Surface Parametrization

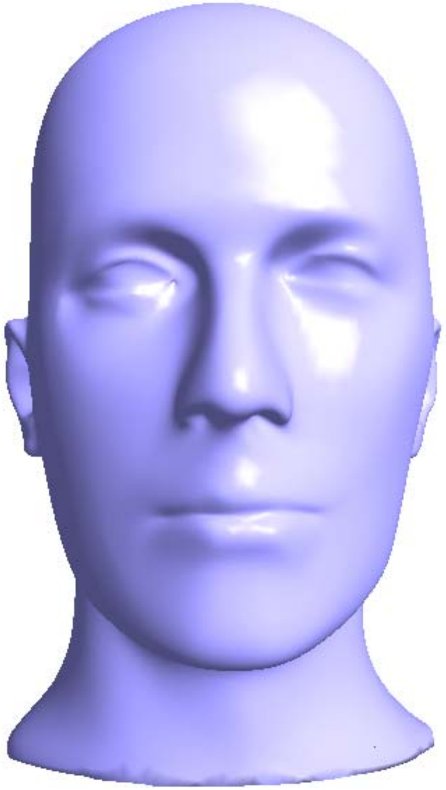
Most slides courtesy of Pierre Alliez and Craig Gotsman

The plan for today

- What is triangle mesh
- What is parameterization and what is it good for:
  - Texture mapping
  - Remeshing
- Parameterization
  - Convex mapping
  - Harmonic mapping

Triangle mesh

- Discrete surface representation
- Piecewise linear surface (made of triangles)



Triangle mesh

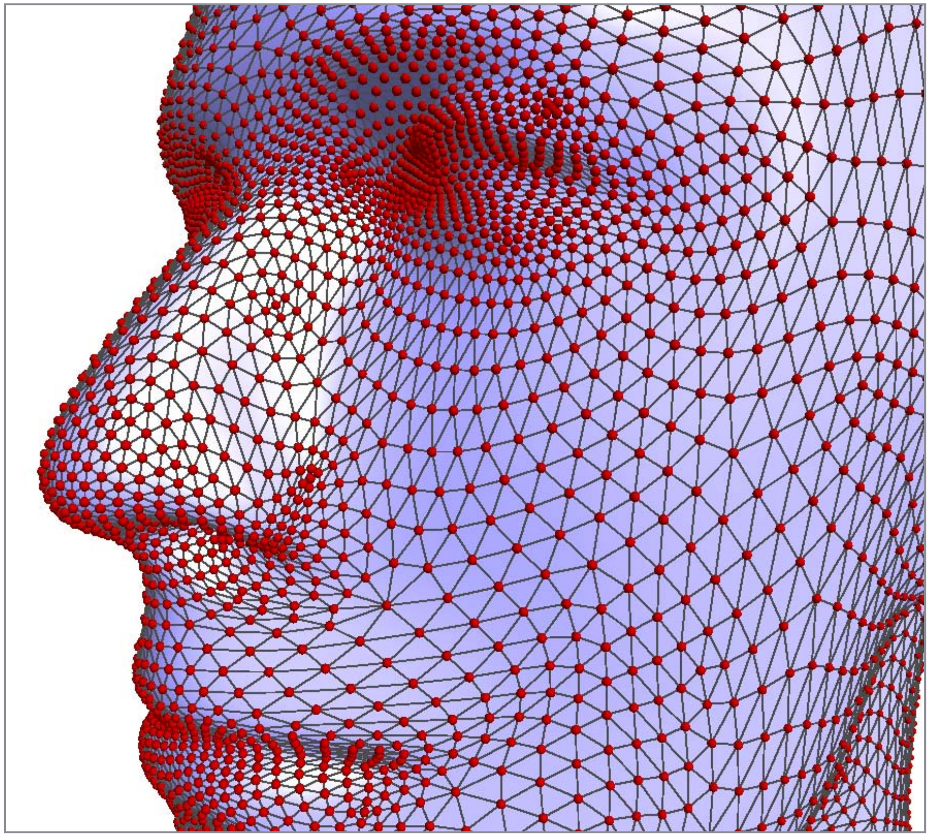
- Geometry:
  - Vertex coordinates

    (x_1, y_1, z_1)

    (x_2, y_2, z_2)

    . . .

    (x_n, y_n, z_n)



- Connectivity (the graph)
  - List of triangles

    (i_1, j_1, k_1)

    (i_2, j_2, k_2)

    . . .

    (i_m, j_m, k_m)

What is a parameterization?

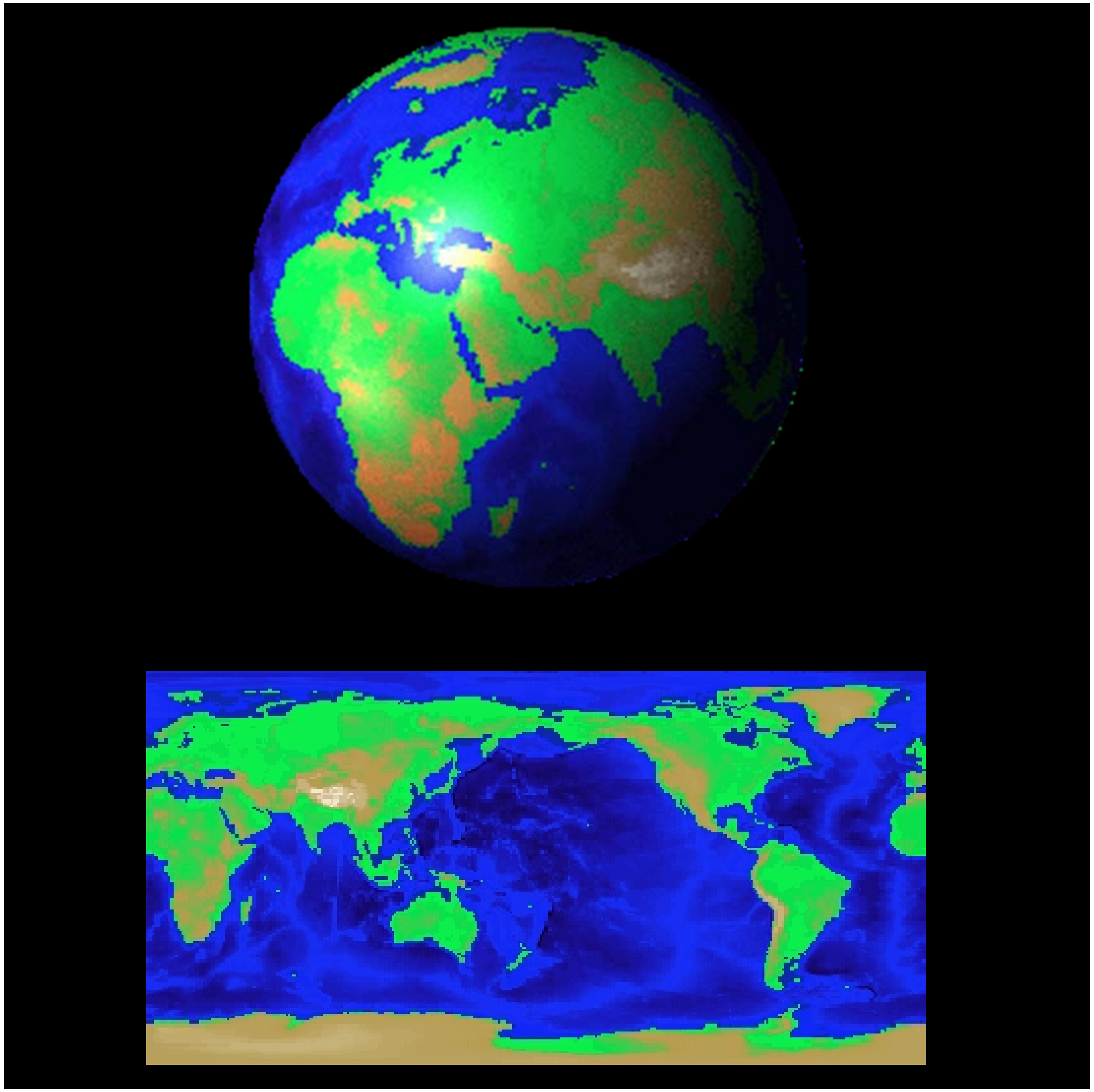
$S \subseteq R^3$ - given surface

$D \subseteq R^2$ - parameter domain

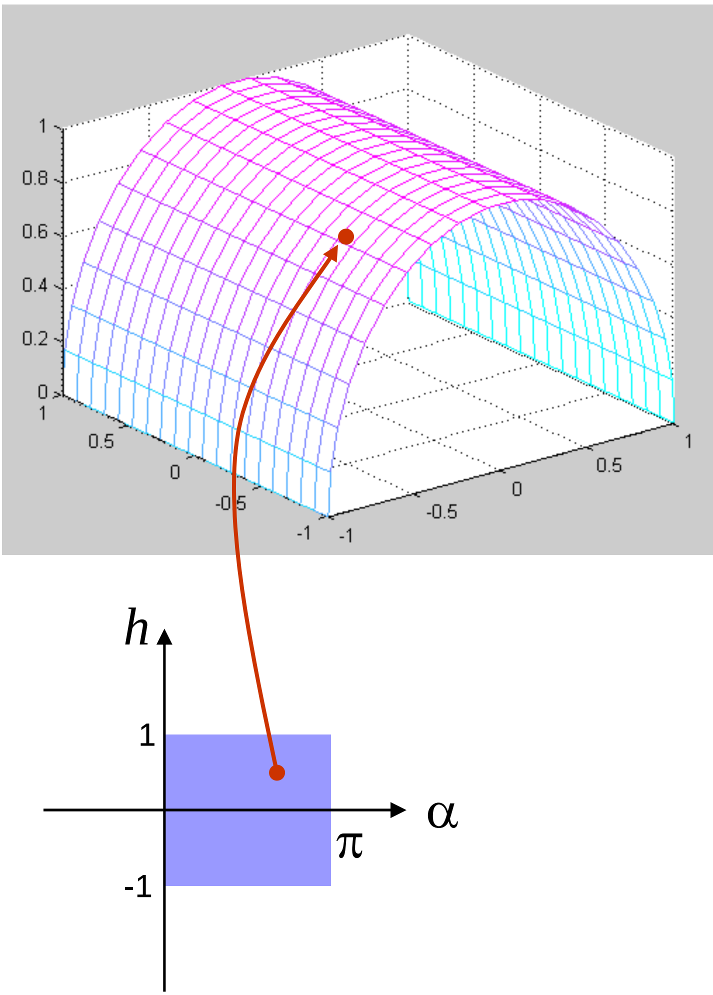
$\mathbf{s} : D \rightarrow S$ *1-1 and onto*

$$\mathbf{s}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$

Example – flattening the earth



Another example:



Parameters: α, h

$D = [0,\pi]\times[-1,1]$

$x(\alpha, h) = cos(\alpha)$

$y(\alpha, h) = h$

$z(\alpha, h) = sin(\alpha)$

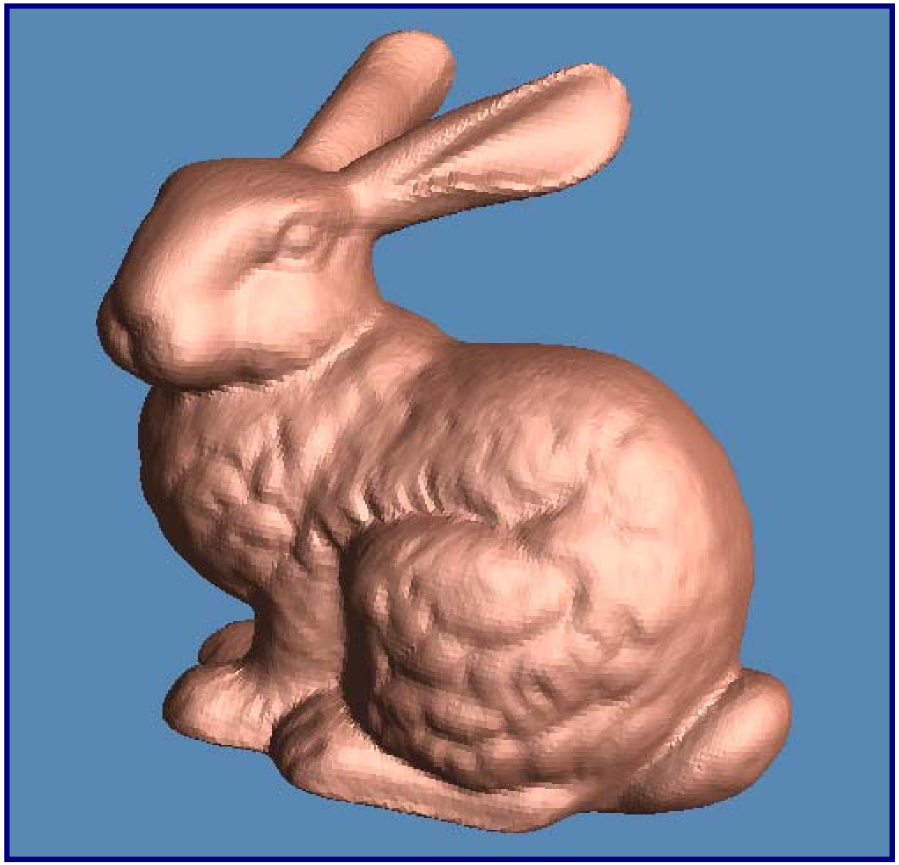
Triangular Mesh

- Standard *discrete* 3D surface representation in Computer Graphics – piecewise linear

