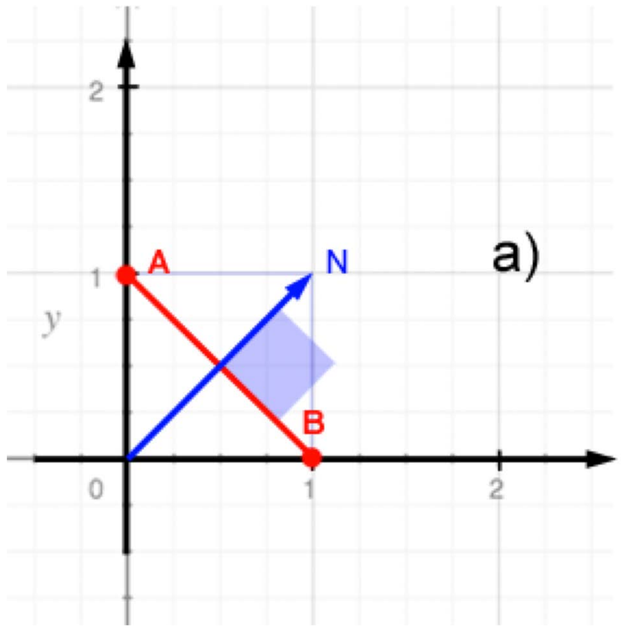
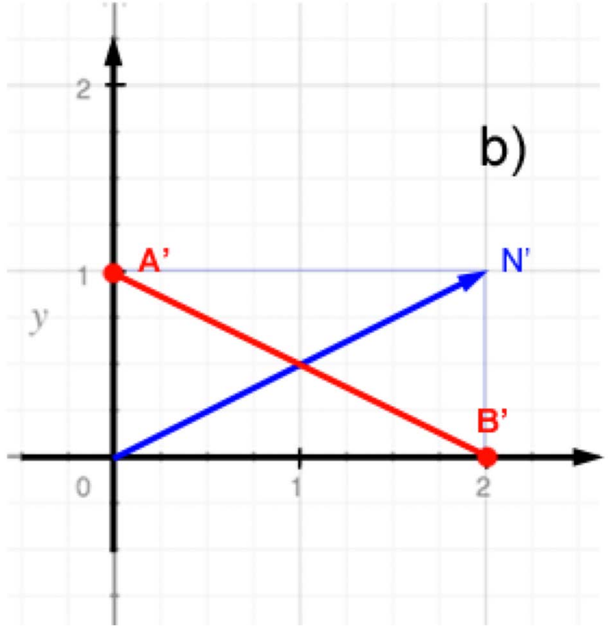
15-Texture Mapping

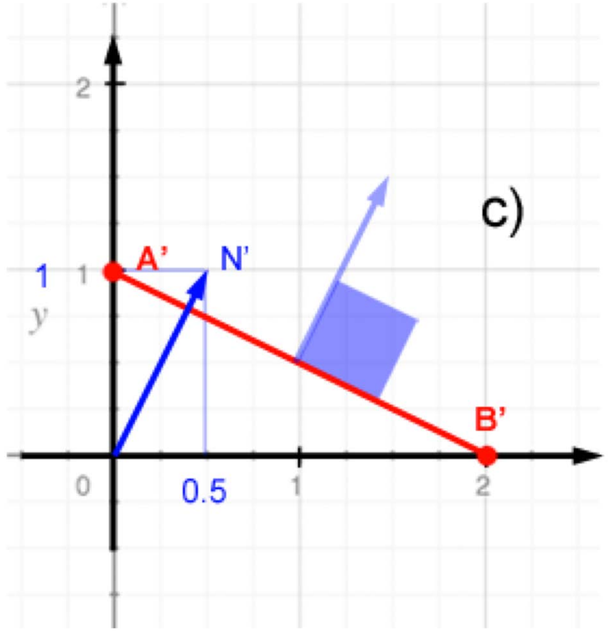
Acknowledgement: Daniele Panozzo

A note on transforming normals

- If you transform a point **v** with a matrix M: **v’** = **Mv** …
- the transformed normal **n’** at the point **v** is **n’ = M^{-T} n**



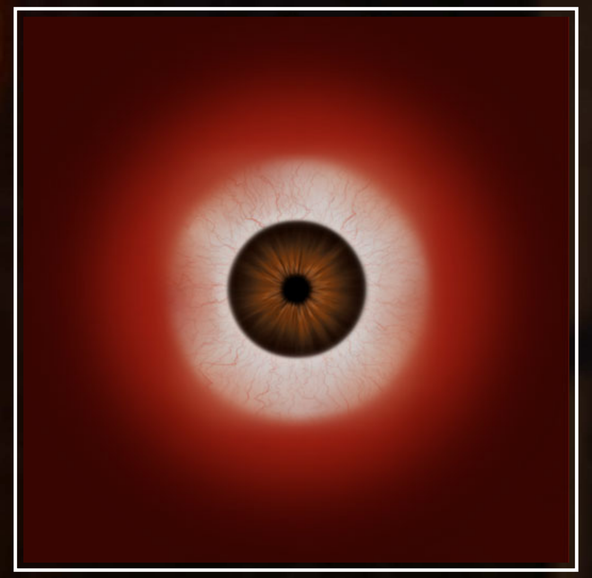


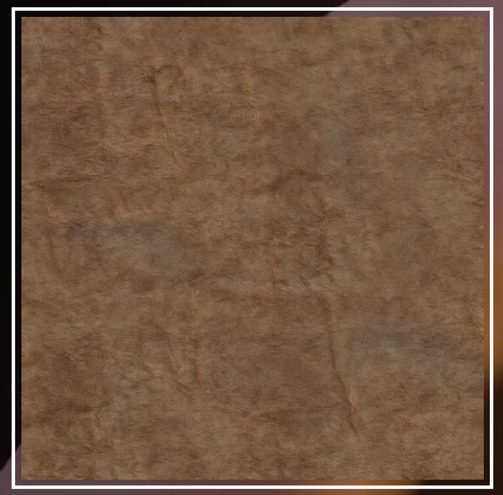


http://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/transforming-normals

15-Texture Mapping

Sintel
Blender Open Movie

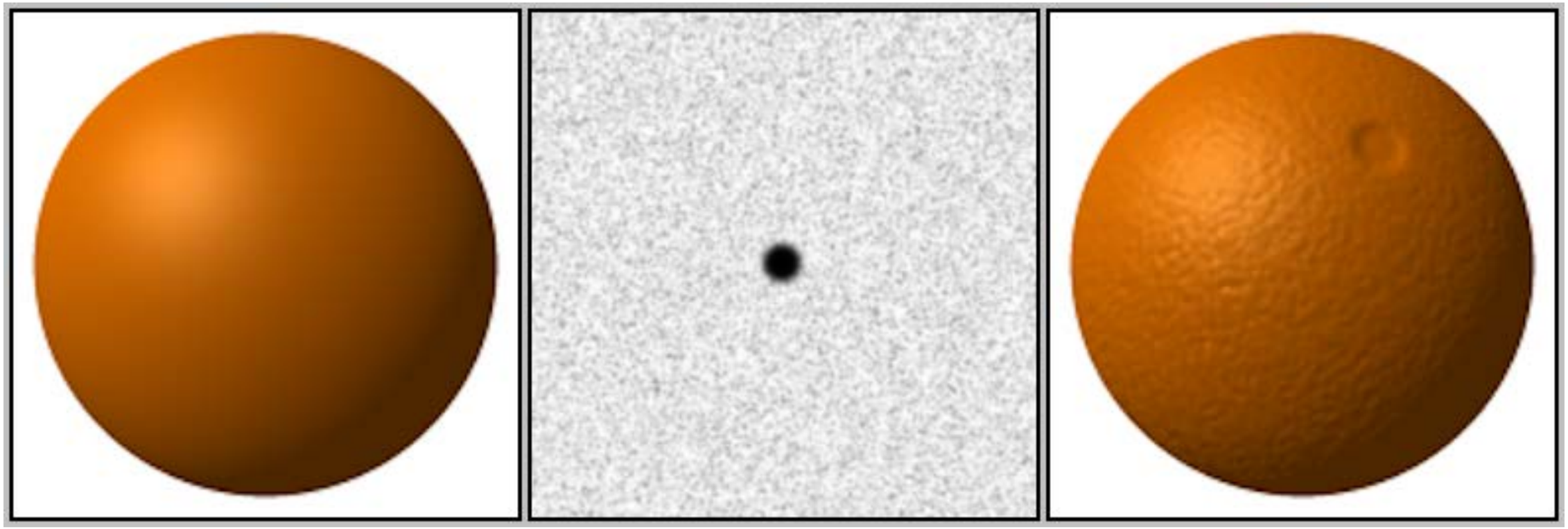




Sintel
Blender Open Movie

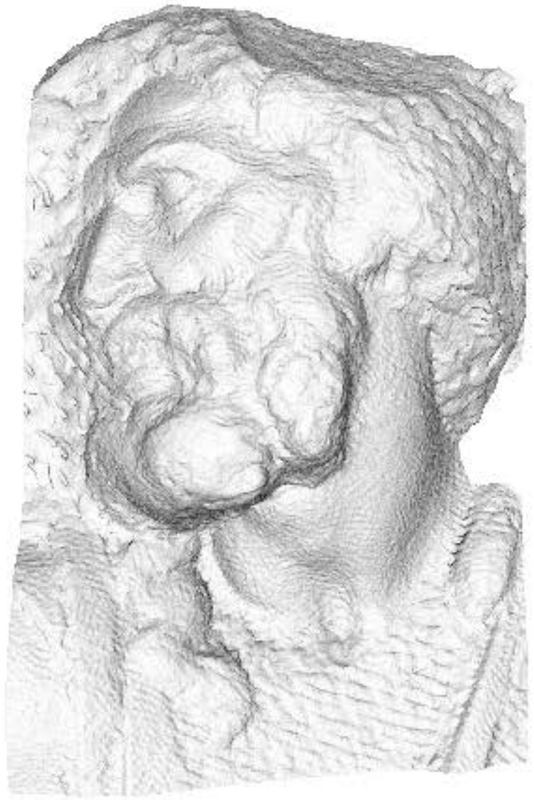
Bump Mapping

Instead of encoding colors in a texture, you encode normals!

