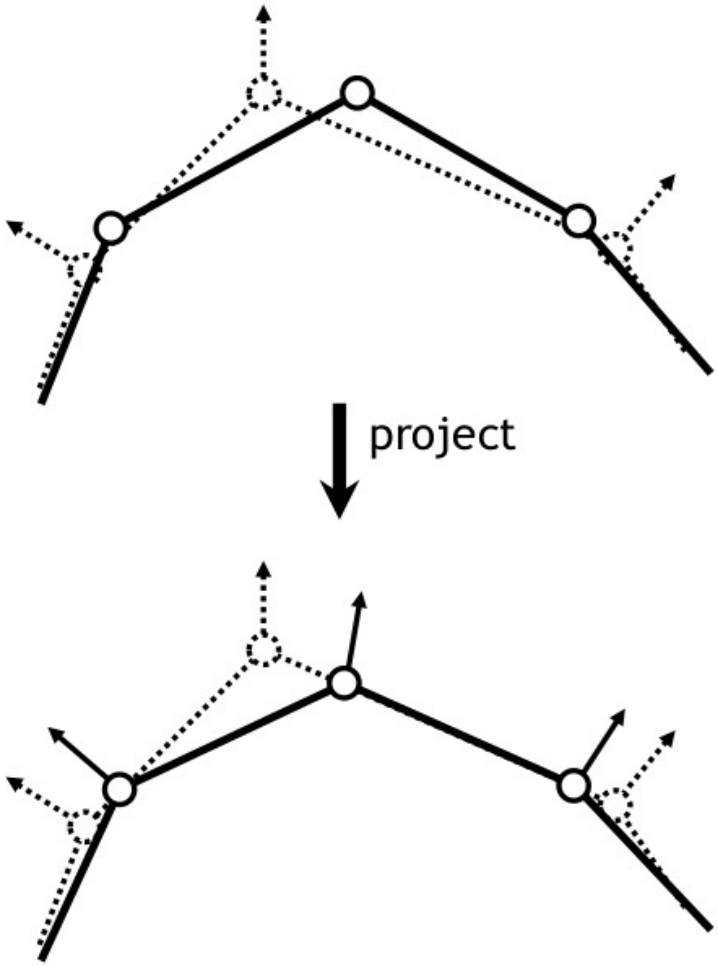
- Uniform Laplacian smoothing
- Bary-center of one-ring neighbors

$$\mathbf{c}_i = \frac{1}{\text{valence}(v_i)} \sum_{j \in N(v_i)} \mathbf{p}_j$$

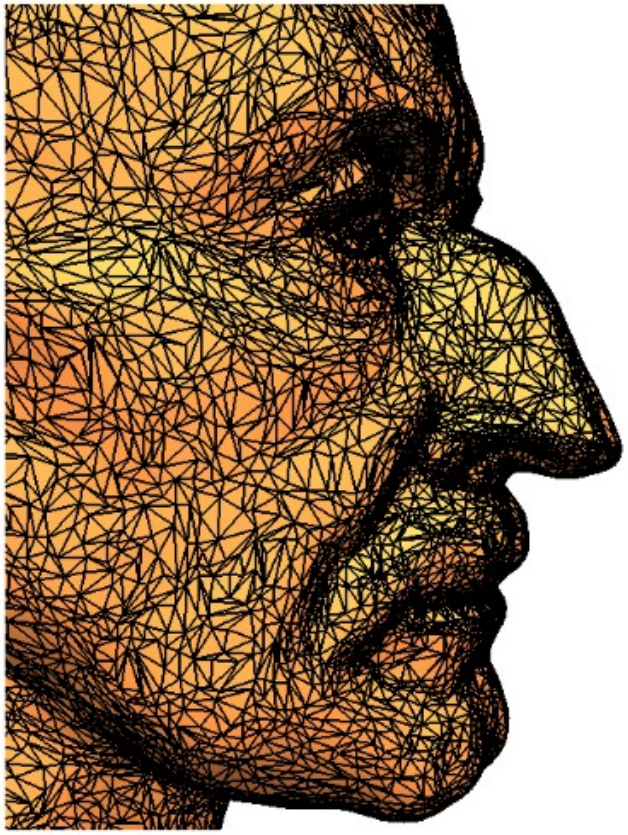


Vertex reprojection

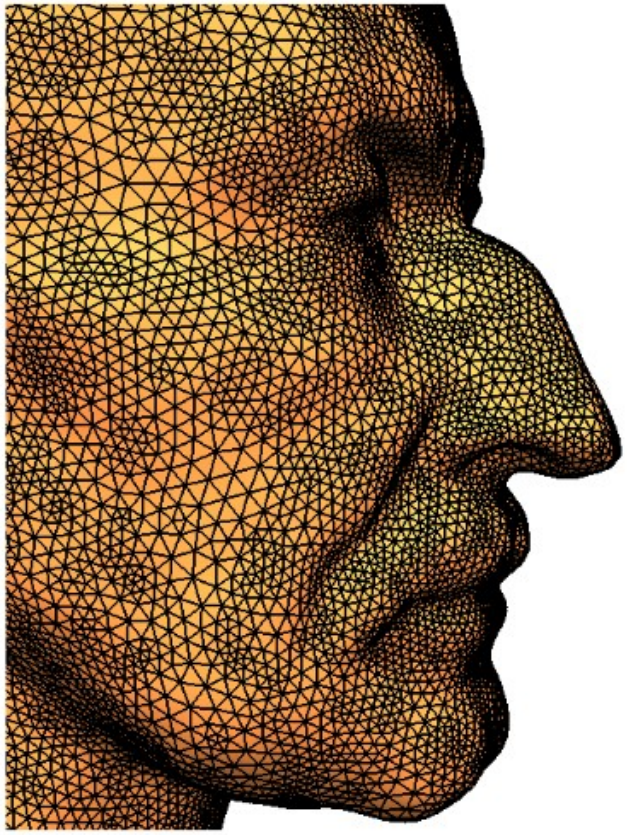
- Project vertices onto original reference mesh
- Assign position & interpolated normal



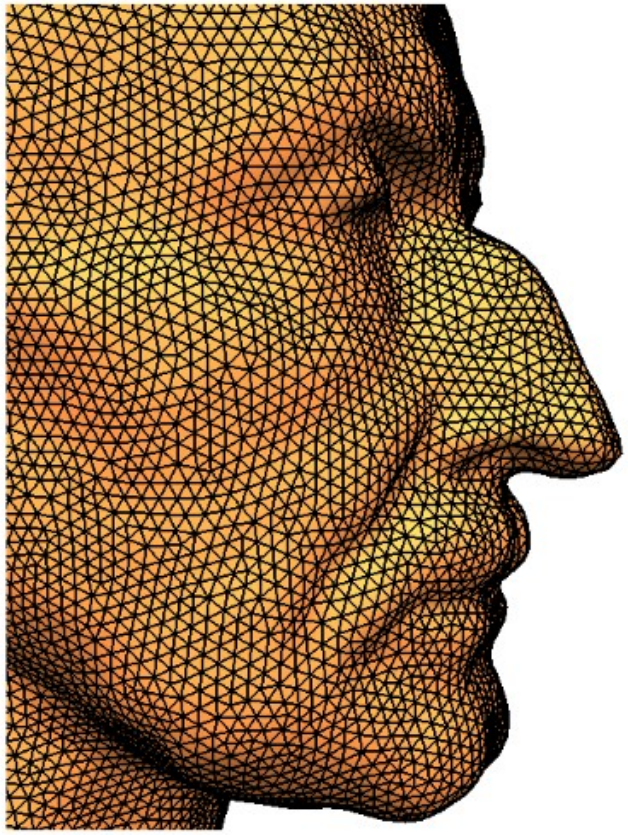
Results



Original



$\left(\frac{1}{2}, 2\right)$



$\left(\frac{4}{5}, \frac{4}{3}\right)$

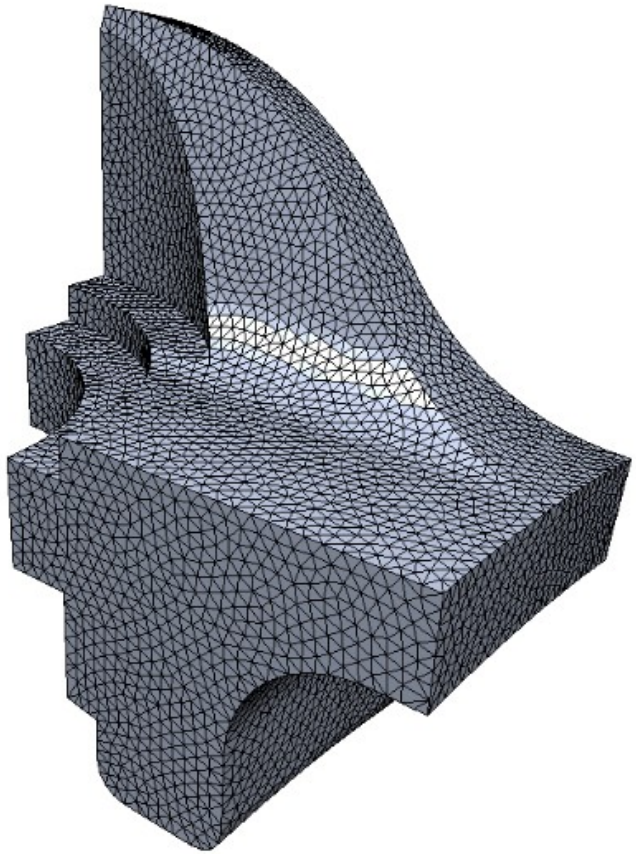
Feature preservation

Define features

- Sharp edges
- Material boundaries

Adjust local operators

- Don’t move corners
- Collapse only along features
- Don’t flip feature edges
- Project to feature curves

