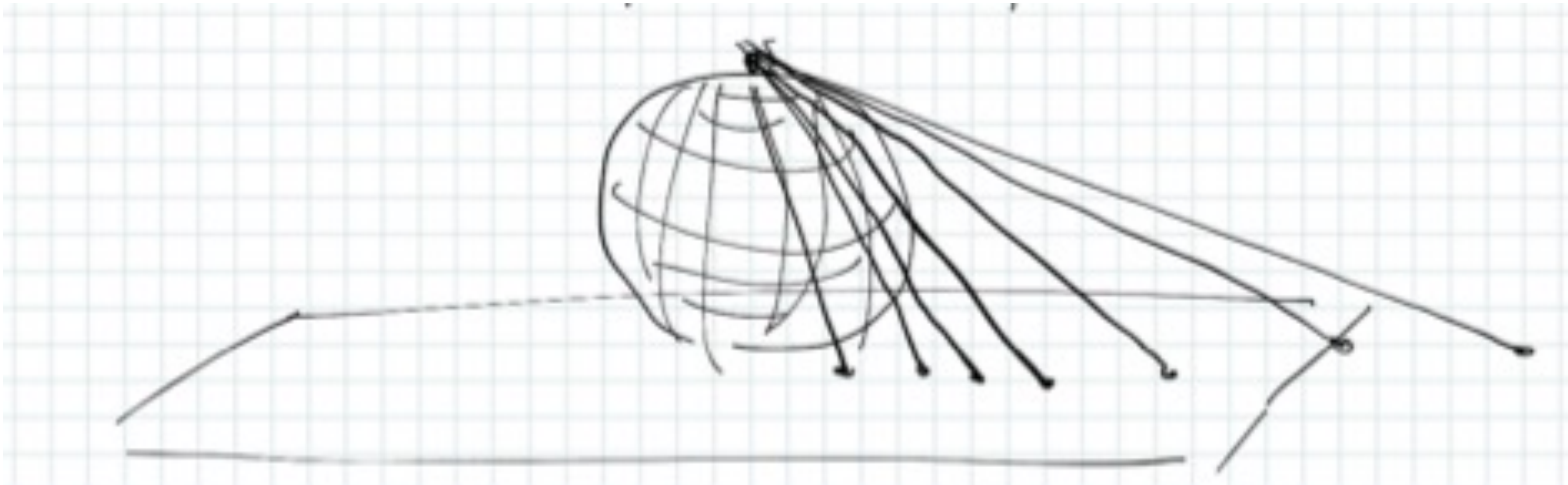
09: Texture Parametrisation, Synthesis & Morphing

Texture Parametrisation

- Assume you don’t have u,v coordinates
- For example, when beginning modelling
- Genus 0 is a sphere – use cartography
- Genus 1 is a torus – double wraparound
- Higher genus is more complex