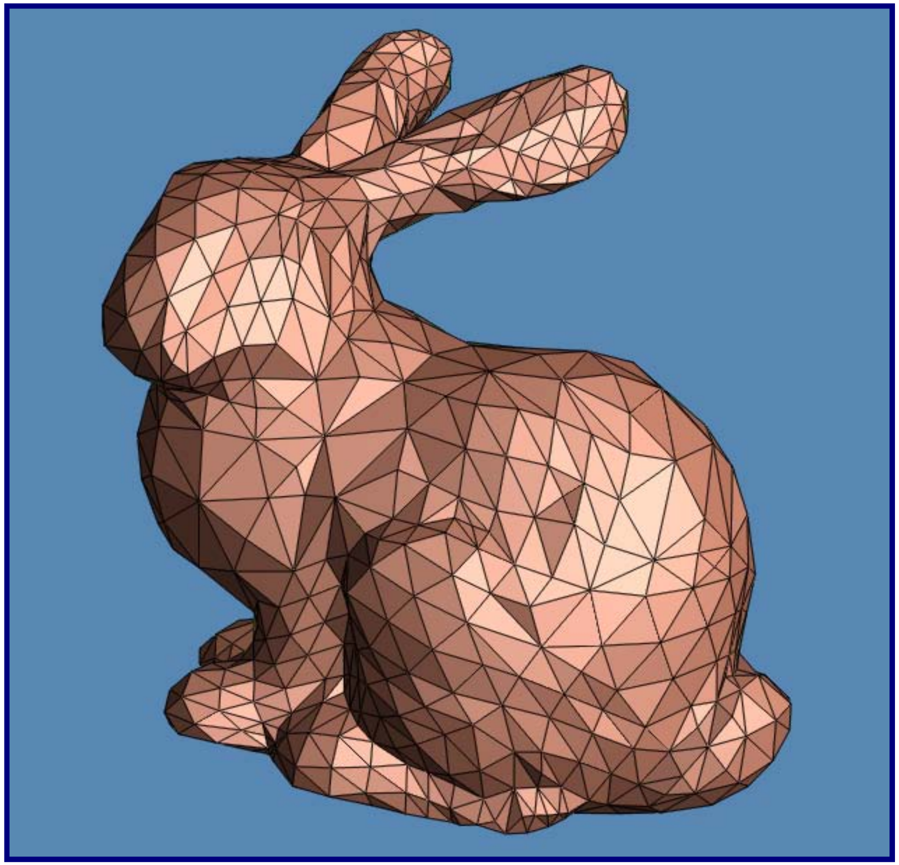
- *Mesh Geometry*: list of vertices (3D points of the surface)

- *Mesh Connectivity or Topology*: description of the faces

Triangular Mesh



Triangular Mesh



Mesh Parameterization

- Uniquely defined by mapping mesh vertices to the parameter domain:

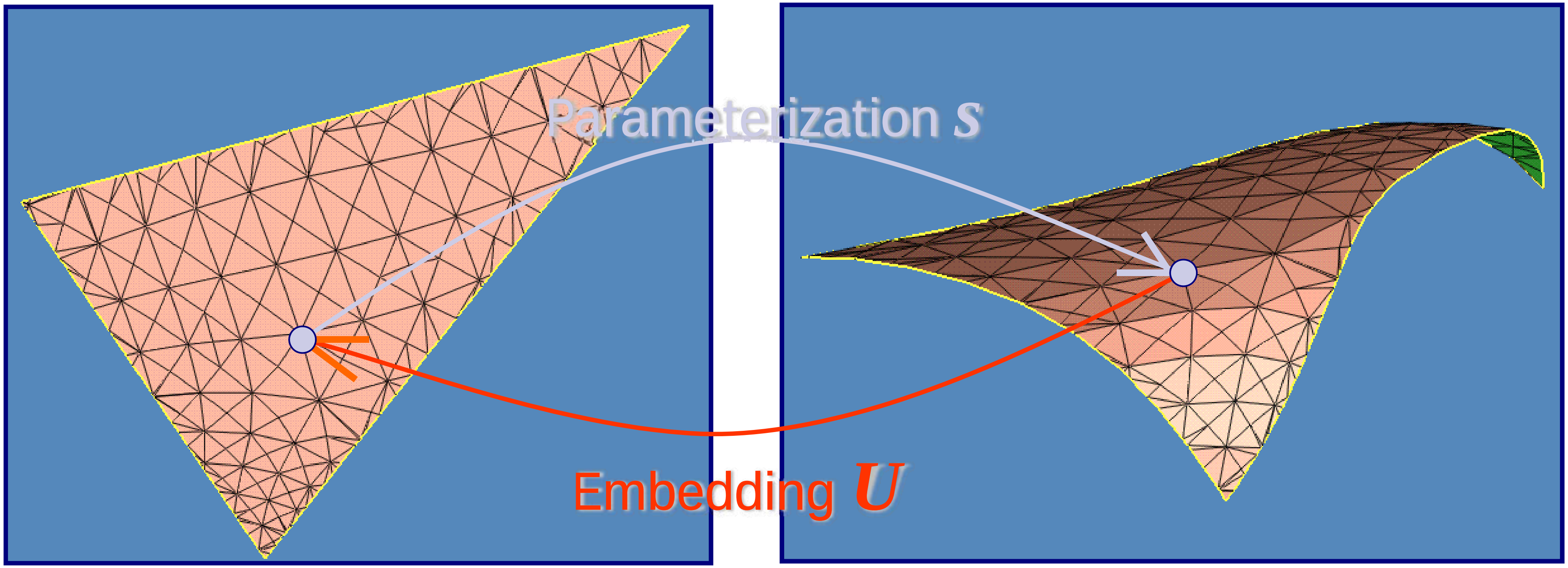
$$U : \{\mathrm{v}_1, \ldots, \mathrm{v}_\mathrm{n}\} \rightarrow D \subseteq R^2$$

$$U(\mathrm{v}_i) = (u_i, v_i)$$

- No two edges cross in the plane (in D)