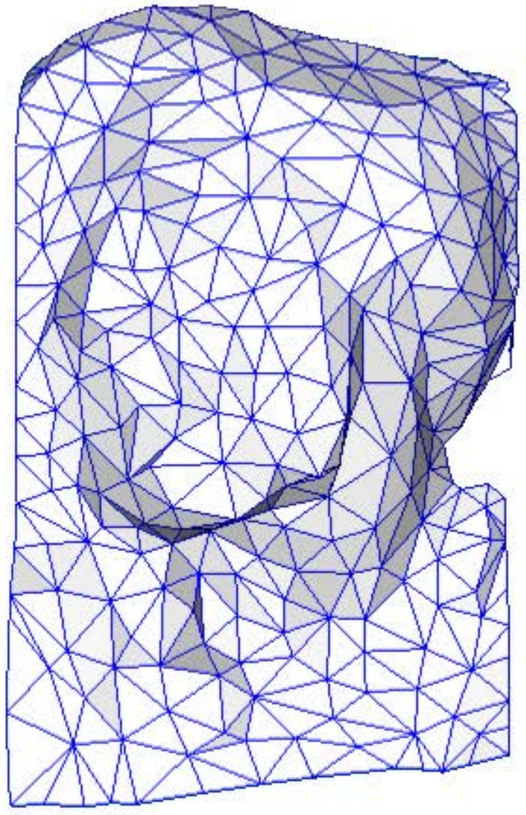


By Bump-map-demo-smooth.png, Orange-bumpmap.png and Bump-map-demo-bumpy.png: Original uploader was Brion VIBBER at en.wikipediaLater version(s)

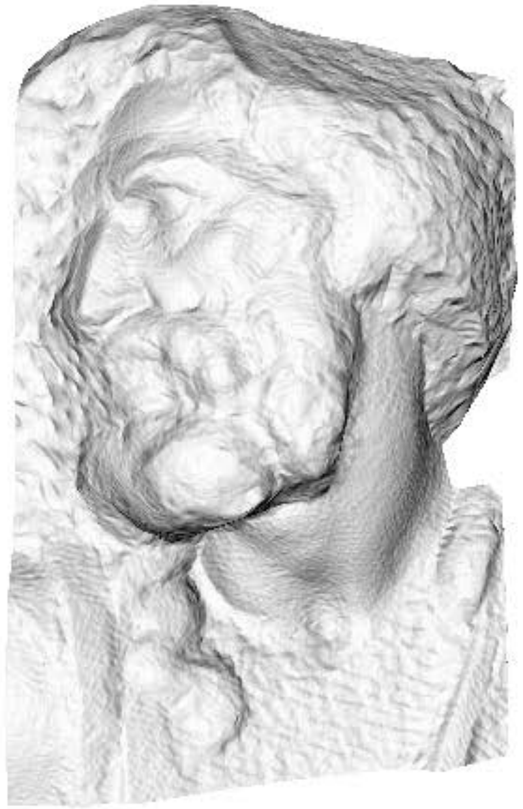
Normal/Bump Mapping



original mesh
4M triangles



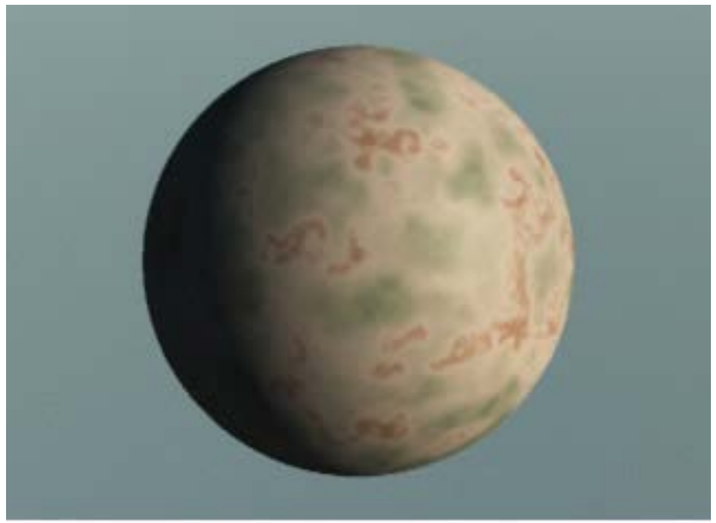
simplified mesh
500 triangles



simplified mesh
and normal mapping
500 triangles

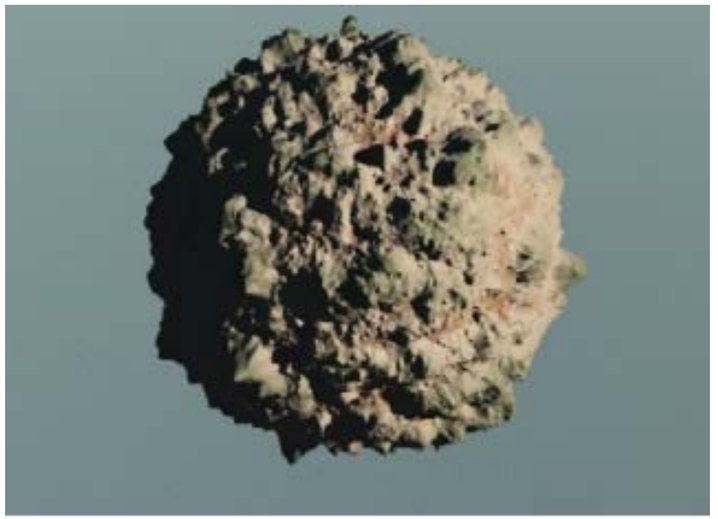
Displacement Mapping

Instead of normals, you encode a displacement.



Original

Bump Mapping



Displacement Mapping

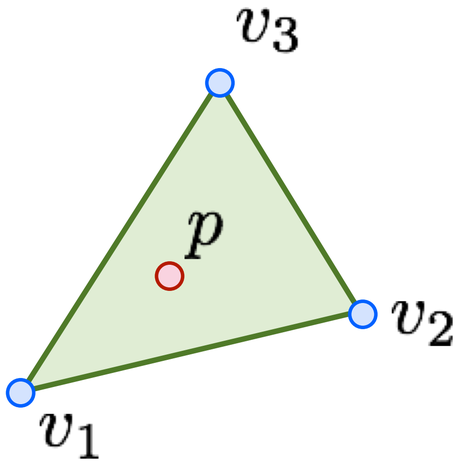
Image courtesy of: http://www.chromesphere.com/Tutorials/Vue6/Optics-Basic.ht

Texture Mapping

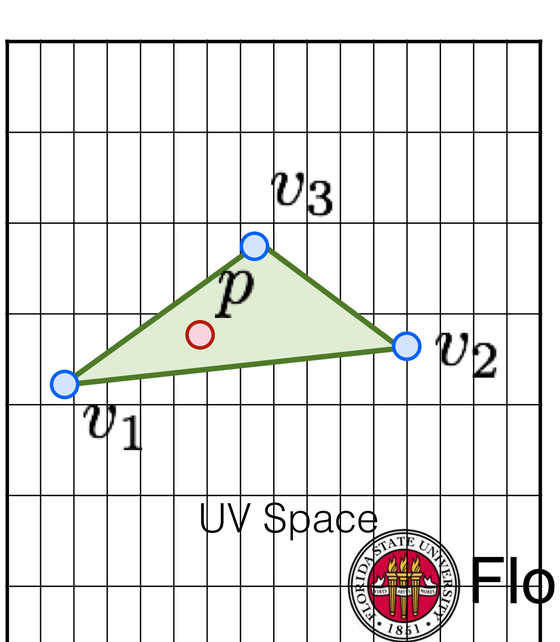
- The idea is the same. Instead of encoding values at vertices of triangles, you encode them in images.
- You gain all the advantages of images (easy to store, compress, interpolate, sample)
- You can encode any property that you want, the most common are:
  - Colors (Texture Mapping)
  - Normals (Bump Mapping)
  - Displacements (Displacement Mapping)

What do you need?