Spherical Parameterisation



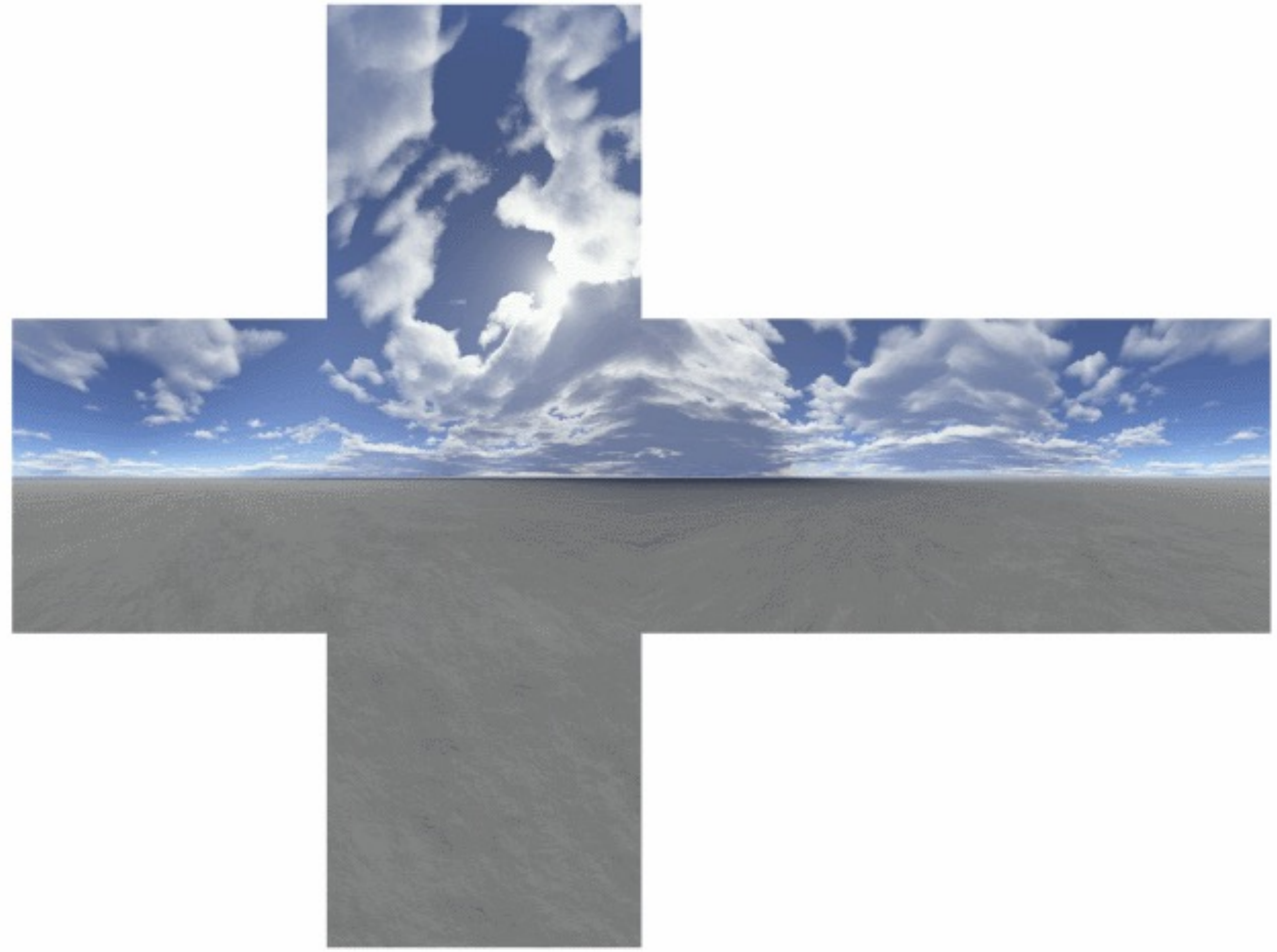
- Place sphere on the texture plane
- Draw ray from N Pole through each point p
- Find u,v coordinates at intersection w/ plane
- N Pole maps to infinity w/ bad distortion

Spherical Parameterisation

From wikipedia

- Latitude / Longitude
- Distorts infinitely near both poles
- Interpolation becomes a problem