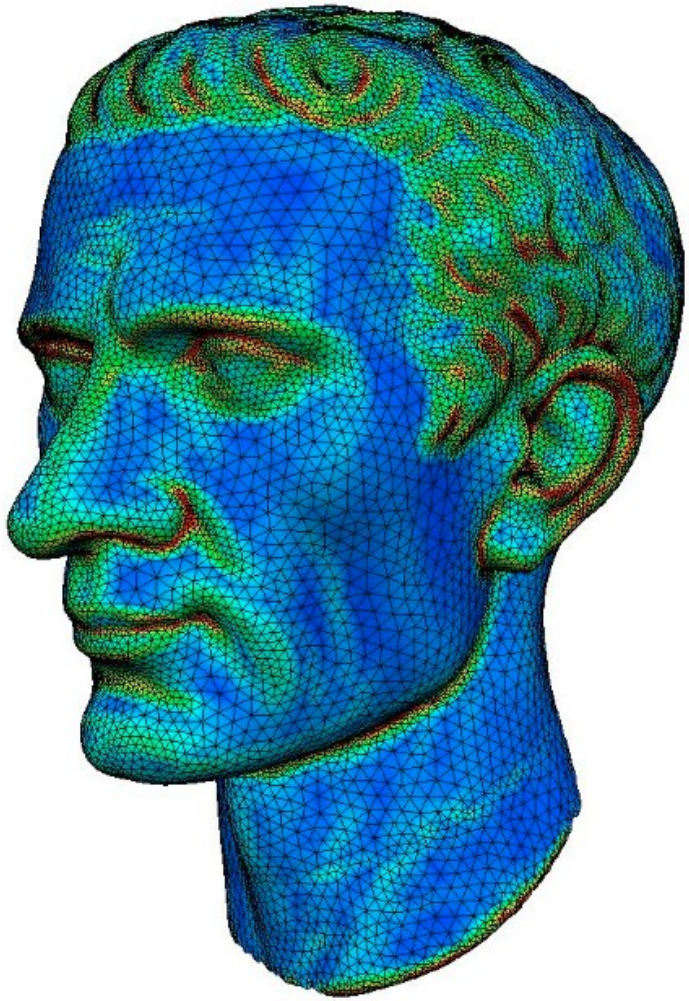


Adaptive remeshing

Precompute max. curvature on reference mesh

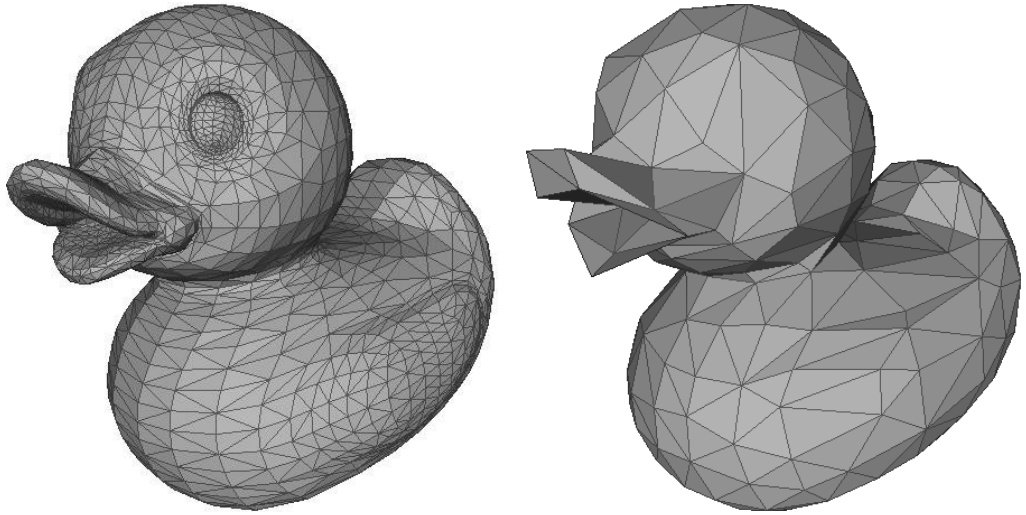
Target edge length locally determined by curvature

Adjust split / collapse criteria

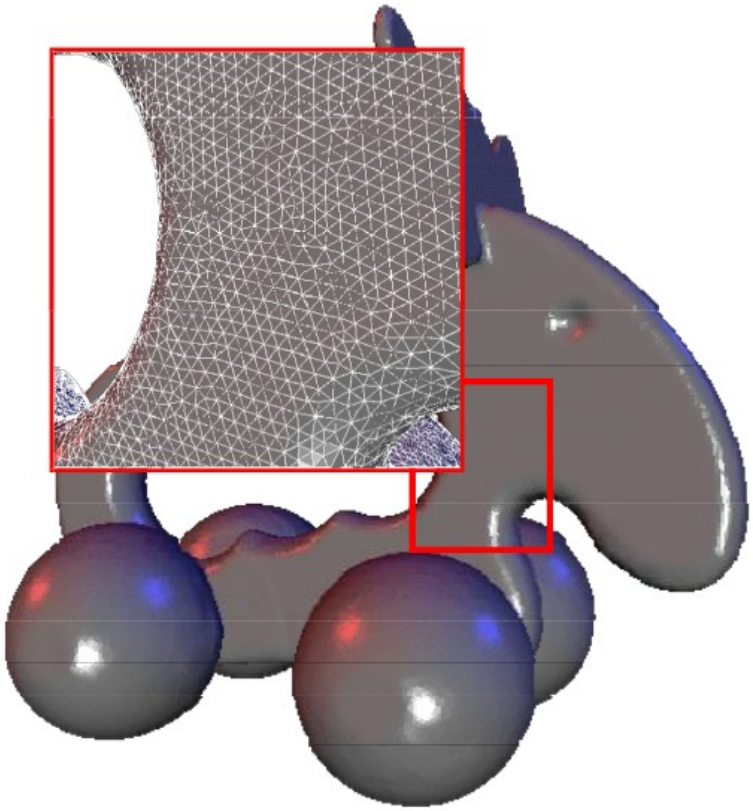


?

