

# Advanced Computer Graphics

## Mesh Processing



G. Zachmann  
University of Bremen, Germany  
[cgvr.cs.uni-bremen.de](http://cgvr.cs.uni-bremen.de)

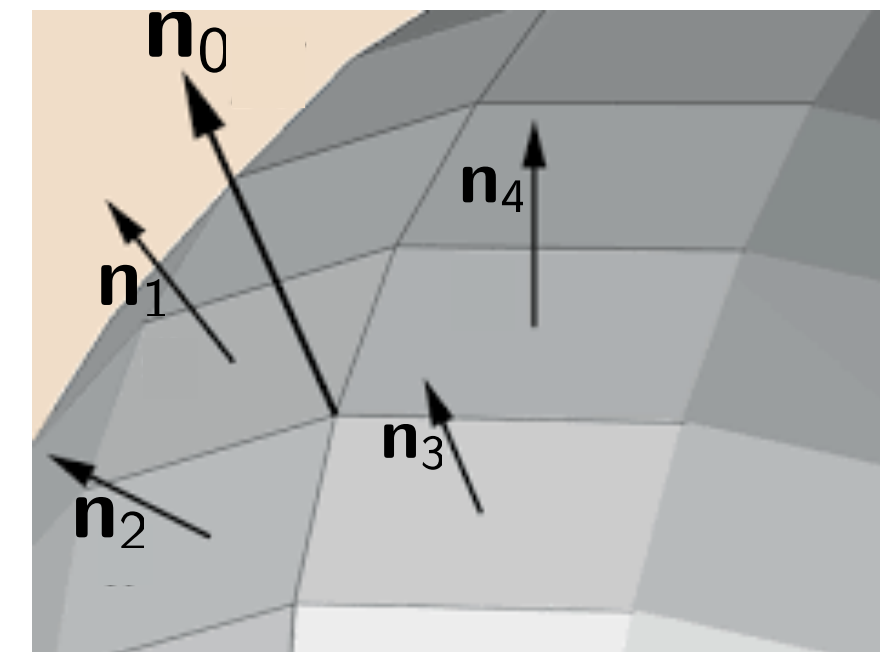
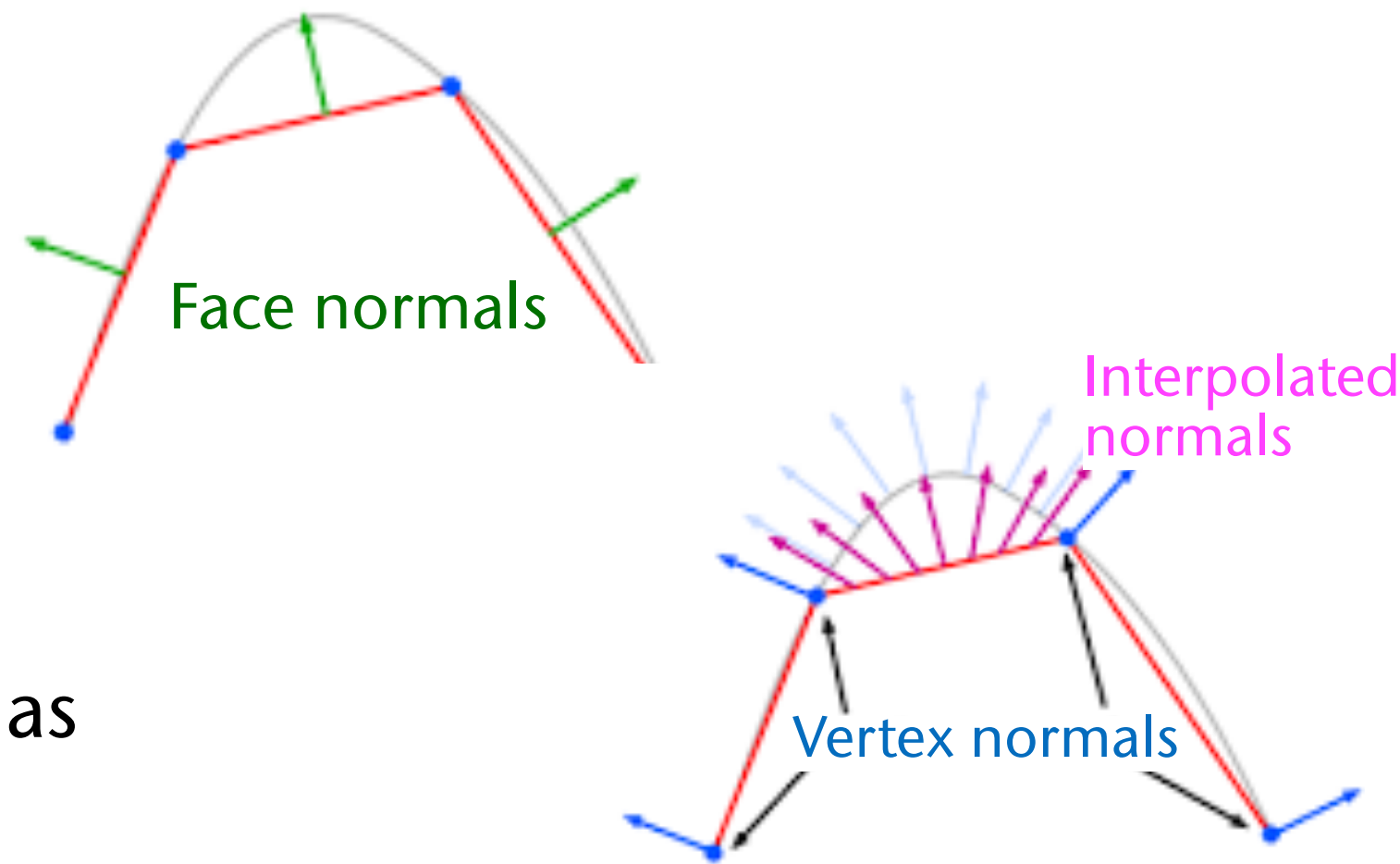
# Vertex Normals

- Polygonal surfaces are (usually) just a linear approximation of smooth surfaces
- Wanted: good vertex normals
  - "Good" = as close as possible to true normals
  - Ansatz: compute vertex normal  $\mathbf{n}_0$  at vertex  $V_0$  as

$$\mathbf{n}_0 = \sum_{i=1}^k w_i \mathbf{n}_i$$

where  $\mathbf{n}_i$  = normal of face given by  $V_0V_iV_{i+1}$  ,  
 $w_i$  = some weight

- Question: which weights give best normals?

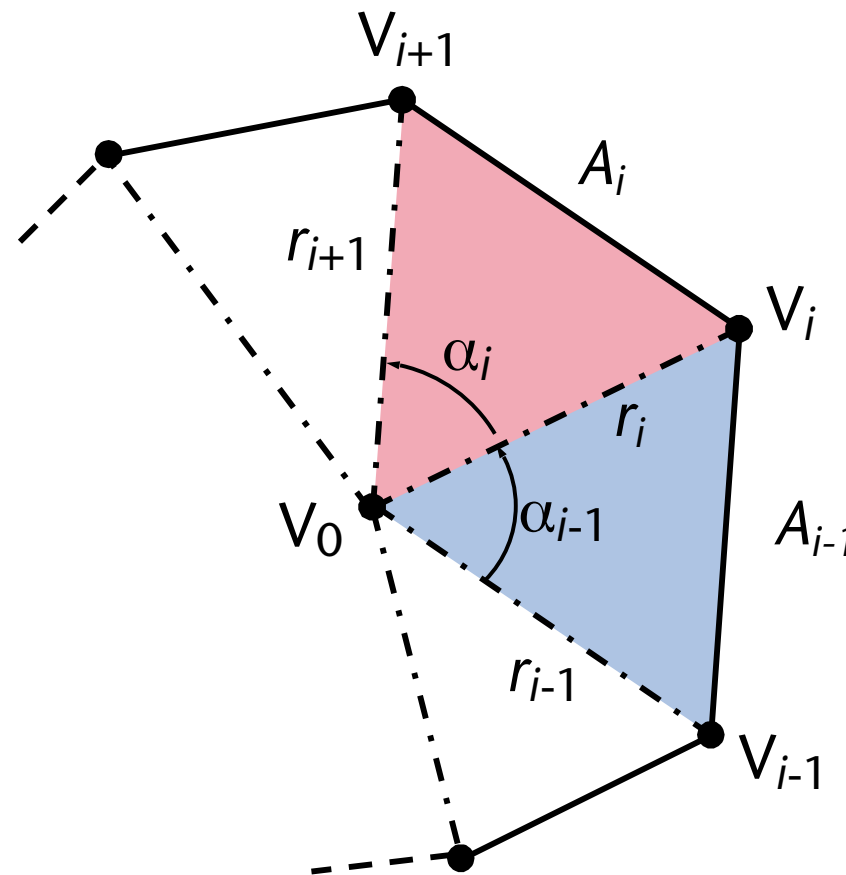


# Weights That Have Been Proposed in the Literature

- No weights, i.e.  $w_i = 1$
- $w_i = A_i$  (area),  $w_i = \alpha_i$ ,  
 $w_i = \frac{1}{r_i r_{i+1}}$  with  $r_i := \|V_i - V_0\|$
- Best (so far) [Nelson Max]:

$$w_i = \frac{\sin(\alpha_i)}{r_i r_{i+1}}$$

- Gives *provably* correct normals for polyhedra inscribed in sphere (= degree 2 surface)
- Smallest RMSE almost everywhere for polygonal approximations of polynomial surface of degree 3



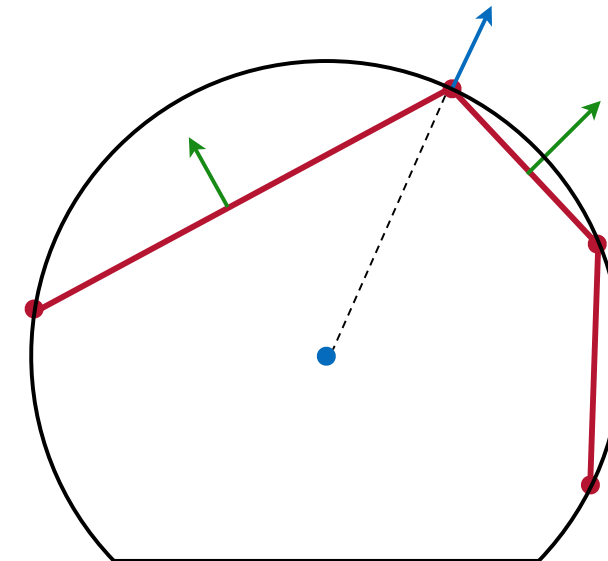
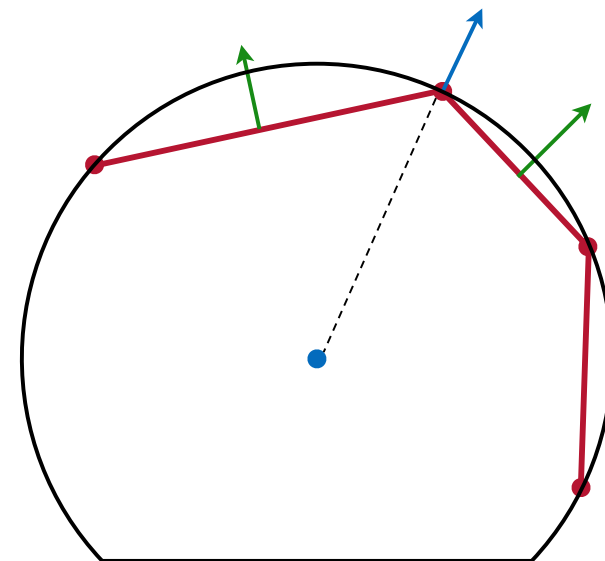
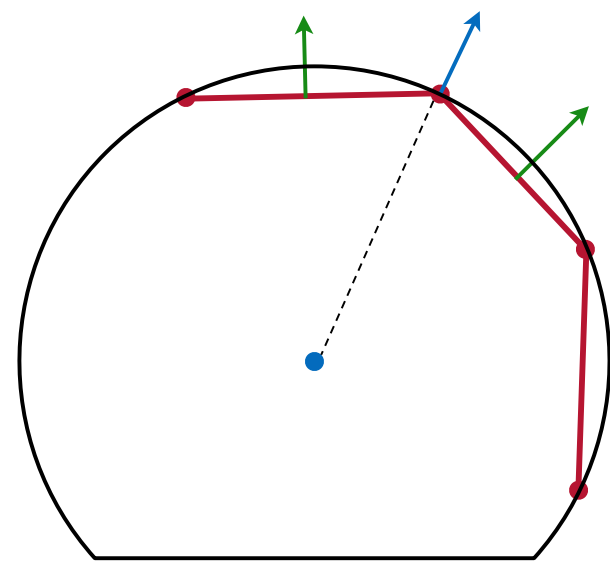
Weights	RMSE
One (no weights)	7.3 – 3.7
$A_i$	6.5 – 2.8
$\alpha_i$	10.7 – 3.4
$\frac{1}{r_i r_{i+1}}$	7.3 – 5.1
Best $(\frac{\sin(\alpha_i)}{r_i r_{i+1}})$	3.0 – 1.5

- Practical computation:

- Remember:  $(V_i - V_0) \times (V_{i+1} - V_0) = \sin(\alpha_i) r_i r_{i+1} \mathbf{n}_i$
- In practice, this allows for easier computation of the vertex normal:

$$\mathbf{n}_0 = \sum_{i=1}^k \frac{(V_i - V_0) \times (V_{i+1} - V_0)}{(V_i - V_0)^2 (V_{i+1} - V_0)^2}$$

- Geometric intuition why *longer* faces should have *smaller* weights:

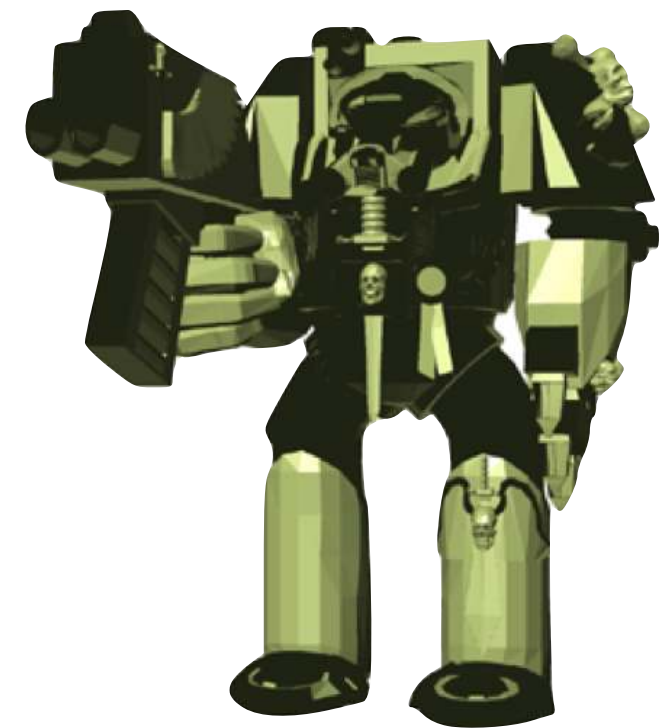


# Consistent Normal Orientation for Meshes

- Problem:
  - Many models consist of many unconnected **patches** (in particular those created with modelling tools)
  - Patches do not necessarily have consistent orientation
- Bad consequences:
  - Two-sided lighting is necessary (slightly slower than one-sided lighting)
  - BSP representation of polyhedra is difficult to construct with inconsistent normals
  - And many more ...