Mesh parameterization $\Leftrightarrow$ mesh embedding

Mesh parameterization



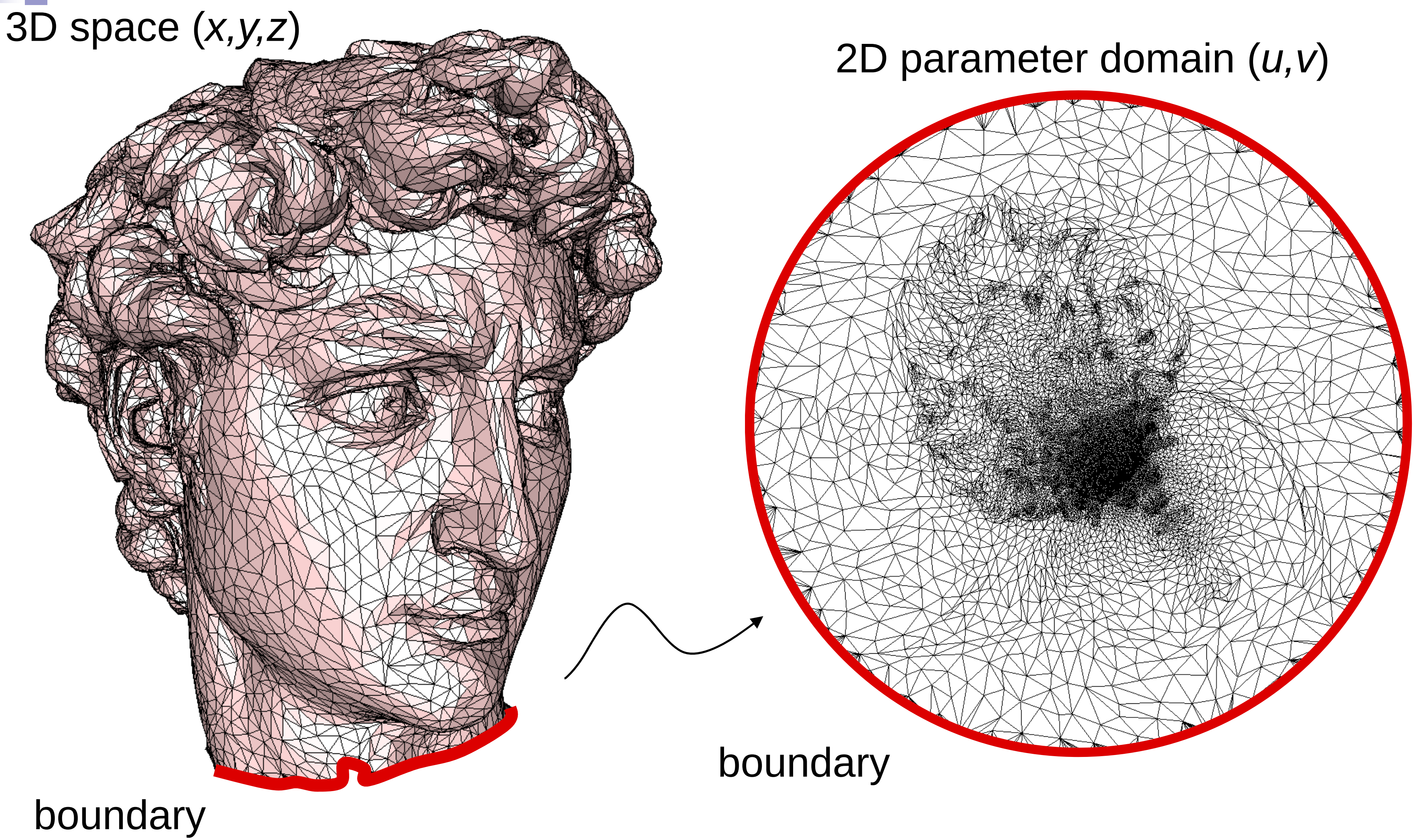
Parameter domain **Mesh surface**

$D \subseteq R^2$

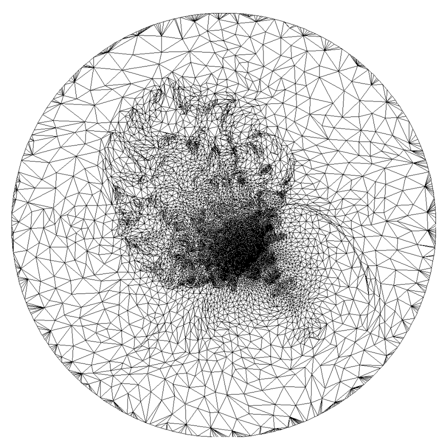
$$\boxed{s = U^{-1}}$$

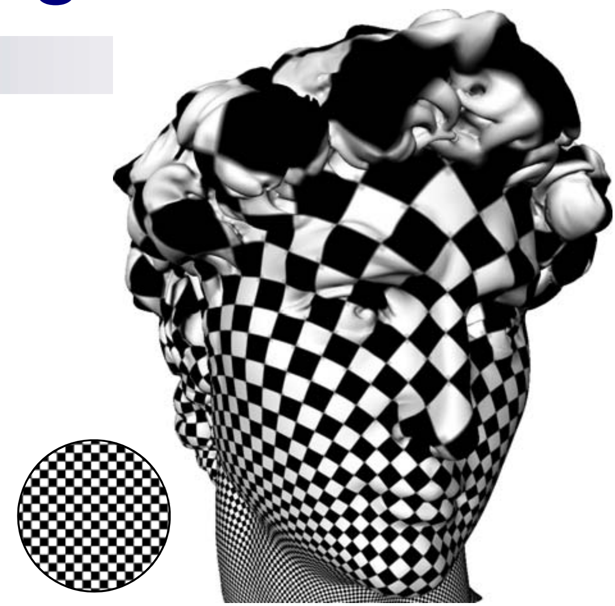
$S \subseteq R^3$

2D parameterization



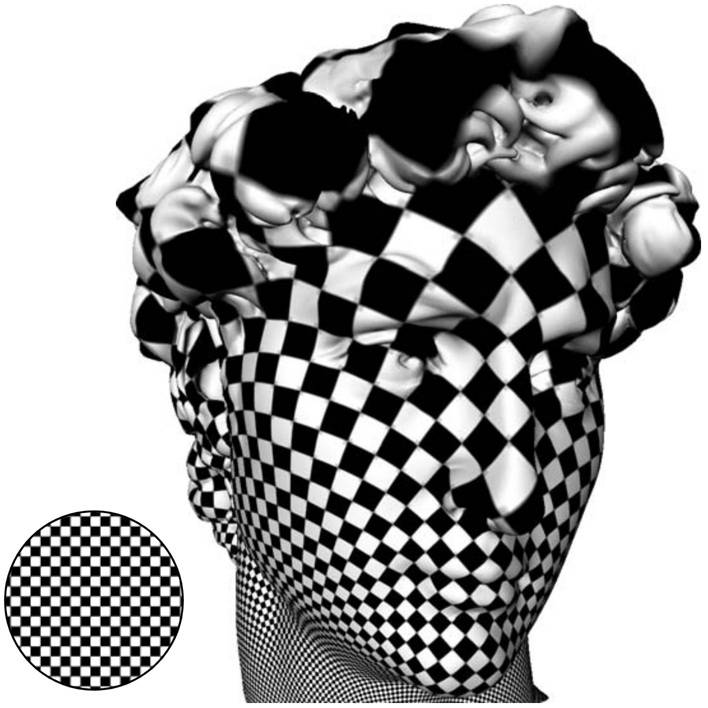
Application - Texture mapping



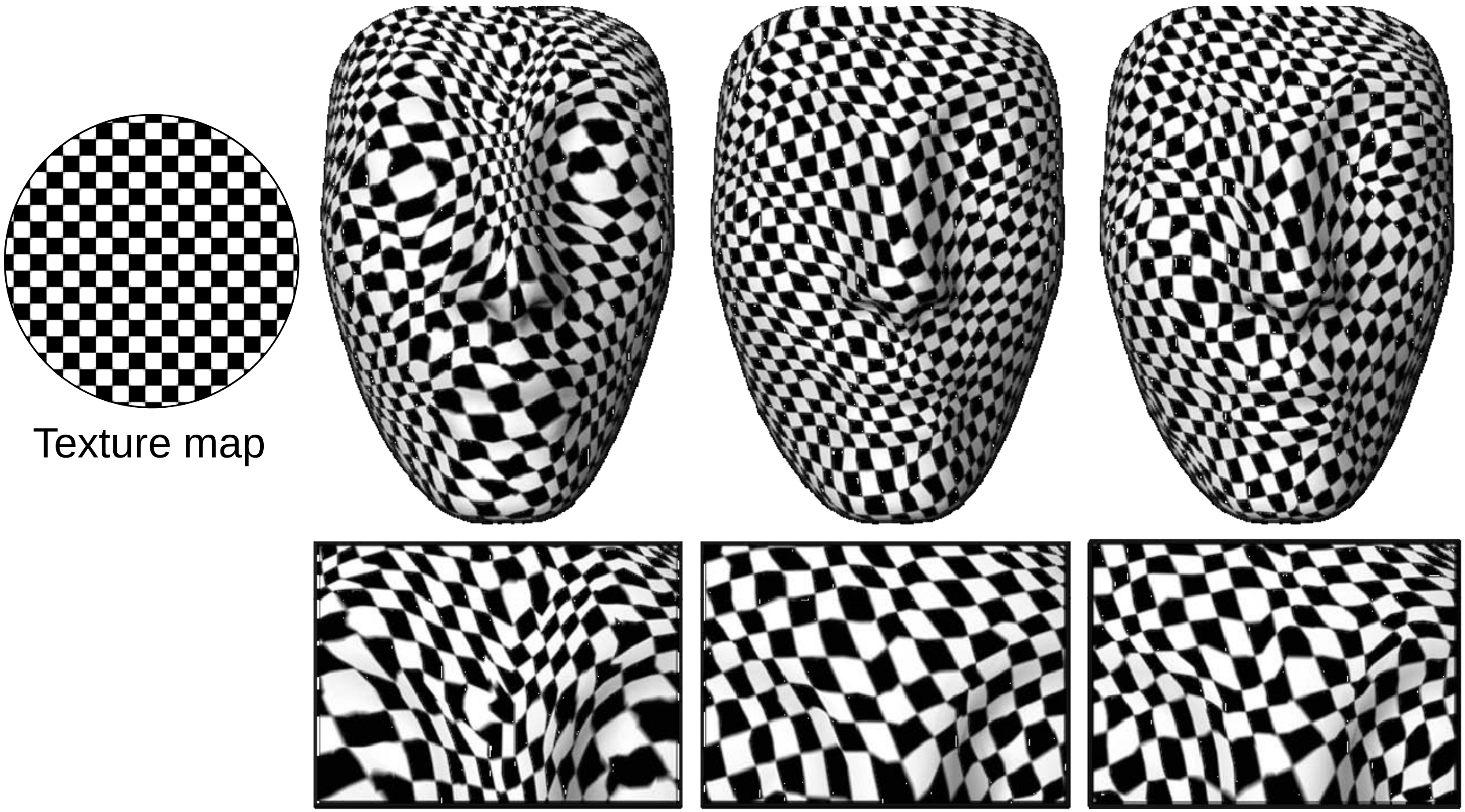


Requirements

- Bijective (1-1 and onto): No triangles fold over.
- Minimal “distortion”
  - Preserve 3D angles
  - Preserve 3D distances
  - Preserve 3D areas
  - No “stretch”