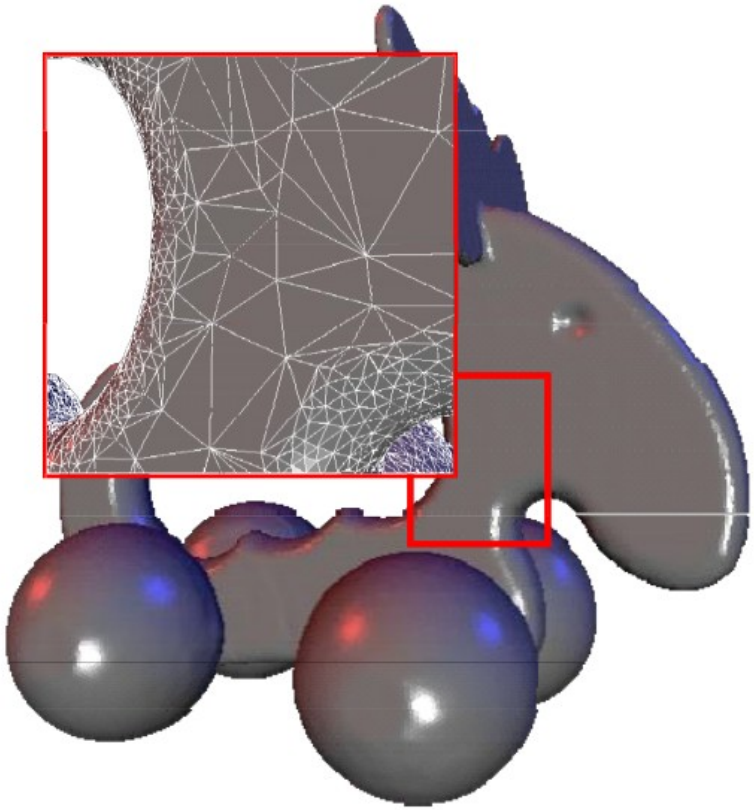
Mesh Simplification



Adaptive tesselation

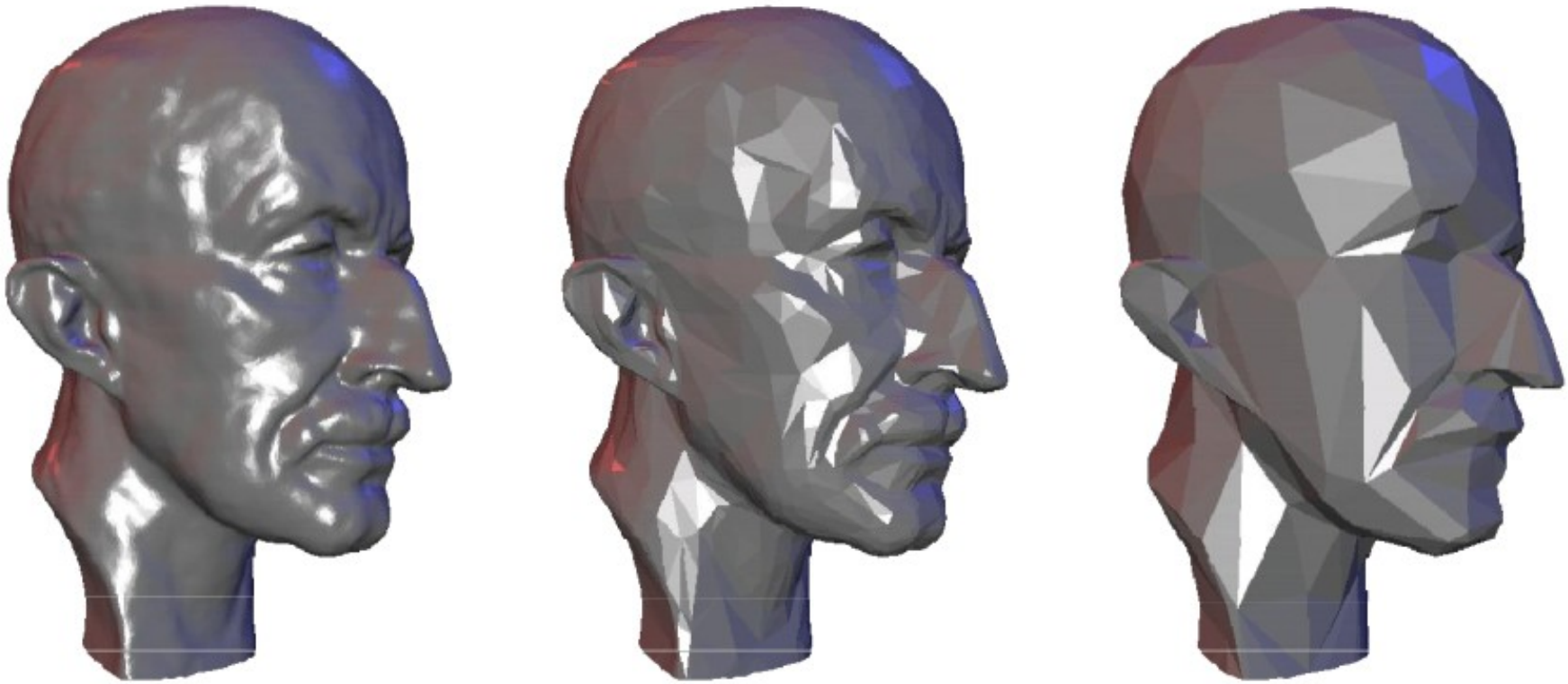


~150k triangles

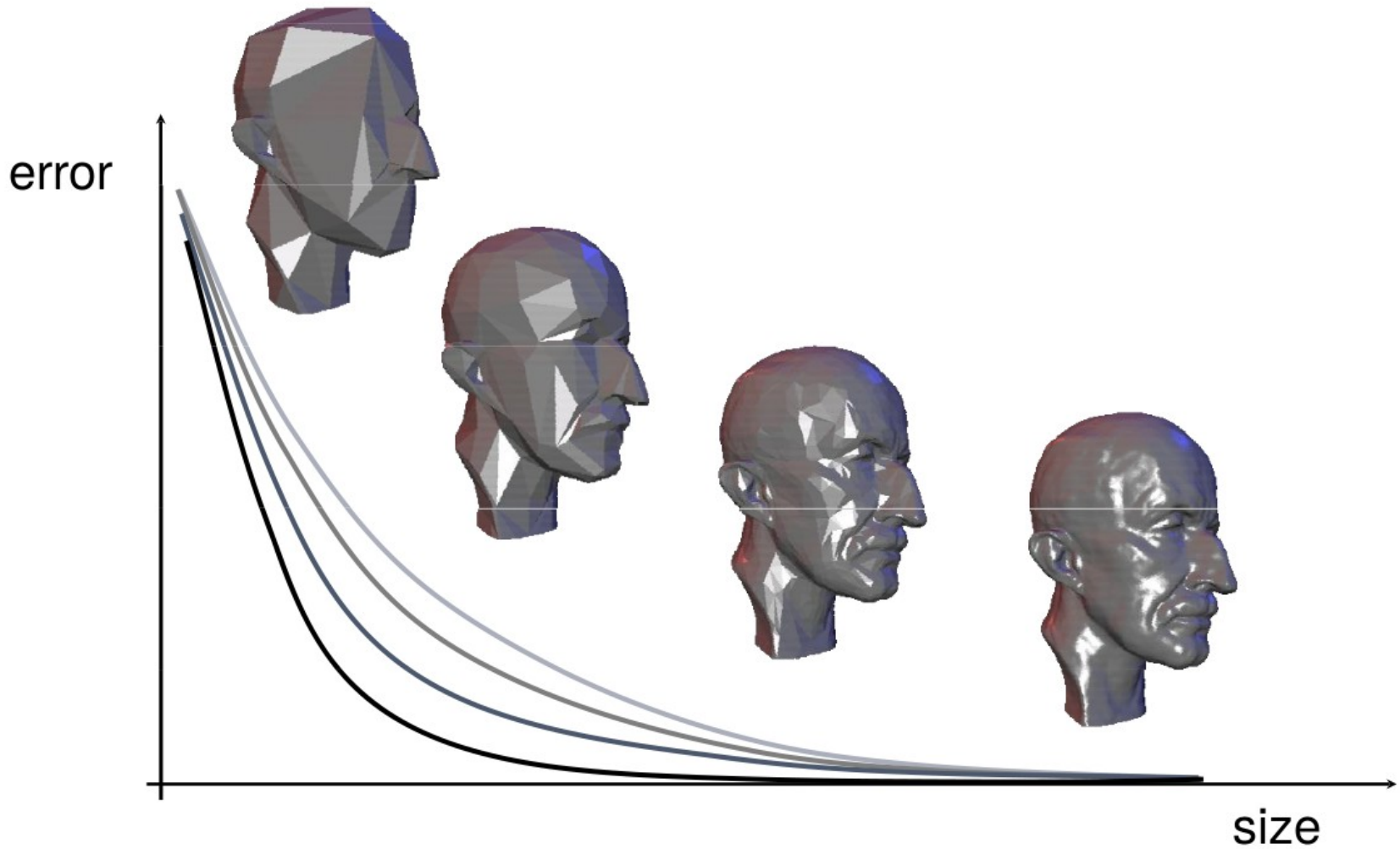


~80k triangles

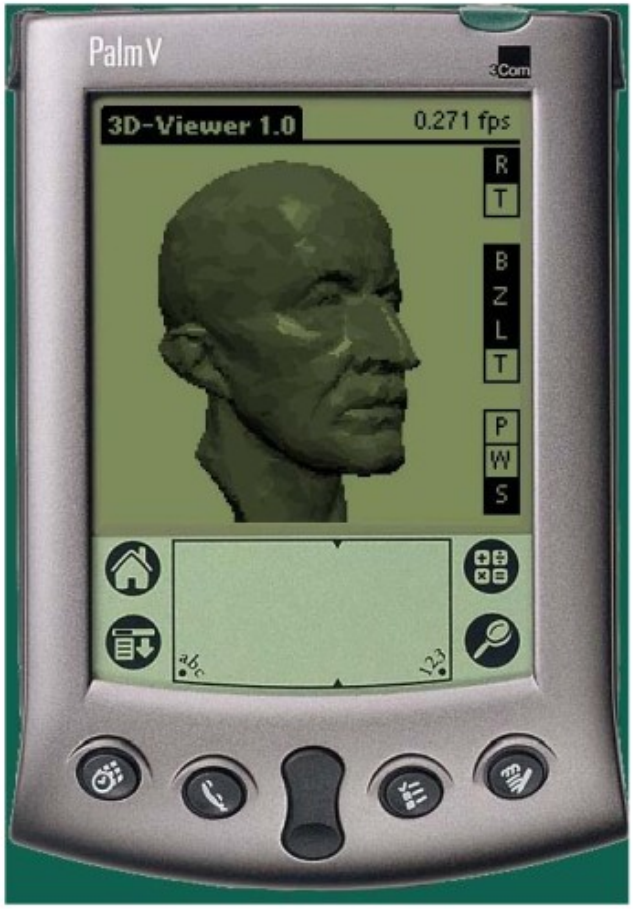
Level of Detail (LoD) rendering



Size/Quality compromise

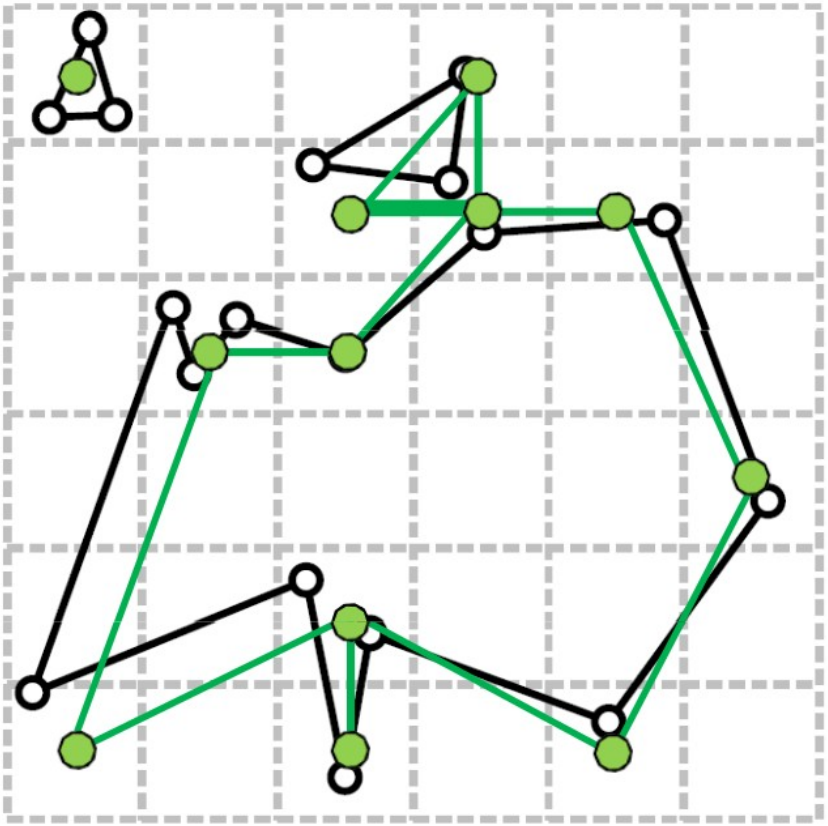


Required for limited capacity hardwares

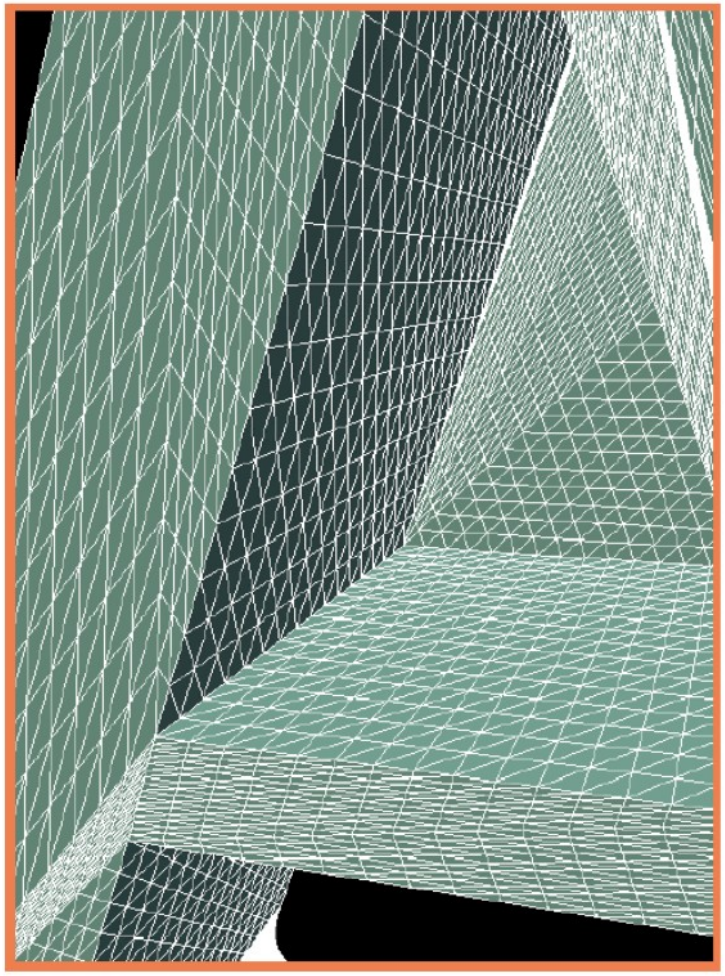


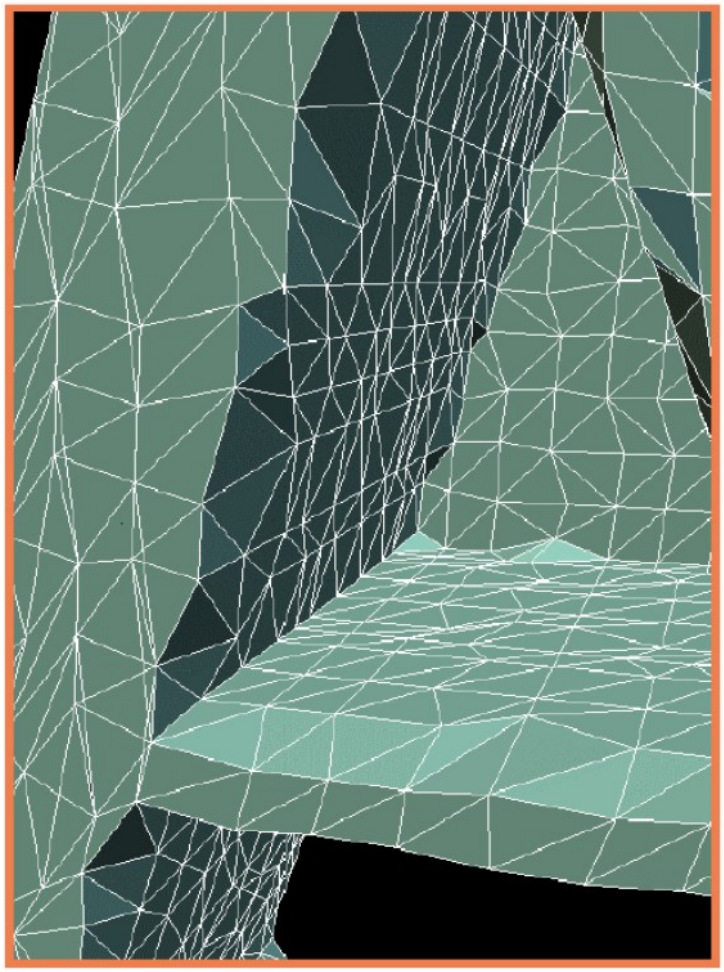
Grid-based Simplification

- Setup a uniform grid
- Average all vertices falling into a single bin
- Keep only non-degenerate triangles (whose corners fall into three different bins)