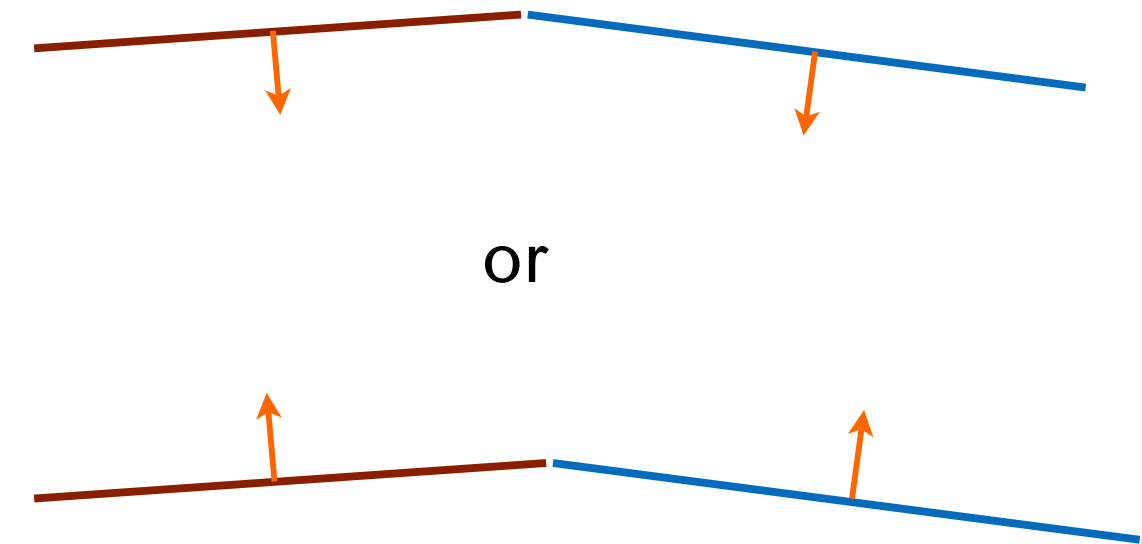
double-sided  
lighting



single-sided lighting

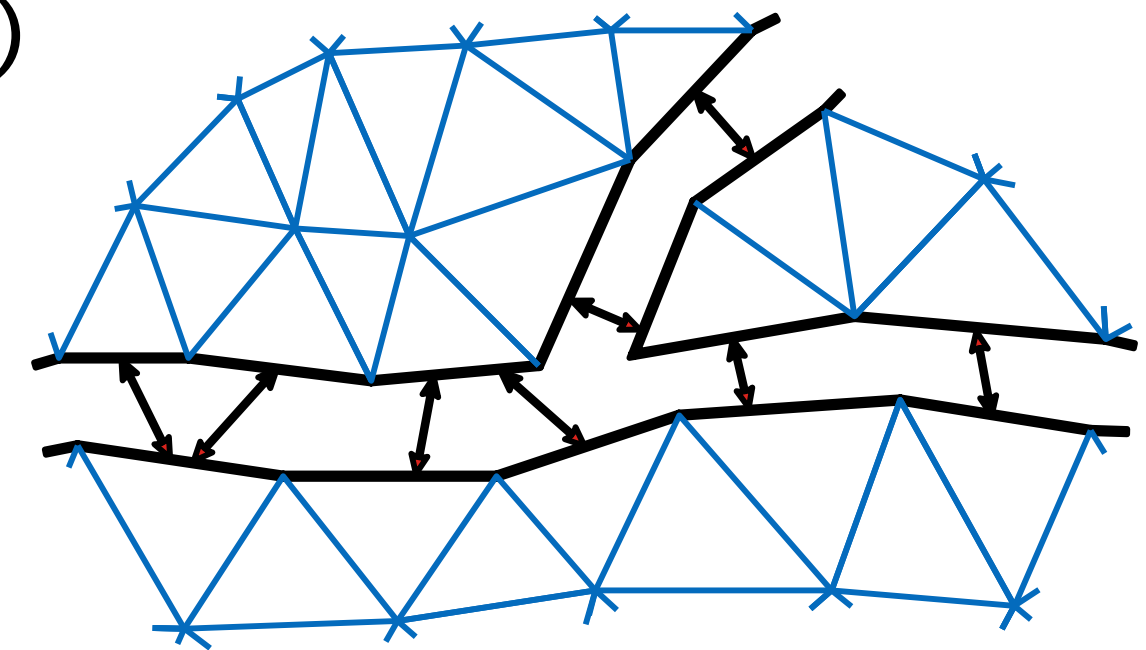
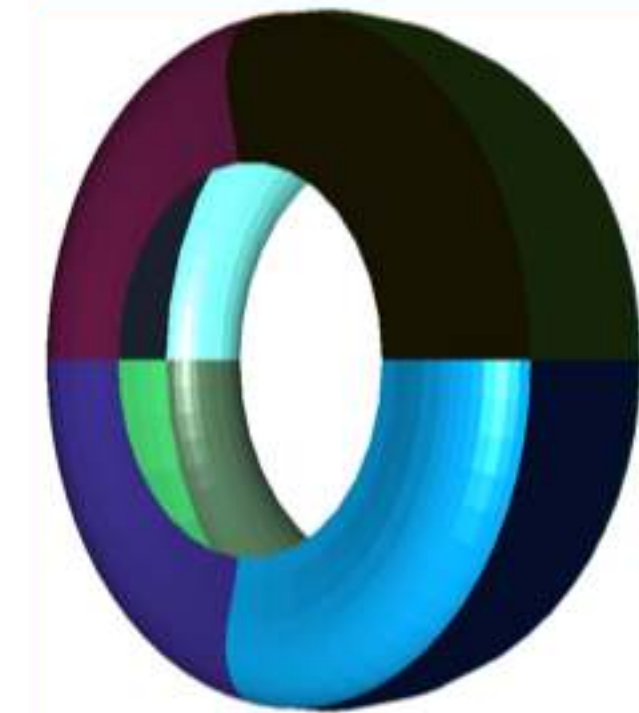


- Idea for a solution: *boundary coherence*  
= patches with common boundaries  
should be oriented consistently
- This is fairly straight-forward to  
implement, provided we have *complete  
neighborhood information* (topology)
  - And assuming the mesh is closed



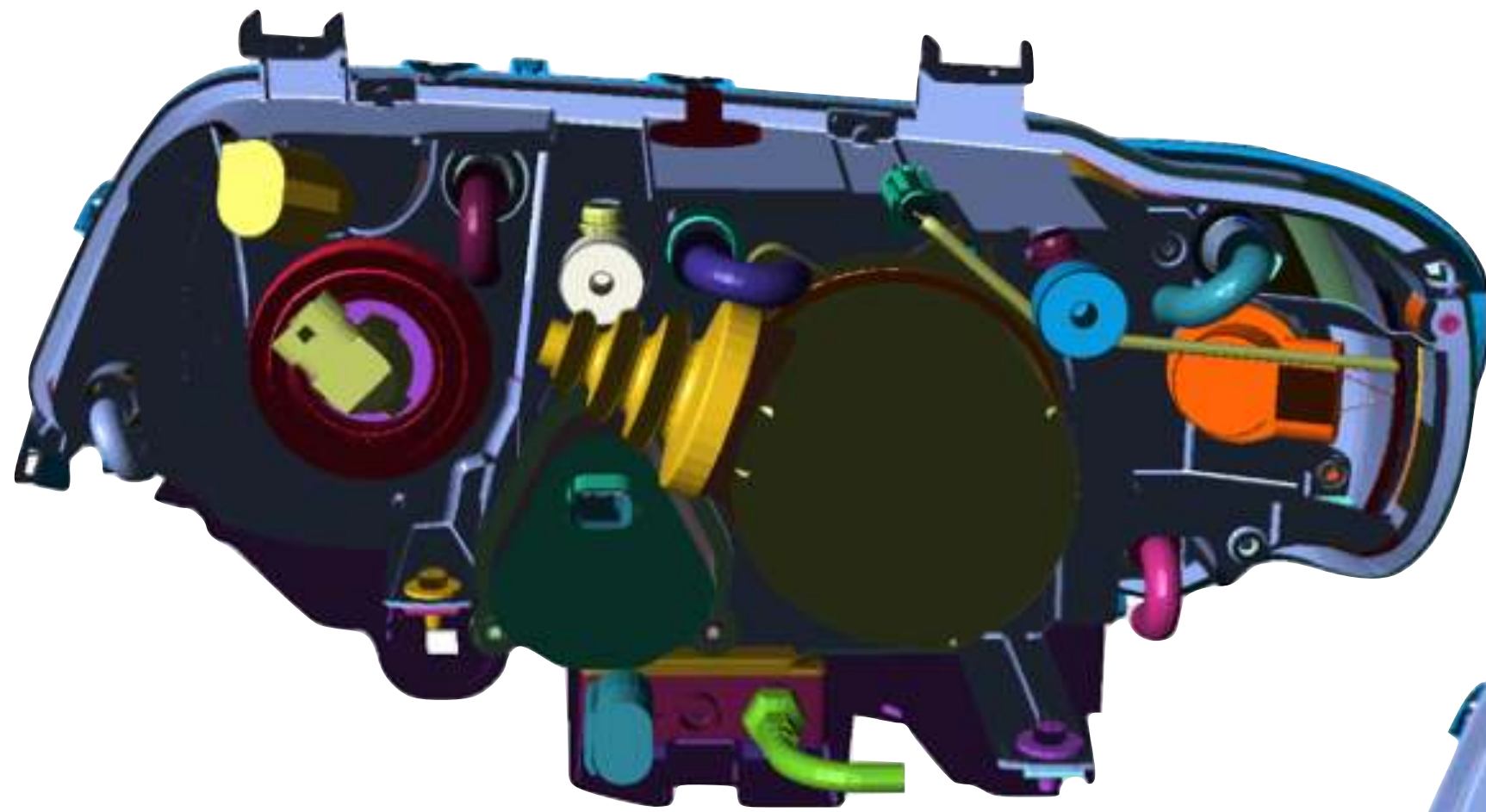
# General Procedure

1. Detect edges incident to only 1 polygon (boundary edges), or incident to more than 2 polygons (non-manifold edges)
2. Partition mesh into 2-manifold patches
3. Orient normals consistently *within* each patch (propagate consistent normal direction from one polygon to the next throughout a patch using BFS)
4. Determine patch-patch boundaries close to each other (which are "meant" to be connected)
5. Propagate normal orientations across those boundaries, too

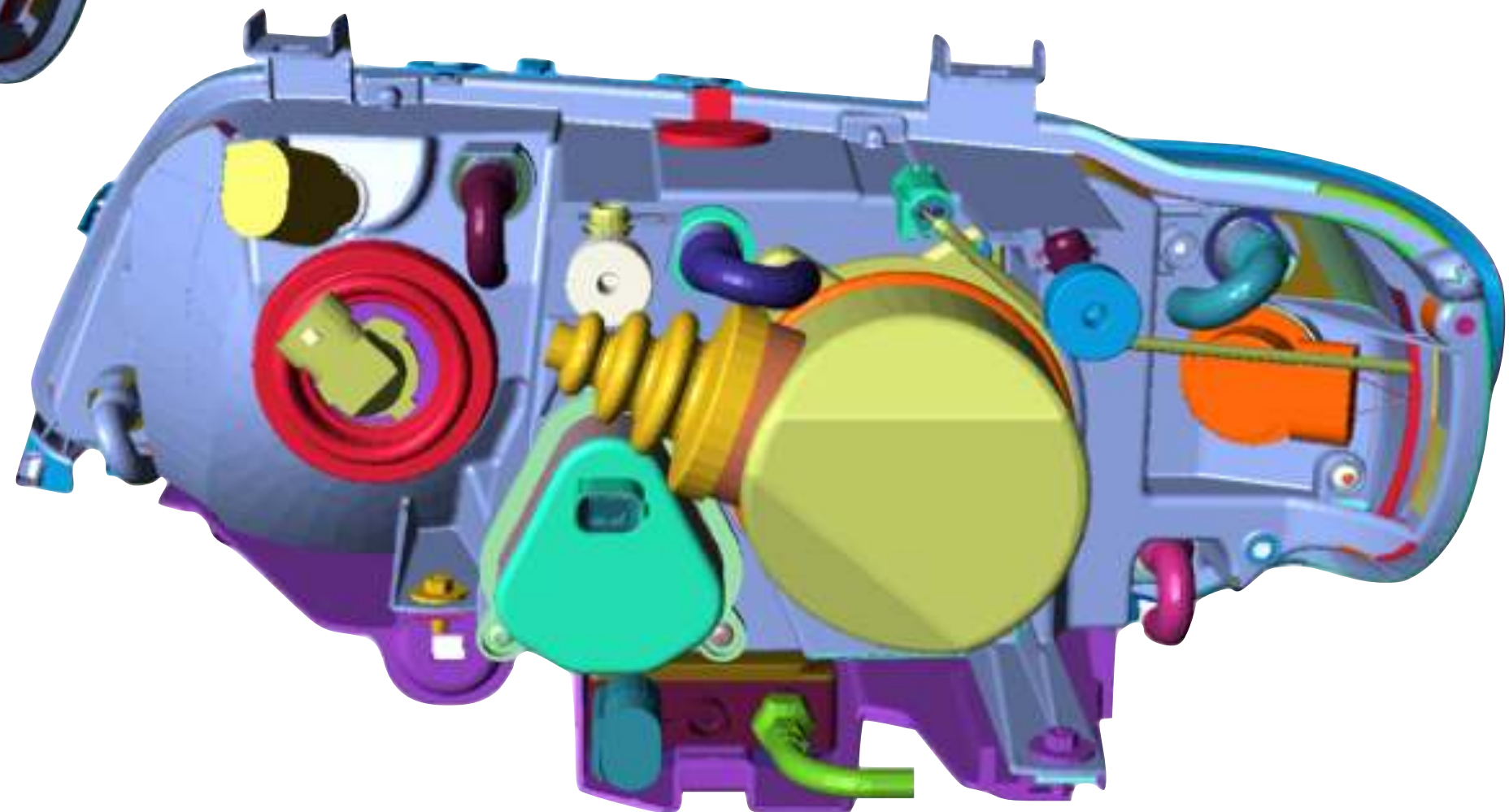




# Results



Before

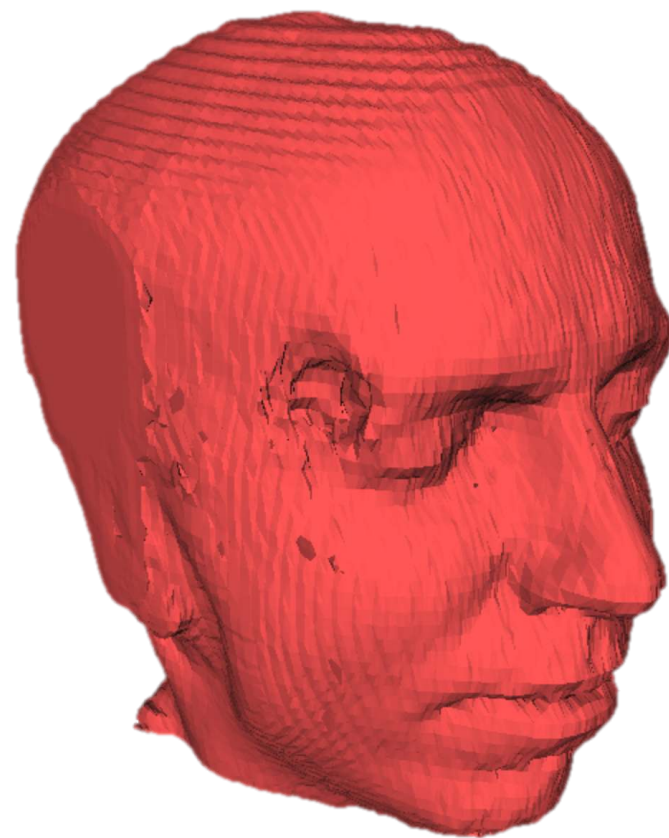


After



# Mesh Smoothing

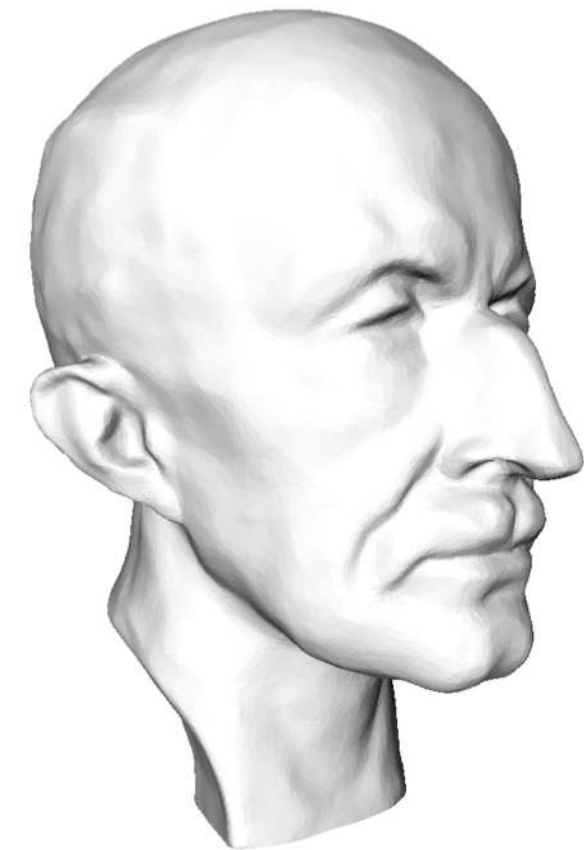
- Frequent problem: meshes are noisy (e.g., from marching cubes, or point cloud reconstruction)



Typical output of  
marching cubes



Output from laser  
scanner after meshing



Desired,  
smoothed mesh

- Idea: "convolve" mesh with a filter (kernel), like Gaussian filter for images

# Digression/Recap: Image Smoothing (Blurring)

- Simple, linear filtering by convolution:
  - $I = I(x, y)$  = input image,  $J = J(x, y)$  = output image