- One additional per-vertex property, the UV coordinates
- An image uploaded to the GPU memory (2D texture)
- The UV coordinates are interpolated inside each triangle, and used to find the corresponding value in the texture
- The texture value is interpolated before it is used in the shader

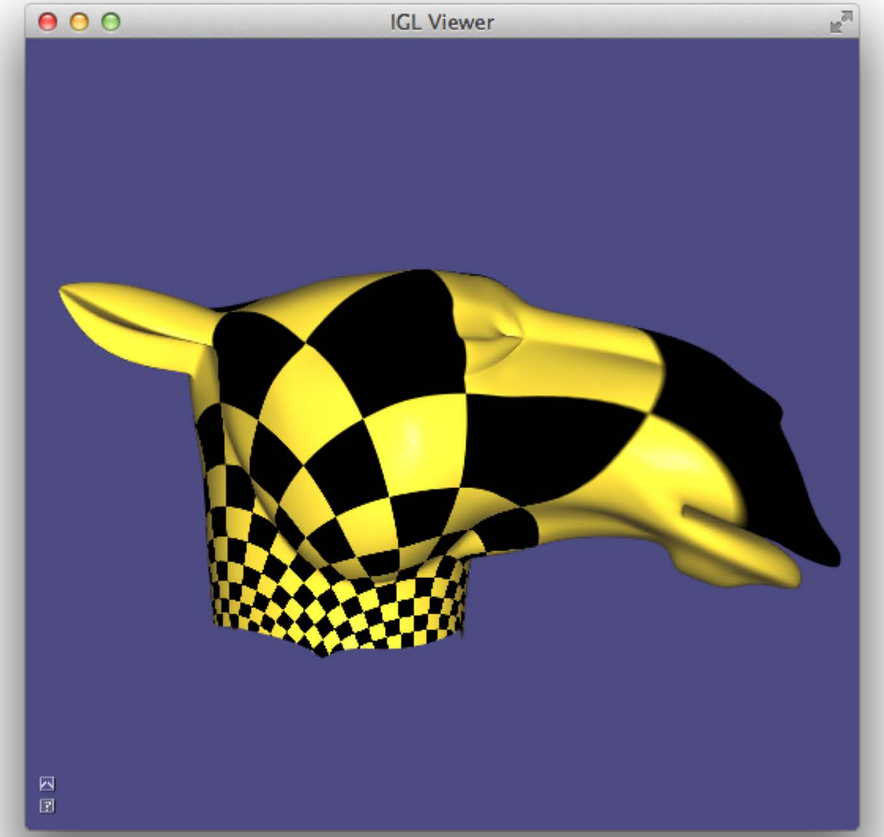


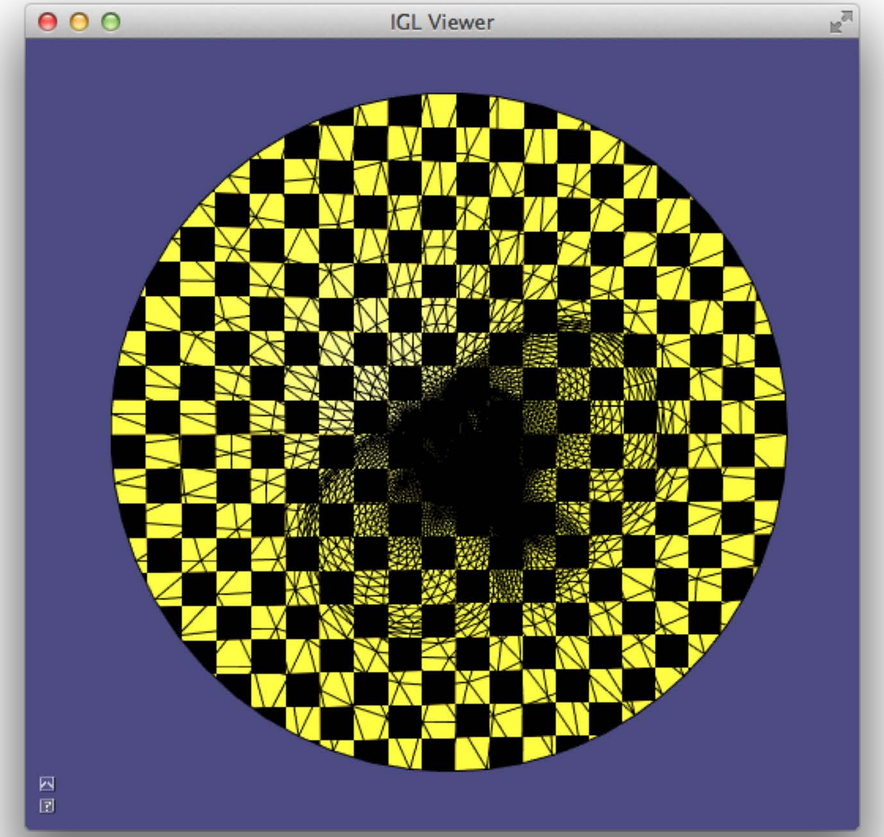
World Coordinates



UV Space

Checkerboards are great to visualize a UV map



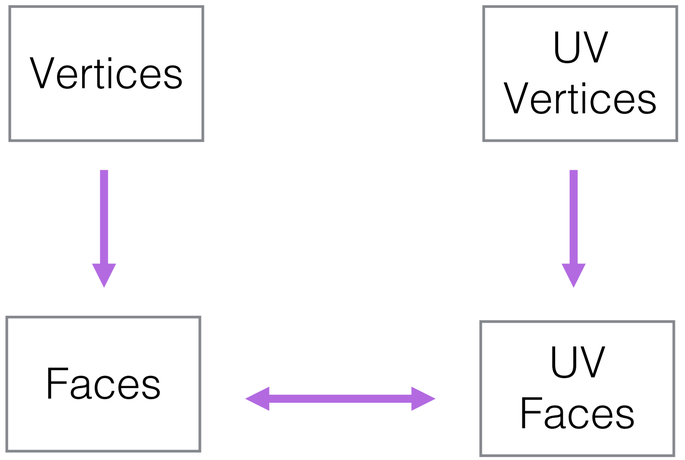


“Seams” are needed for complex objects

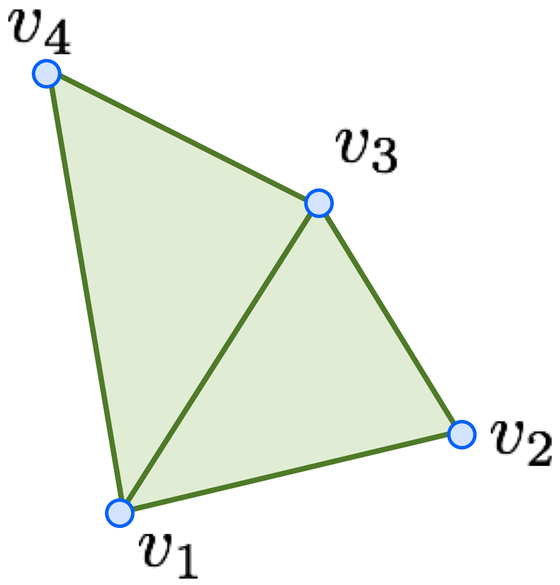
Image from Vallet and Levy, techreport INRIA

How are UV maps encoded?

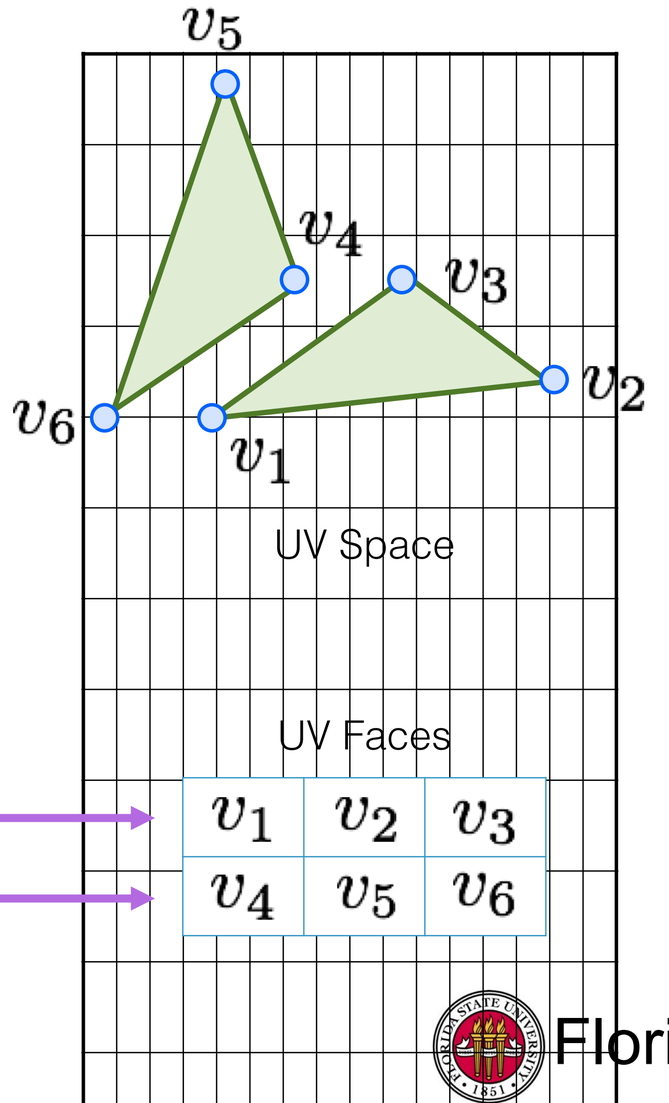
- 2 versions of the mesh are stored, one for the triangles in 3D and one for those in 2D
- The faces of the 2 meshes are in one-to-one correspondence
- OpenGL ***does not support this*** you need to duplicate all the vertices on the seams and pass one single mesh. An easy (and inefficient) way to do this is by duplicating all vertices and not using an element buffer.