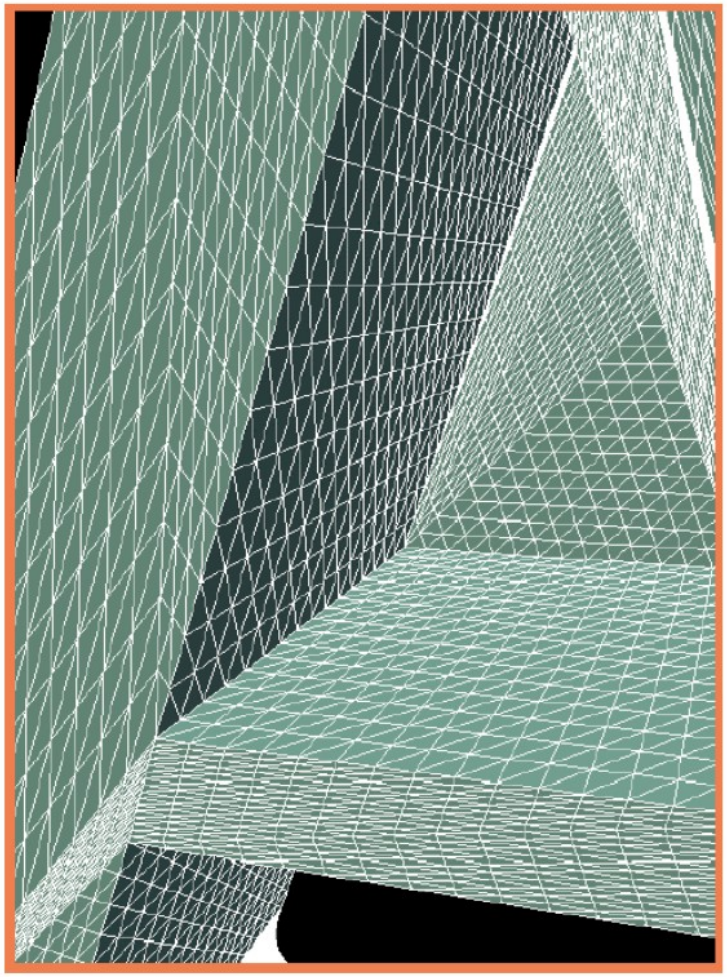
Grid-based Simplification

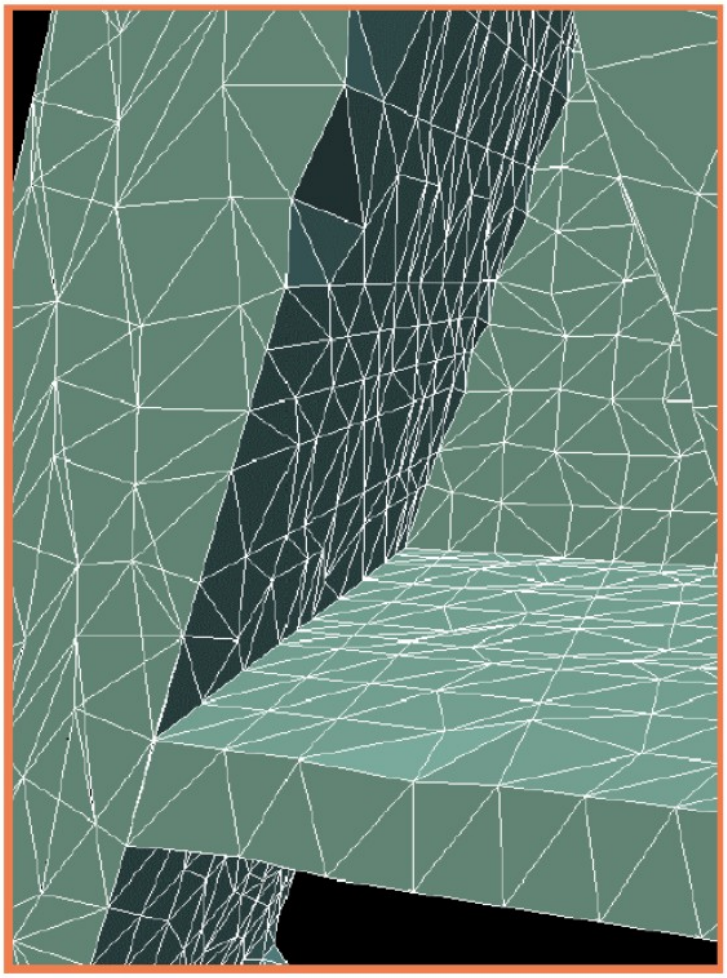




Representant point : using average position

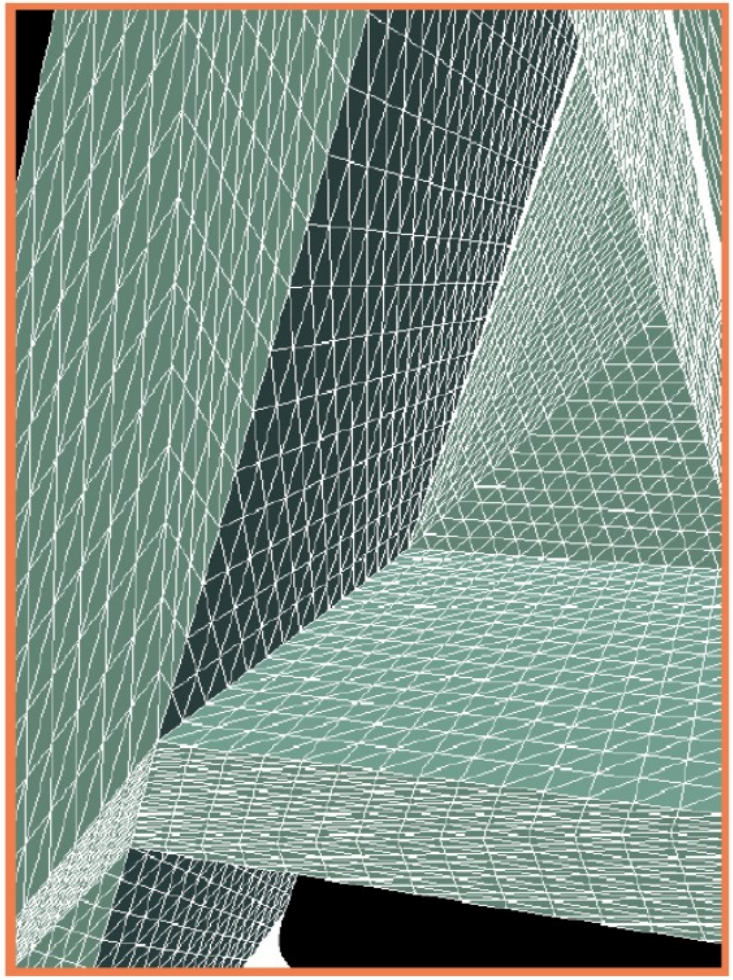
Grid-based Simplification

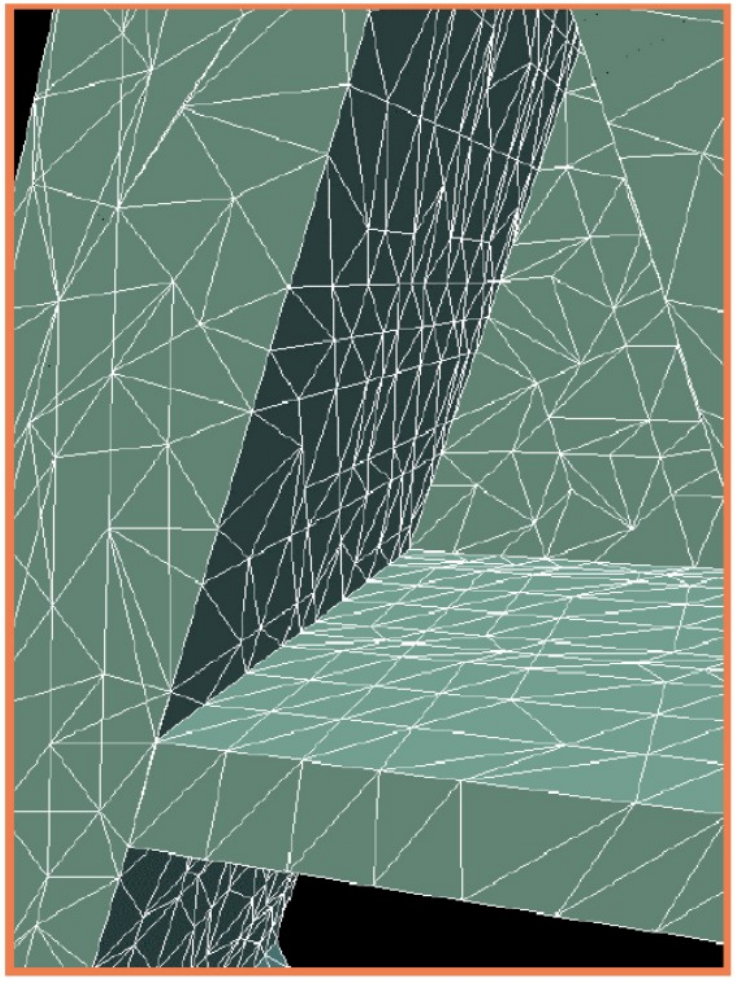




Representant point : using median position

Grid-based Simplification





Representant point : using QEF minimizer

Priors on QEF

Squared distance between x and plane Pi : $d^2(x, P_i) = \left(n_i{}^T x - d_i\right)^2$

Using homogeneous coordinates : $\bar{x} = (x, 1) \quad \bar{n}_i = (n_i, -d_i)$

$$d^2(x, P_i) = \left(\bar{n}_i{}^T \bar{x}\right)^2 = \bar{x}^T \bar{n}_i \bar{n}_i{}^T \bar{x} =: \bar{x}^T Q_i \bar{x}$$

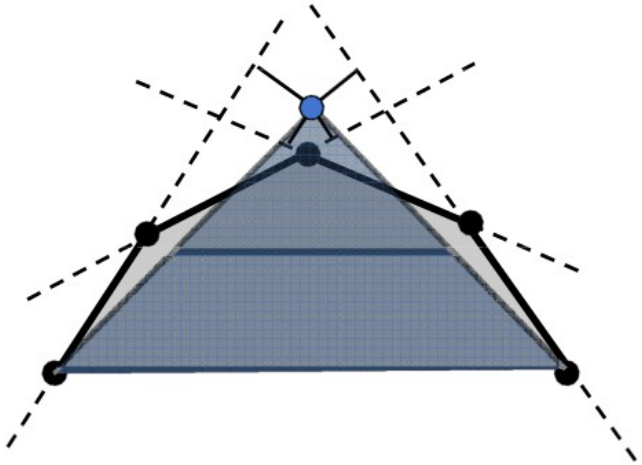
Symmetric 4x4 matrix (quadric)

The sum of squared distances to an arbitrary number of planes is still a quadric !

$$E(x) = \sum_{t_i \in R} \bar{x}^T Q_i \bar{x} = \bar{x}^T \left(\sum_{t_i \in R} Q_i\right) \bar{x} := \bar{x}^T Q \bar{x}$$

Quadric Error Function (QEF) optimization

$$E(x) = \sum_{t_i \in R} \bar{x}^T Q_i \bar{x} = \bar{x}^T \left(\sum_{t_i \in R} Q_i \right) \bar{x} := \bar{x}^T Q \bar{x}$$