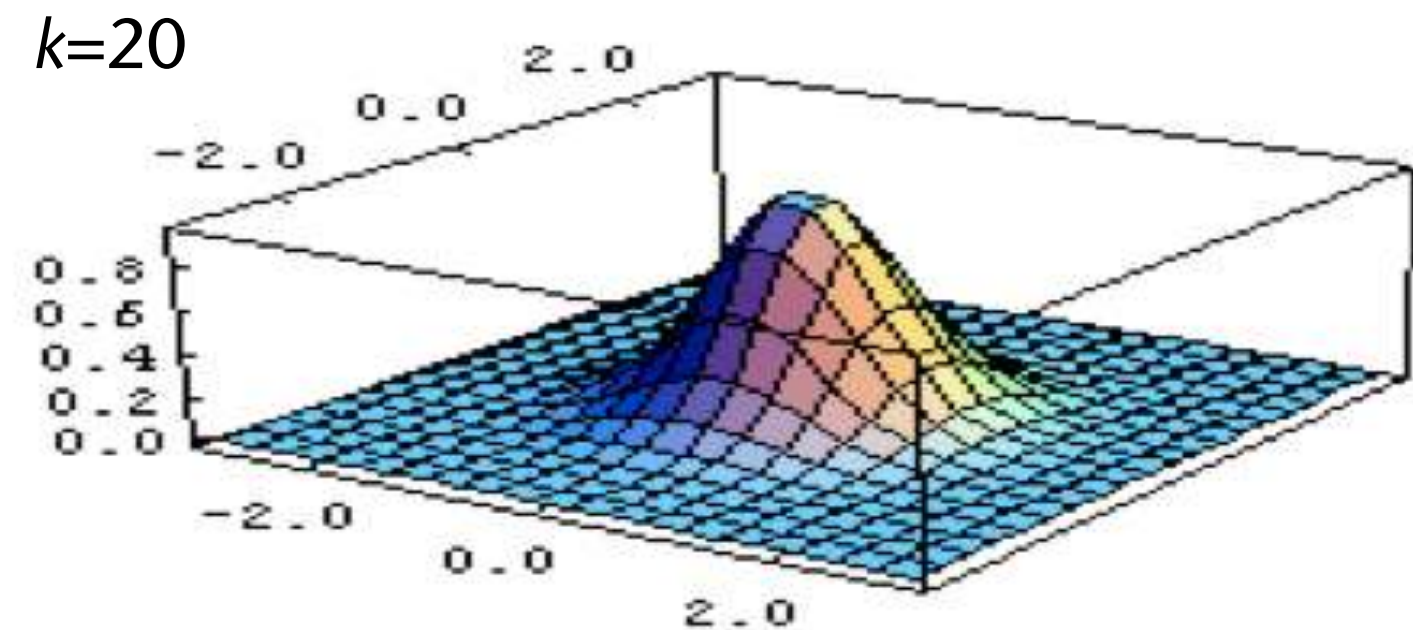
$$J(x, y) = \sum_{\substack{i=-k, \dots, +k \\ j=-k, \dots, +k}} I(x + i, y + j) H(i, j)$$

- $H$  is called a **kernel**,  $k$  = kernel width
- Sequential algorithm to construct  $J$ :
  - **Slide** a  $k \times k$  **window** across  $I$
  - At every pixel of  $I$ , compute weighted average of  $I$  inside window, weighted by  $H$

# Examples

- Gaussian kernel

$$H = \frac{1}{16} \begin{matrix} & k=3 \\ \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \end{matrix}$$

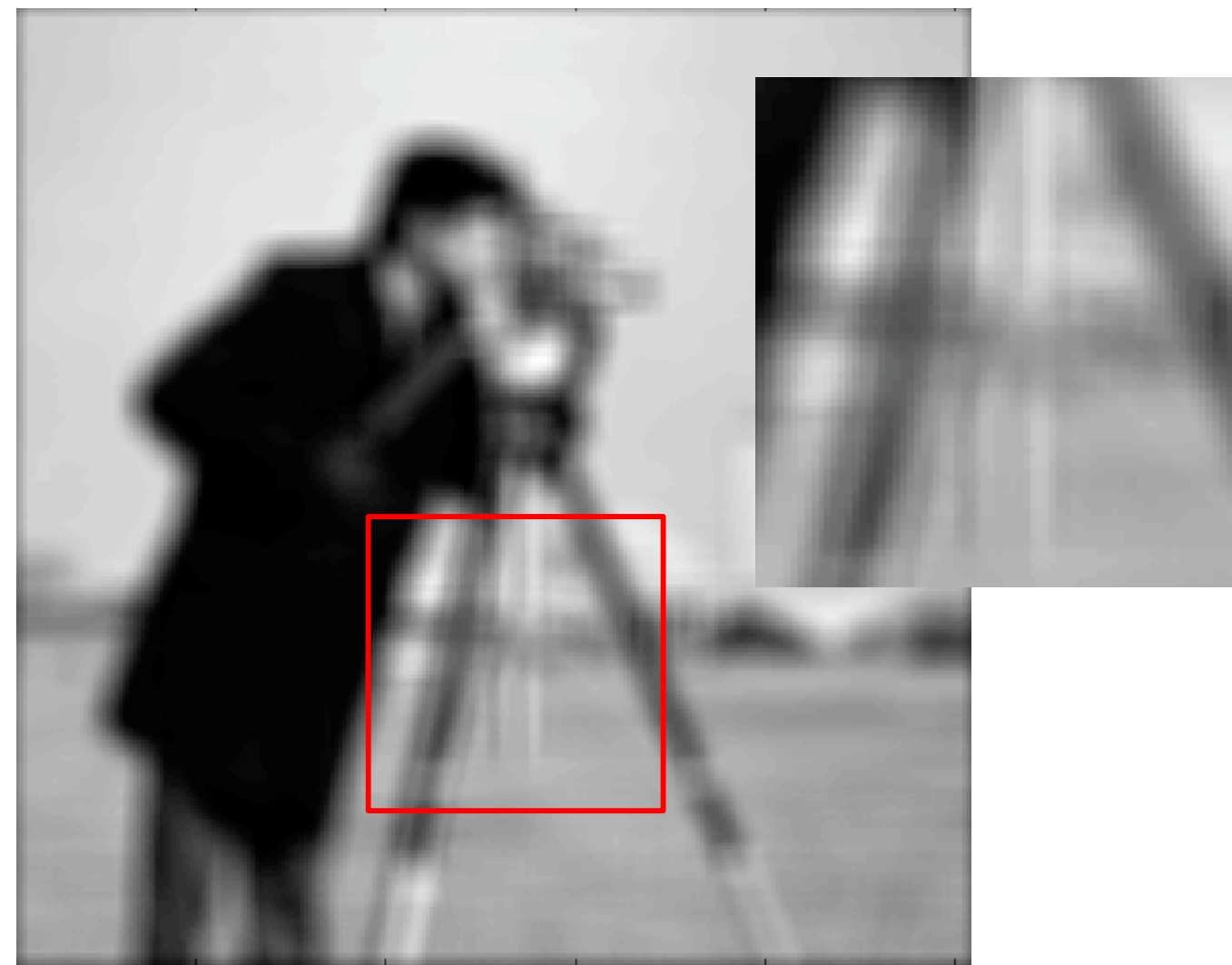


- Box filter (= simple averaging):

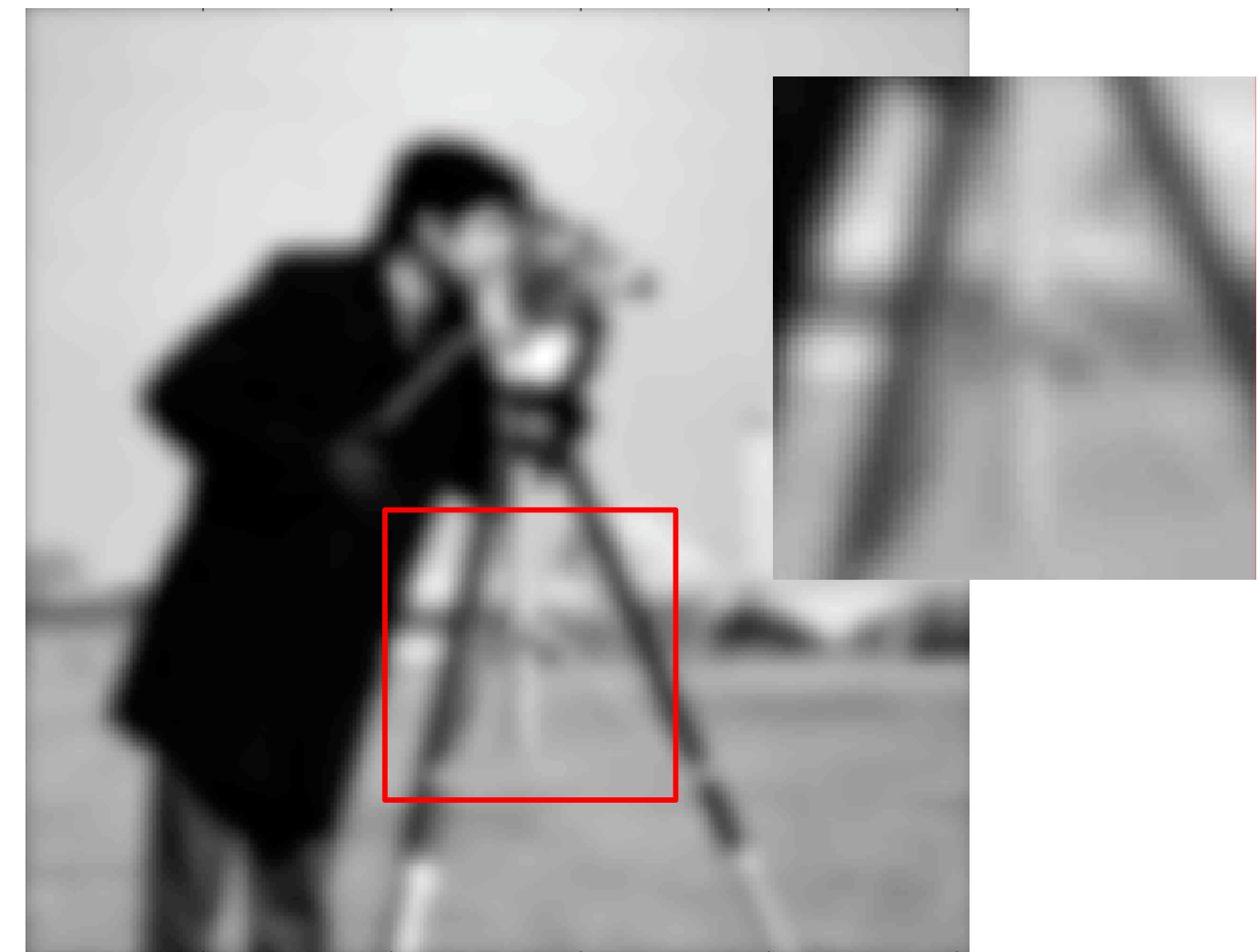
$$H = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Box