Cube Map



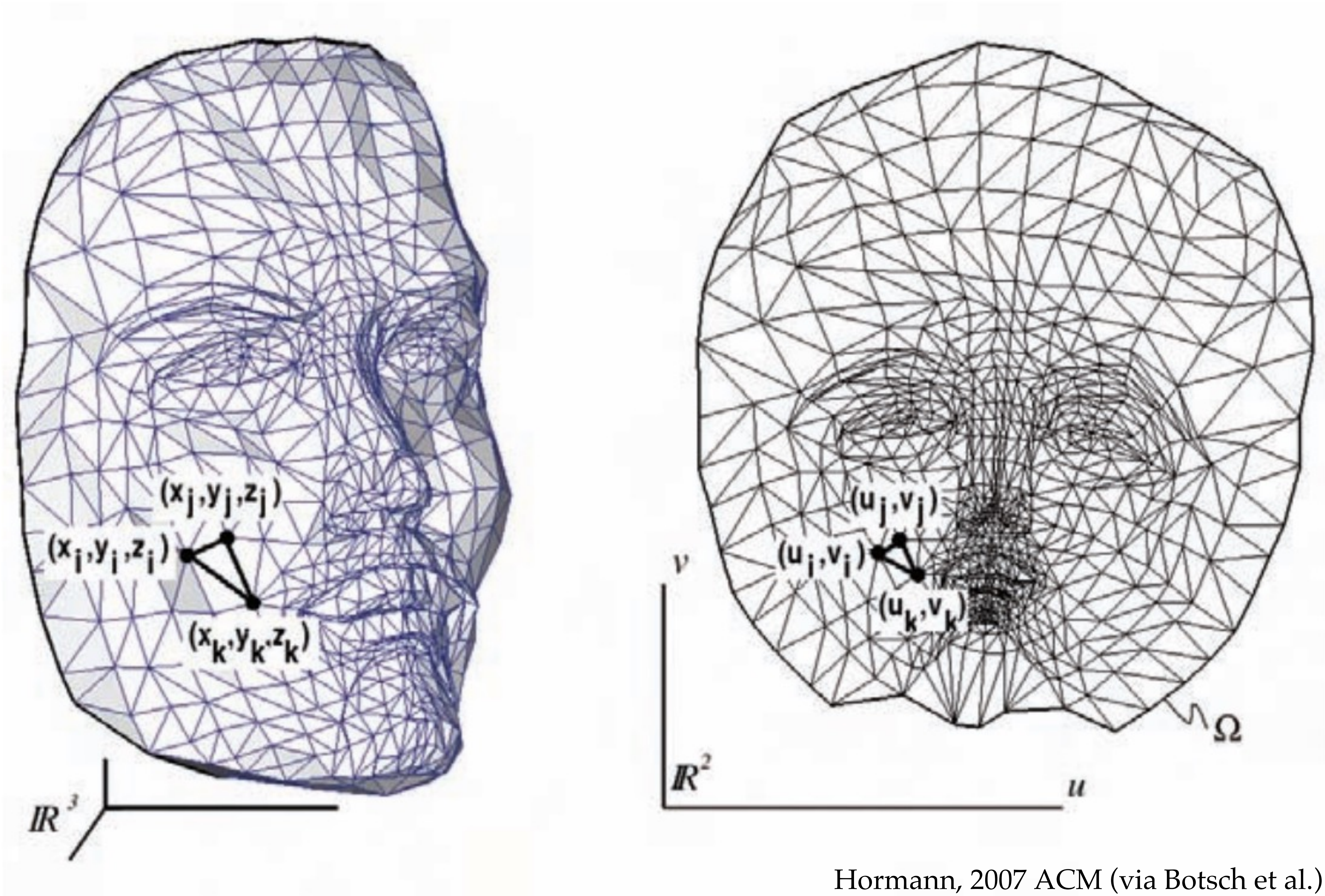
From Epic Games forum

- Use half of the texture space
- Often used for skyboxes
- But limits distortion

Practical Parameterisation

- Separate surface into connected components
- Separate each surface into patches
  1. Pay an artist to assign u,v coords
  2. Cut the surface into patches – i.e. artist
  3. Expand random vertices until patches meet
- Then assign u,v coordinates inside patches
  - But how?

An Example



Hormann, 2007 ACM (via Botsch et al.)

Assumptions

- We want to parameterise a patch
- Formally, patch is homeomorphic to a disk
- Treat it as a function $f: \mathbb{R}^3 \rightarrow \mathbb{R}^2$
- We need an input $\{\boldsymbol{x}_i \in \mathbb{R}^3\}$ in a mesh
  - Barycentric interpolation on triangles
- Output is texture coordinates $\{\boldsymbol{u}_i \in \Omega\}$

Theorem (Tutte, 1960)

- Given a triangle mesh homeomorphic to a disk
  - We can enforce this with half-edge
- With a convex boundary polygon
  - We can choose this
- If the coordinates of each interior vertex are
- A convex combination of their neighbours
- Then you have a valid parameterisation

Setup

- We want a convex combination of neighbours:

$\boldsymbol{u}_i = \sum_{j \in N_1(v_i)} a_{ij} \boldsymbol{u}_j$ (sum over 1-ring)

$0 \leq a_{ij} \leq 1$ (convex coordinates)

$\sum_{j \in N_1(v_i)} a_{ij} = 1$ (weights sum to 1)