QEF

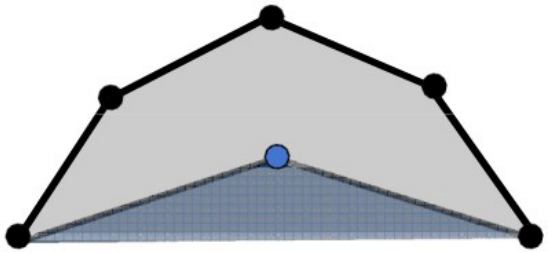
- Easy to construct
- Easy to minimize :

$$Q = \begin{bmatrix} A & -b \\ -b^T & c \end{bmatrix}$$

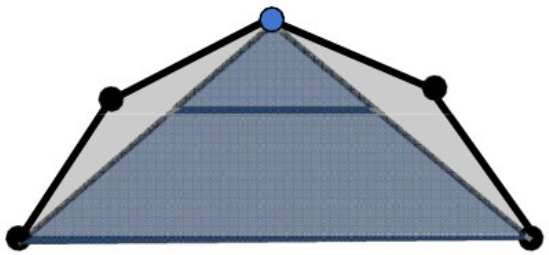
$$Ax = b$$

- Soved using, e.g., SVD

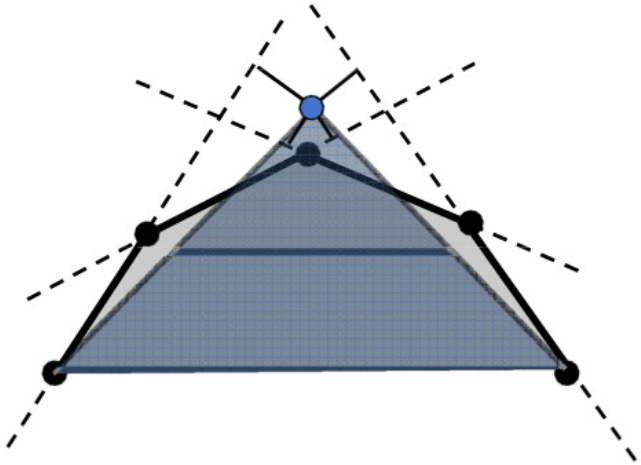
Comparison



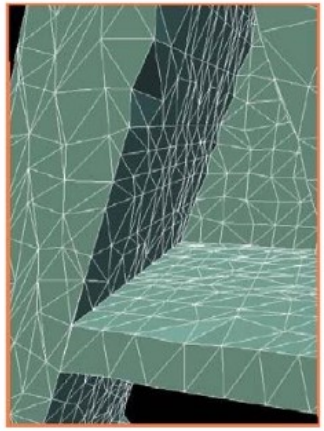
mean



median

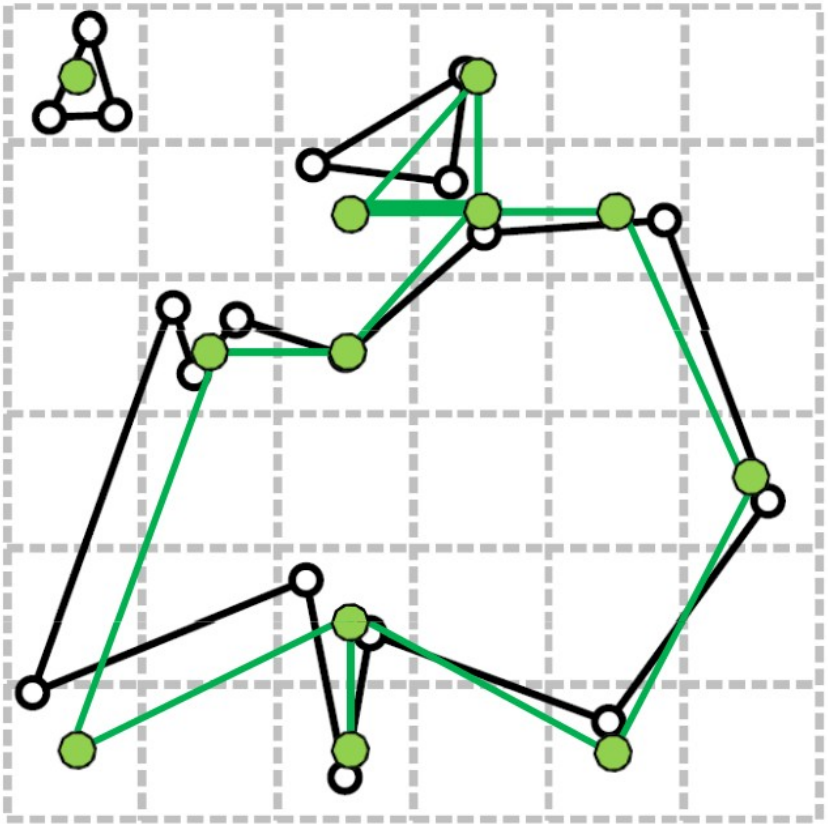


QEF



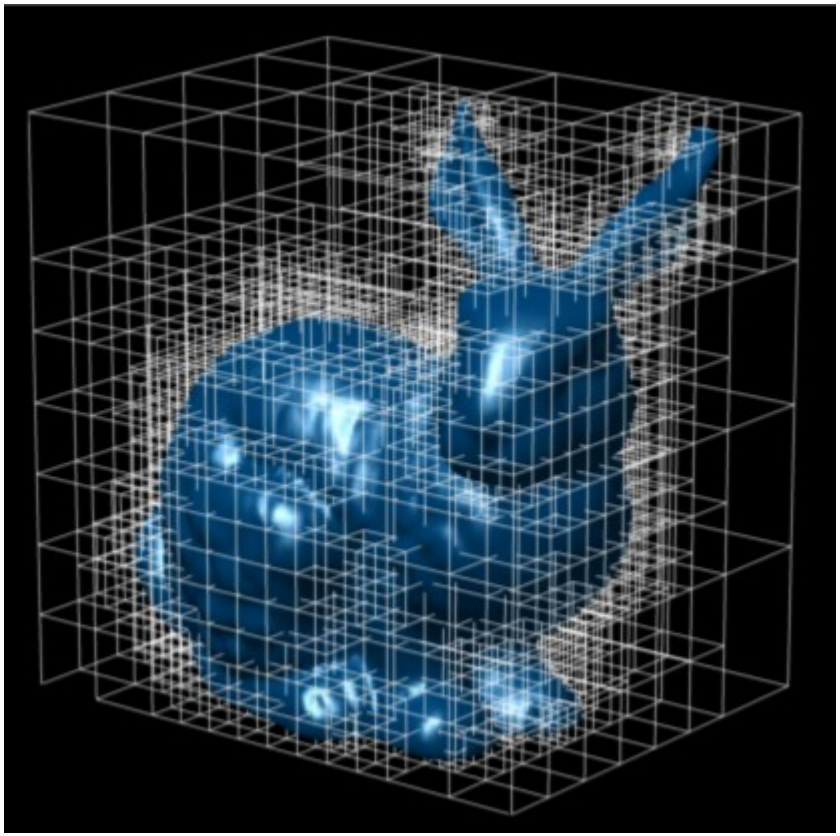
Grid-based Simplification

- Extremely simple to implement (+)
- Very costly for large grids (-)
- Uniform level of detail (-)



Adaptivity

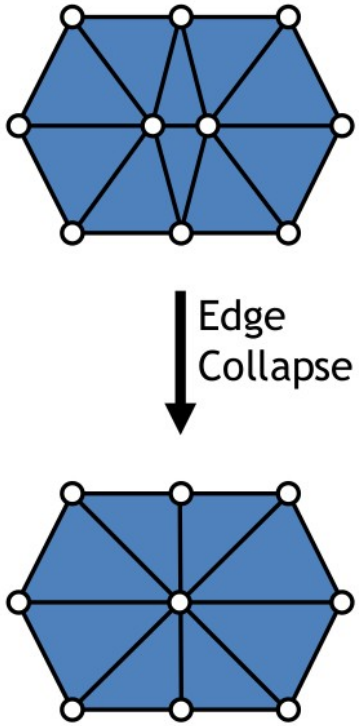
- Use an octree instead of a uniform grid
- Not much more complicated
- Allocate only cells around the input vertices (no memory wasted)



?

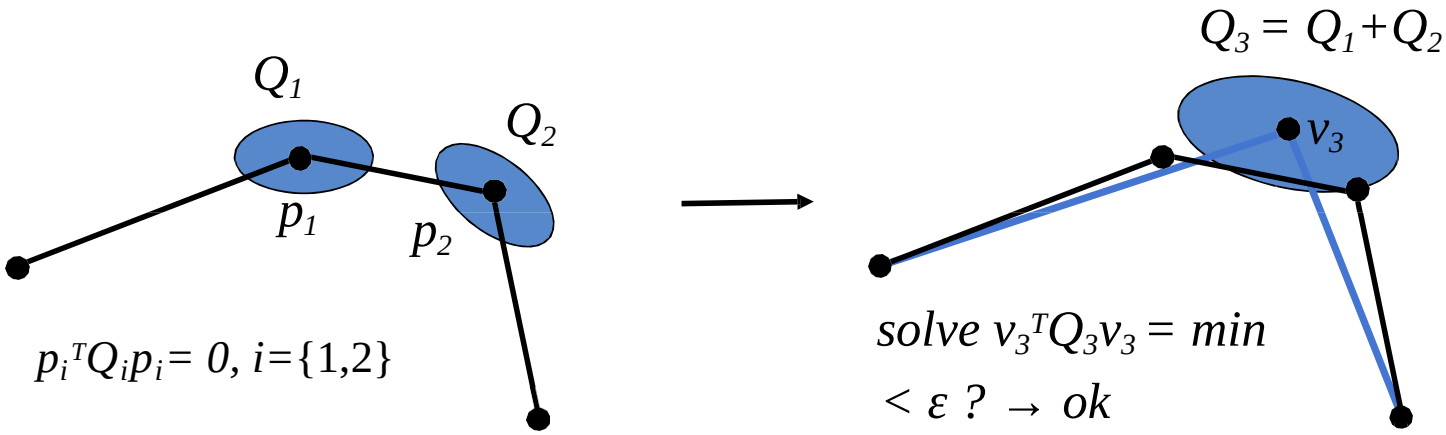
Edge-decimation algorithms

- Uses simple concepts already seen (edge collapse, QEF)
- Iterative
- Continuous level of detail
- Simple to implement
- The basis for lots of algorithms