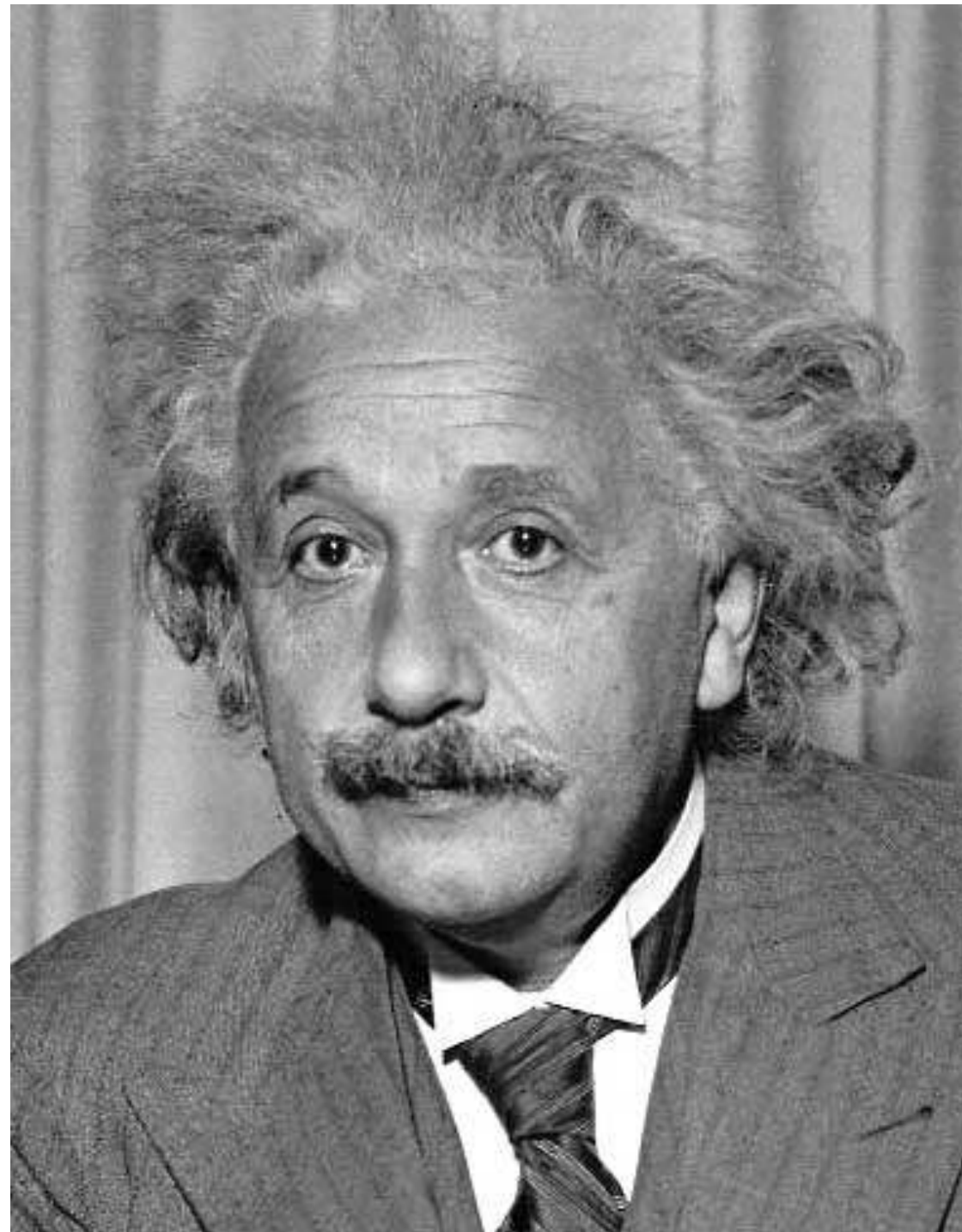


Gaussian

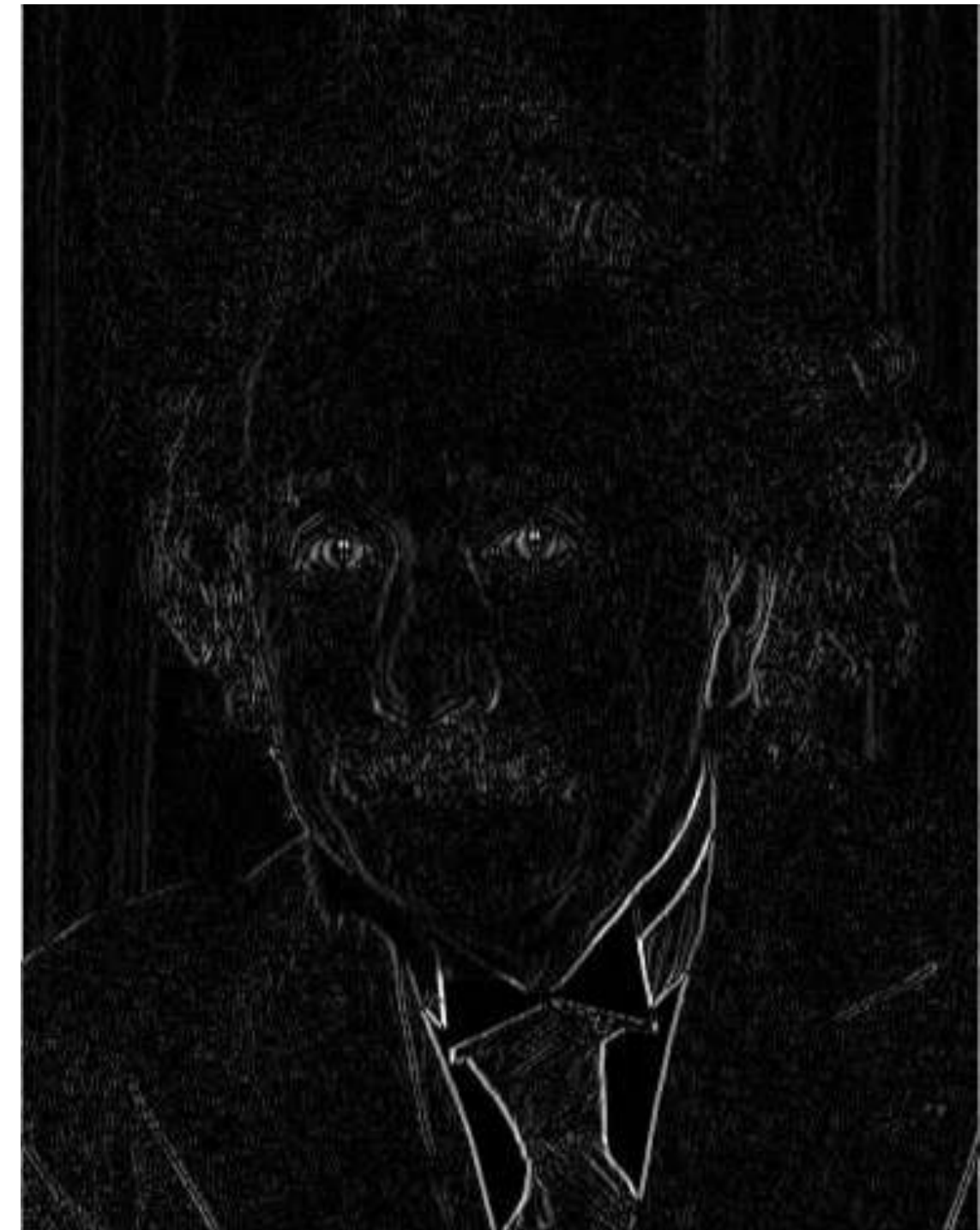




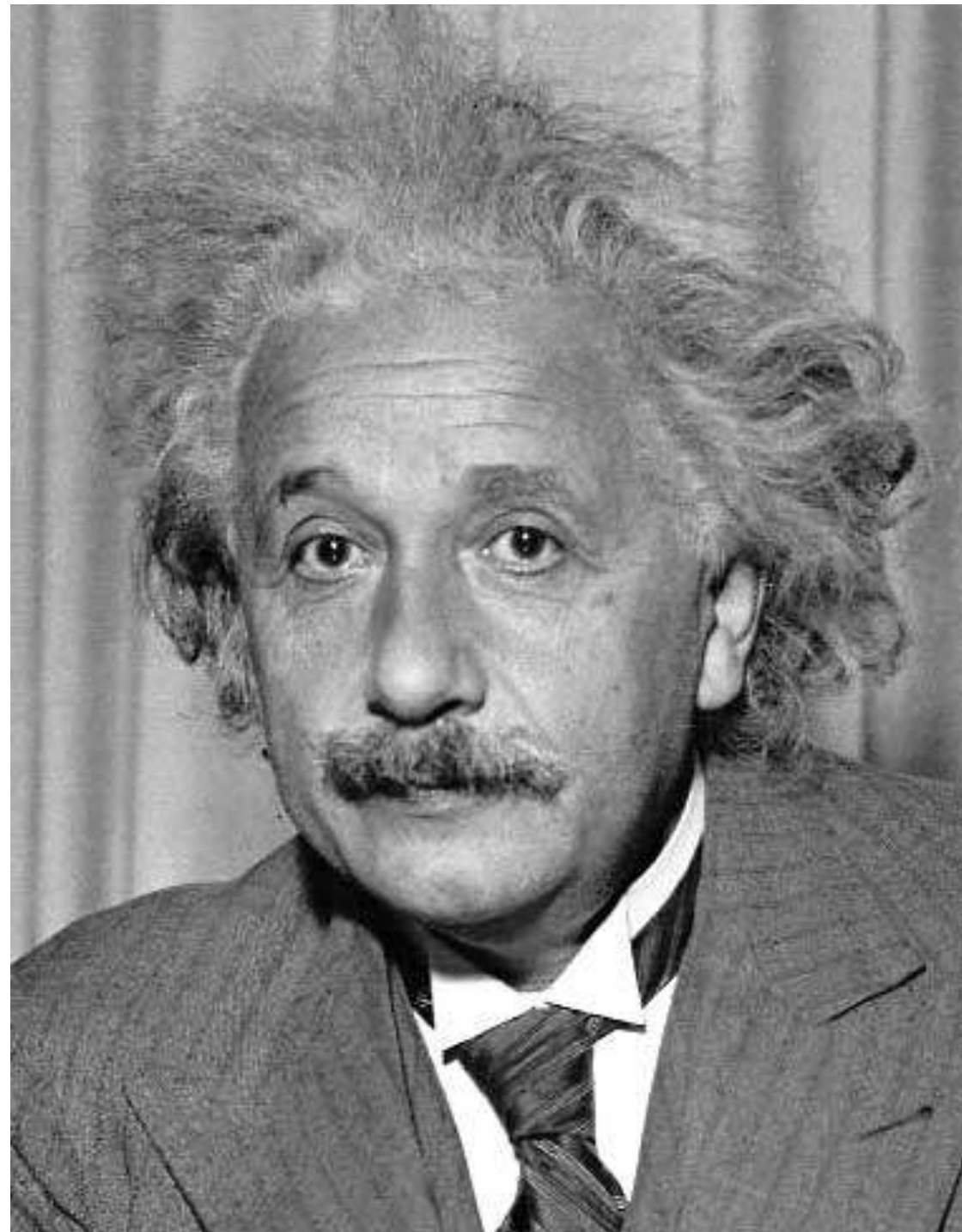
# Digression: Edge Extraction



Vertical  
Sobel  
Operator

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$


Vertical edges (absolute value)



Horizontal  
Sobel  
Operator

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$


Horizontal edges (absolute value)



- Problem: we can't simply apply the convolution idea to meshes!
- Why not?
- Meshes don't have a canonical , tensor-structure-like parameterization!
  - I.e., usually there is no parameterization like  $x$  and  $y$  in the plane
- Goal: filter *without* parameterization

# Laplacian Smoothing

- Idea:
  - Consider edges as springs
  - For a vertex  $\mathbf{v}_0$ , determine its position of *least* energy within its 1-ring

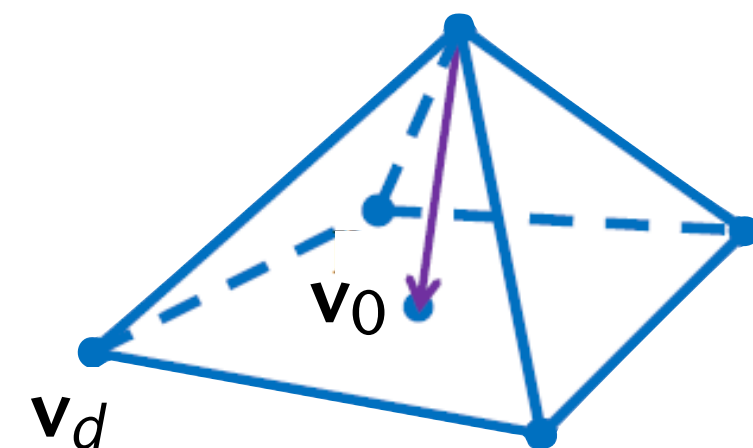
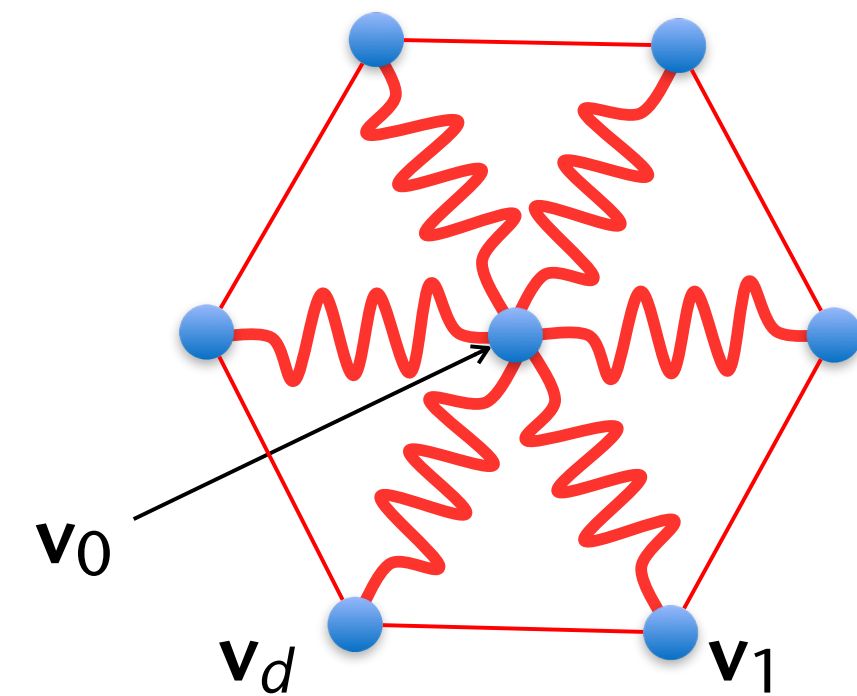
- Energy of  $\mathbf{v}_0$ : 
$$E = \frac{1}{2} \sum_{i=1}^d \|\mathbf{v}_i - \mathbf{v}_0\|^2$$

- Necessary condition for minimum: derivative equals zero

$$\frac{dE}{d\mathbf{v}_0} = \sum_{i=1}^d (\mathbf{v}_i - \mathbf{v}_0) = 0$$

- Iterative procedure: 
$$\mathbf{v}'_0 = \frac{1}{d} \sum_{i=1}^d \mathbf{v}_i$$

Sometimes a.k.a.  
"umbrella operator"





- Generalization: introduce "influence" of adjacent vertices and "speed"

$$\Delta \mathbf{v}_0 = \sum_{i=1}^k w_i (\mathbf{v}_i - \mathbf{v}_0), \quad \text{with } \sum w_i = 1, w_i \geq 0$$

$$\mathbf{v}'_0 = \mathbf{v}_0 + \lambda \Delta \mathbf{v}_0$$

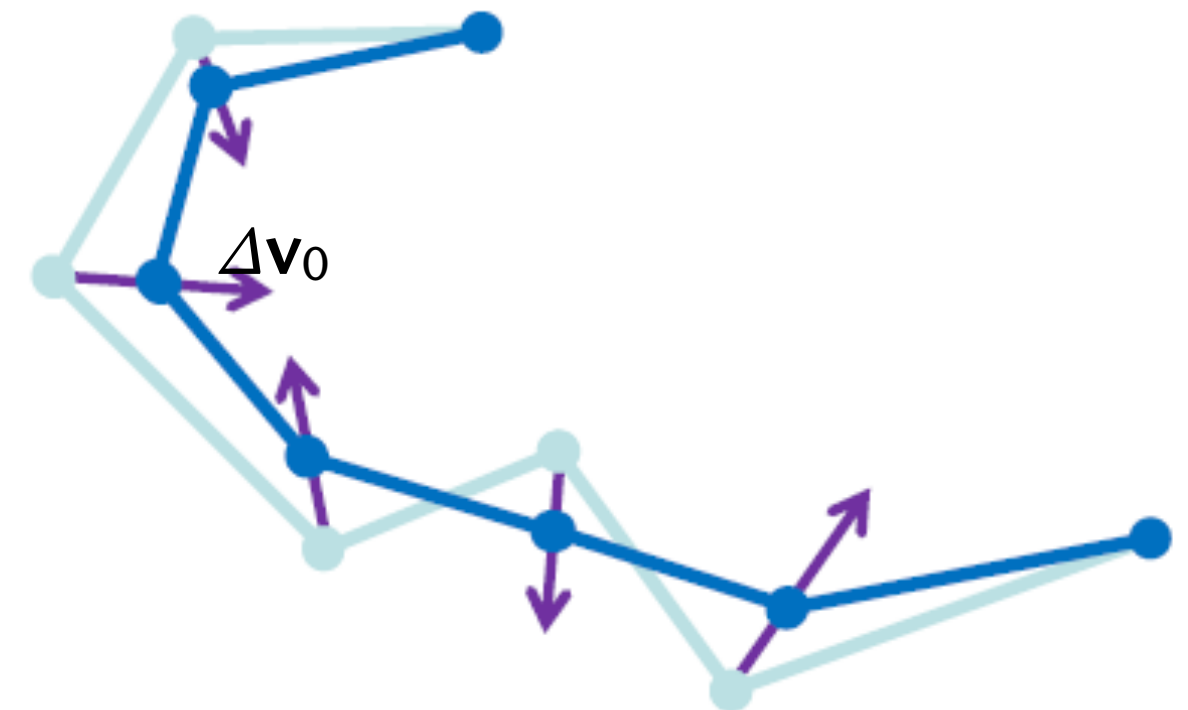
- Simplest form of the weights:

$$\Delta \mathbf{v}_0 = \frac{1}{d} \sum_{i=1}^d (\mathbf{v}_i - \mathbf{v}_0)$$

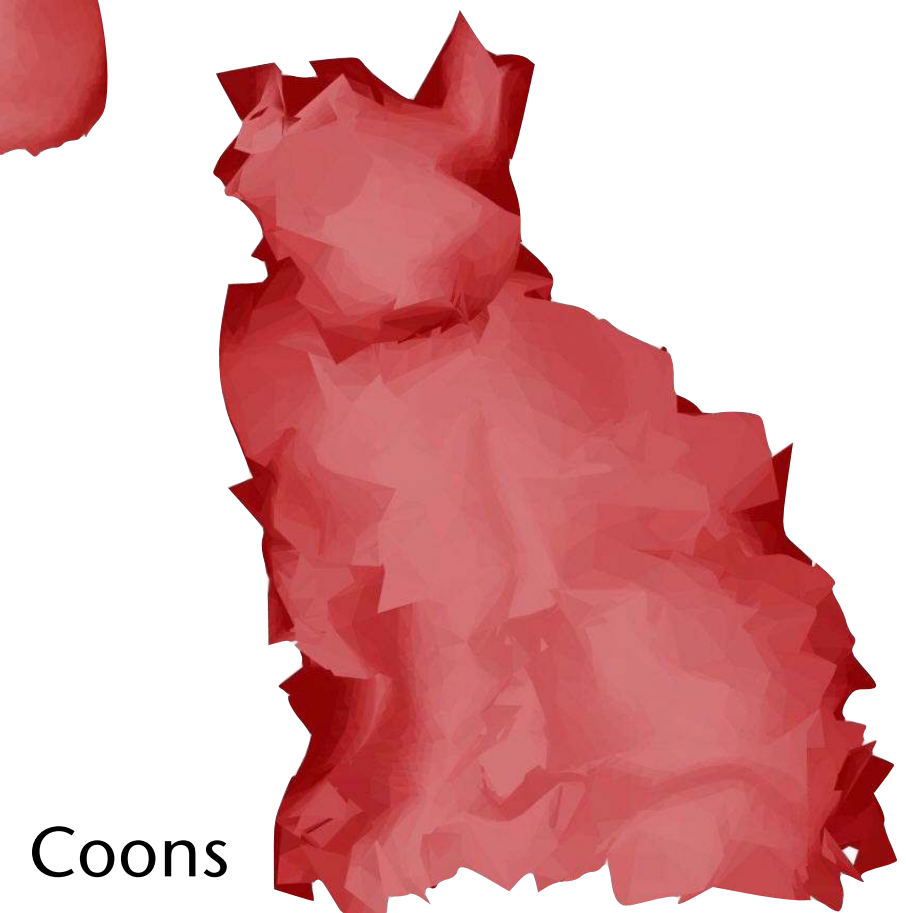
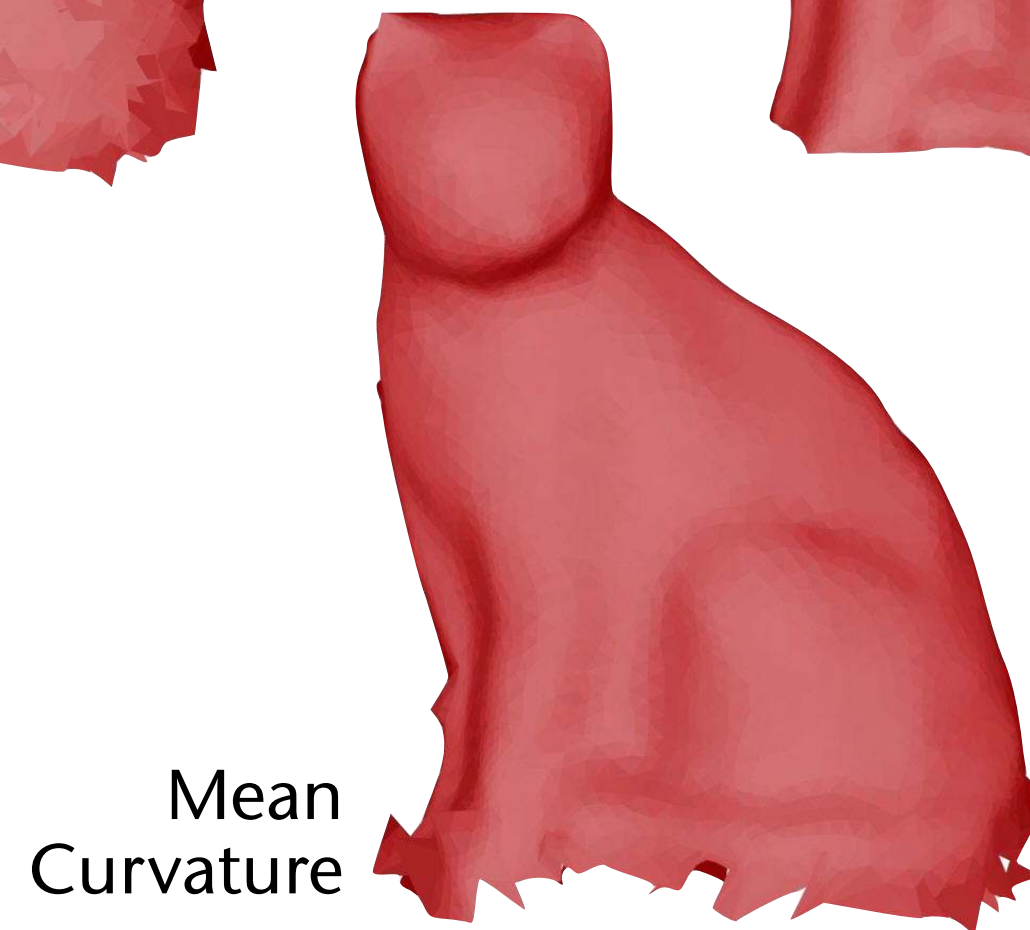
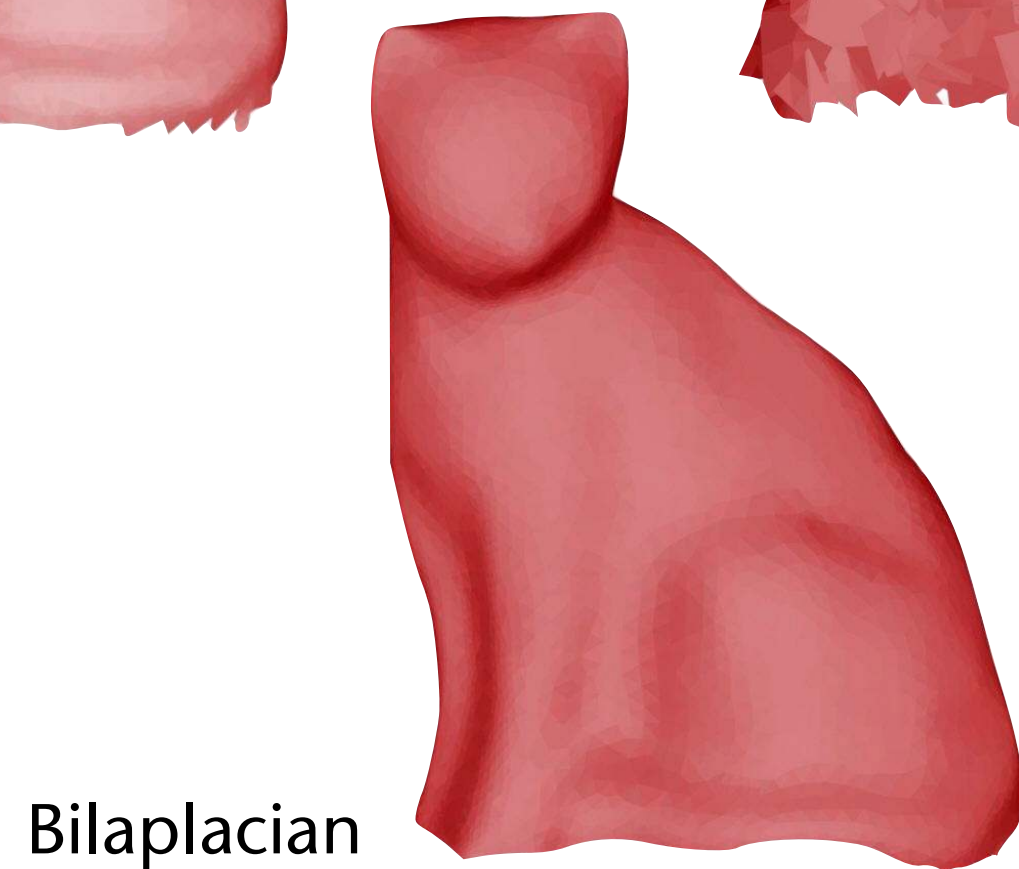
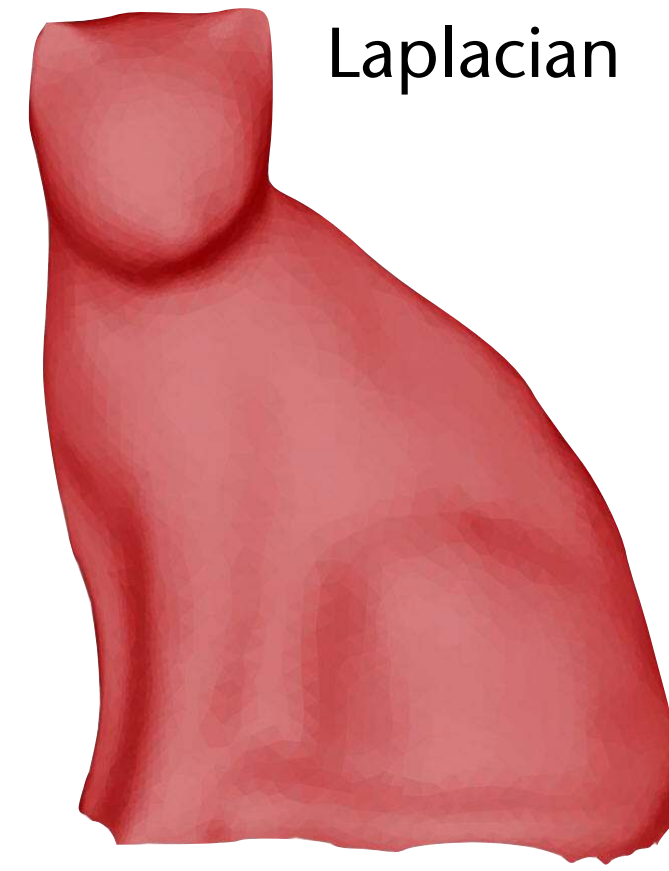
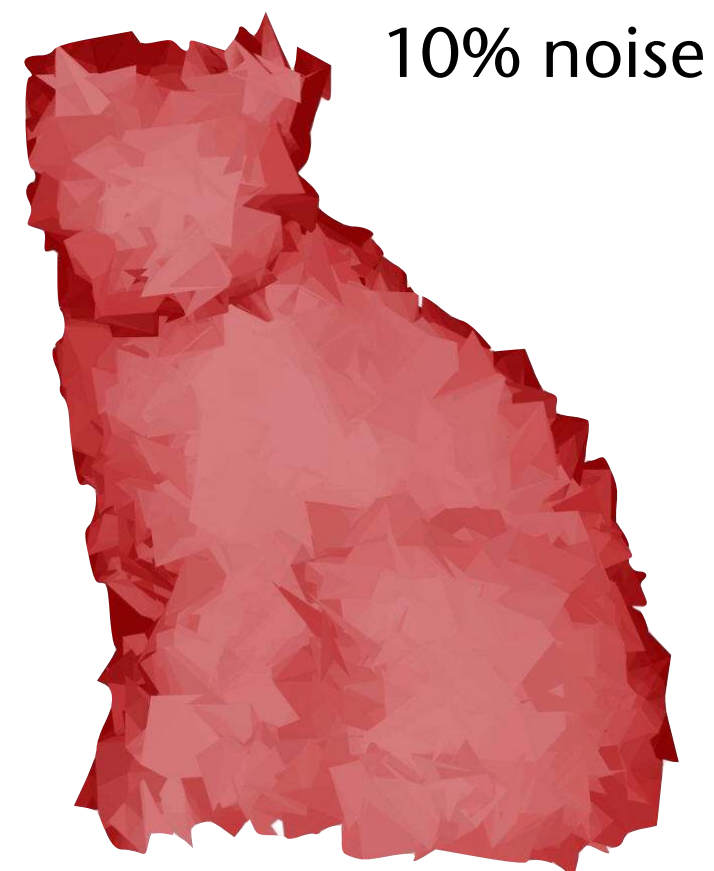
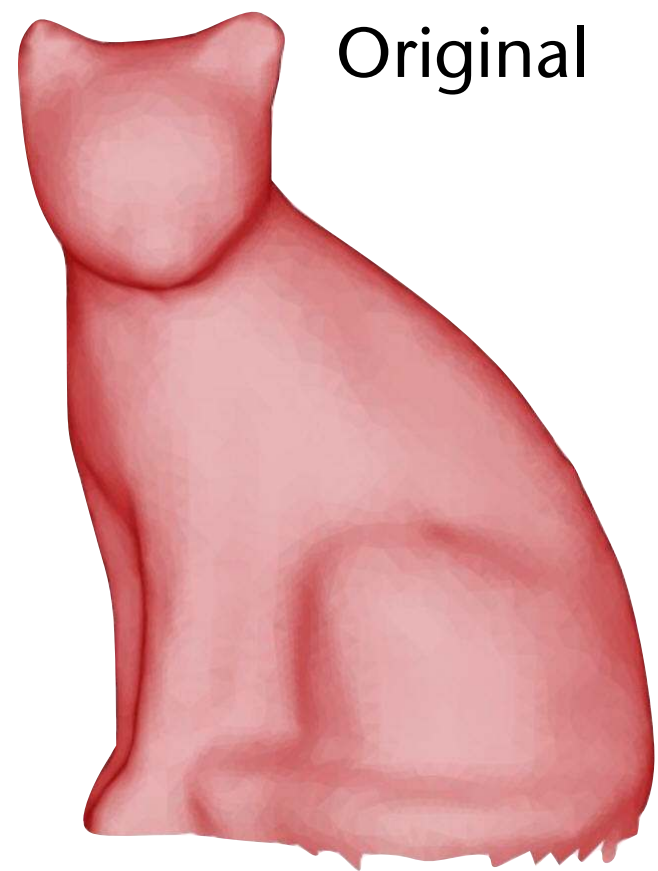
where  $d = \text{degree of } \mathbf{v}_0 = \text{number of neighbors}$

- Better weights are  $w_i = \frac{1}{\|\mathbf{v}_i - \mathbf{v}_0\|}$  or  $w_i = e^{-\|\mathbf{v}_i - \mathbf{v}_0\|^2}$  ("better" by experiment)

(see chapter "Object Representations" for more)

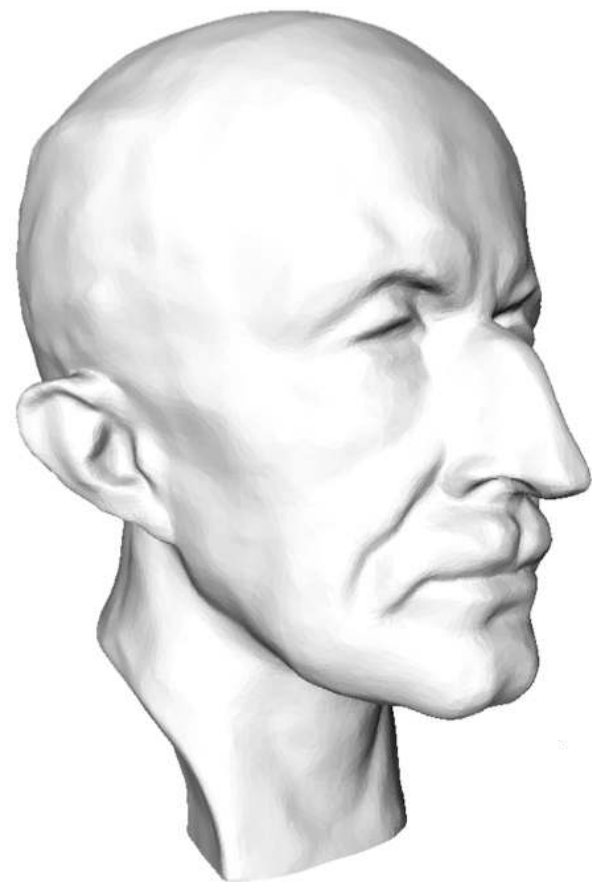


# Comparison with Other Smoothing Operators (not presented here)





# Problem: Laplace-Smoothing Causes Shrinking



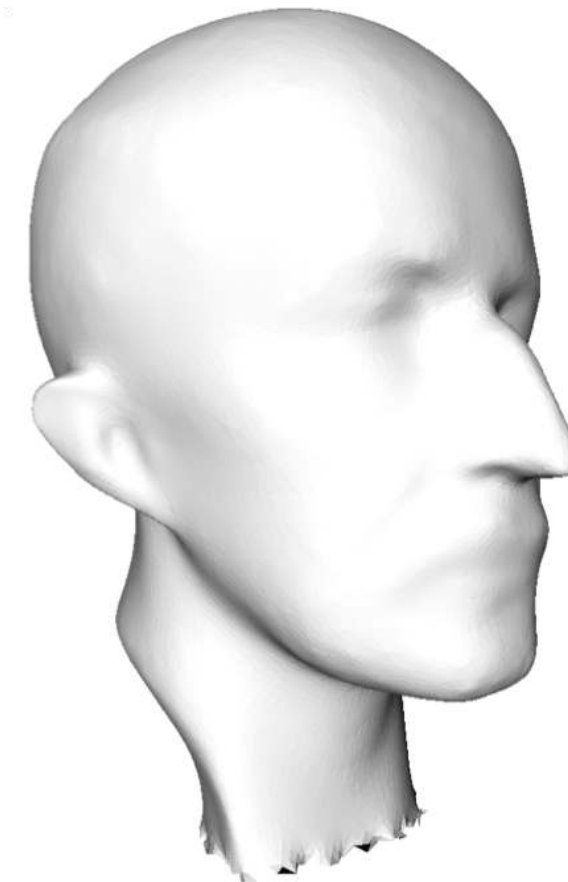
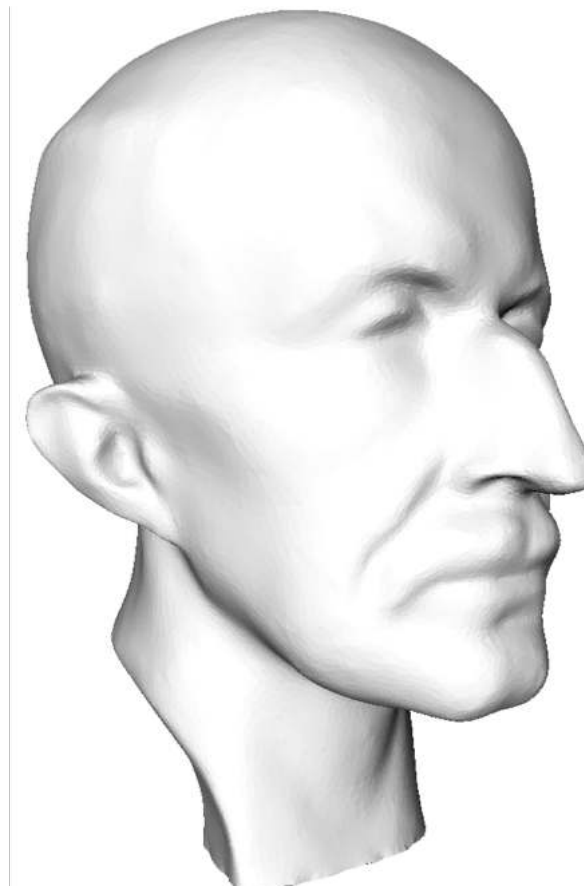
Original

With noise



After 4 iterations

After 10



After 80

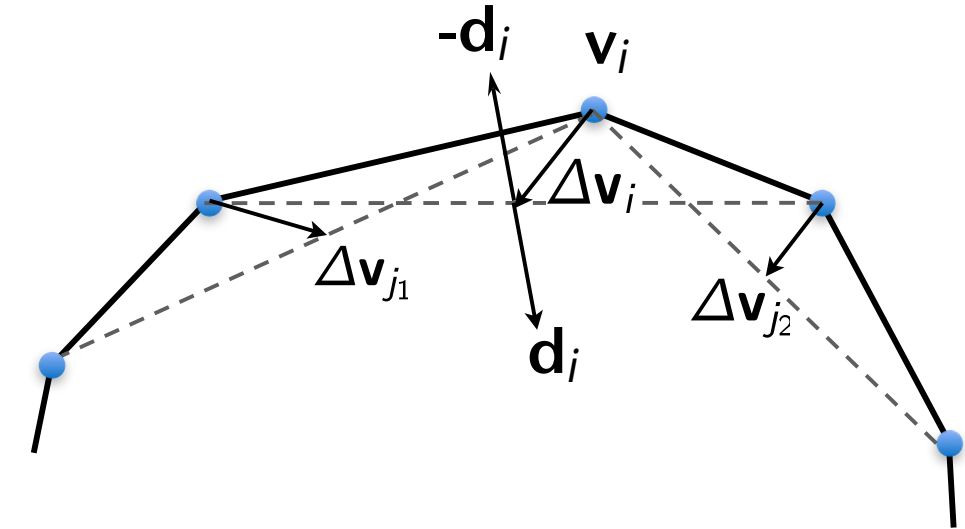
After 400



# A Simple Extension to Prevent Shrinking

- Like before, for every  $\mathbf{v}_i$  compute

$$\Delta \mathbf{v}_i = \frac{1}{d} \sum_{j \in \mathcal{N}(i)} (\mathbf{v}_j - \mathbf{v}_i)$$



- Average all neighboring  $\Delta$ 's (*including* the own  $\Delta$ ):

$$\mathbf{d}_i = \frac{1}{d + 1} \sum_{j \in \mathcal{N}(i) \cup i} \Delta \mathbf{v}_j$$

- Push the new vertex towards the 1-ring equilibrium *and outwards* away from the local direction of contraction ( $\mathbf{d}_i$ ):

$$\mathbf{v}'_i = \mathbf{v}_i + \lambda (\alpha \Delta \mathbf{v}_i - (1 - \alpha) \mathbf{d}_i)$$