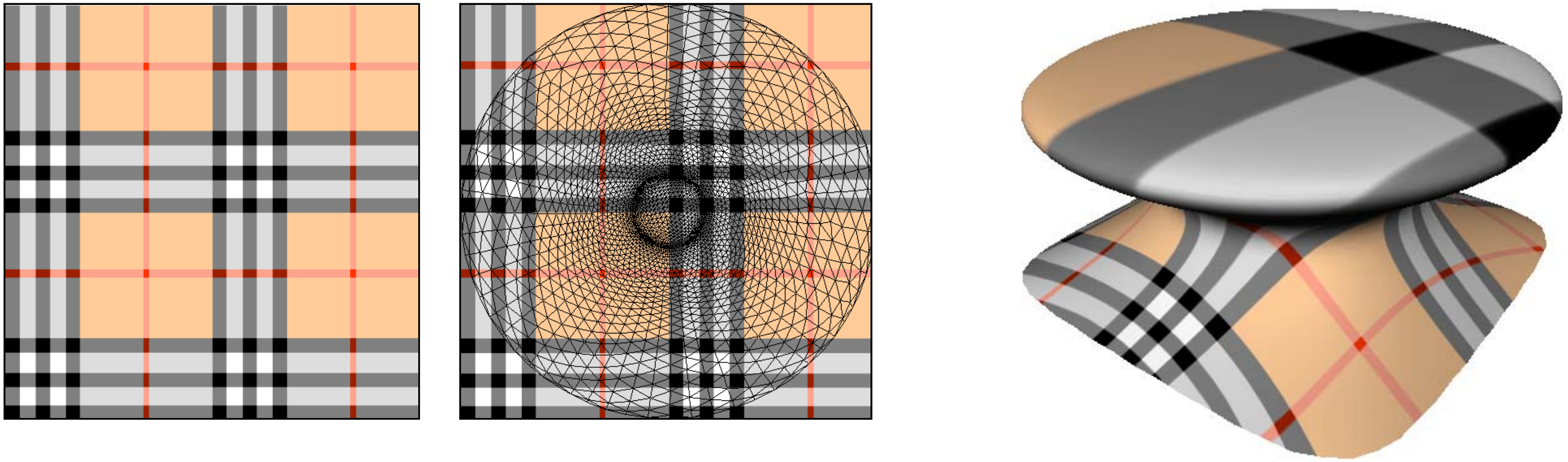


Distortion minimization

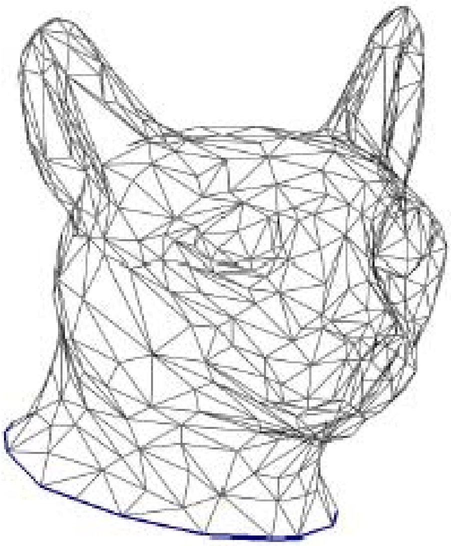


Kent et al '92 Floater 97 Sander et al '01

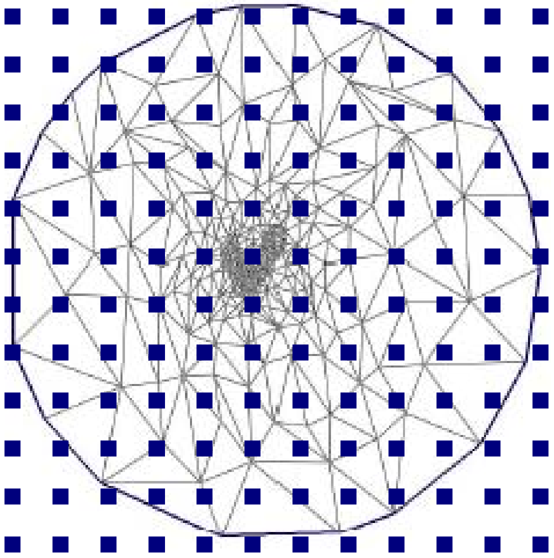
More texture mapping



Resampling problems



Cat mesh



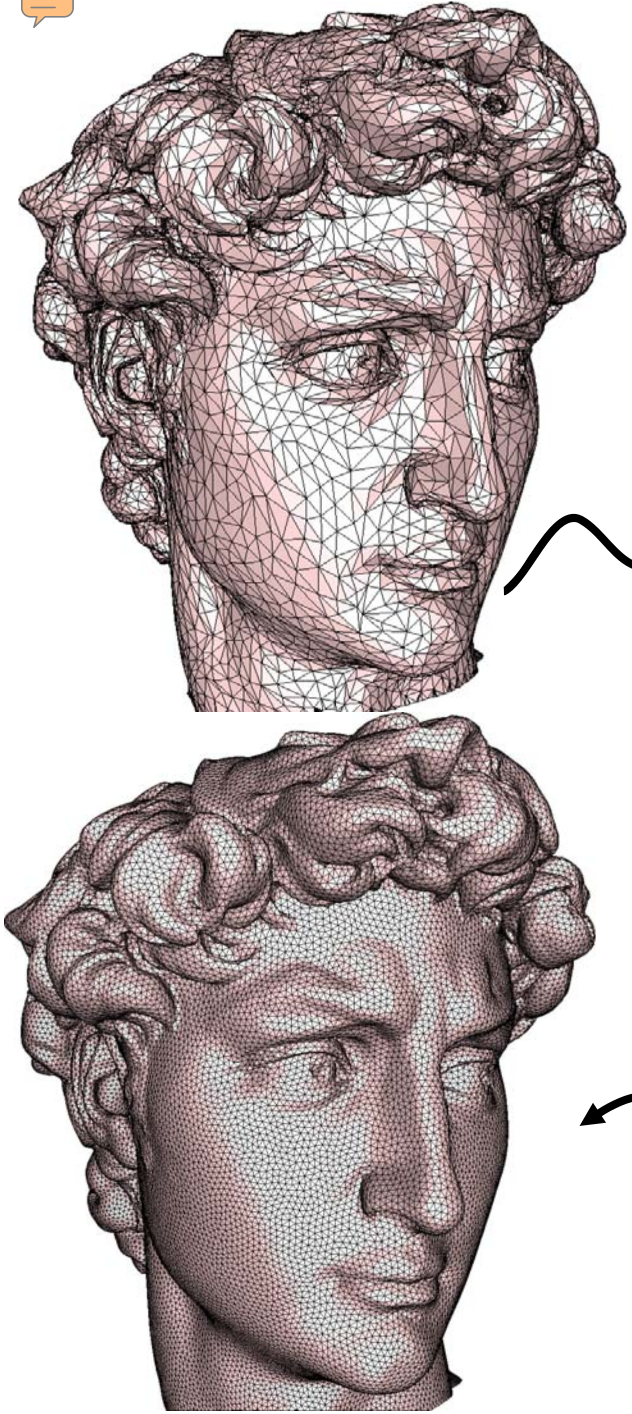
Distorting embedding

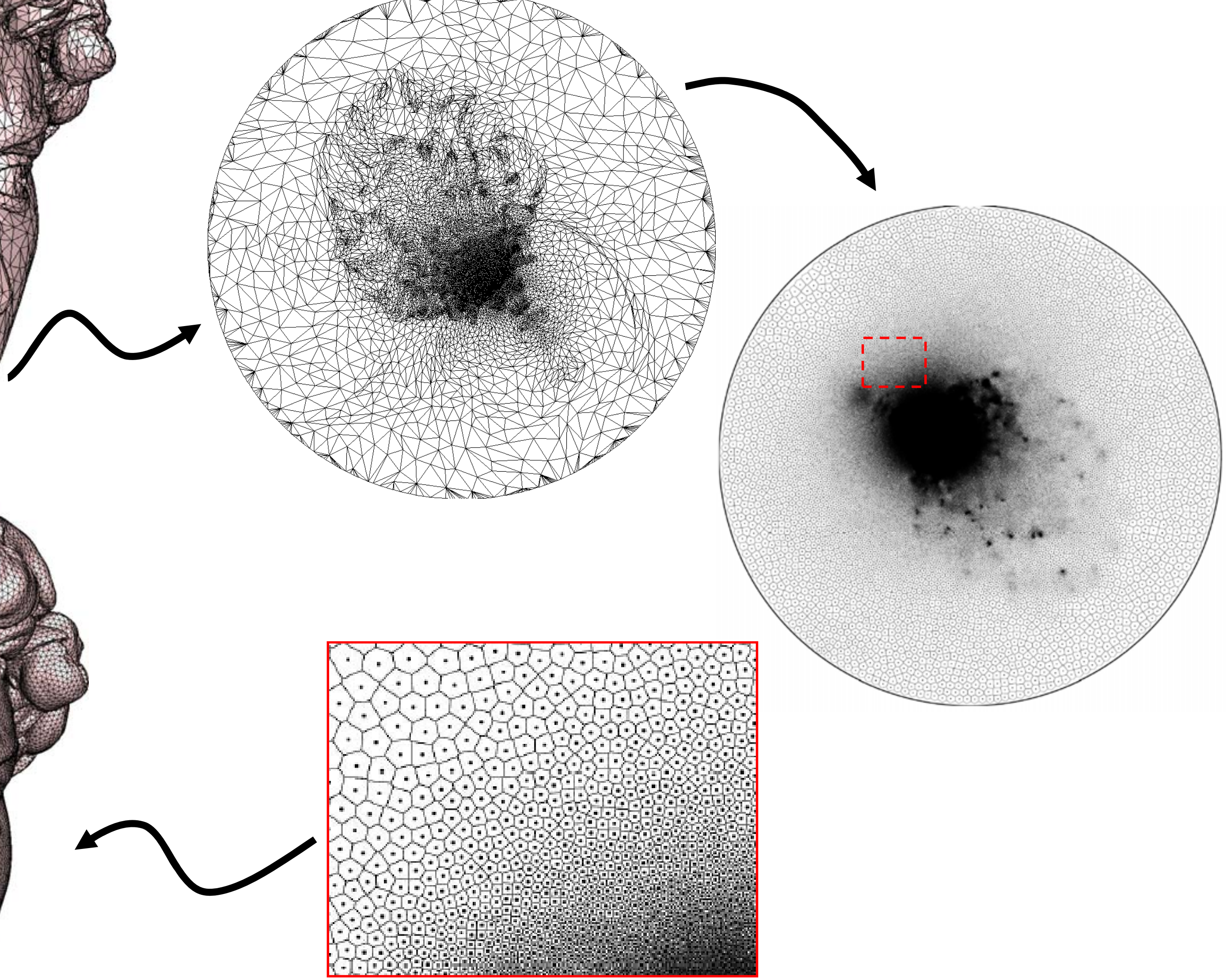
Resampling on regular grid

Applications

- Texture Mapping
- Remeshing
- Surface Reconstruction
- Morphing
- Compression

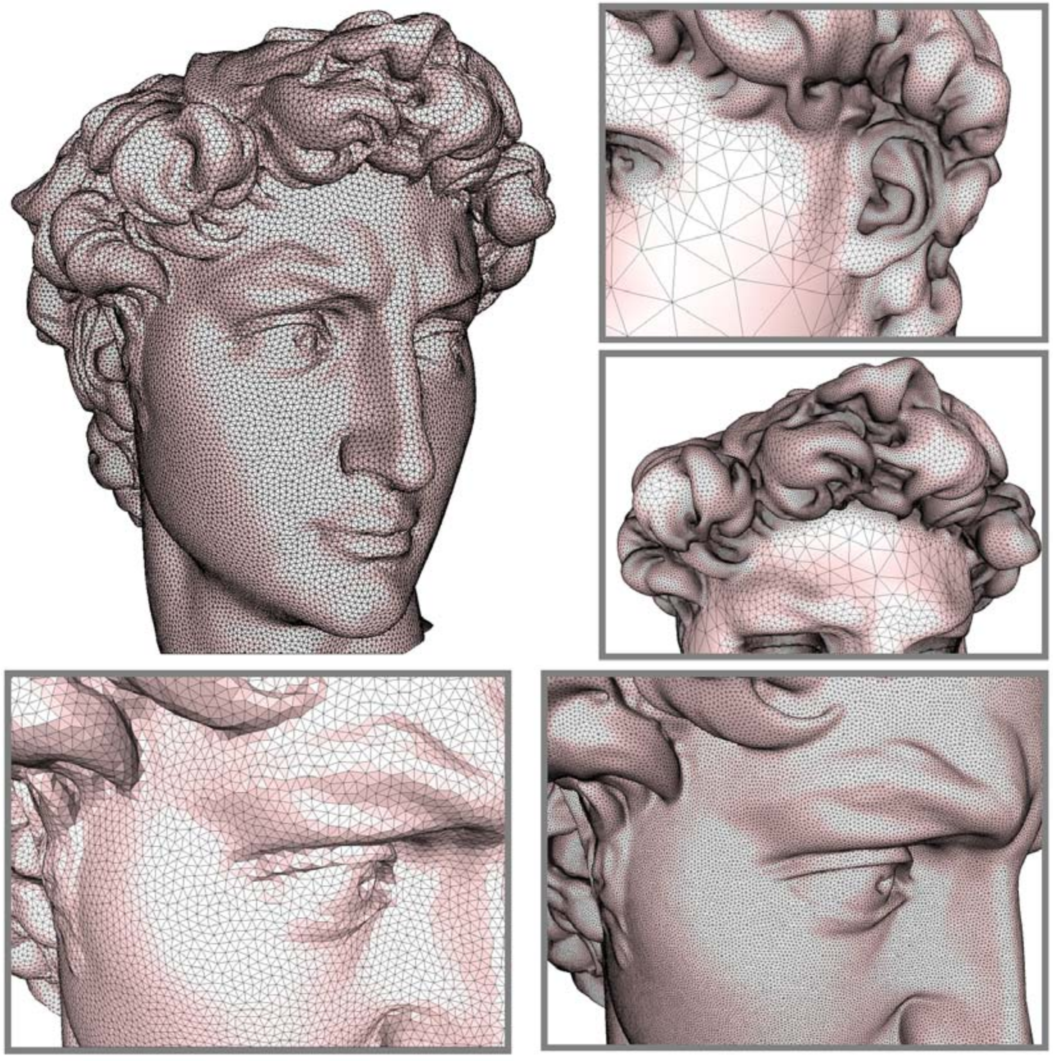