A minimal example



World Coordinates



UV Space

Faces

v_1	v_2	v_3
v_1	v_3	v_4

UV Faces

v_1	v_2	v_3
v_4	v_5	v_6

Texture Mapping in OpenGL

Based on: https://open.gl/textures

Create the texture

- Similarly to all other opengl objects, we have to create it first
- Then bind it
- The pixels in the texture will be addressed using texture coordinates during drawing operations. These coordinates range from 0.0 to 1.0 where (0,0) is conventionally the bottom-left corner and (1,1) is the top-right corner of the texture image

```
GLuint tex;
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE_2D, tex);
```

Load the texture in GPU memory

- Similar to VBOs, you first allocate an array on CPU side, and then upload it to the GPU

```
// Black/white checkerboard
float pixels[] = {
    0.0f, 0.0f, 0.0f,   1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,   0.0f, 0.0f, 0.0f
};
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 2, 2, 0, GL_RGB, GL_FLOAT, pixels);
```

Vertex Shader

- In the vertex shader, you need to have the UV coordinates for each vertex …

```
float vertices[] = {
//  Position      Color             Texcoords
    -0.5f,  0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, // Top-left
     0.5f,  0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, // Top-right
     0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, // Bottom-right
    -0.5f, -0.5f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f  // Bottom-left
};
```

- and pass them to the fragment shader

```
...
in vec2 texcoord;
out vec3 Color;
out vec2 Texcoord;
...
void main()
{
    Texcoord = texcoord;
```

Fragment Shader

- In the fragment shader, you can directly read the values in the textures, indexing them using 2 coordinates

```
#version 150

in vec3 Color;
in vec2 Texcoord;

out vec4 outColor;

uniform sampler2D tex;

void main()
{
    outColor = texture(tex, Texcoord) * vec4(Color, 1.0);
}
```

Texture Units

- Each texture unit can have ***1 texture binded***
- If you want to have more than 1 texture per object you need to use multiple texture units (usually you have ~48)
- Usually, it is best to pack all textures of a single object in one texture

Texture Units

```
GLuint textures[2];
glGenTextures(2, textures);

int width, height;
unsigned char* image;

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textures[0]);
image = SOIL_load_image("sample.png", &width, &height, 0, SOIL_LOAD_RGB);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE, image);
SOIL_free_image_data(image);
glUniform1i(glGetUniformLocation(shaderProgram, "texKitten"), 0);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, textures[1]);
image = SOIL_load_image("sample2.png", &width, &height, 0, SOIL_LOAD_RGB);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE, image);
SOIL_free_image_data(image);
glUniform1i(glGetUniformLocation(shaderProgram, "texPuppy"), 1);
```

Texture Wrapping

- The clamping can be set per coordinate, where the equivalent of (x,y,z) in texture coordinates is called (s,t,r).

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Texture Filtering

GL_NEAREST

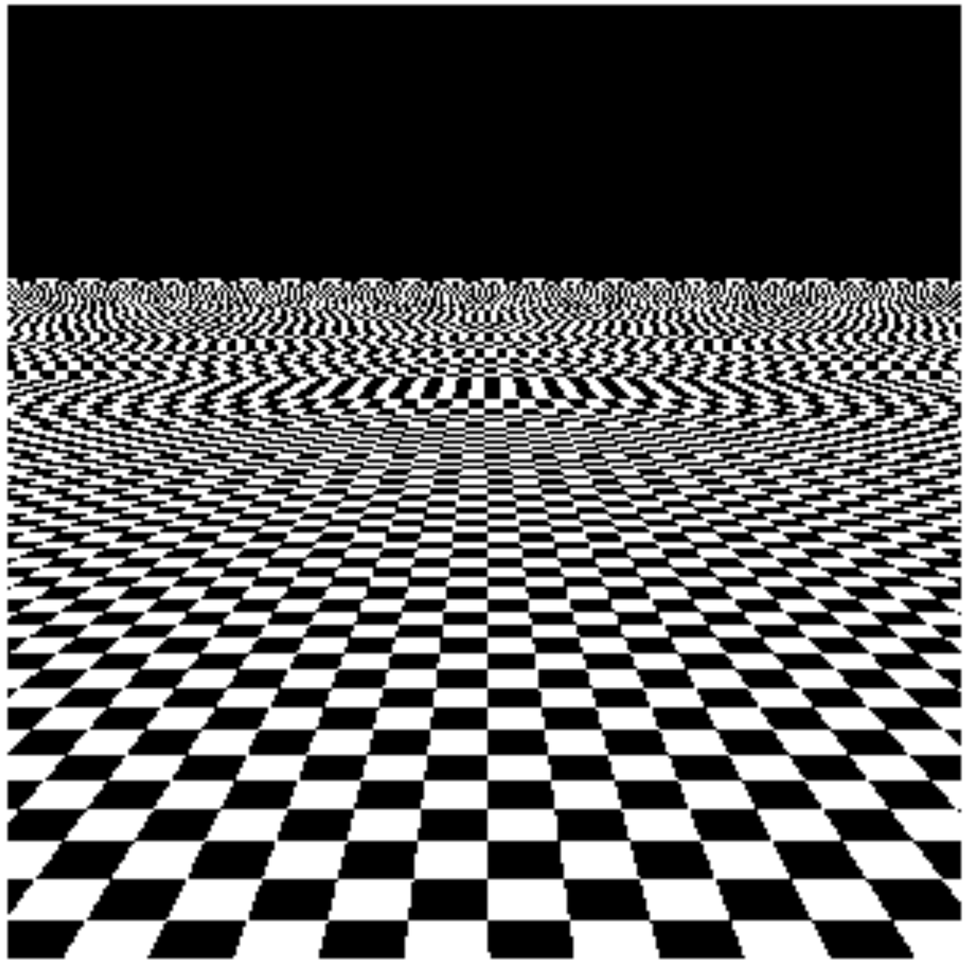
GL_LINEAR

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Moire Pattern

http://photo.net/digital-darkroom-forum/00W8gC

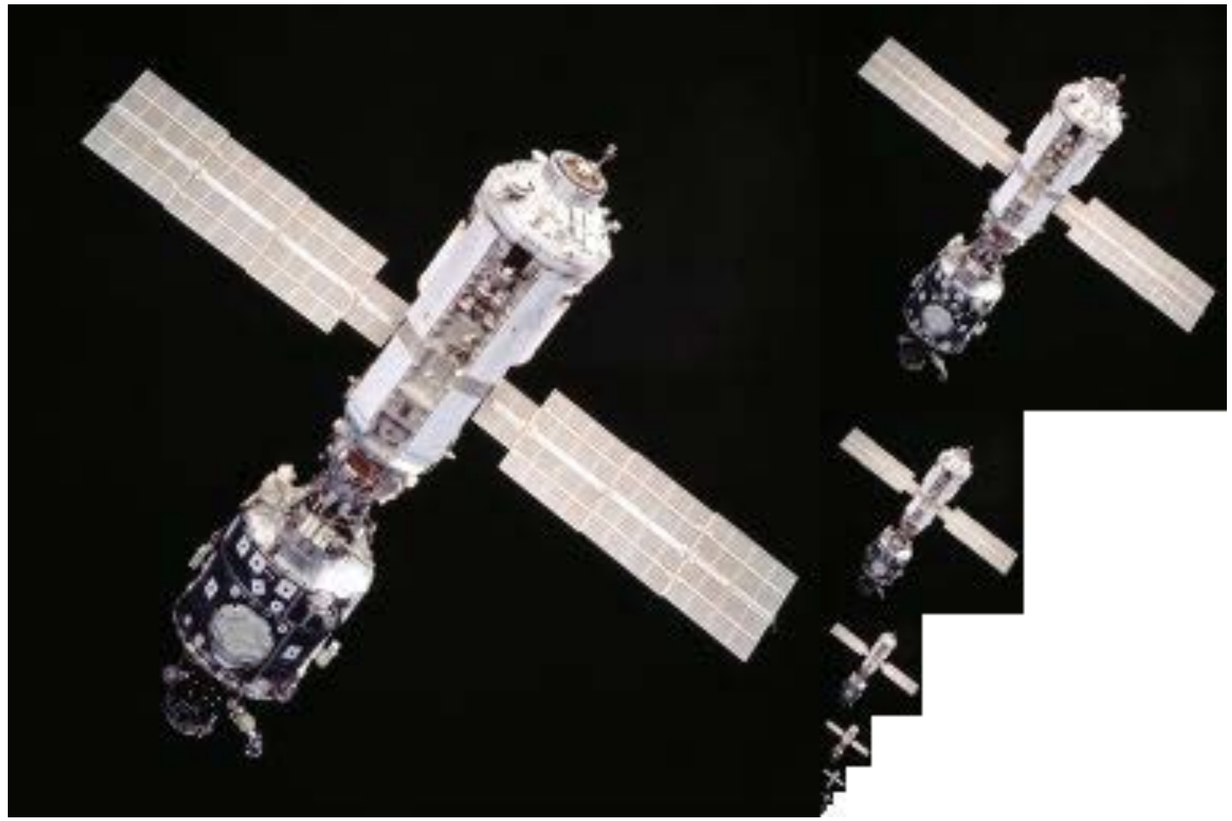
Mipmapping



Mipmapping

- Moire pattern can appear if the resolution of the texture is much higher than the sampling rate
- A good solution for this problem is mipmapping and it only requires one line in opengl

```
glGenerateMipmap(GL_TEXTURE_2D);
```