# Comparison

Laplacian  
smoothing



Smoothing  
with pushback



# Global Laplacian Smoothing

- Given: mesh  $M = (V, E, F)$ ,  $V = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ ,  $\mathbf{v}_i = (x_i, y_i, z_i)$
- Sought: mesh  $M'$  with vertices  $\mathbf{v}_i'$  such that
  - $M'$  is smoother than  $M$ , and
  - $M'$  approximates  $M$
- If  $M'$  was perfectly smooth (i.e., a plane), we could find weights s.t.

$$\forall i : \sum_{j \in \mathcal{N}(\mathbf{v}_i')} w_{ij} (\mathbf{v}_j' - \mathbf{v}_i') = 0 \quad (1)$$

- This can be written as 3 systems of linear equations, one for x coords, one for y coords, one for z
  - In the following, we will deal with the x coords – y and z work similarly

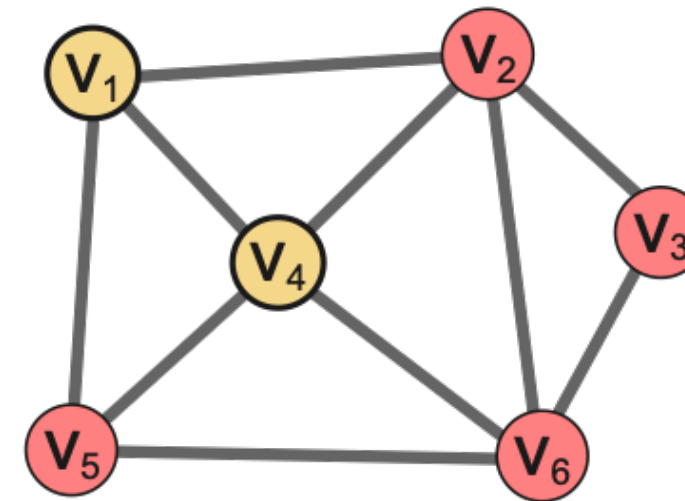
- Consider the  $x$  coords; write (1) as  $\mathbf{L} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = 0$

where  $\mathbf{L}$  is a  $n \times n$  matrix, with  $L_{ij} = \begin{cases} -1 & , i = j \\ w_{ij} & , (i, j) \in E \\ 0 & , \text{else} \end{cases}$

- Definition:  $\mathbf{L}$  is called the **Laplacian** of the mesh
  - In a sense,  $\mathbf{L}$  encodes the adjacency of the mesh
- Analogously, construct a system of equations of  $y$  and  $z$

- Example: for sake of simplicity, use  $w_{ij} = \frac{1}{d_i}$

$$\mathbf{L} = \begin{pmatrix} -1 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 1/4 & -1 & 1/4 & 1/4 & 0 & 1/4 \\ 0 & 1/2 & -1 & 0 & 0 & 1/2 \\ 1/4 & 1/4 & 0 & -1 & 1/4 & 1/4 \\ 1/3 & 0 & 0 & 1/3 & -1 & 1/3 \\ 0 & 1/4 & 1/4 & 1/4 & 1/4 & -1 \end{pmatrix}$$

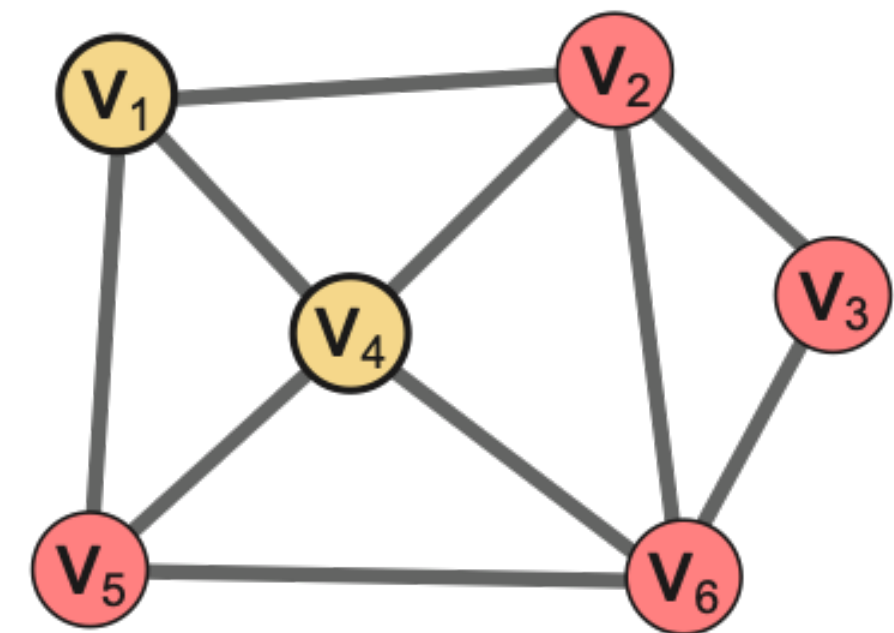


- Warning:  $\mathbf{L}$  has rank  $n-1$ ,  $n = \#$  vertices
- "Proof" by example: vector  $\mathbf{x} = (1, \dots, 1)^T$  is a solution to  $\mathbf{L}\mathbf{x} = 0$  (and for all  $\alpha$ ,  $\mathbf{L}(\alpha\mathbf{x}) = 0$ , too)
  - Check for yourself: ist that so?



- Solution: "anchor" one vertex, i.e., fix its position
- For instance, in our example, add condition  $\mathbf{v}'_1 = \mathbf{v}_1$  :

$$\begin{pmatrix} -1 & 1/3 & 0 & 1/3 & 1/3 & 0 \\ 1/4 & -1 & 1/4 & 1/4 & 0 & 1/4 \\ 0 & 1/2 & -1 & 0 & 0 & 1/2 \\ 1/4 & 1/4 & 0 & -1 & 1/4 & 1/4 \\ 1/3 & 0 & 0 & 1/3 & -1 & 1/3 \\ 0 & 1/4 & 1/4 & 1/4 & 1/4 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ \vdots \\ 0 \\ x_1 \end{pmatrix}$$



- This system now has a unique solution

- Avoiding shrinking: introduce another constraint requiring the barycenters of the new triangles be the same as the barycenters of the old ones

$$\forall (i, j, k) \in F : \frac{1}{3}(\mathbf{v}'_i + \mathbf{v}'_j + \mathbf{v}'_k) = \frac{1}{3}(\mathbf{v}_i + \mathbf{v}_j + \mathbf{v}_k) \quad (2)$$

- Write (1) and (2) as 
$$\begin{pmatrix} \mathbf{L} \\ \mathbf{B} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{b} \end{pmatrix} \quad (3)$$

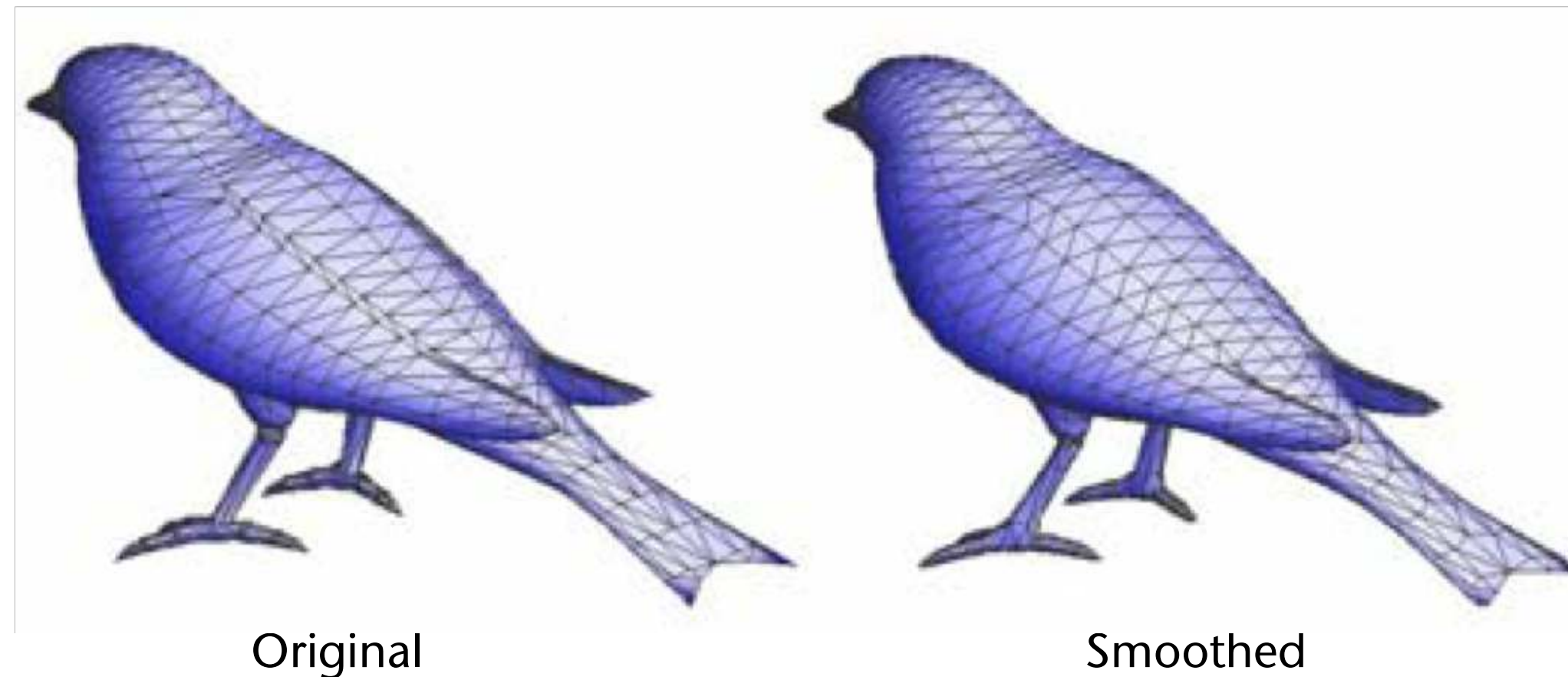
where  $\mathbf{B}$  is a  $m \times n$  matrix,  $m$  = number of triangles, and  $\mathbf{b}$  is a column vector with  $m$  entries, where the  $k$ -th row corresponds to triangle  $F_k = (i_1, i_2, i_3)$  and  $B_{ki} = \frac{1}{3}$ , for  $i = i_1, i_2, i_3$ , 0 elsewhere, and  $b_k = \frac{1}{3}(x_{i_1} + x_{i_2} + x_{i_3})$

- Solve (over-determined) system (3), which has the form  $\mathbf{Ax} = \mathbf{c}$  in the least squares sense:

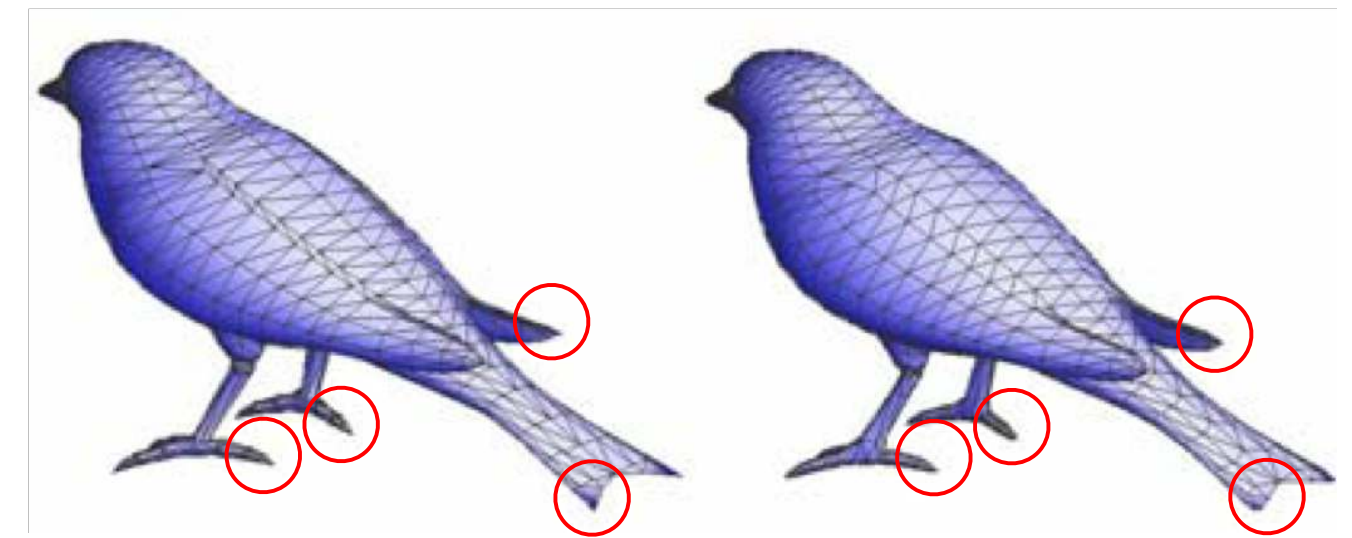
$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{c}$$

- In real life, use a sparse solver, e.g., TAUCS or OpenNL

- Results:



- Further requirement: certain points ("features") should be maintained
- Solution: introduce more constraints
  - Pick **feature points**  $\mathbf{v}_{i_1}, \dots, \mathbf{v}_{i_k}$



- Either by user, or by automatic salient point detectors

- Add constraint  $\mathbf{v}'_{i_l} = \mathbf{v}_{i_l}$ ,  $l = 1, \dots, k$  (4)

- Add equations (4) to system (3):

$$\begin{pmatrix} \mathbf{L} \\ \mathbf{B} \\ \mathbf{C} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{b} \\ \mathbf{c} \end{pmatrix}$$

where  $\mathbf{C}$  is a matrix containing

in every row  $l$  just one 1 at position  $i_l$ ,  $1 \leq l \leq k$ , and  $\mathbf{c} = (x_{i_1}, \dots, x_{i_k})$

- Again, we do this for x-, y-, and z-coordinates separately

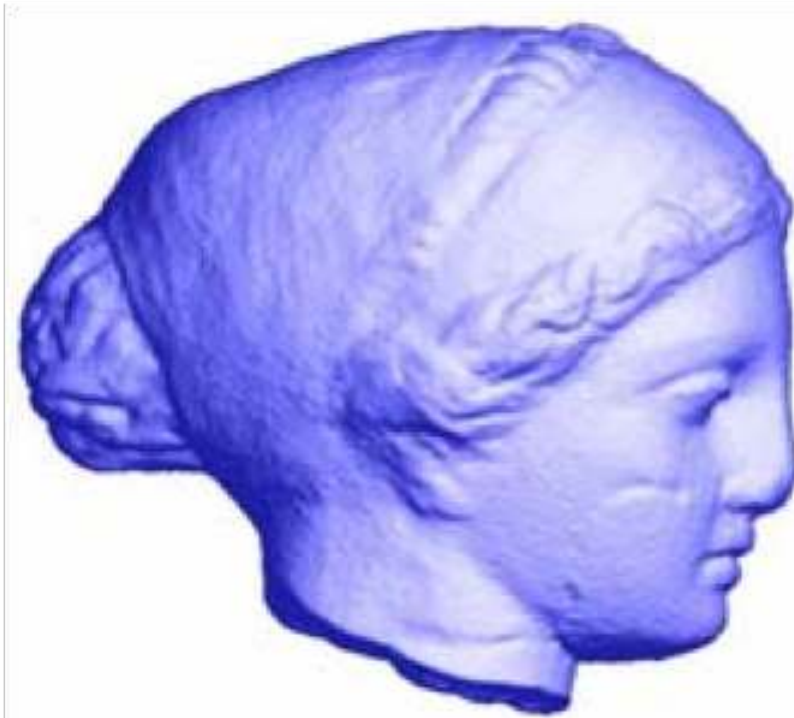


# Results

Noisy original



Smoothed



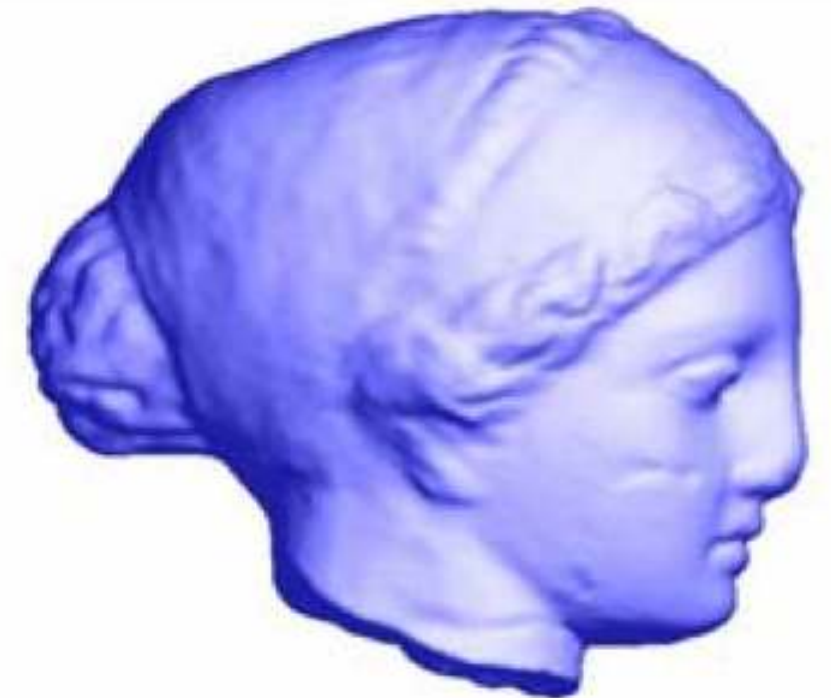
Noisy original



Laplacian smoothing



Bilateral smoothing



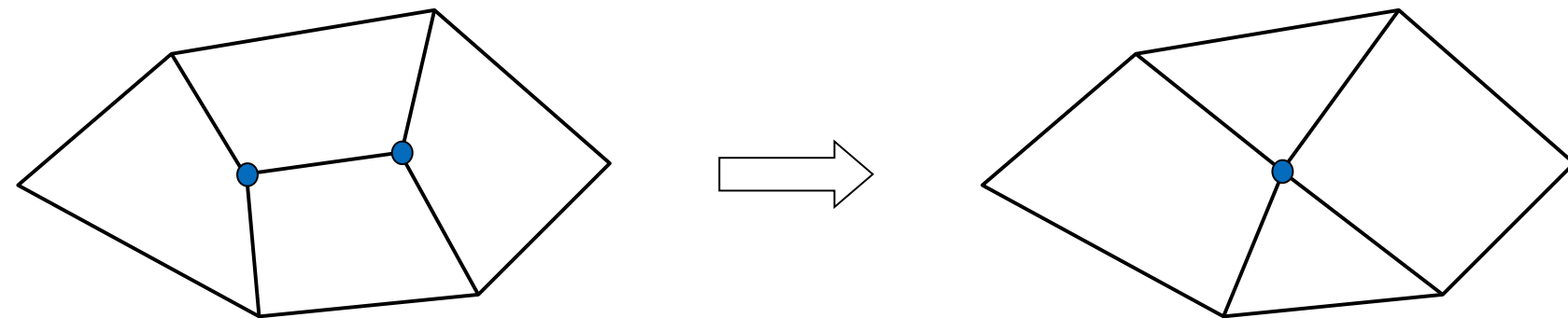
Global smoothing

# Mesh Simplification

- **Simplification:** Generate a coarse mesh from a fine (hi-res) mesh
  - While maintaining certain criteria (will not be discussed further here)