Remeshing



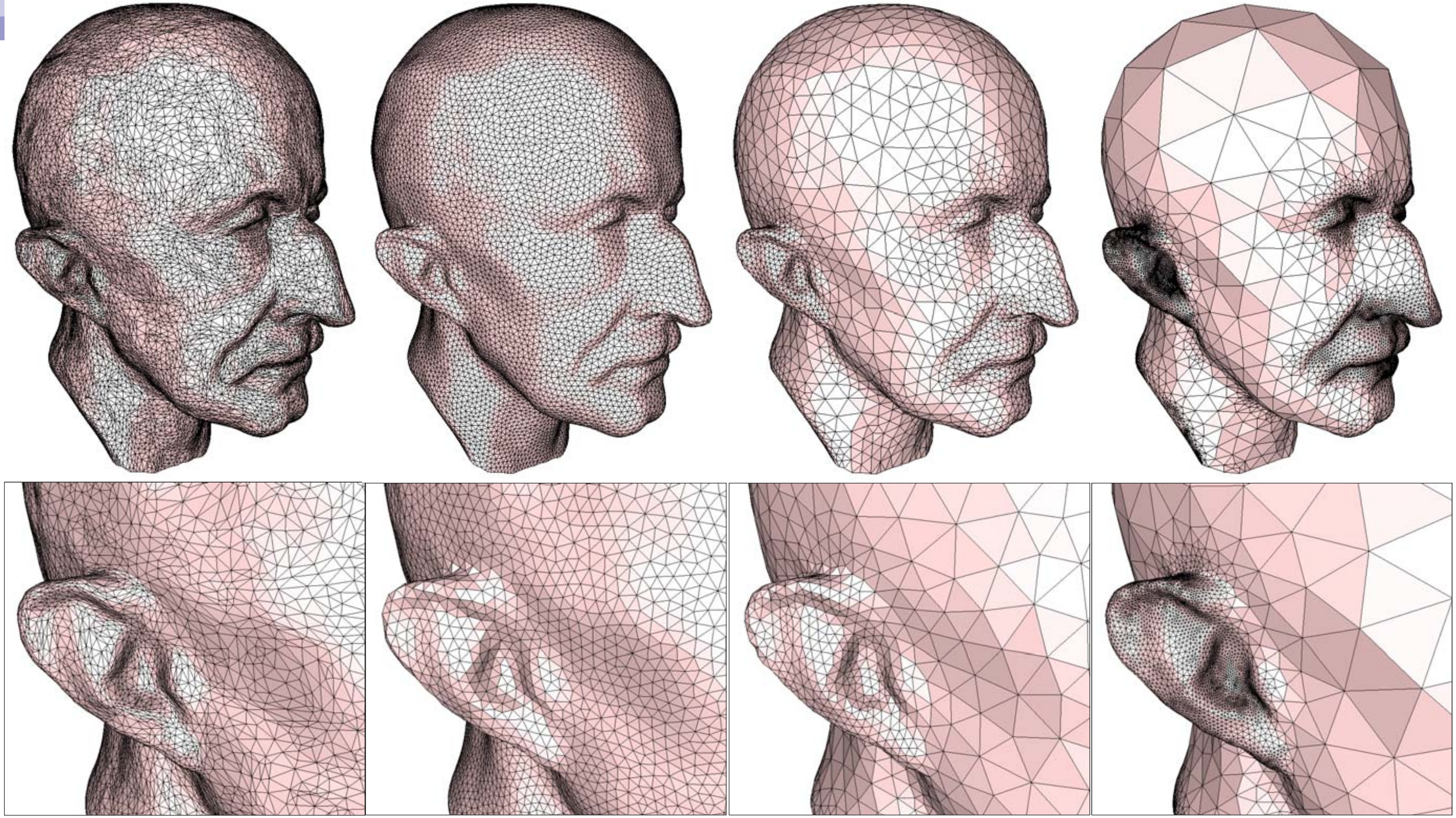


Remeshing

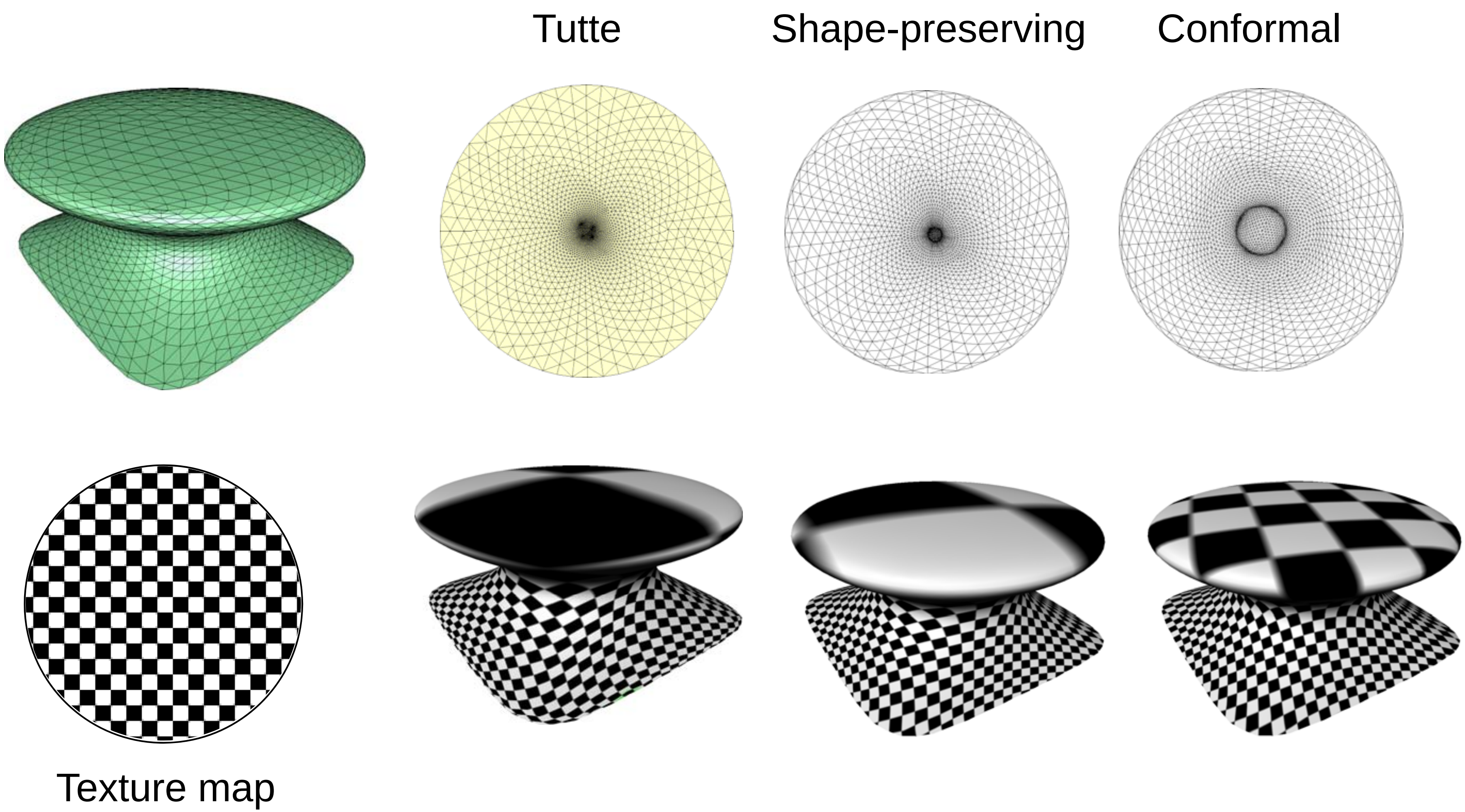
Remeshing



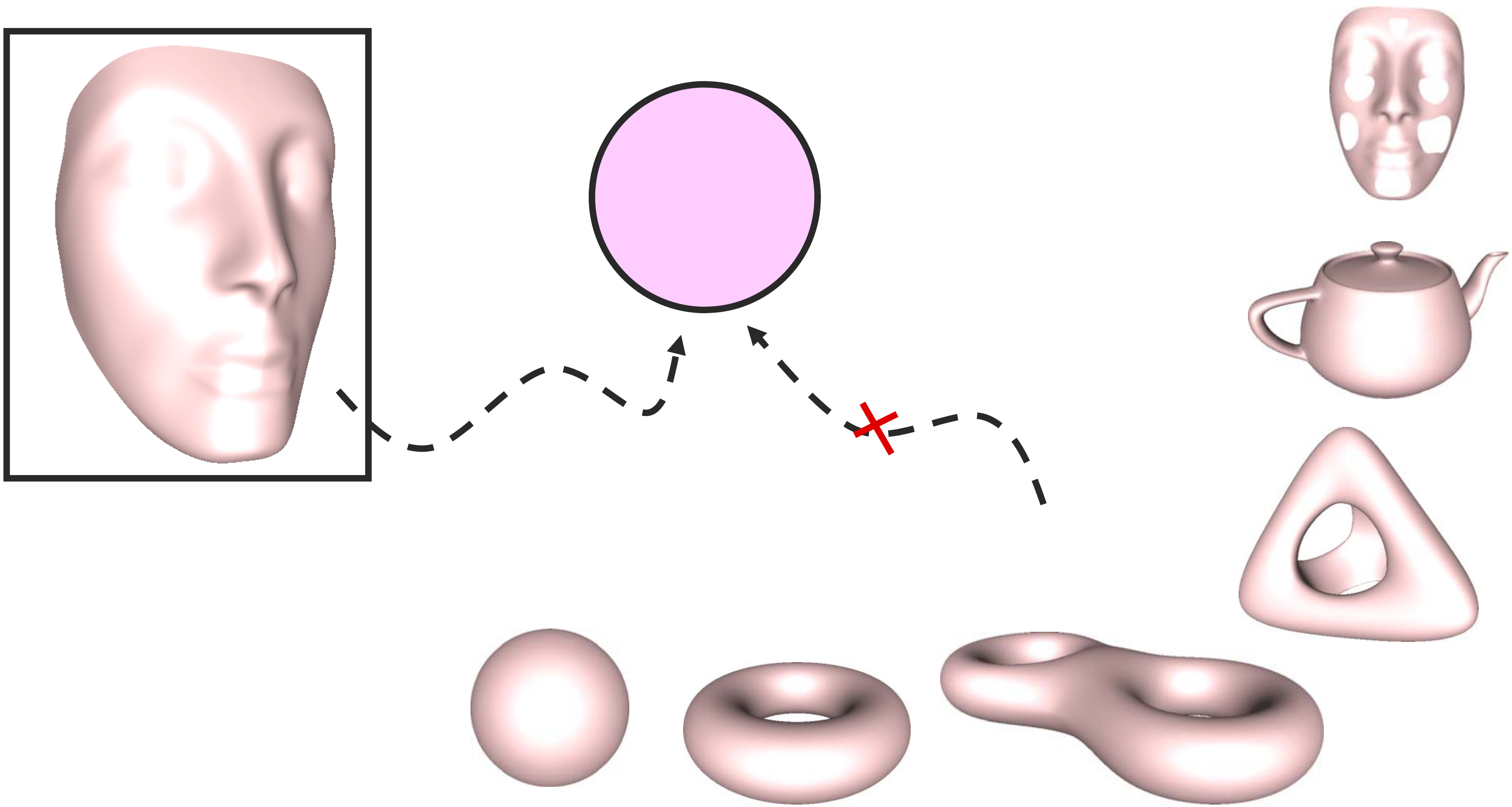
More remeshing examples



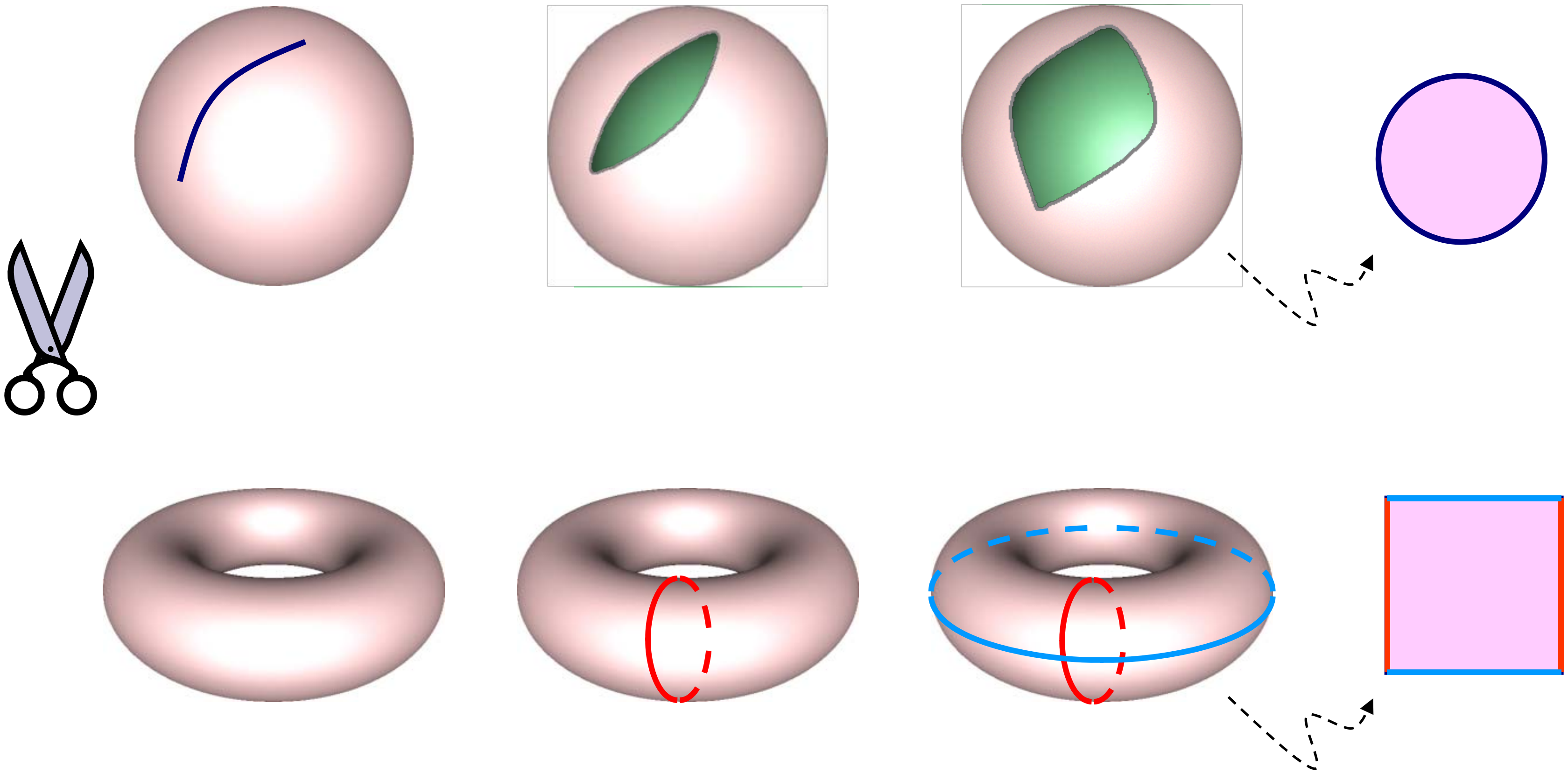
Conformal parametrization



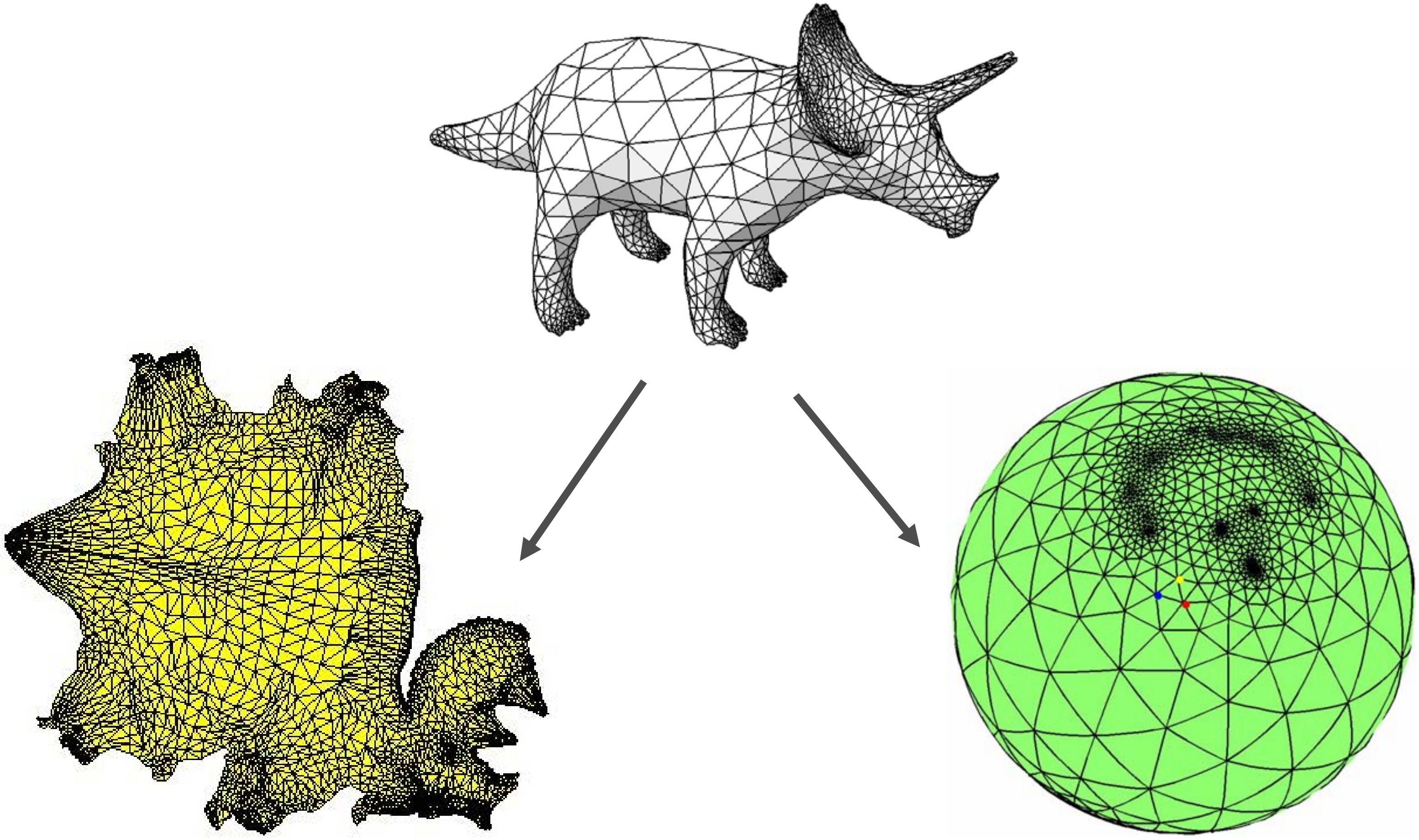
Non-simple domains



Cutting



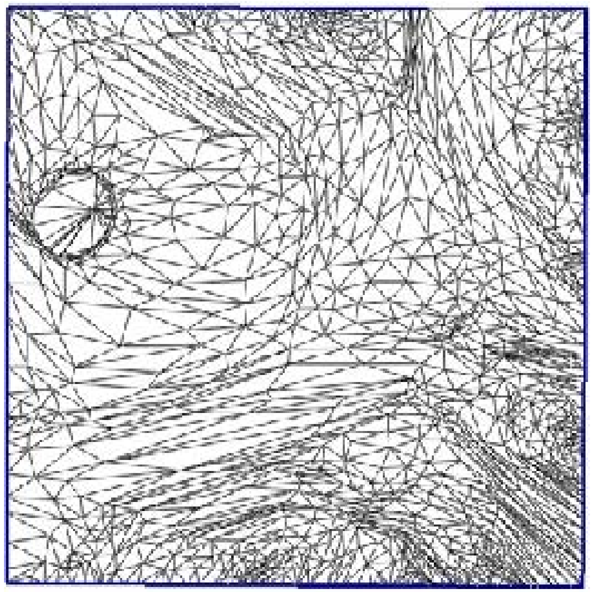
Parameterization of closed genus-0 triangle meshes



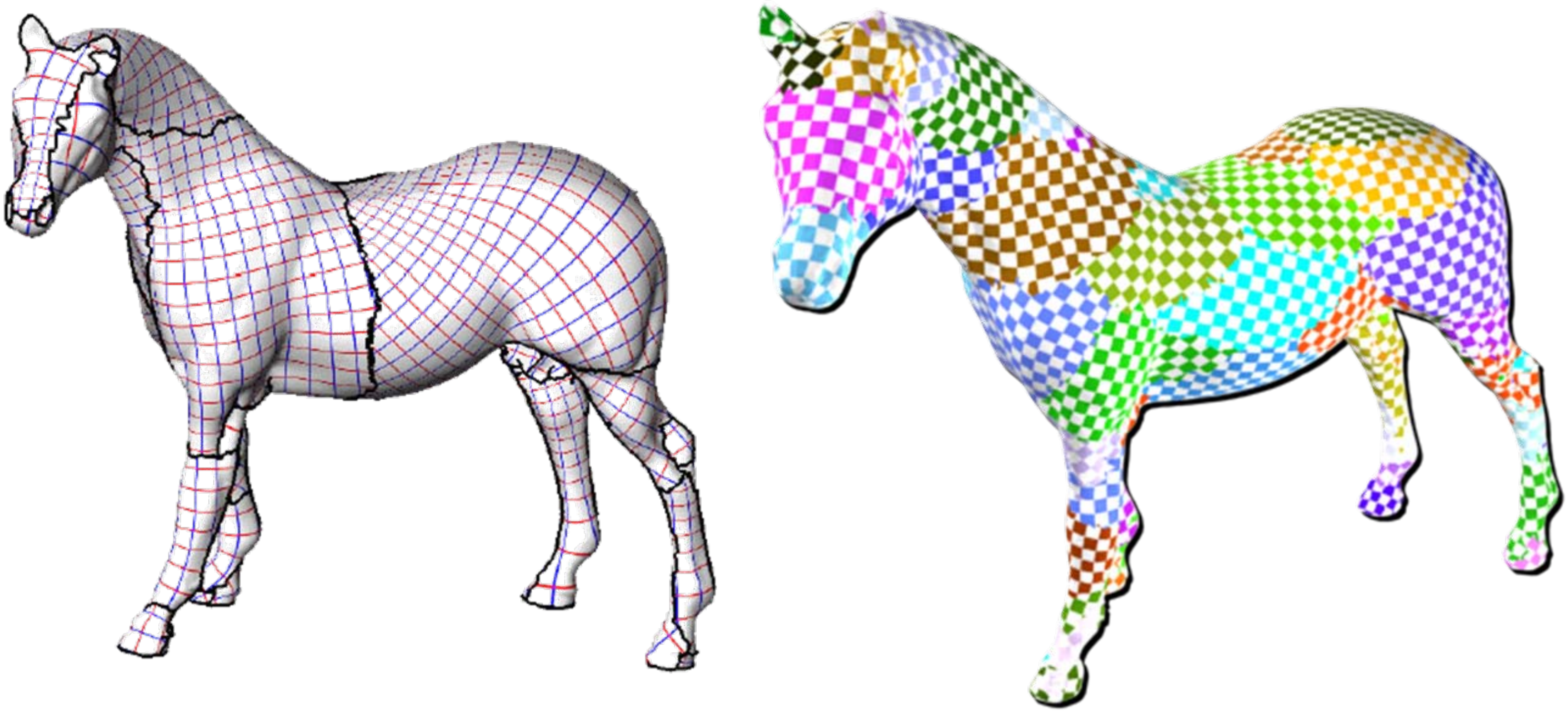
Non-Constrained Planar

Spherical

Introducing seams (cuts)