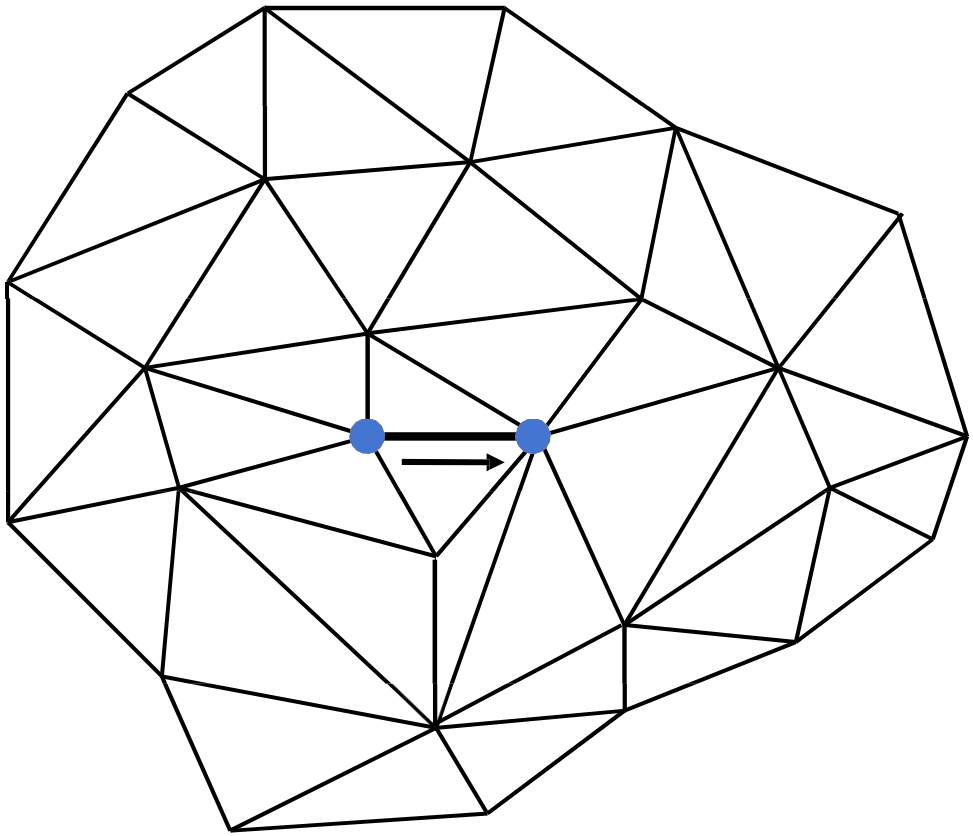


Edge-decimation algorithms

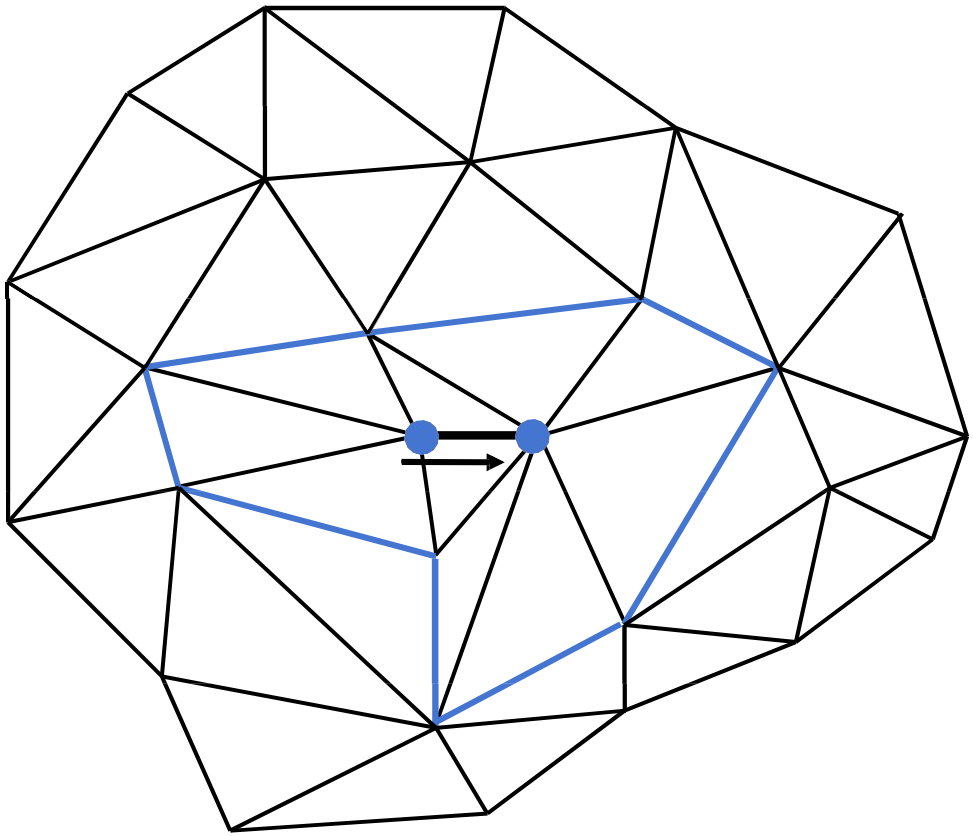
- Pick an edge (v_i,v_j), and collapse it to a new vertex v. 2 adjacents triangles are removed.
- The cost of a collapse is given by its QEF (a QEF is associated with each initial input vertex, and the QEF of a new vertex is the sum of the QEFs of the collapsed vertices). The new vertex position is the minimizer of the QEF