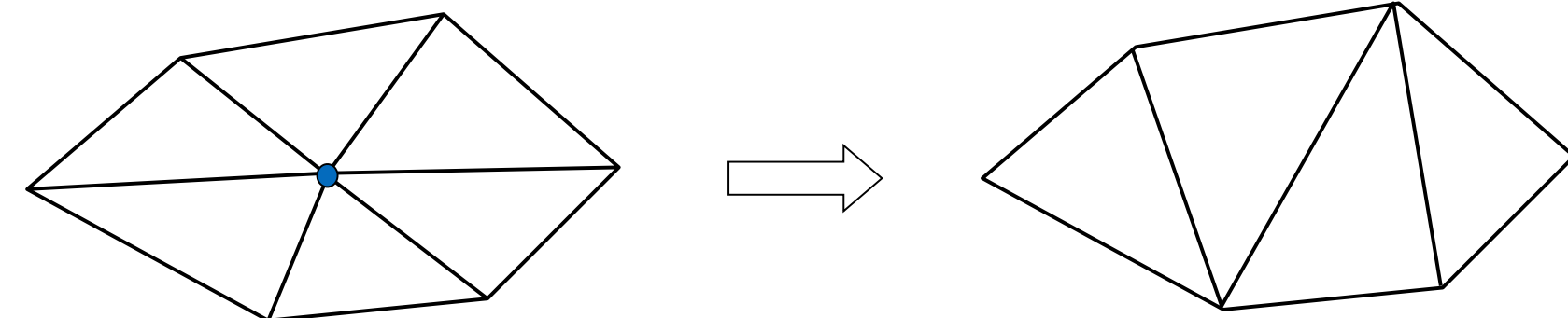
- Elementary operations:

- Edge collapse:



(More details in the course  
"Virtual Reality .." )

- Vertex removal:



- All edges incident to the vertex are needed



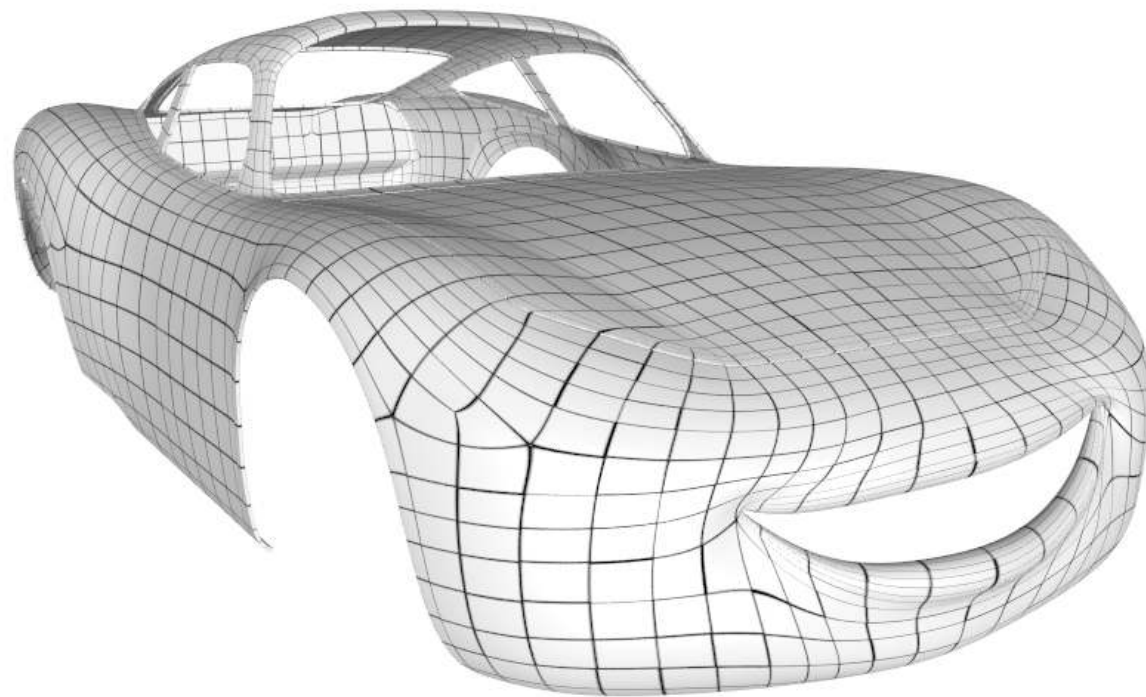
# Subdivision Surfaces: One of the First Movies



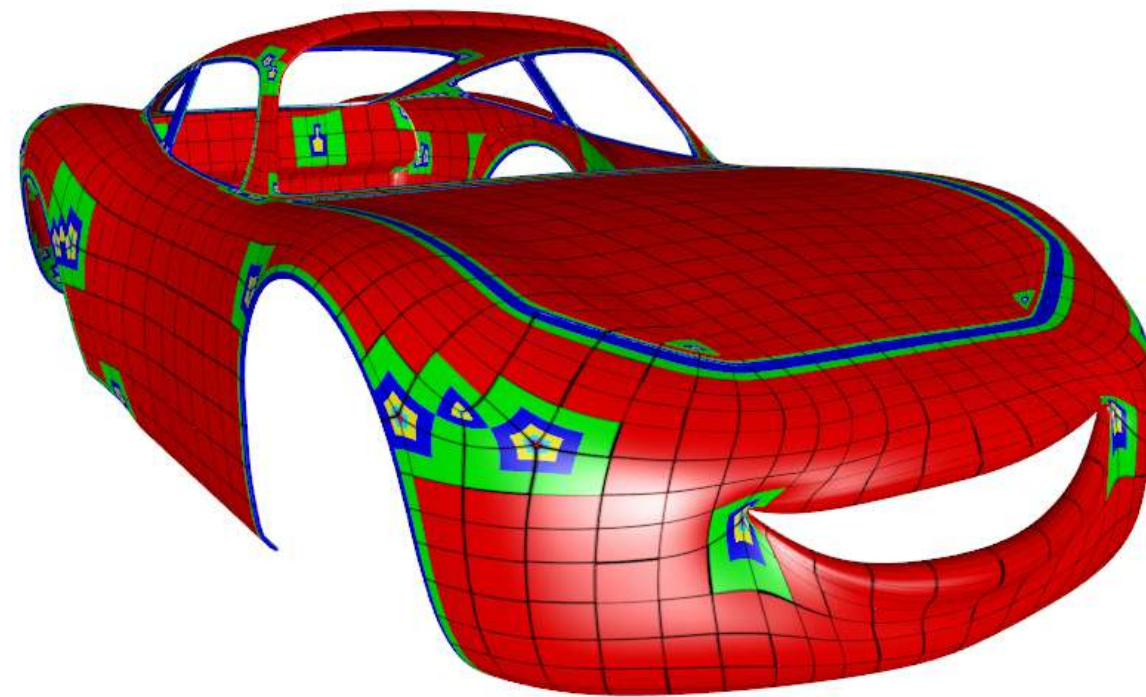
[Pixar: "Geri's Game"]

# Examples from Animation Films

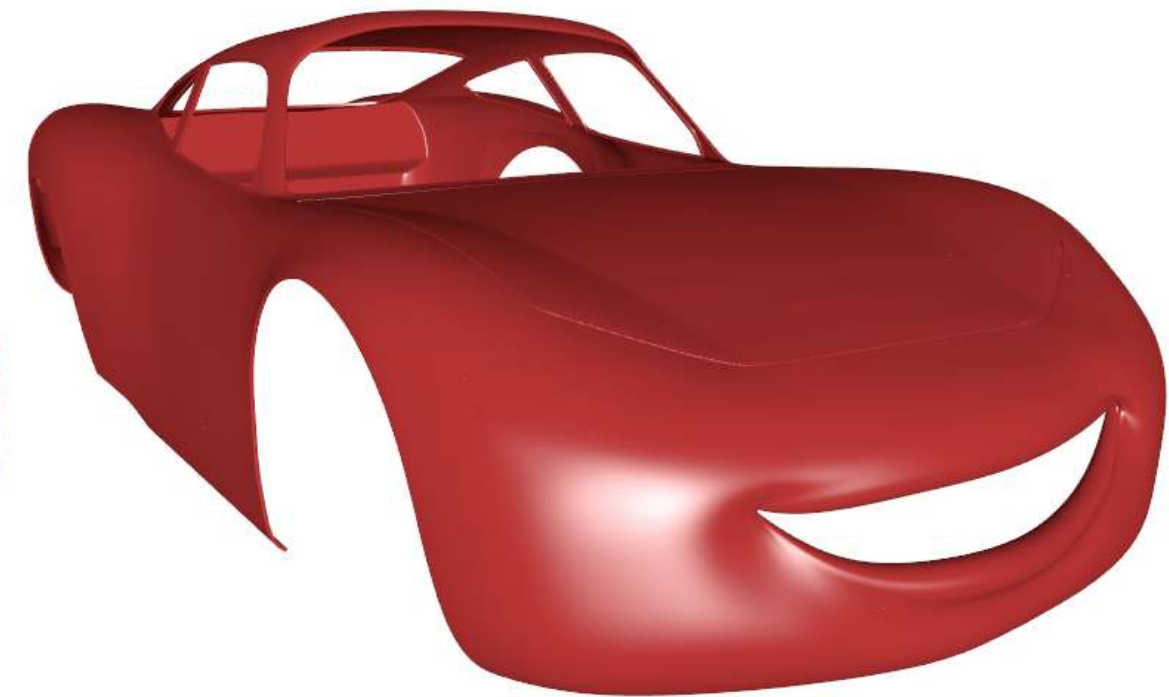
Input base mesh



Subdivision patch structure



Final model



[Nießner et al., 2012]



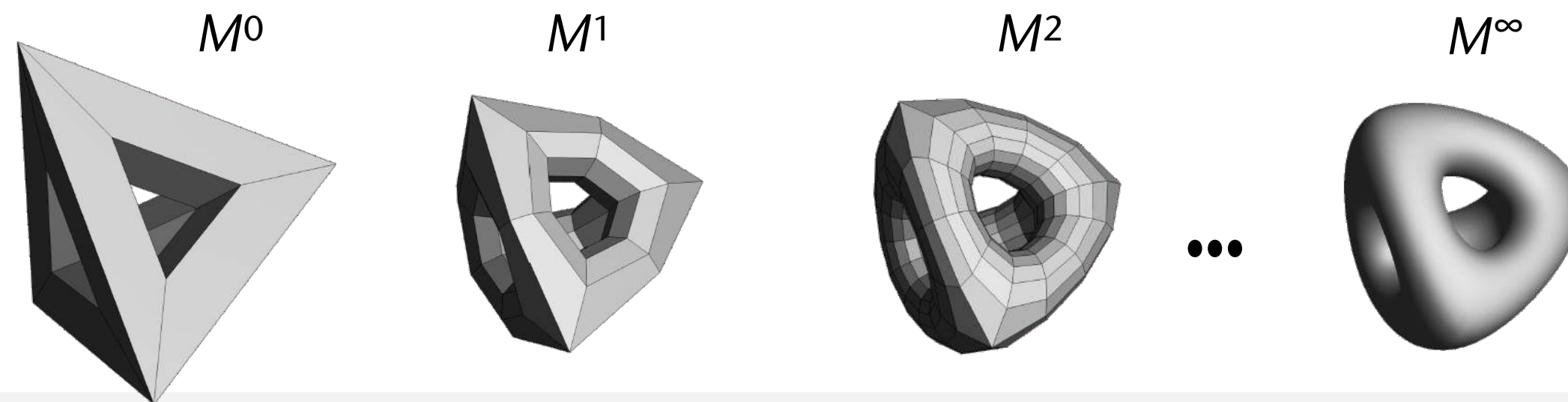
# Example from Games

- Used to create high-poly models that are then used to bake texture maps (normal map, specular map, etc.) for the low-poly in-game models



# Basic Idea of Subdivision

- Start with a (simple) mesh  $M^0$ , called **control mesh**
- In each iteration  $i$ :
  1. Refinement: subdivide edges and faces of  $M^i$ 
    - Some schemes split vertices ("dual" subdivision schemes)
  2. Weighted averaging: calculate new positions by averaging neighboring vertices
    - Results in a new mesh  $M^{i+1}$  (**generation  $i+1$** )
- Ideally, the mesh converges to a **limit surface**



# The Catmull-Clark Subdivision Scheme

- Let  $p_i$  = vertices of the "old" mesh generation
- For each face, calculate a new "face point"

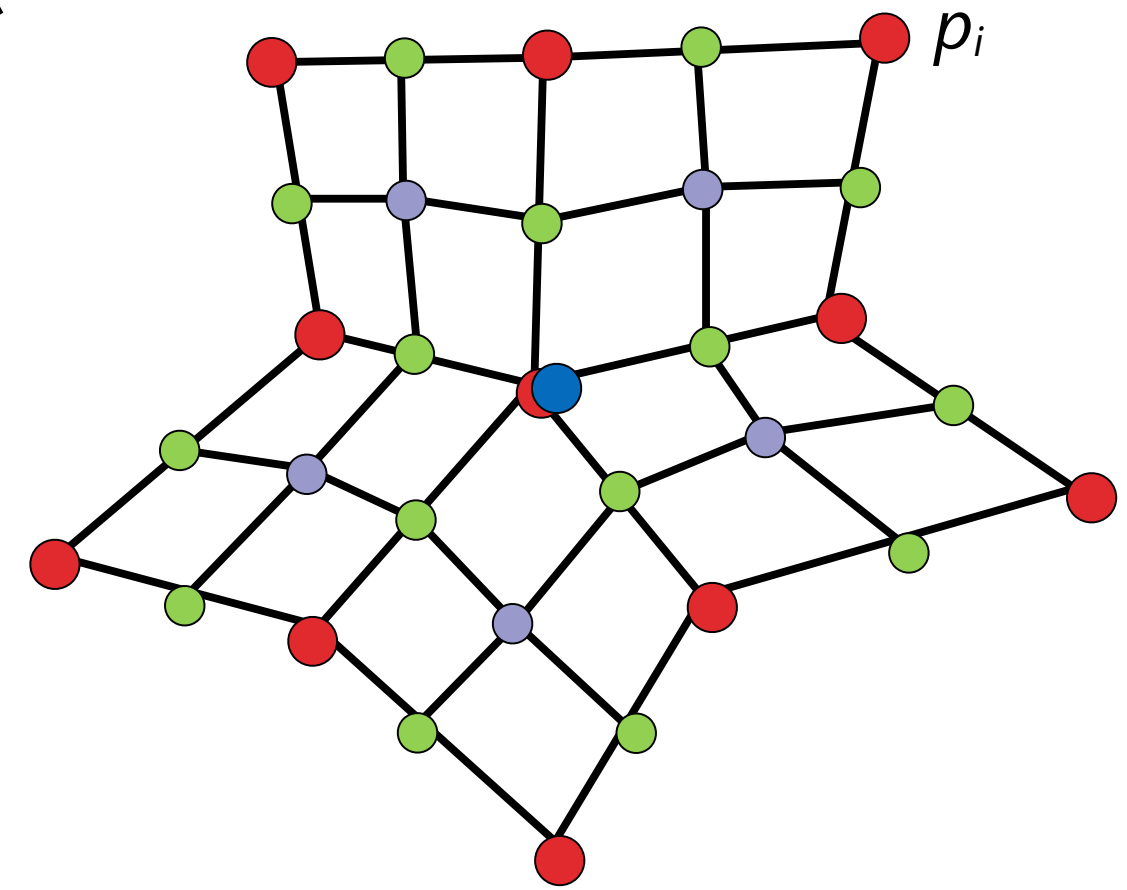
$$f = \frac{1}{k} \sum_{i=1}^k p_i$$

- For each edge, calculate a new "edge point":

$$e = \frac{1}{4}(p_1 + p_2 + f_1 + f_2)$$

- For each old vertex,  $p$ , calculate a new "vertex point":

$$p' = \frac{1}{m}q + \frac{2}{m}r + \frac{m-3}{m}p$$



$k$  = # old vertices incident to the face (valence)