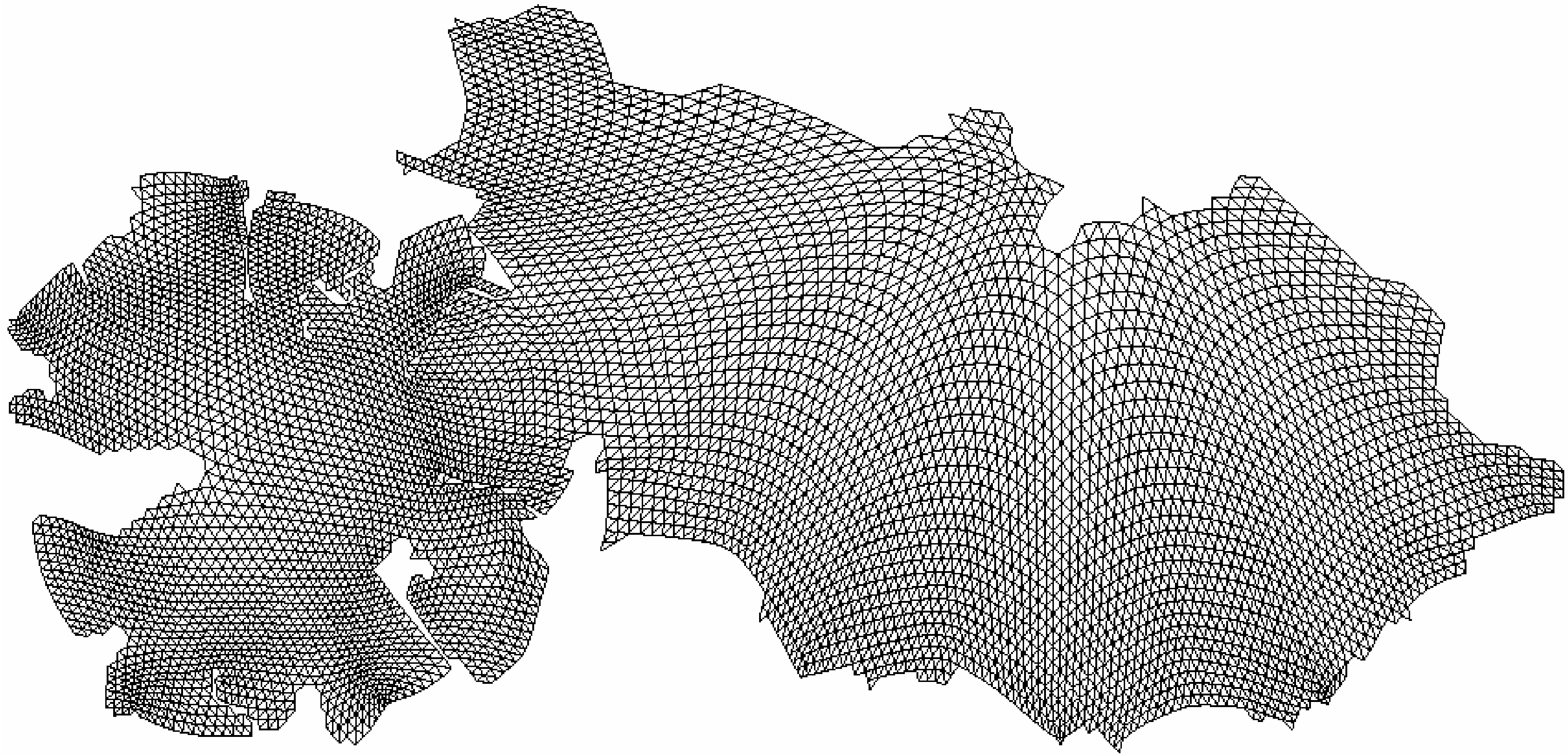


Partition

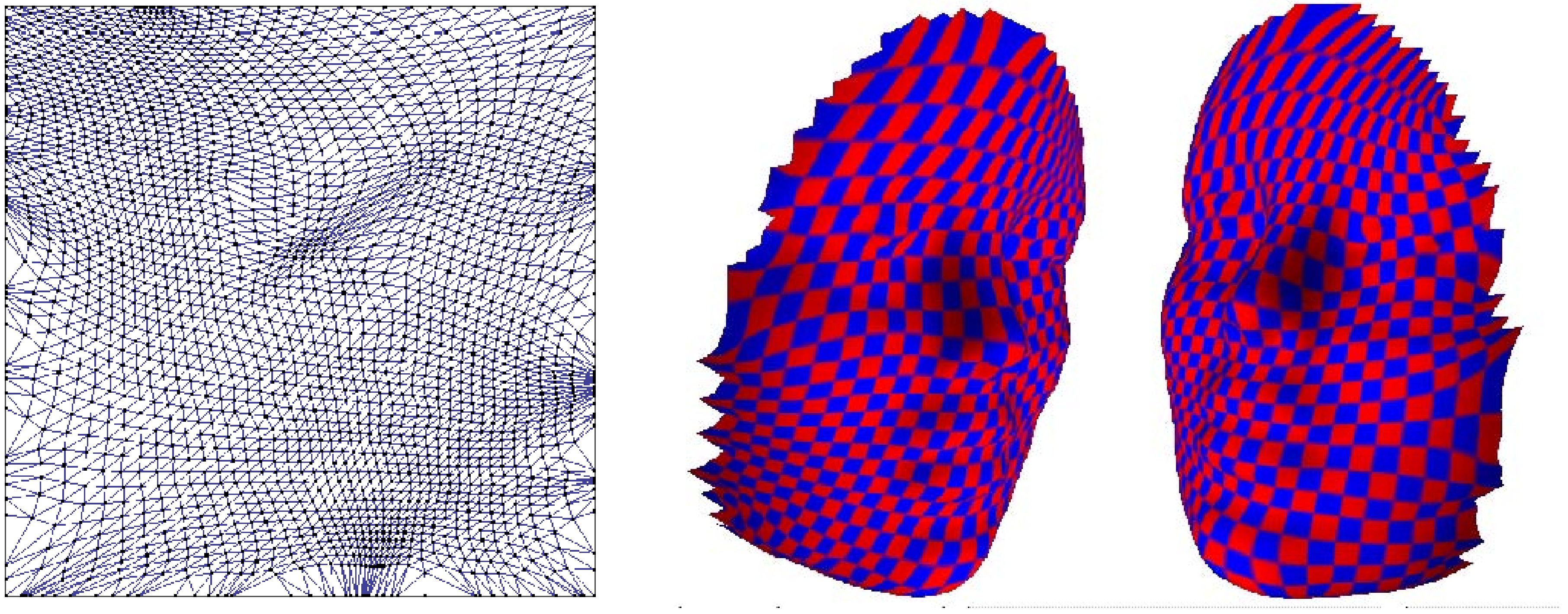


Introducing seams (cuts)

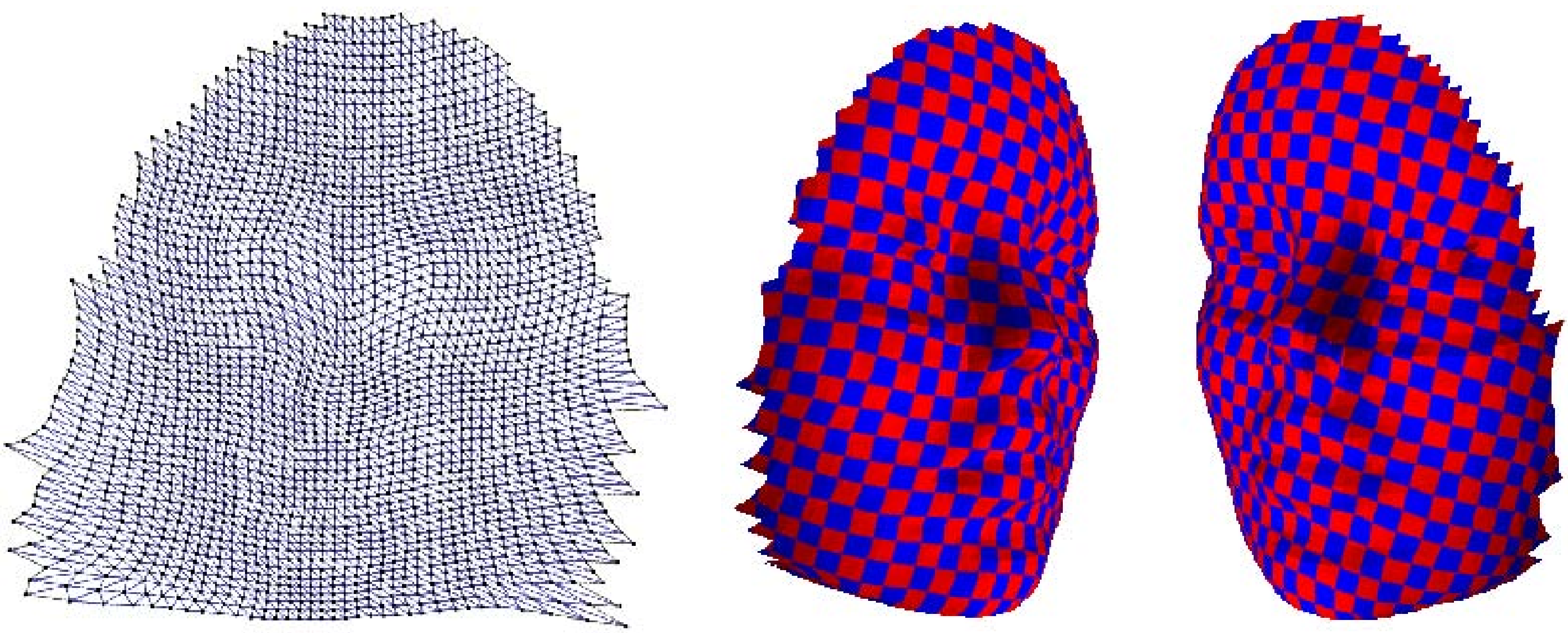
Introducing seams (cuts)



Bad parameterization



Better…(free boundary)



Partition – problems

- Discontinuity of parameterization
- Visible artifacts in texture mapping
- Require special treatment
  - Vertices along seams have several (u,v) coordinates
  - Problems in mip-mapping

Make seams short and hide them

Summary

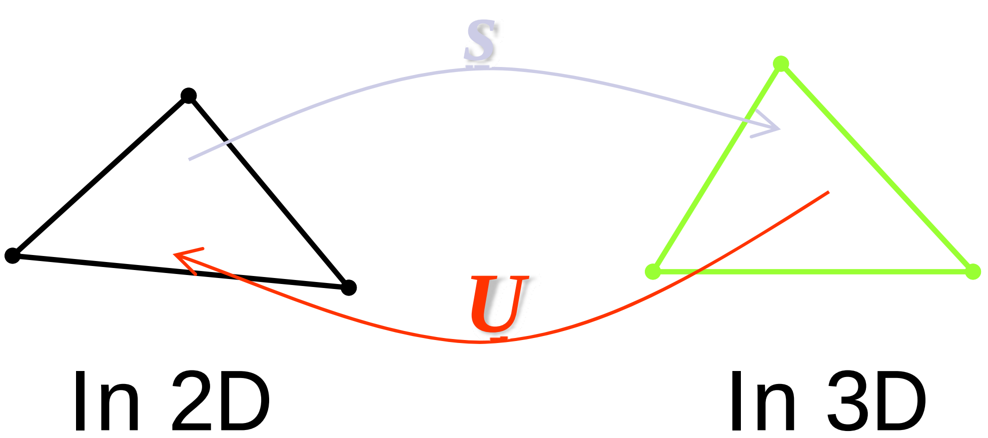
- “Good” parameterization = non-distorting
  - Angles and area preservation
  - Continuous param. of complex surfaces cannot avoid distortion.