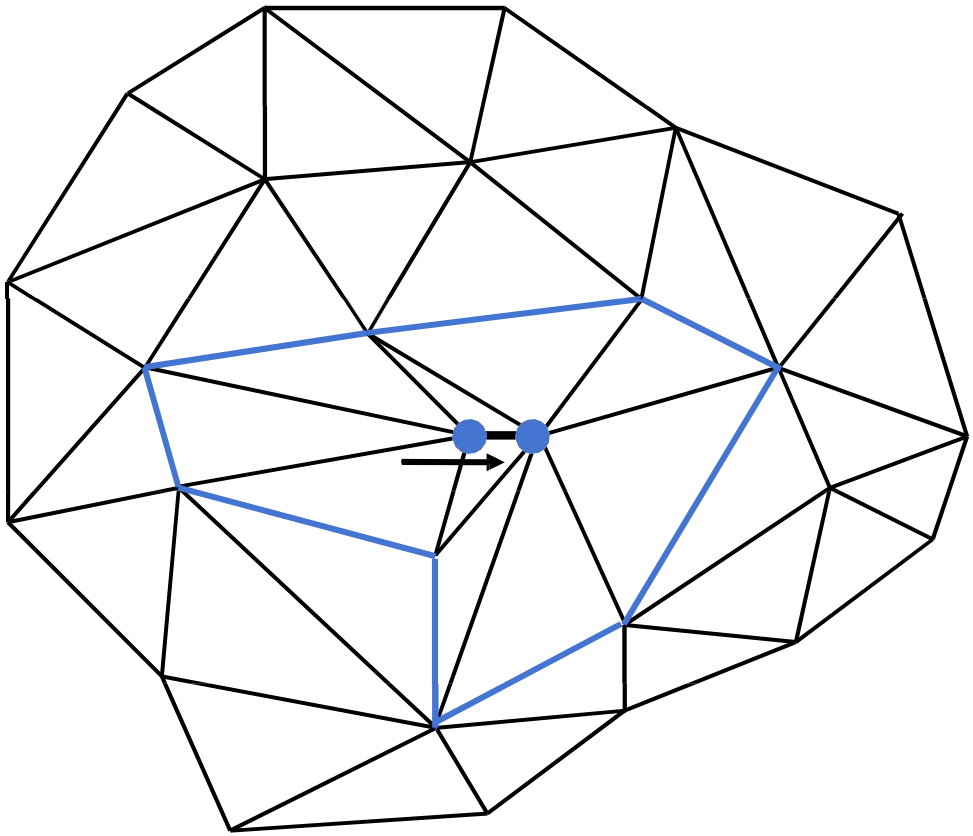
Edge-decimation algorithms



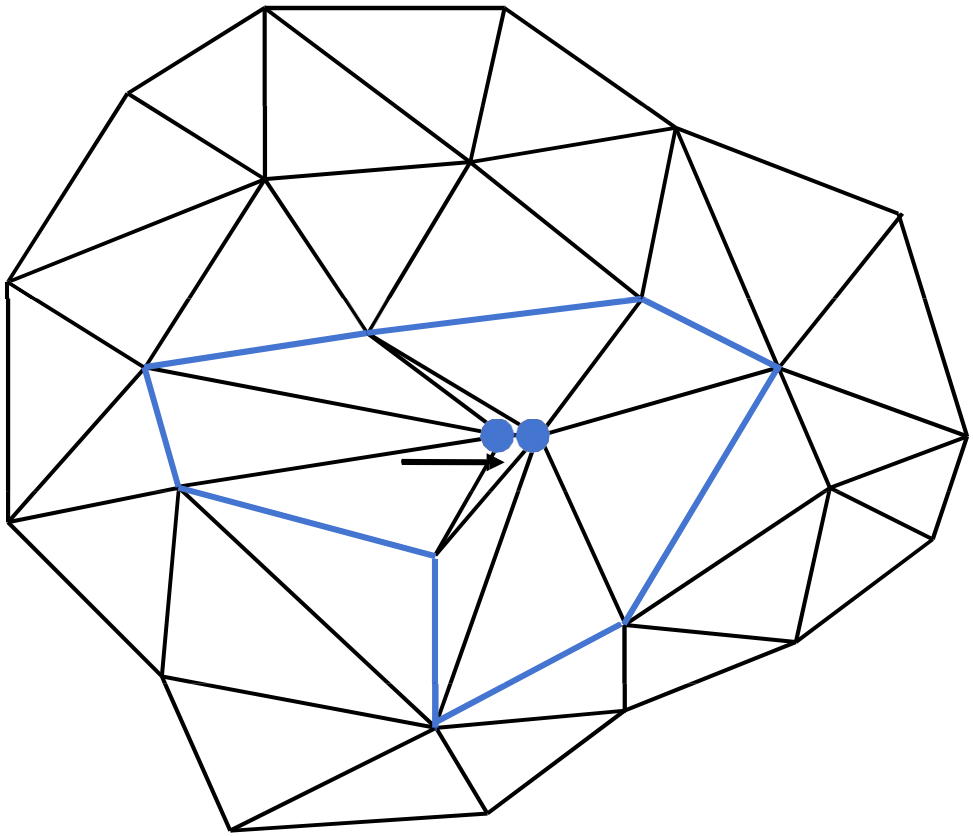
Edge-decimation algorithms



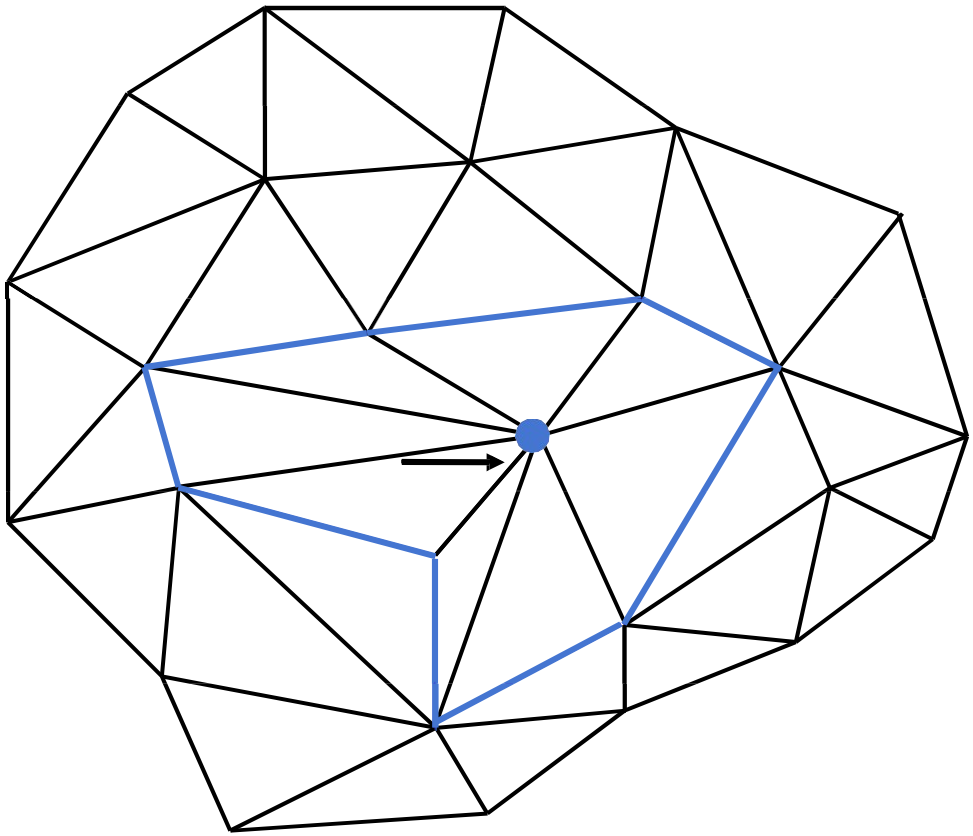
Edge-decimation algorithms



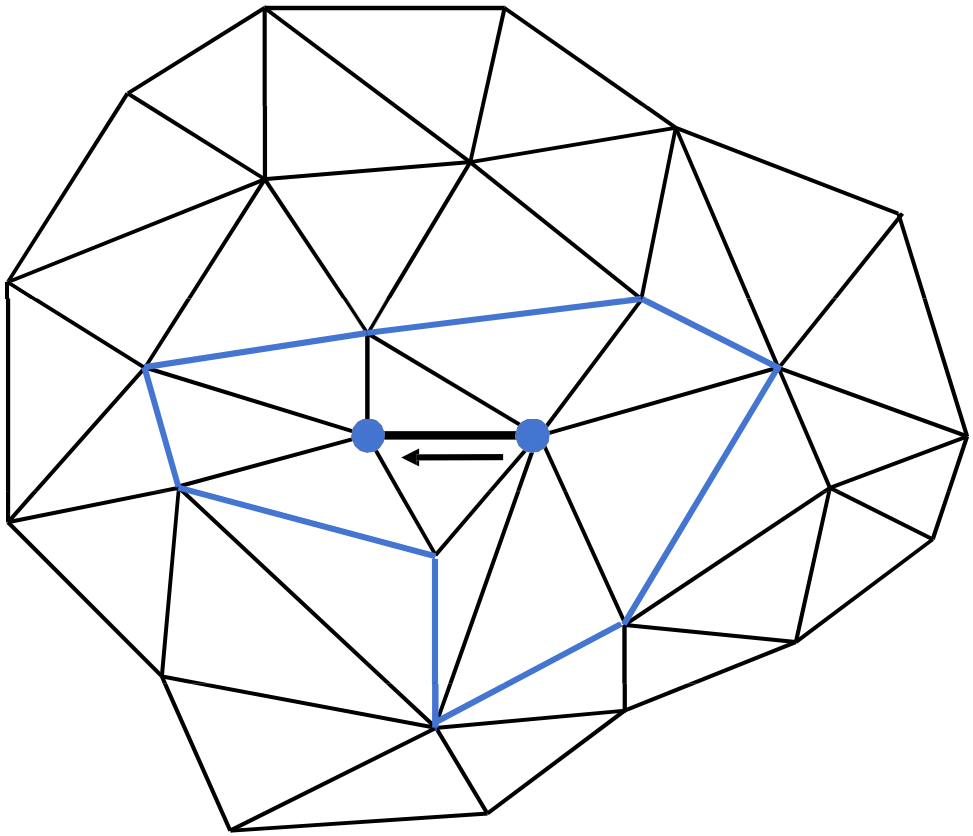
Edge-decimation algorithms



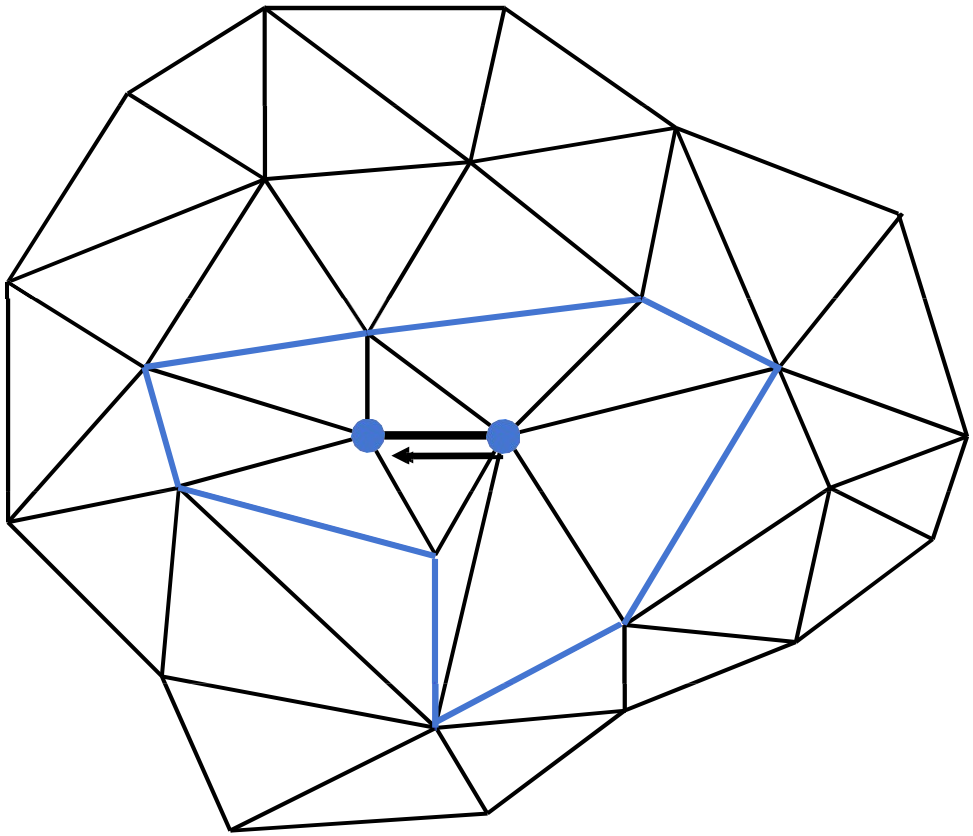
Edge-decimation algorithms



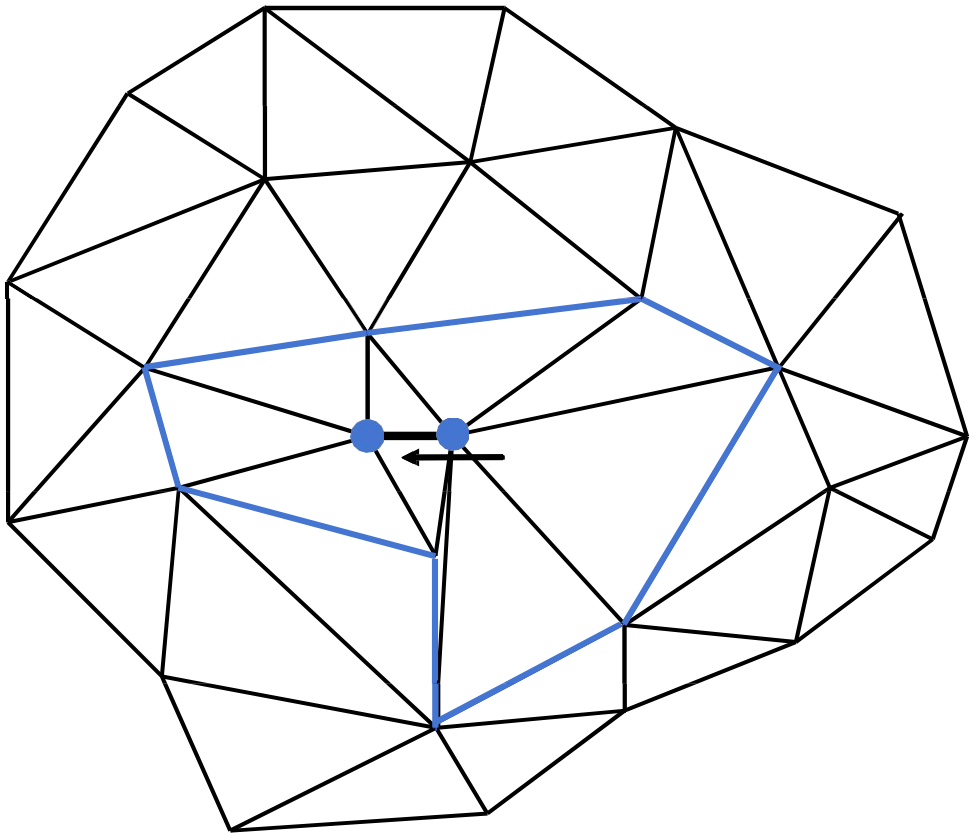
Edge-decimation algorithms



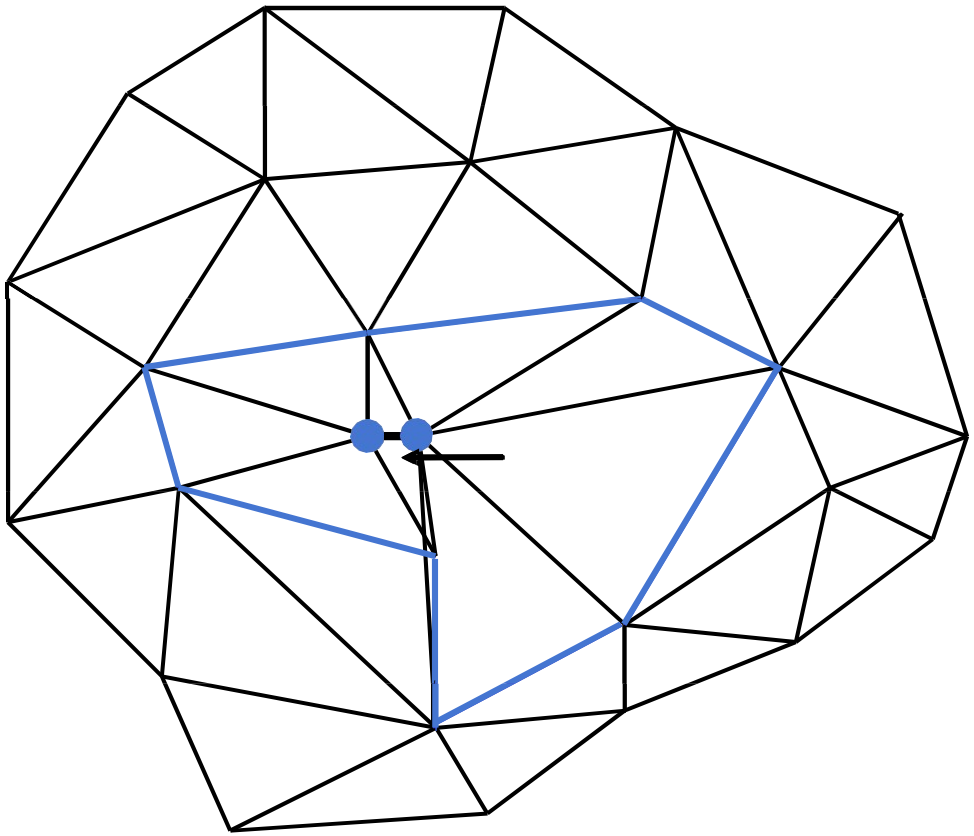
Edge-decimation algorithms



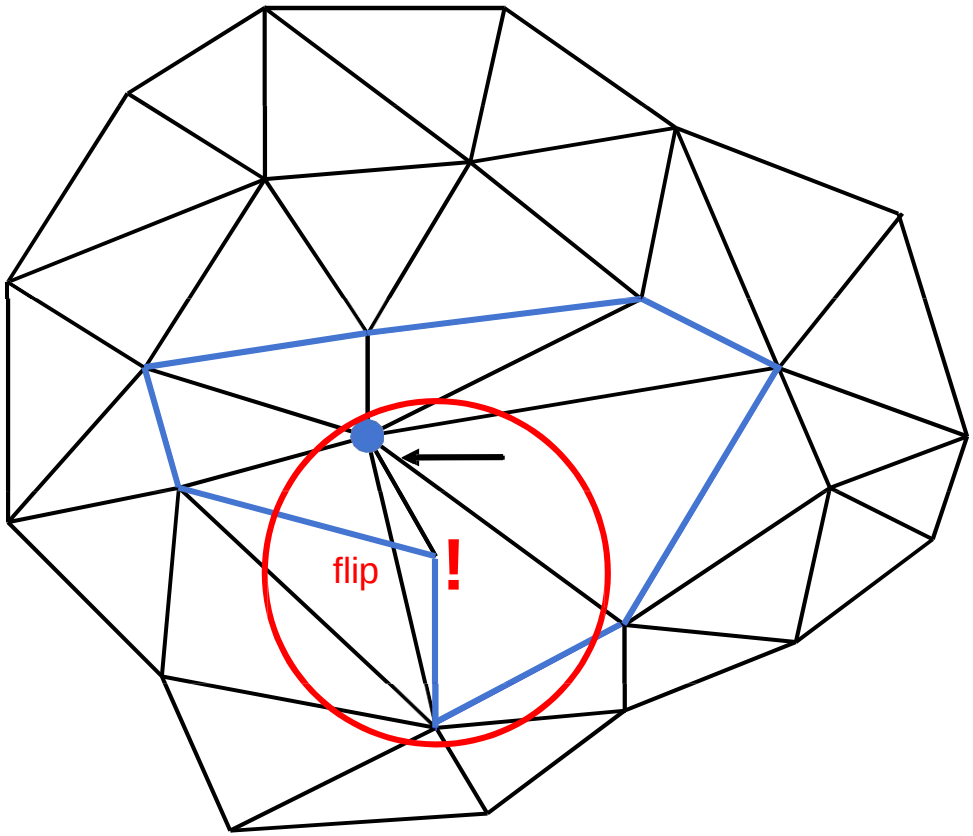
Edge-decimation algorithms



Edge-decimation algorithms

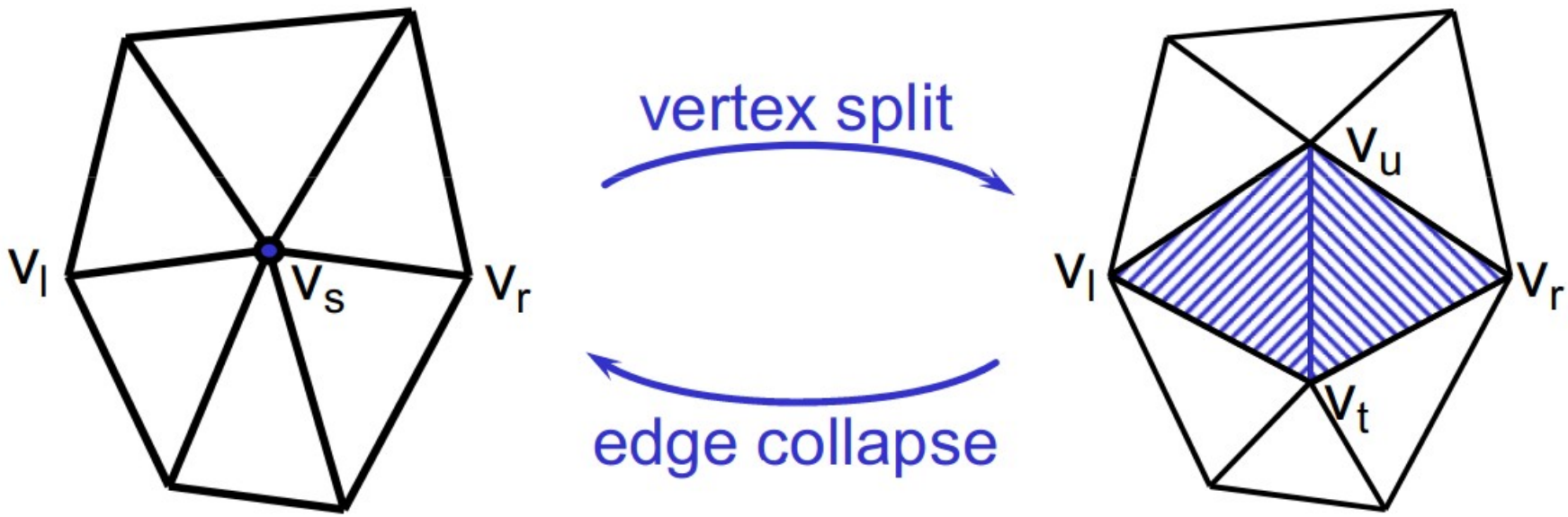