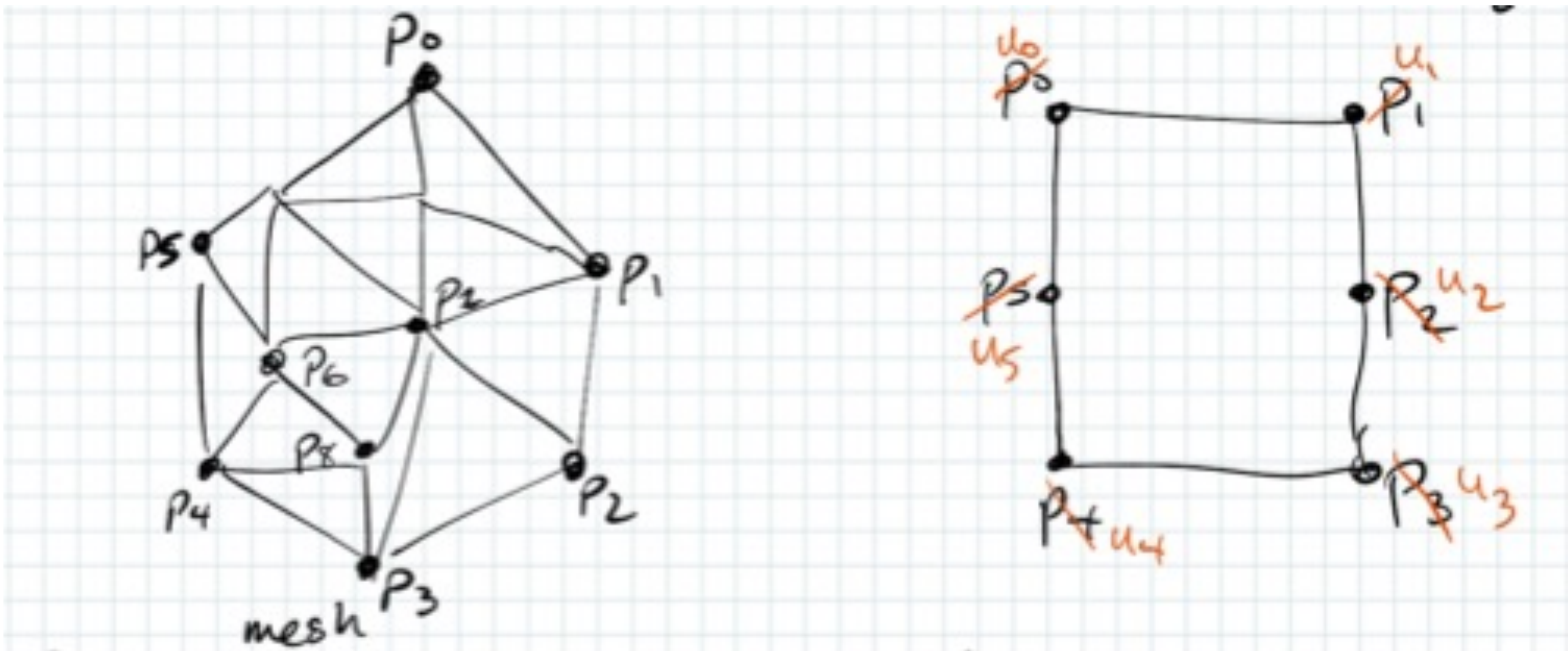
$a_{ij} = 0$ if $j \notin N_1(v_i)$ (neighbours only)

$a_{ii} = - \sum_{j \in N_1(v_i)} a_{ij}$ (so $\sum_j a_{ij} = 0$)

Setting Coefficients

- What weights should we use?
  - How about 1 for every neighbour
  - Divided by vertex degree
  - Notice that $a_{ij} \neq a_{ji}$ in general
- Each vertex is at the barycentre of its 1-ring
- And we can compute it iteratively
  - Technically, Gauss-Seidel solver

Floater's Algorithm



- Choose an exterior face
- Lay out exterior polygon, choosing $\boldsymbol{u}_i$
- Then iterate the remaining vertices

Interior / Exterior Vertices

- Put the exterior vertices at the beginning
  - $\boldsymbol{u}_0 \cdots \boldsymbol{u}_{b-1}$ are treated as constants
- And the interior vertices at the end
  - $\boldsymbol{u}_b \cdots \boldsymbol{u}_{n-1}$ are treated as variables

$$\sum_{j=b}^{n-1} a_{ij}\boldsymbol{u}_j = -\sum_{j=0}^{b-1} a_{ij}\boldsymbol{u}_j \quad \text{For all i≥b}$$