- “Good” partition/cut:
  - Large patches, minimize seam length
  - Align seams with features (=hide them)

Mesh parameterization

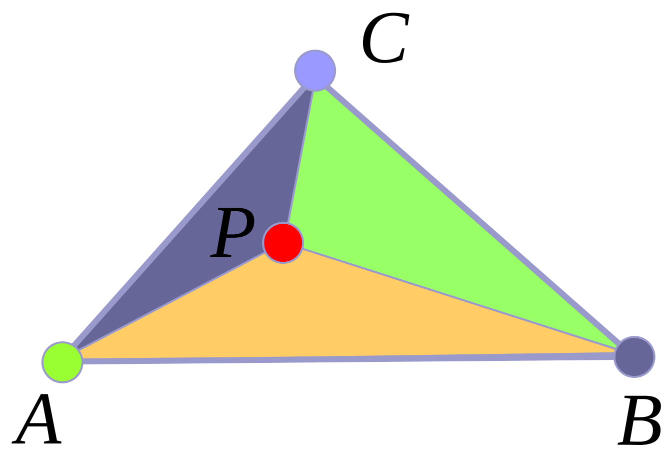
s and *U* are piecewise-linear

Linear inside each mesh triangle



A mapping between two triangles is a *unique affine* mapping

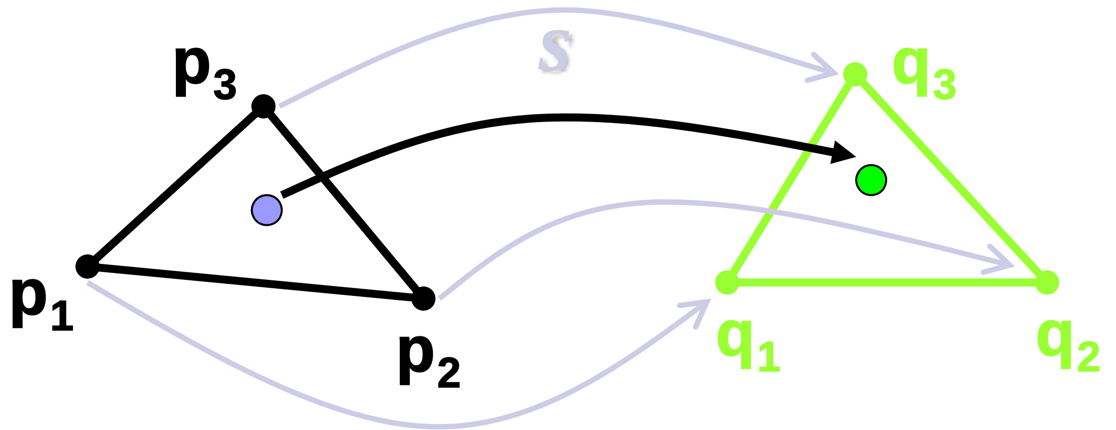
Barycentric coordinates



$$\vec{P} = \frac{\langle P,\ B,\ C\rangle}{\langle A,\ B,\ C\rangle}\vec{A} + \frac{\langle P,\ C,\ A\rangle}{\langle A,\ B,\ C\rangle}\vec{B} + \frac{\langle P,\ A,\ B\rangle}{\langle A,\ B,\ C\rangle}\vec{C}$$

$\langle \cdot,\ \cdot,\ \cdot \rangle$ denotes the (signed) area of the triangle

Mapping triangle to triangle

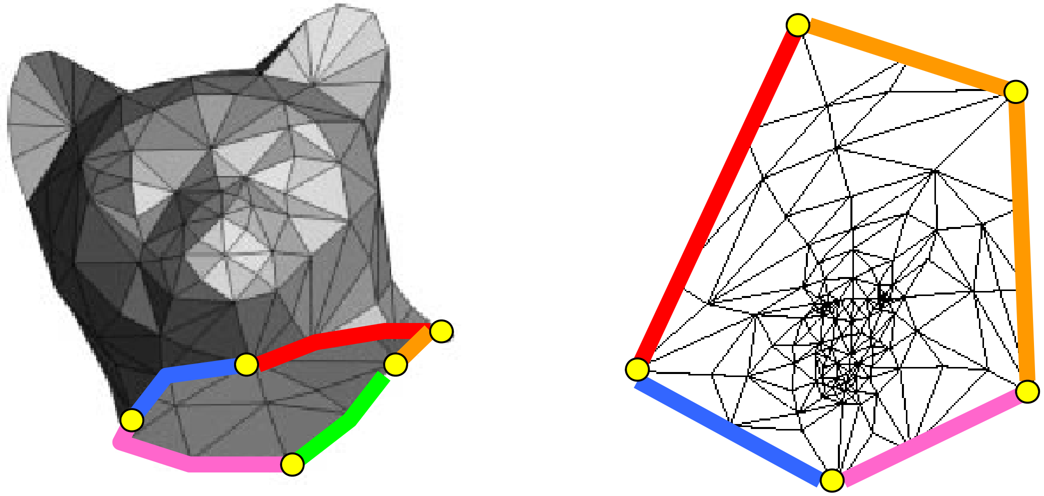


$$\mathbf{s}(\mathbf{p}) = \frac{\langle \mathbf{p}, p_2, p_3 \rangle}{\langle p_1, p_2, p_3 \rangle} q_1 + \frac{\langle \mathbf{p}, p_3, p_1 \rangle}{\langle p_1, p_2, p_3 \rangle} q_2 + \frac{\langle \mathbf{p}, p_1, p_2 \rangle}{\langle p_1, p_2, p_3 \rangle} q_3$$

Some techniques

Convex mapping (Tutte, Floater)

- Works for meshes equivalent to a disk
- First, we map the boundary to a convex polygon
- Then we find the inner vertices positions



$\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ – inner vertices; $\boldsymbol{v}_n, \boldsymbol{v}_{n+1}, \ldots, \boldsymbol{v}_N$ – boundary vertices

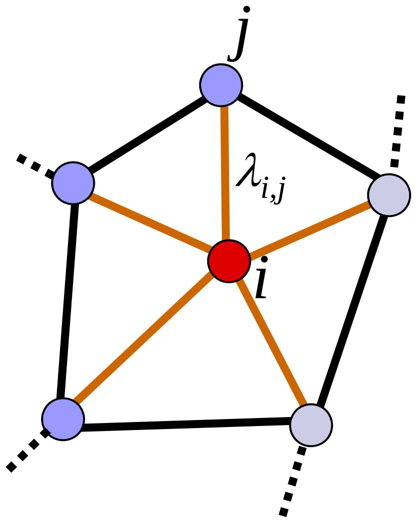
Inner vertices

- We constrain each inner vertex to be a weighted average of its neighbors:

$$\boxed{\boldsymbol{v}_i = \sum_{j \in N(i)} \lambda_{i,j} \boldsymbol{v}_j, \quad i = 1, 2, \ldots, n}$$

$$\lambda_{i,j} = \begin{cases} 0 & i, j \ \text{ are not neighbors} \\ > 0 & (i, j) \in E \ \text{ (neighbours)} \end{cases}$$

$$\sum_{j \in N(i)} \lambda_{i,j} = 1$$



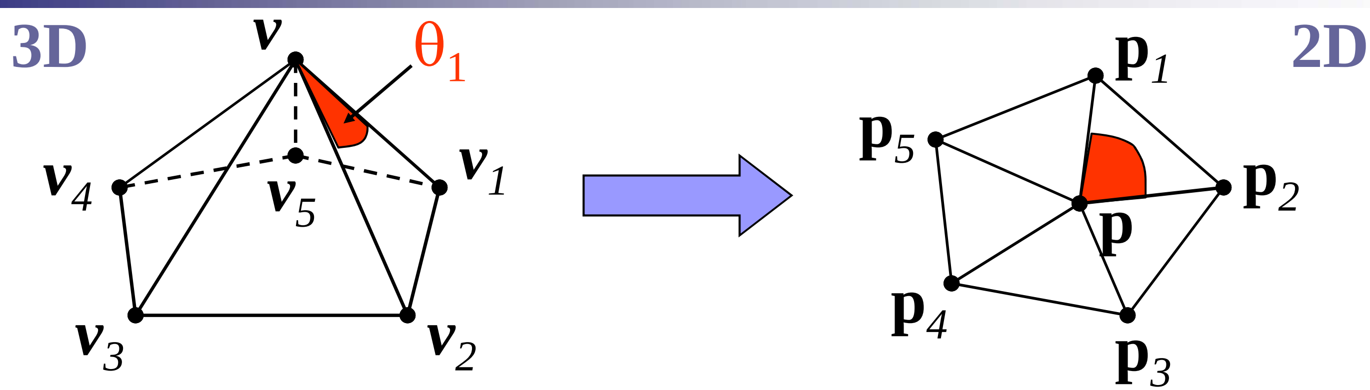
Linear system of equations

$$\boldsymbol{v}_i - \sum_{j \in N(i)} \lambda_{i,j} \boldsymbol{v}_j = 0, \quad i = 1, 2, \ldots, n$$

$$\boldsymbol{v}_i - \sum_{j \in N(i) \setminus B} \lambda_{i,j} \boldsymbol{v}_j = \sum_{k \in N(i) \cap B} \lambda_{i,k} \boldsymbol{v}_k, \quad i = 1, 2, \ldots, n$$

$$\begin{pmatrix} 1 & & -\lambda_{1,j_1} & & -\lambda_{1,j_{d1}} \\ & 1 & & & \\ & & 1 & & \\ & -\lambda_{4,j_1} & & \ddots & \\ & & -\lambda_{n,j_5} & & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \\ \vdots \\ \boldsymbol{v}_n \end{pmatrix} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \\ \\ \sigma_n \end{pmatrix}$$

Shape preserving weights



To compute $\lambda_1, \ldots, \lambda_5$, a local embedding of the patch is found:

1) $\| \mathbf{p}_i - \mathbf{p} \| = \| \mathbf{x}_i - \mathbf{x} \|$

2) $angle(\mathbf{p}_i, \mathbf{p}, \mathbf{p}_{i+1}) = (2\pi / \Sigma \theta_i)\ angle(\mathbf{v}_i, \mathbf{v}, \mathbf{v}_{i+1})$

$$\exists\, \lambda_i\,, \begin{cases} \mathbf{p} = \Sigma\, \lambda_i\, \mathbf{p}_i \\ \lambda_i > 0 \\ \Sigma\, \lambda_i = 1 \end{cases} \quad \Rightarrow$$ use these λ as edge weights.

Linear system of equations

- A unique solution always exists
- Important: the solution is legal (bijective)

- The system is sparse, thus fast numerical solution is possible
- Numerical problems (because the vertices in the middle might get very dense…)

Harmonic mapping

- Another way to find inner vertices
- Strives to preserve angles (conformal)
- We treat the mesh as a system of springs.
- Define spring energy:

$$\boxed{E_{harm} = \frac{1}{2} \sum_{(i,j)\in E} k_{i,j} \left\| \boldsymbol{v}_i - \boldsymbol{v}_j \right\|^2}$$

where $\boldsymbol{v}_i$ are the flat position (remember that the boundary vertices $\boldsymbol{v}_n, \boldsymbol{v}_{n+1}, \ldots, \boldsymbol{v}_N$ are constrained).

Energy minimization – least squares

- We want to find such flat positions that the energy is as small as possible.
- Solve the linear least squares problem!

$$\boldsymbol{v}_i = (x_i, y_i)$$

$$E_{harm}(x_1,\ldots,x_n,y_1,\ldots,y_n) = \frac{1}{2}\sum_{(i,j)\in E} k_{i,j}\left\|\boldsymbol{v}_i - \boldsymbol{v}_j\right\|^2 =$$

$$= \frac{1}{2}\sum_{(i,j)\in E} k_{i,j}\left((x_i - x_j)^2 + (y_i - y_j)^2\right).$$

E_{harm} is function of 2n variables

Energy minimization – least squares

- To find minimum: $\nabla E_{harm} = 0$

$$\boxed{\begin{aligned} \frac{\partial}{\partial x_i} E_{harm} &= \frac{1}{2} \sum_{j \in N(i)} 2 k_{i,j} (x_i - x_j) = 0 \\ \frac{\partial}{\partial y_i} E_{harm} &= \frac{1}{2} \sum_{j \in N(i)} 2 k_{i,j} (y_i - y_j) = 0 \end{aligned}}$$

- Again, $x_{n+1}, \ldots, x_N$ and $y_{n+1}, \ldots, y_N$ are constrained.

Energy minimization – least squares

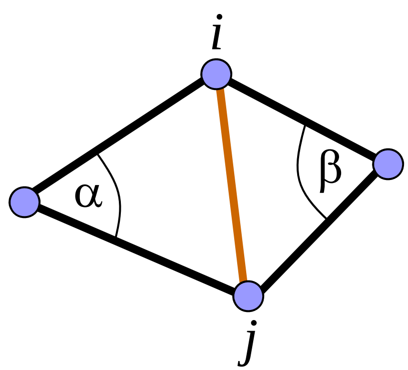
- To find minimum: $\nabla E_{harm} = 0$

$$\boxed{\begin{aligned} &\sum_{j \in N(i)} k_{i,j}(x_i - x_j) = 0, \quad i = 1, 2, \ldots, n \\ &\sum_{j \in N(i)} k_{i,j}(y_i - y_j) = 0, \quad i = 1, 2, \ldots, n \end{aligned}}$$

- Again, $x_{n+1},\ldots, x_N$ and $y_{n+1}, \ldots, y_N$ are constrained.

The spring constants $k_{i,j}$

- The weights $k_{i,j}$ are chosen to minimize angles distortion:
  - Look at the edge (i, j) in the 3D mesh
  - Set the weight $\boxed{k_{i,j} = \cot \alpha + \cot \beta}$



3D

Discussion

- The results of harmonic mapping are better than those of convex mapping (local area and angles preservation).
- But: the mapping is not always legal (the weights can be negative for badly-shaped triangles…)
- Both mappings have the problem of fixed boundary – it constrains the minimization and causes distortion.
- There are more advanced methods that do not require boundary conditions.