UV Unwrapping
(a.k.a.) Mesh Parameterization

Acknowledgement: Olga Sorkine-Hornung

Projections

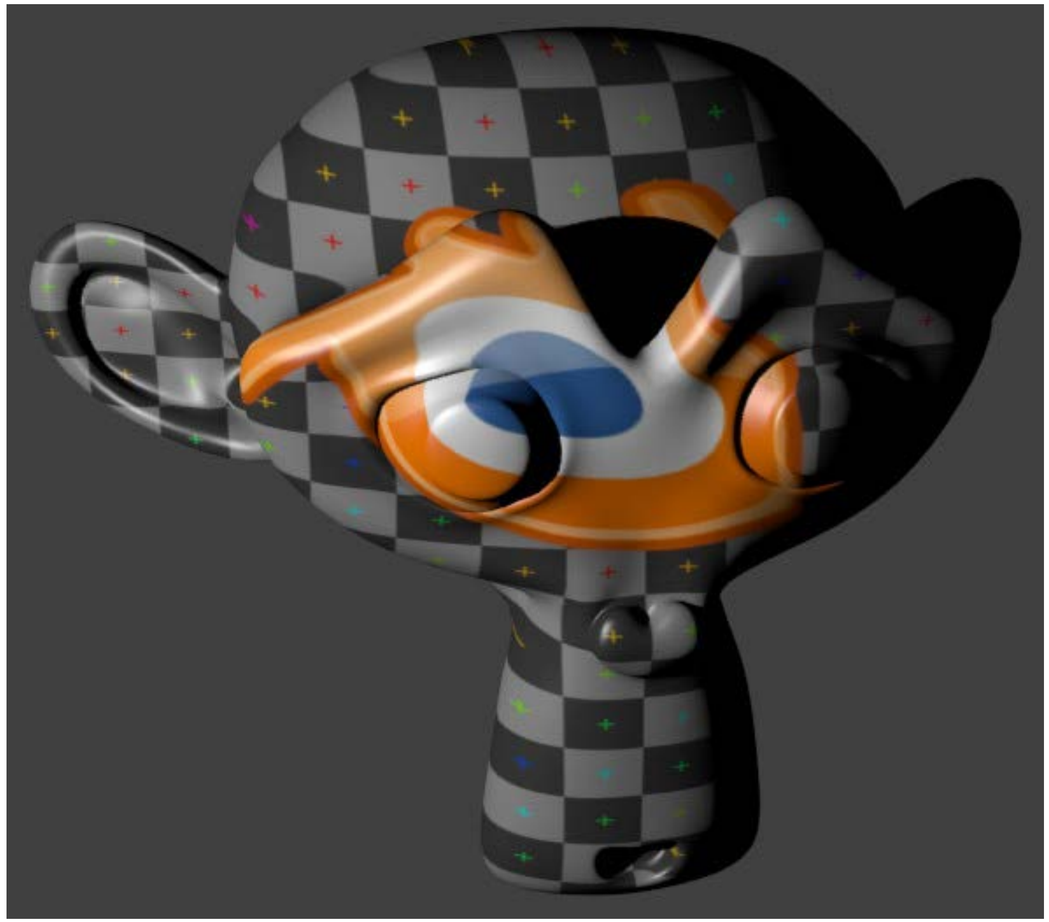
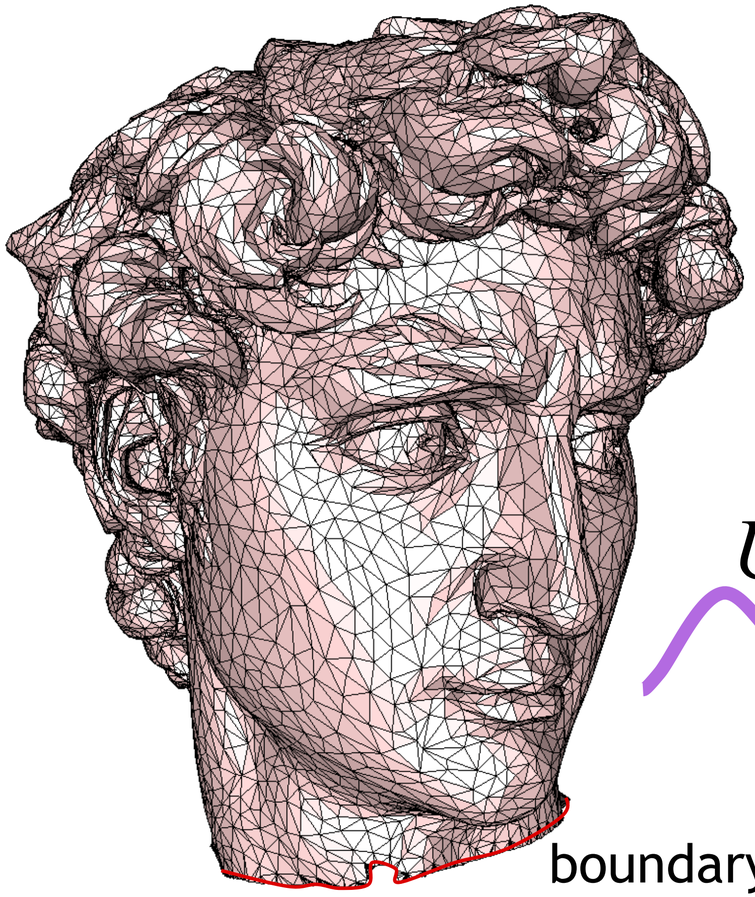


Image Courtesy of Blender

Surface Parameterization

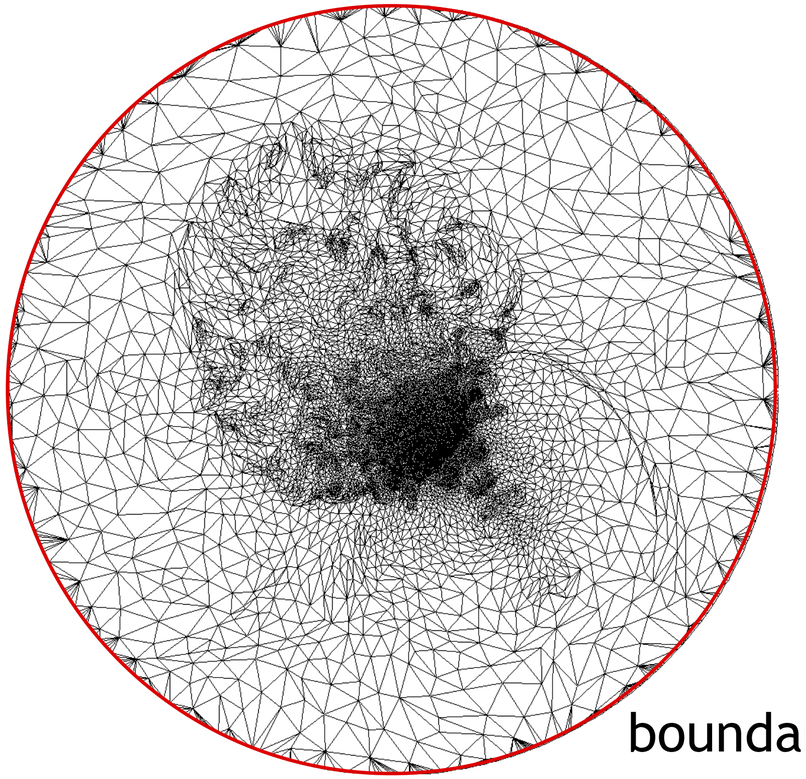
3D space (x,y,z)



boundary

U

2D parameter domain (u,v)