- Rewrite as $A\boldsymbol{u} = \boldsymbol{v}$ and solve

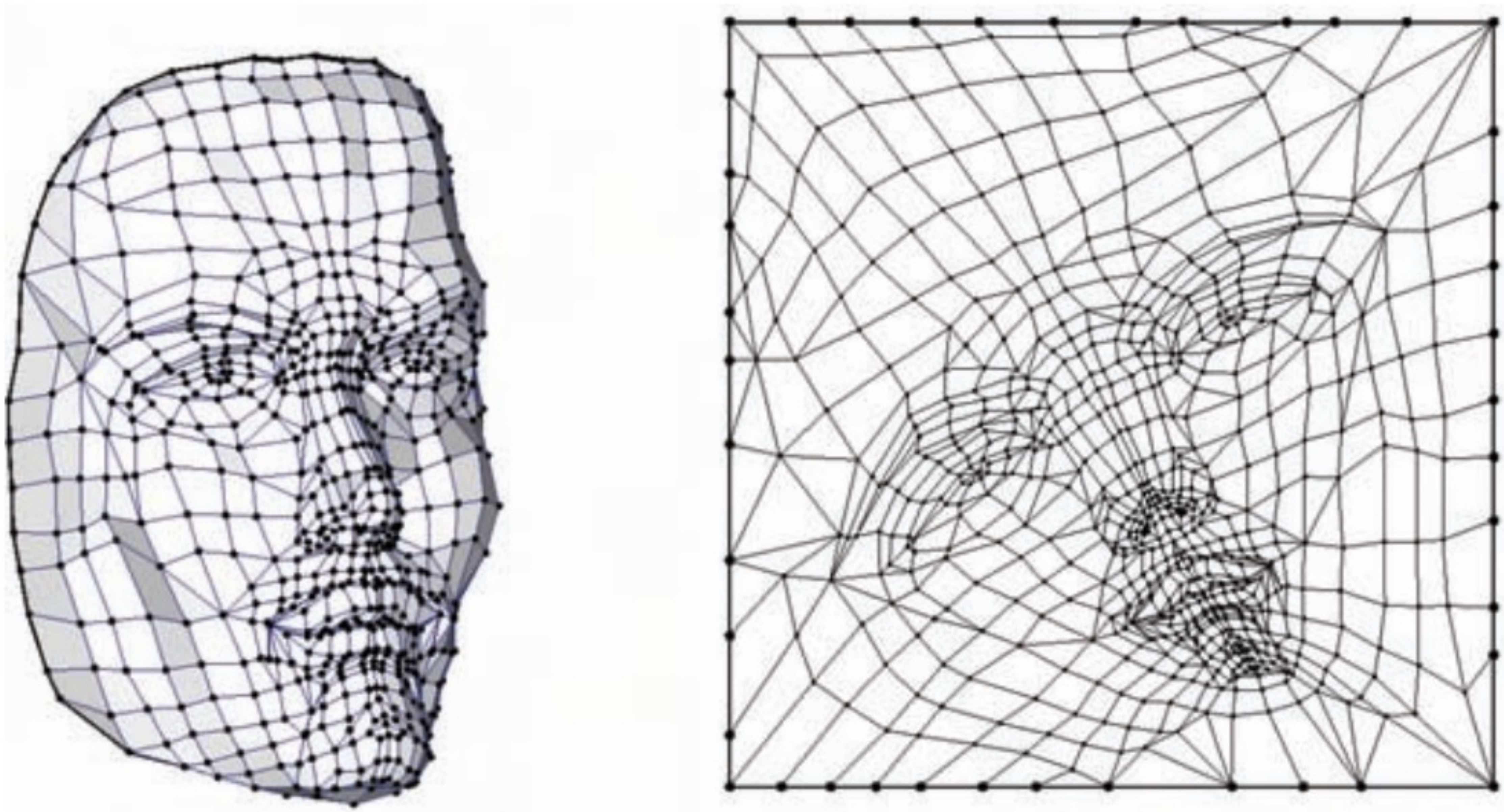
Shortcut

- Set all interior points to the centre
- Then iterate as many times as necessary
  - Recompute each interior $\boldsymbol{u}_i$ from its 1-ring
  - Boundary vertices stay fixed
- Works for up to n = 5000 vertices or so

```
while (not done)
```

$$\boldsymbol{u}_i \leftarrow \frac{1}{-a_{ii}} \sum_{j \in N_1(v_i)} a_{ij} \boldsymbol{u}_j$$