Angle-based Flattening (ABF)

[Sheffer and de Sturler 2001]

- Angle-preserving parameterization
- The energy functional is formulated using the flat mesh angles only!
- Allows free boundary

Angle-based Flattening (ABF)

[Sheffer and de Sturler 2001]

- The goal: minimize the difference

$$\sum_{i=1}^{N} \left(\alpha_i - \beta_i\right)^2$$

where β_i are angles of original (3D) mesh and α_i are the unknowns (the flat mesh)

The angles equations (constraints)

All angles are positive:

$$\alpha_i > 0 \quad (1)$$

Angles around an inner vertex in 2D sum up to 2π

$$\sum_{j \text{ around } i} \alpha_j = 2\pi \quad (2)$$

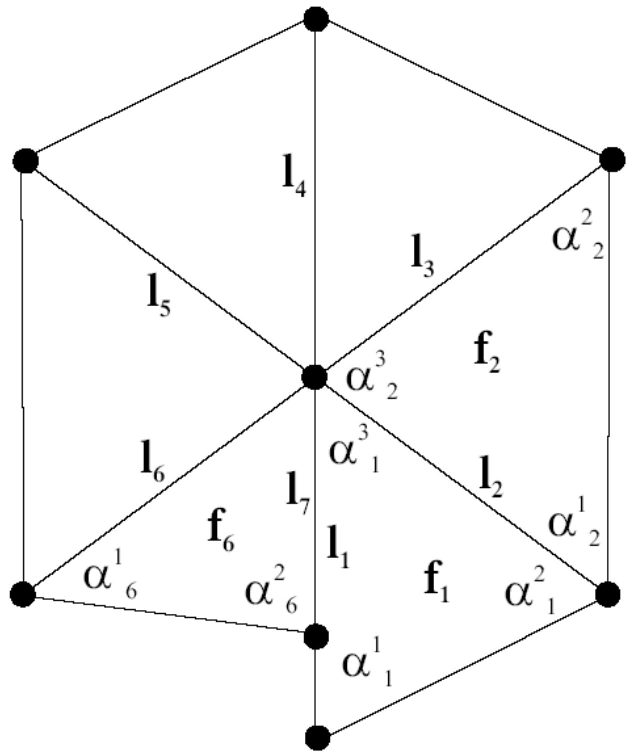
Angles in a triangle sum up to π

$$a_{i_1} + a_{i_2} + a_{i_3} = \pi \quad (3)$$

The angles equations (constraints)

- Finally, something like the sine theorem must hold:

$$(4) \qquad \frac{\prod_{j=1}^{N(i)} \sin \alpha_j}{\prod_{j=1}^{N(i)} \sin \tilde{\alpha}_j} = 1$$



The final optimization:

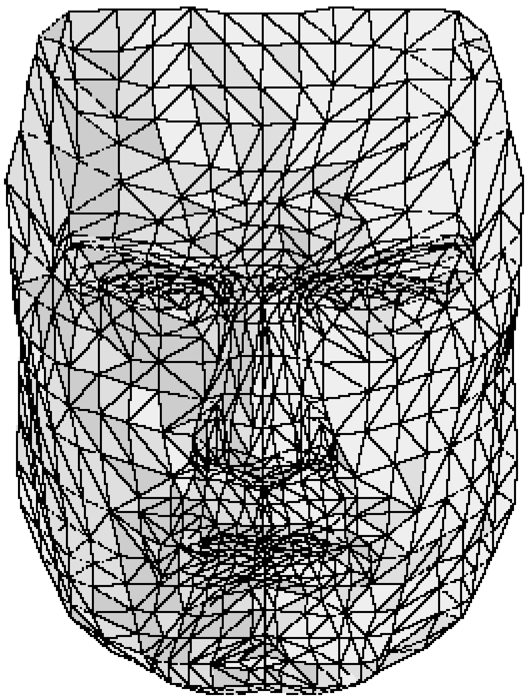
- We minimize

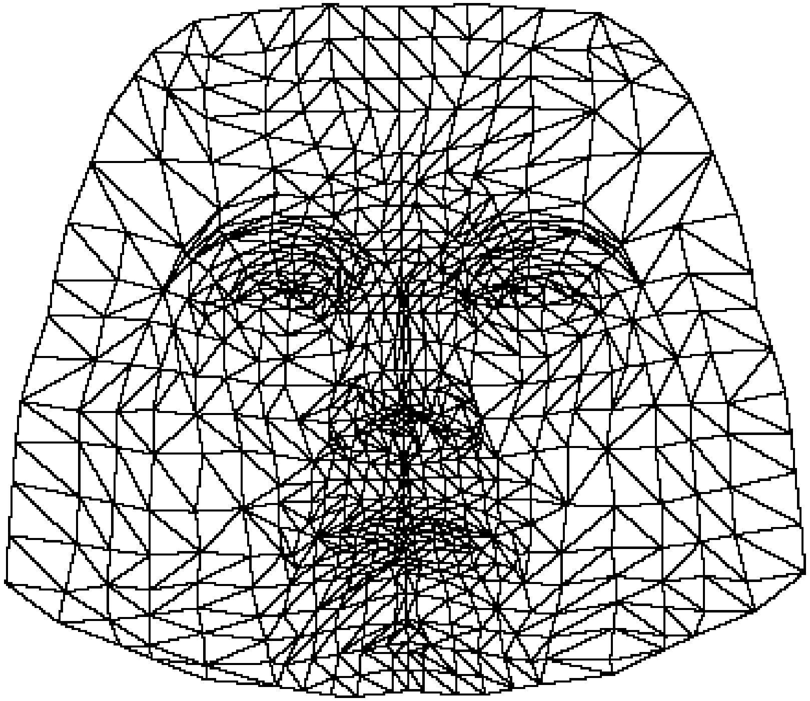
$$\sum_{i=1}^{N} \left(\alpha_i - \beta_i\right)^2$$

under the 4 constraints

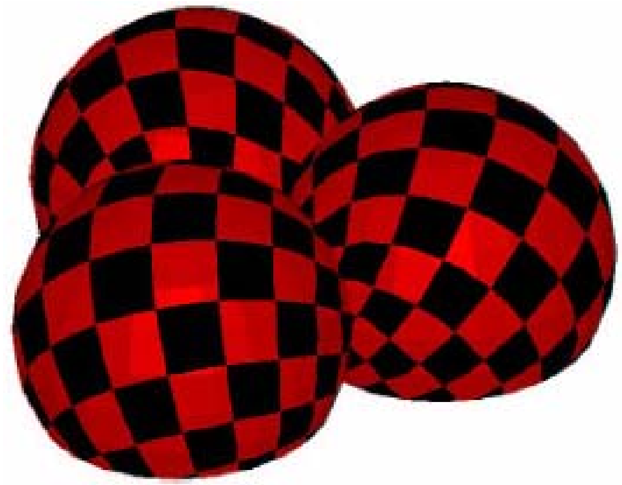
- It's enough to fix one triangle in the plane to define the whole flat mesh

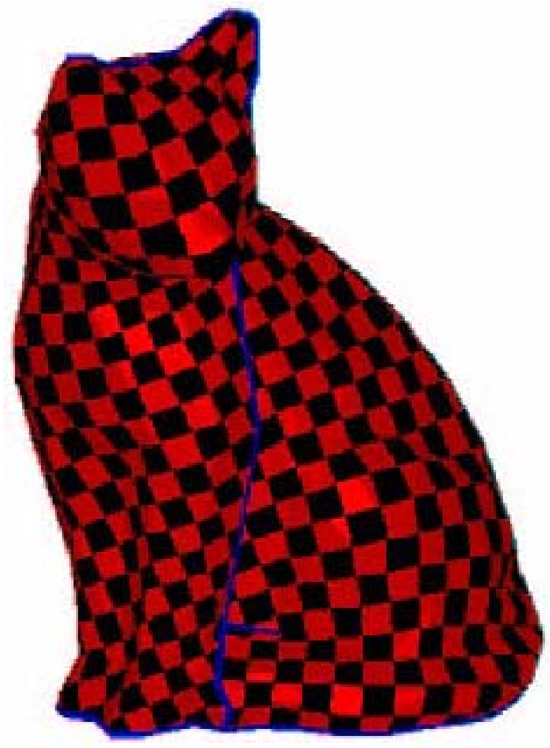
Results



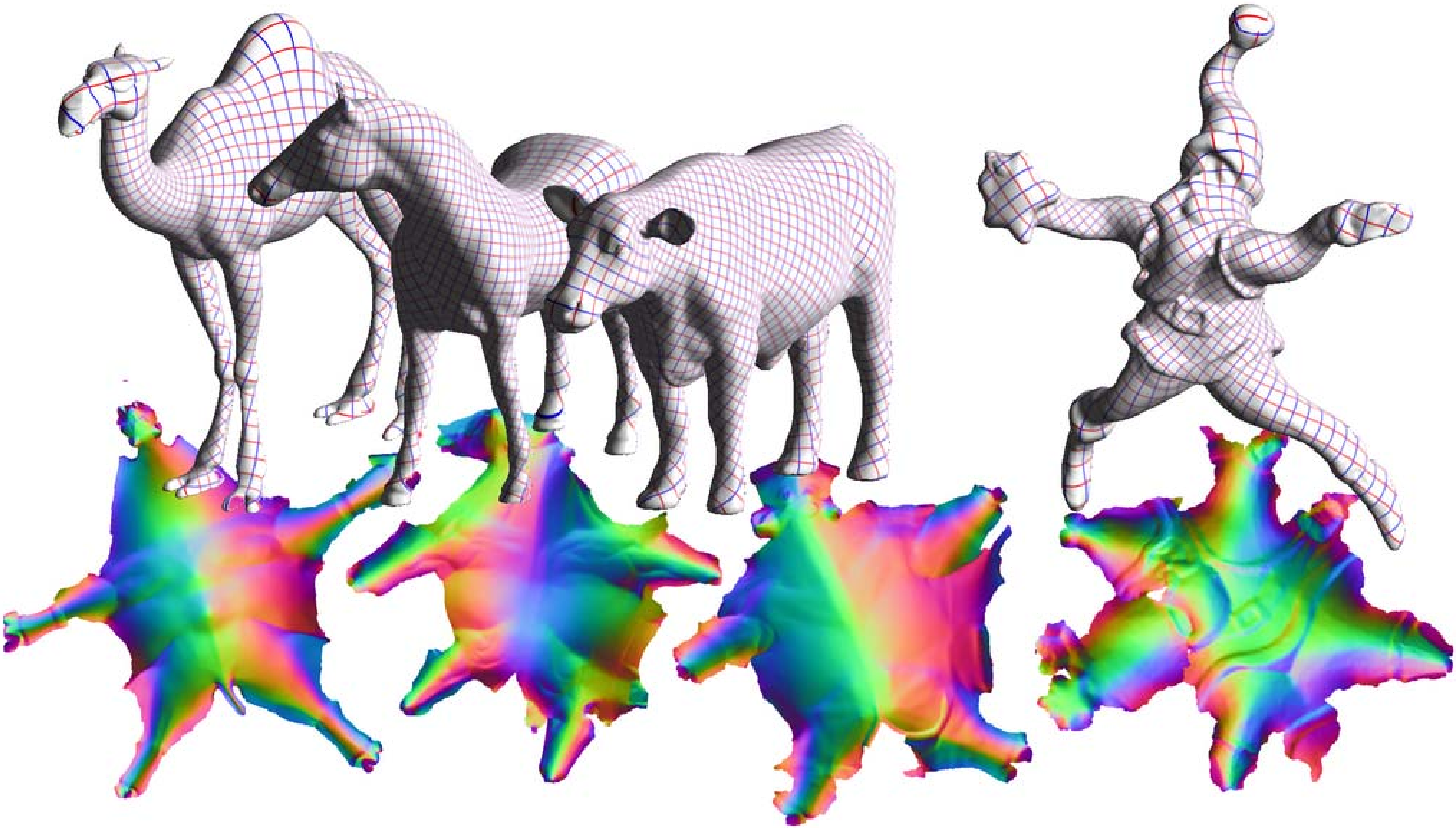


Results

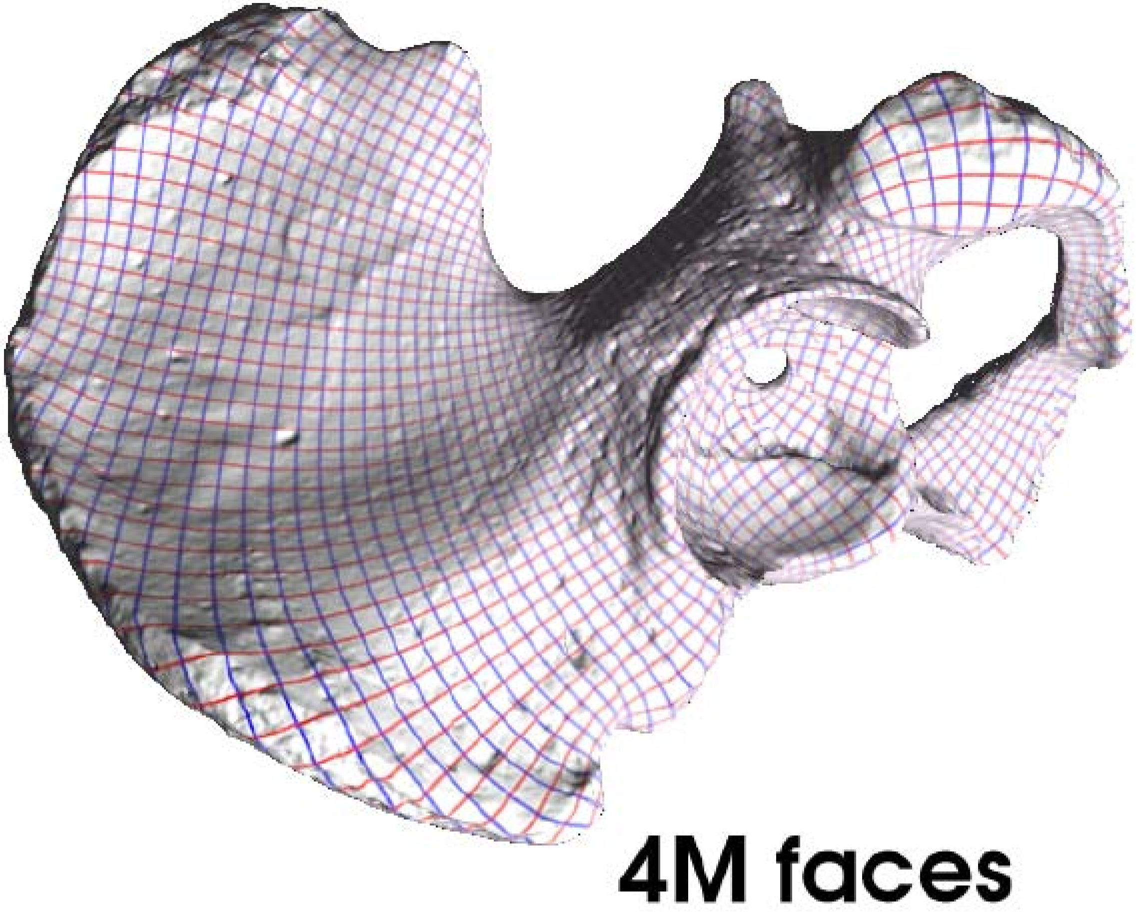




Results



Results



Discussion

- Pros:
  - Angle preserving
  - Always valid (at least internally)
  - No rigid boundary constraints
- Cons:
  - Non-linear optimization
    - Expensive (but now a multi-grid method exists)
  - Building the mesh from angles can be unstable

Thanks