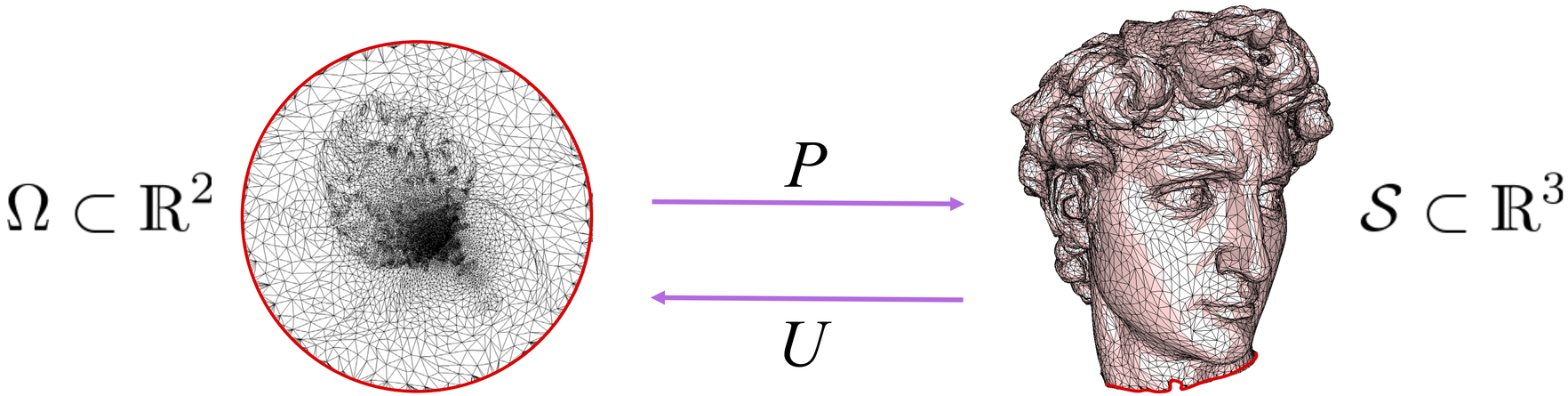
boundary

Parameterization – Definition

- Mapping P between a 2D domain Ω and the mesh S embedded in 3D (the inverse = flattening)
- Each mesh vertex has a corresponding 2D position:

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

- Inside each triangle, the mapping is affine (barycentric coordinates)

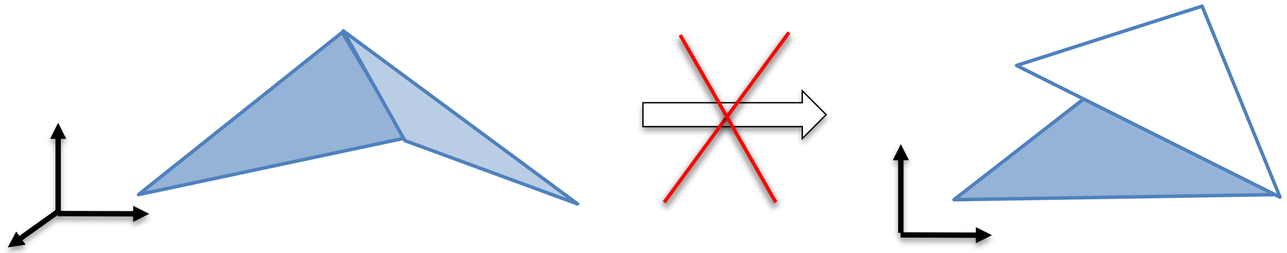


What is a good parametrization?

- It depends on the application, but usually:
  - Bijectivity
  - Number of cuts and charts
  - Geometric distortion

Bijectivity

- Locally bijective (1-1 and onto): No triangles fold over.



- Globally bijective: locally bijective + no “distant” areas overlap

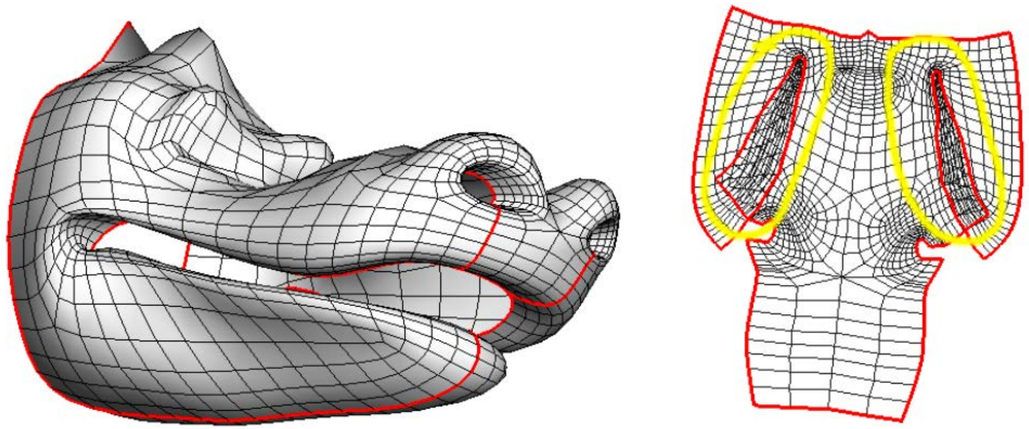
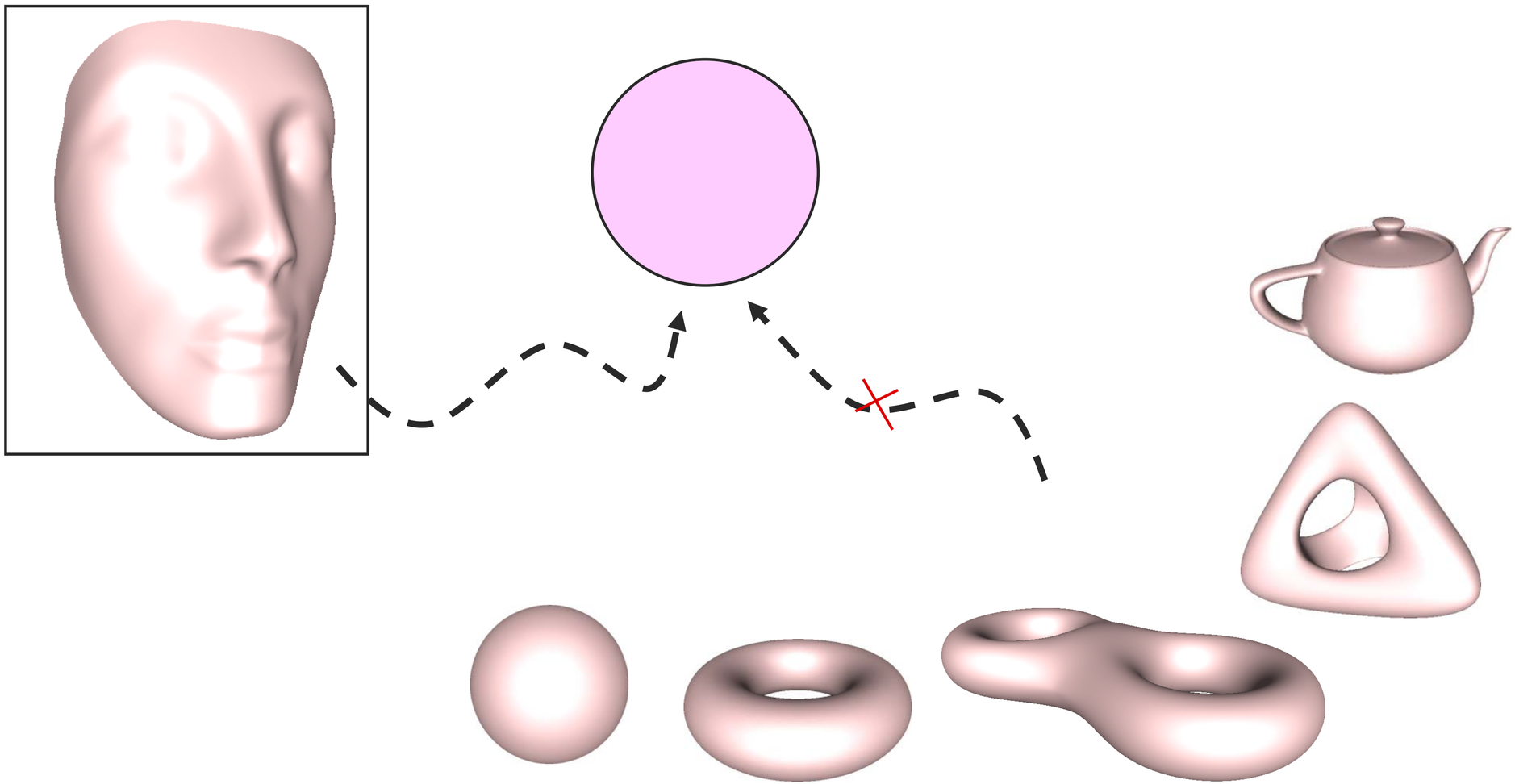
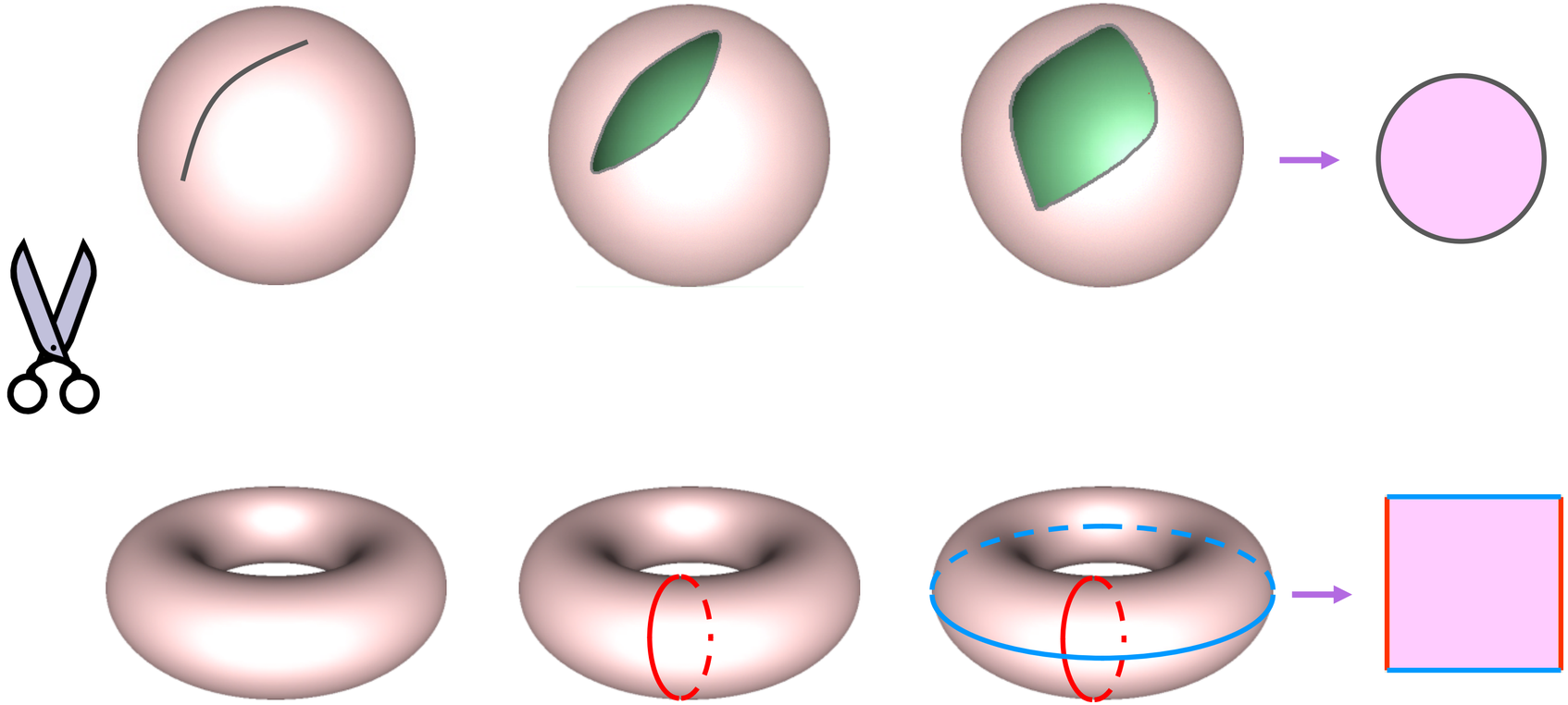