Result



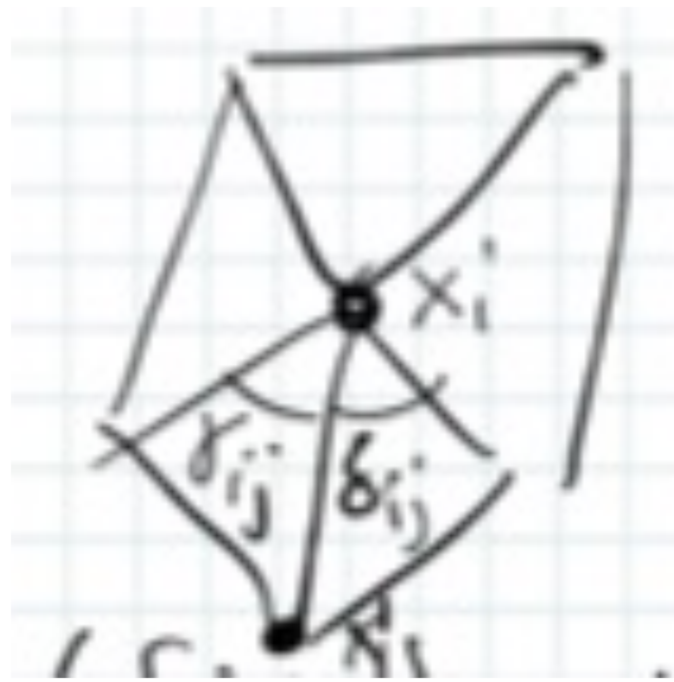
Hormann, 2007 ACM (via Botsch et al.)

- Works well, but distorts texture

Weight Choices

- Uniform weights cause this distortion
- Cotangent weights can be negative
  - Unless you guarantee no triangles are obtuse
  - Or you subdivide
  - Or you add another hack
    - Such as triangle area or perimeter angle

Floater, 2003



- Substitute something similar to cotangent
  - $a_{ij} = \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}\left(tan\frac{\delta_{ij}}{2} + tan\frac{\gamma_{ij}}{2}\right)$
- This works OK in practice, is always positive
- And there are many other hacks

UNIVERSITY OF LEEDS

Problems

- While Floater's algorithm works, it distorts
- Instead, consider the ellipse of anisotropy
  - Based on eigenvectors of the Jacobian matrix
- We can compute this as a function of $\boldsymbol{u}_i$
- Apply our favourite optimization technique
- **Least Squares Conformal Mapping** (LSCM)
  - Implemented in Blender
  - But still has distortion

Texture Synthesis

- Given a mesh M with
  - Vertex positions P
  - Texture coordinates U
  - Attribute values A
- Render M in 2D
  - Using U as vertex positions, not P
- And the texture will hold the attribute

Deformation & Morphing

- Two related problems
- Deformation – used for modelling
- Morphing – used for animation
- Both require mapping x,y,z meshes to u,v
  - But not necessarily u,v you start with

Deformation Modelling

- Artist "grabs" a point on a surface
  - We need to discuss how to do this
- And drags it to modify shape
- So, given (x,y) in screen space
  - Find corresponding (u,v) in texture space
  - And find closest vertex to drag

Picking Points in 3D

1. Raytrace & find intersection (slow)
2. Specialised render code (no-one uses this)
3. Back-buffer hack:
    a. Render in false colour (u,v -> RGB)
    b. Read the pixel under the mouse
    c. Now you know the u,v coordinates

Deformation

- We've found a vertex
- What direction do we drag?
  - Perpendicular to surface
- Typically also affects 1-ring or 2-ring
- Sometimes a circular area in texture cords
- How much do they move?
- Gaussian weighting based on distance

Morphing

- Given a surface S_1 at time t_1
- And a surface S_2 at time t_2
- Construct intermediate surfaces to animate
- But no guarantees that vertices match
  - So picking nearest vertex won't work
  - No guarantees that it's 1-1 and onto

Solution

- Generate an intermediate surface
- Then morph twice in the (u,v) space
- Store (x,y,z) for each surface in a texture map
- Find (u,v) coordinates for each vertex in S_2
- I.e we know $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$
- We need $\boldsymbol{x}$ for an arbitrary $\boldsymbol{u}$

Rasterise to Texture

- Render triangles to texture domain Ω
- I.e. use (u,v) as (x,y) and (x,y,z) as (r,g,b)
- Now we have a valid texture
  - And $\boldsymbol{u}_i$ maps to $\boldsymbol{x}_i$
- Then all we have to do is generate keyframes
  - Which means talk to He

Intermediate Surface

- Generate a distance field
  - Choose the medial axis (half-way points)
- {Artist, Algorithm} chooses a correspondence
  - By identifying landmarks / features
  - And pairing them up
- Many, many hacks & heuristics