Edge-decimation algorithms



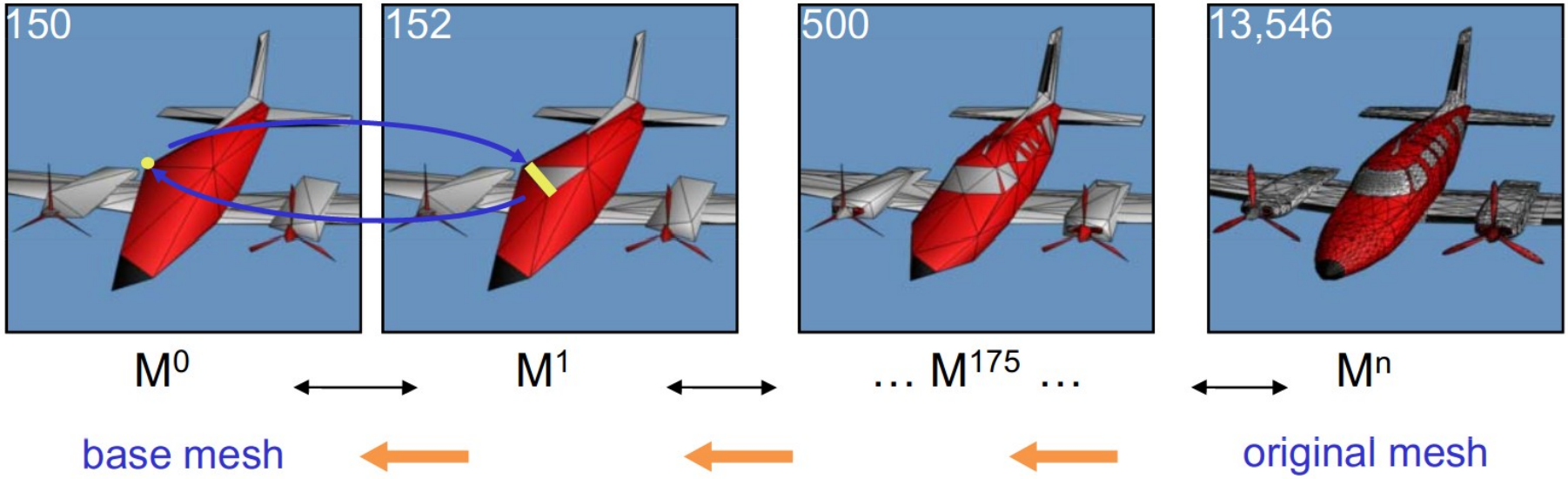
Progressive meshes

- It is very easy to record all edge collapse operations (and reverse as well)



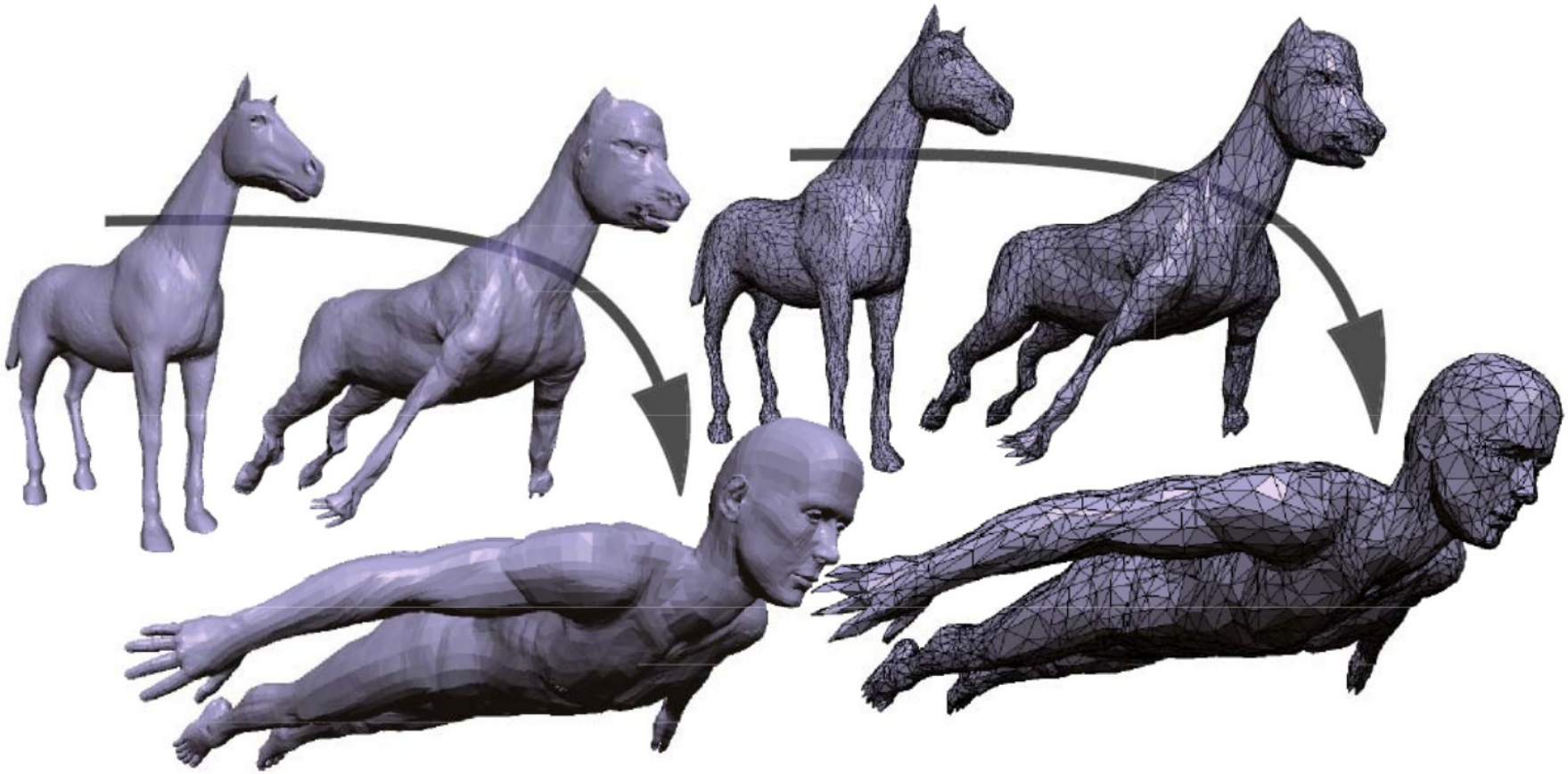
Progressive meshes

- Noting M(k+1) the mesh obtained from decimating one edge of mesh M(k), and M0 the input mesh,
- All meshes {M(k)}_k can be efficiently stored as M0 and a set of N small collapse/split operations (for N vertices in M0)
- The hierarchy defines a « progressive mesh » :



Extension to animated meshes

- Simply sum the QEFs over all the input frames, and decimate jointly an edge in all the frames.



?