image from “Least Squares Conformal Maps”, Lévy et al., SIGGRAPH 2002

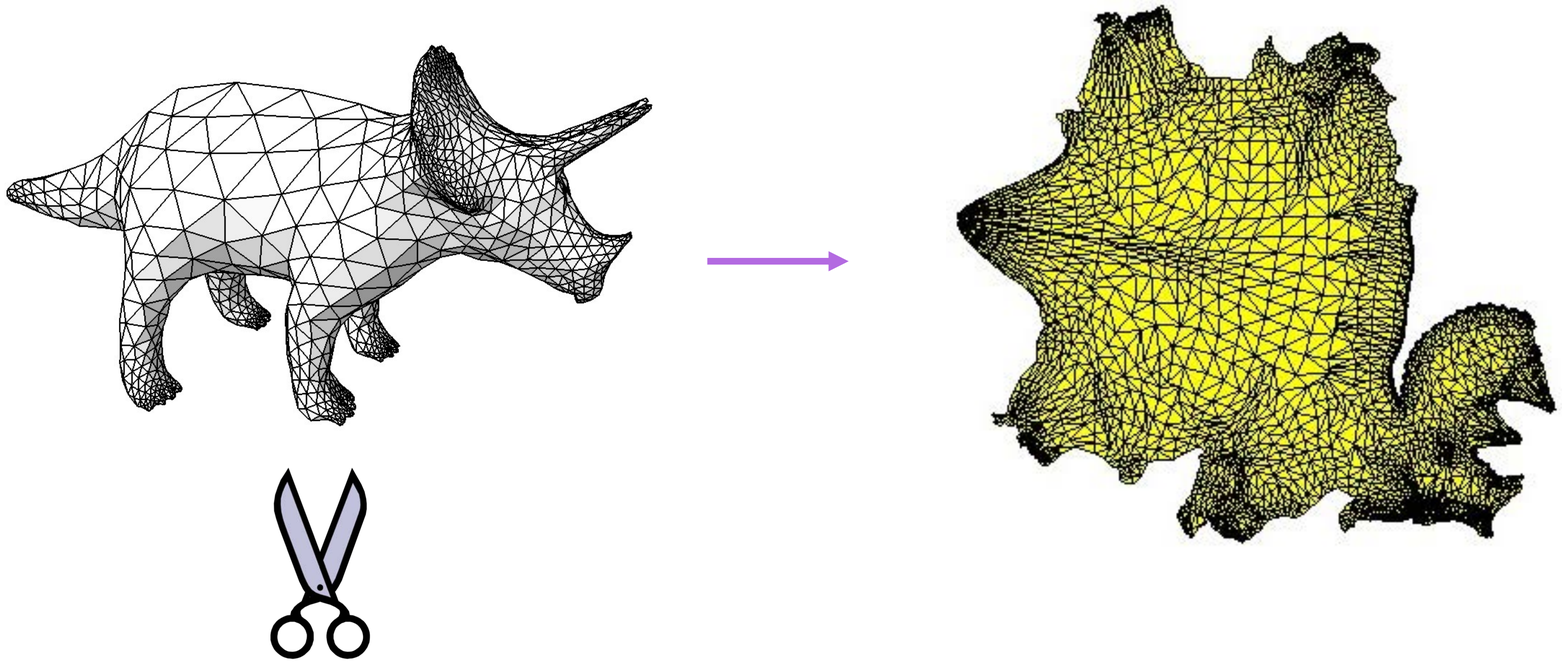
Bijectivity: Non-Disk Domains



Topological Cutting

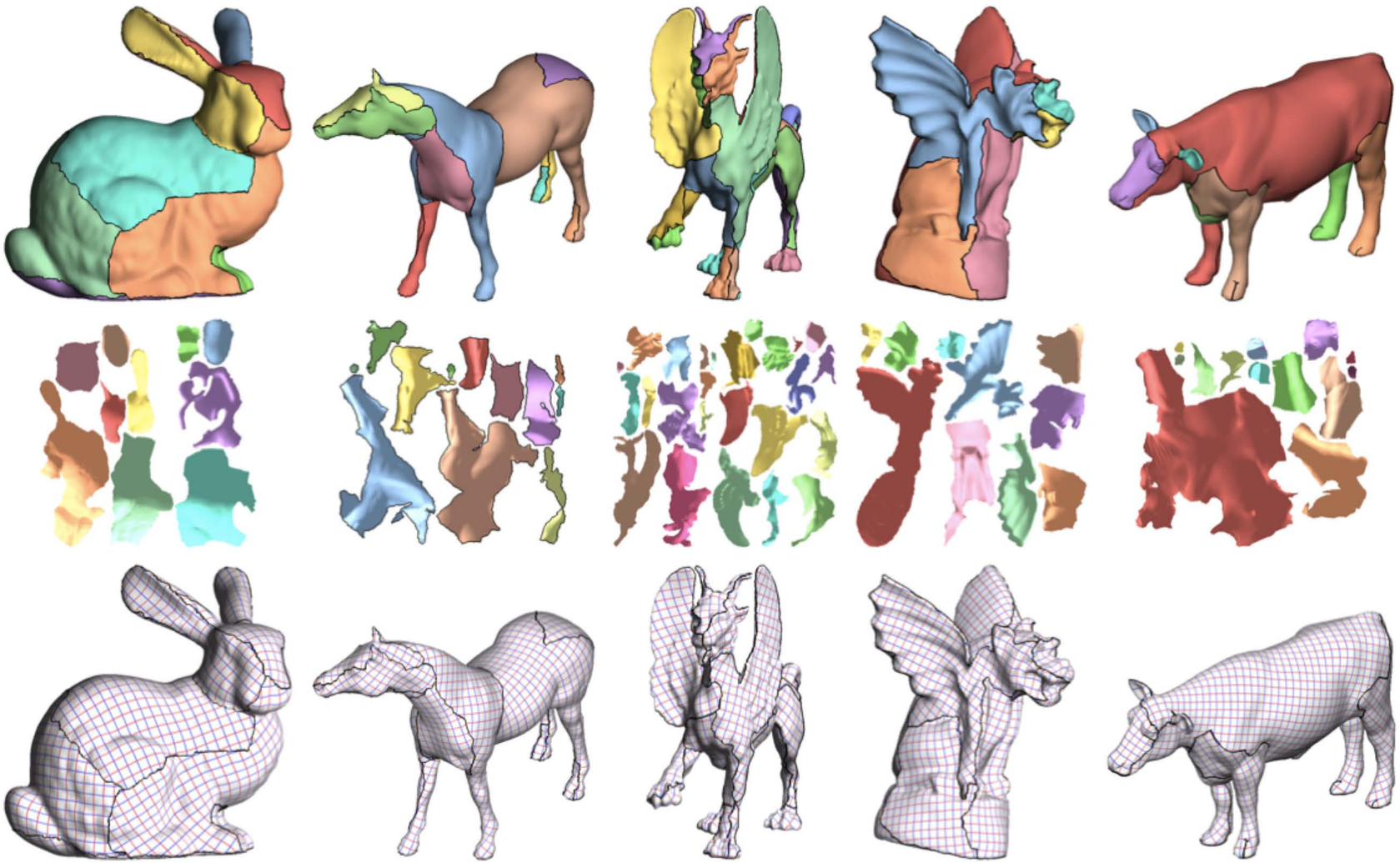


Topological Cutting