$p_1, p_2$  = old vertices incident to the edge

$f_1, f_2$  = new face point of the faces incident to the edge

$m$  = # faces/edges incident to old vertex (valence)

$q$  = average of incident face points

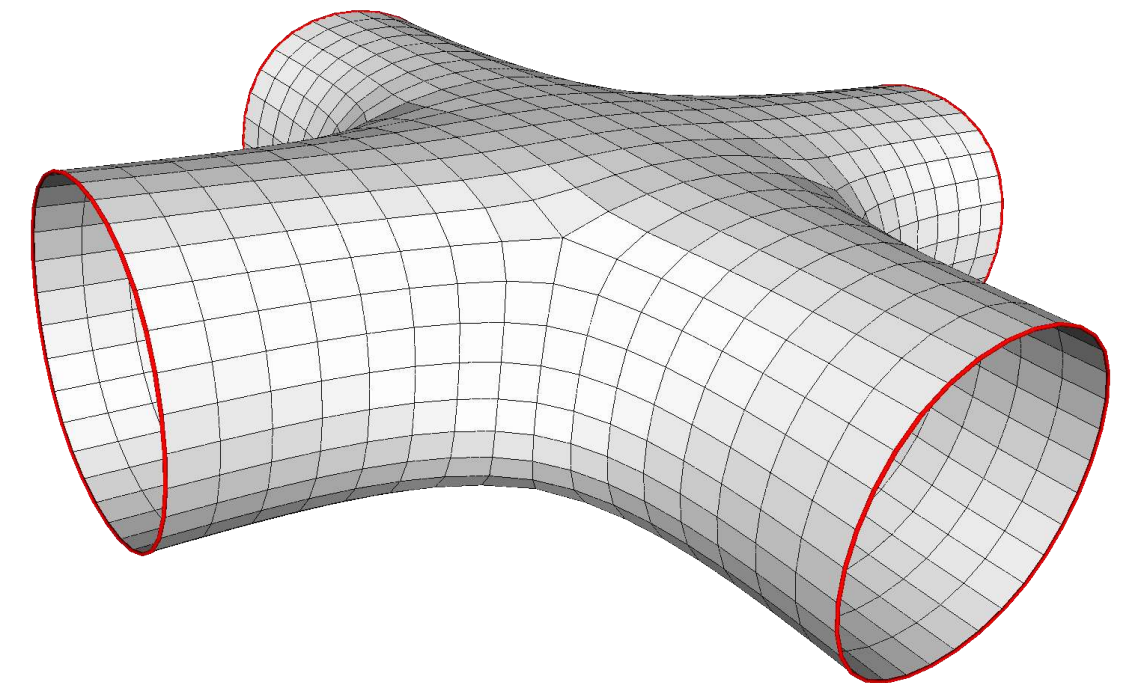
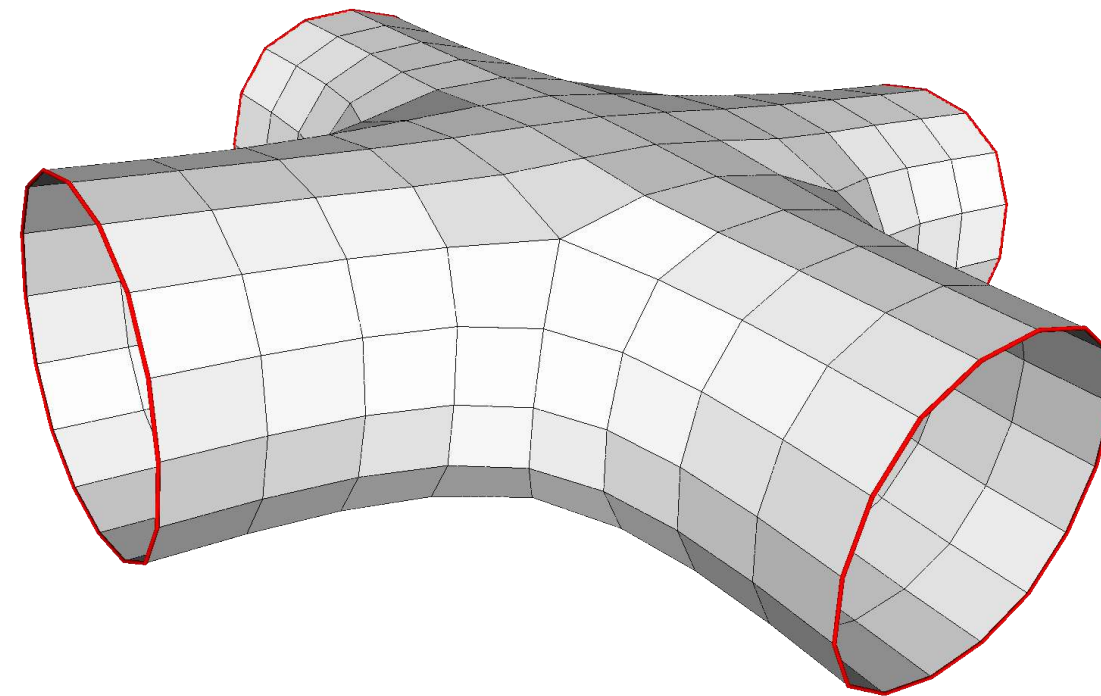
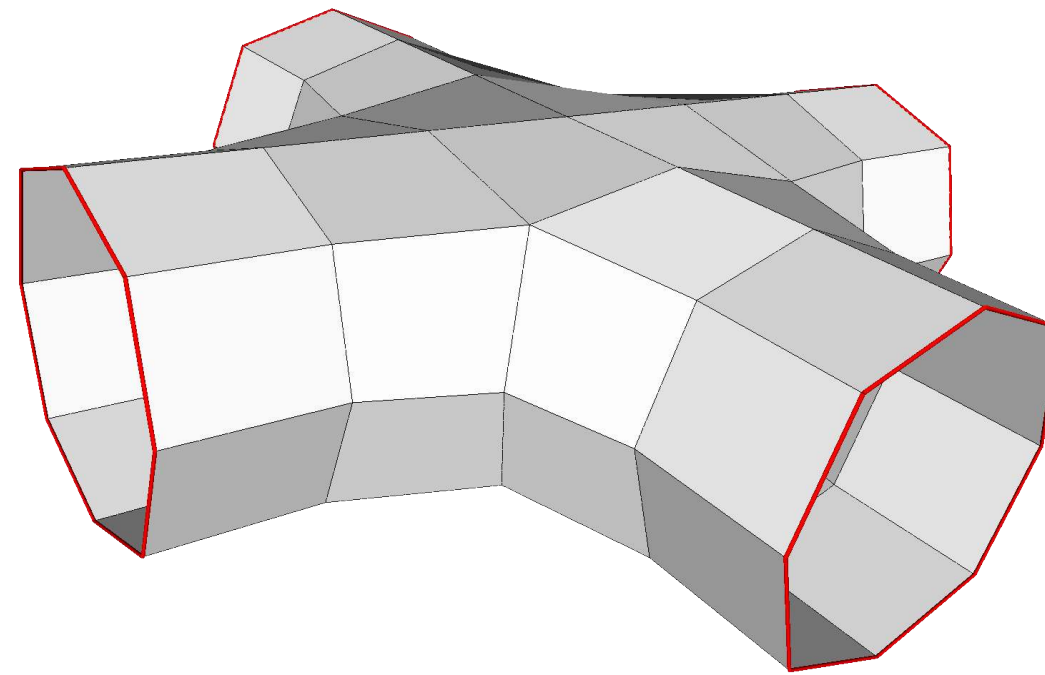
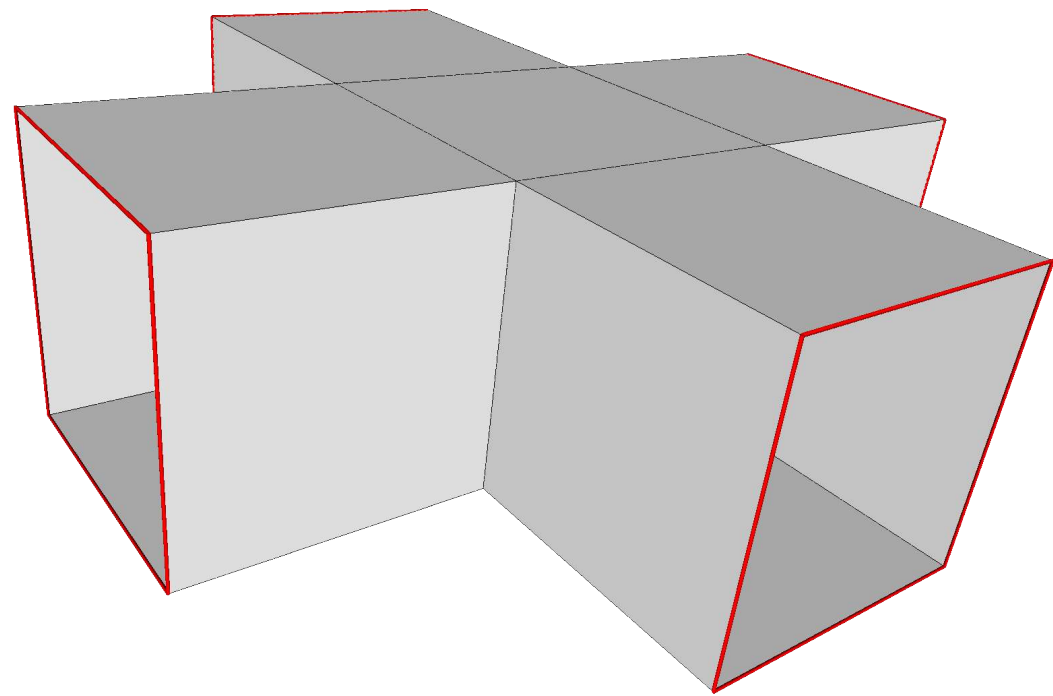
$r$  = average of incident edge points

$$q = \frac{1}{m} \sum_{i=1}^m f_i$$

$$r = \frac{1}{m} \sum_{i=1}^m e_i$$



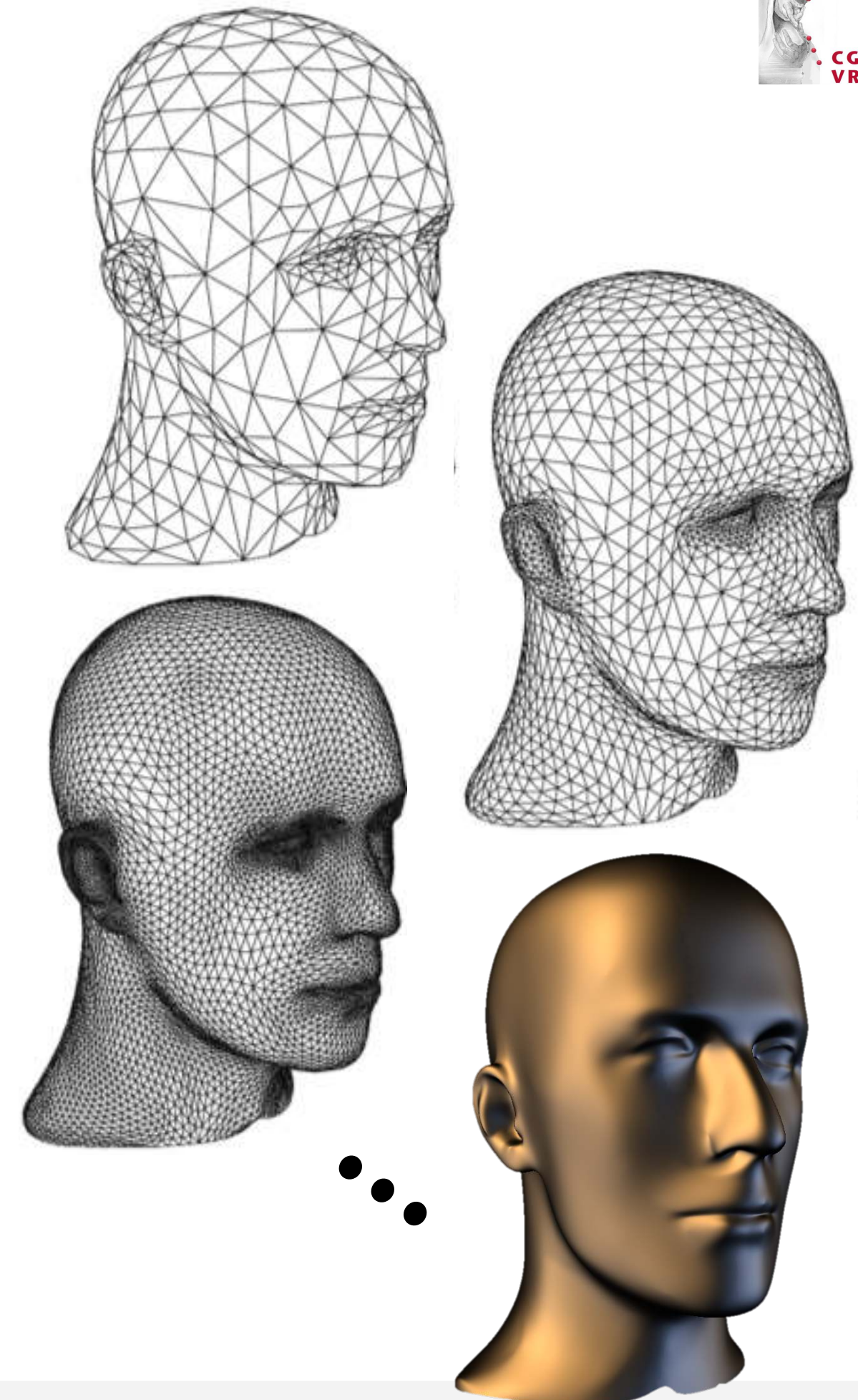
# Catmull-Clark in Action





# Advantages

- Modelers and animators (artists) like object descriptions that are ...
  - Easy to understand and control
  - Smooth, but creases can be added easily when needed
  - Offer different levels of detail, and LoD's can be made adaptive, e.g., view-dependent
  - Well-suited for animation, i.e., easy to deform
  - Allow for arbitrary topology (with holes and borders)
  - Compact (in terms of memory usage)



# Subdivision Schemes ("Subdivision Zoo")

## Common schemes:

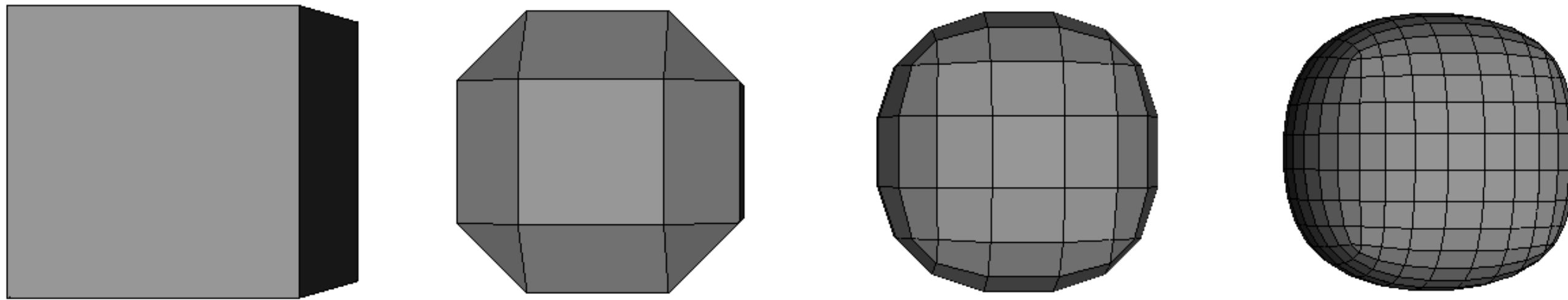
- Catmul Clark
- Doo-Sabin
- Loop
- Butterfly – Nira Dyn
- ...many more

## Classification by:

- Mesh type: tris, quads, hex..., combination
- Face / vertex split (a.k.a. "primal" / "dual" scheme)
- Interpolating / Approximating
- Smoothness
- Linear/non-linear
- ...

# Catmull-Clark vs Doo-Sabin

Doo-Sabin



Catmull-Clark