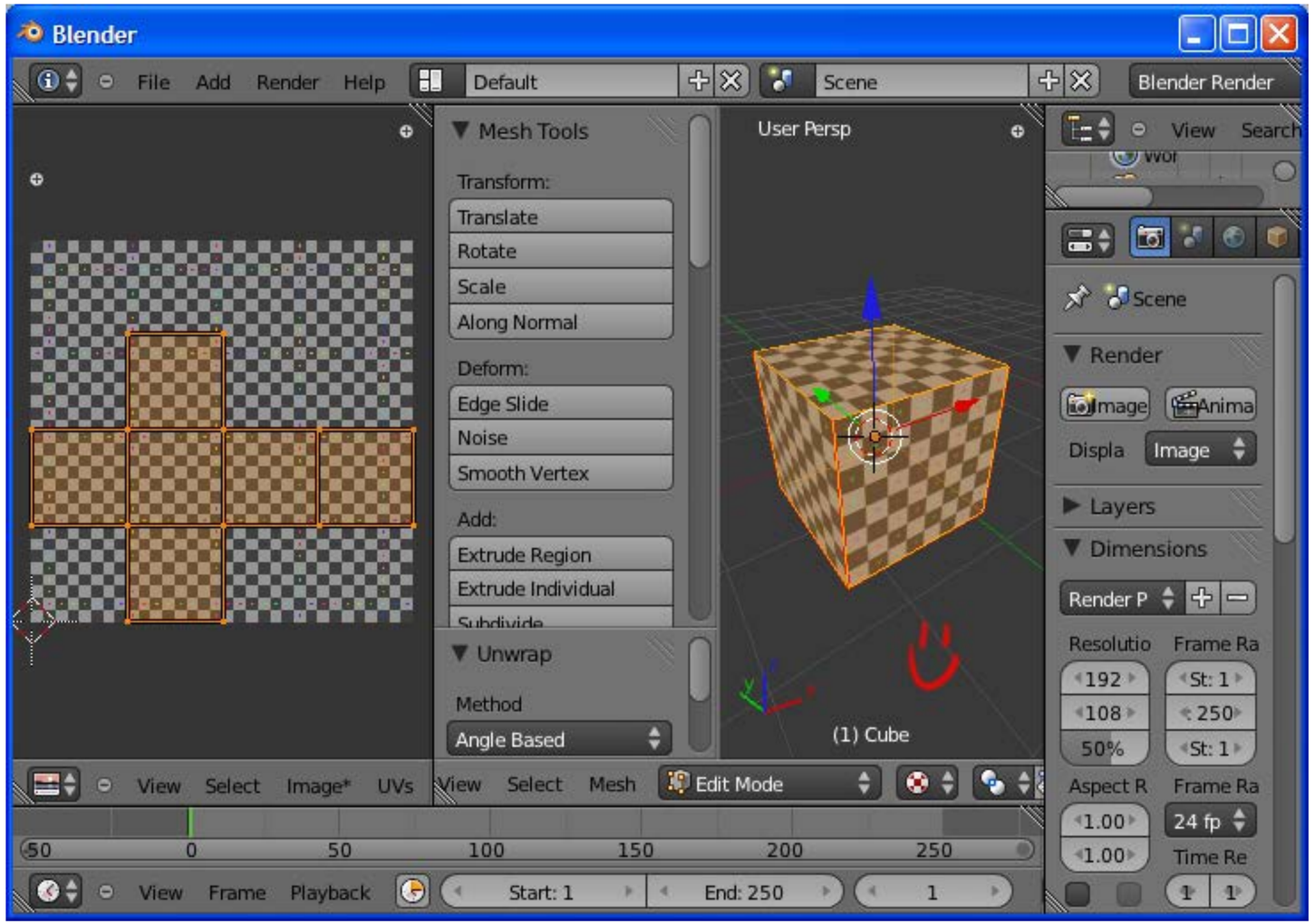
A. Sheffer, J. Hart:
Seamster: Inconspicuous Low-Distortion Texture Seam Layout, IEEE Vis 2002
http://www.cs.ubc.ca/~sheffa/papers/VIS02.pdf

Segmentation



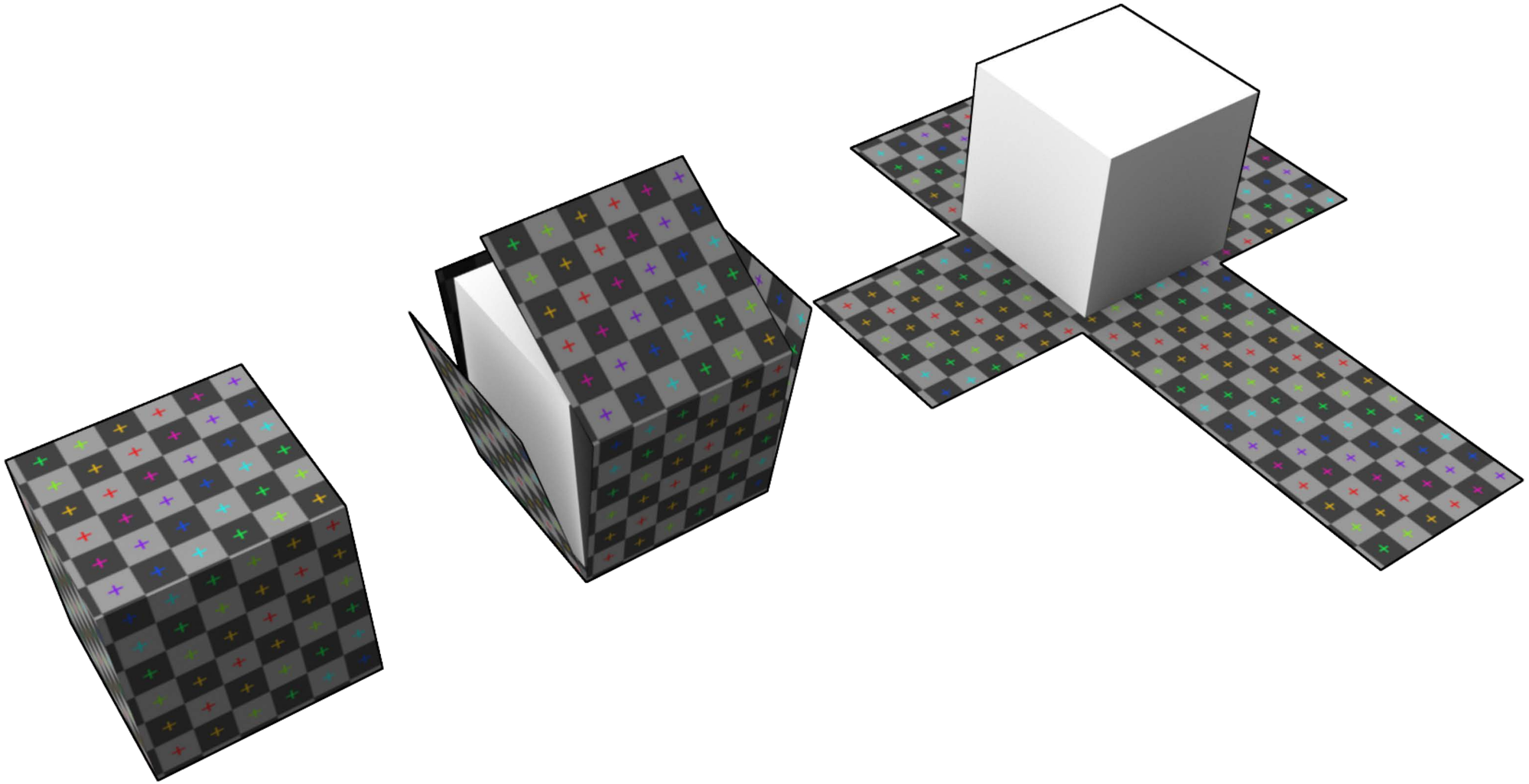
D-Charts: Quasi-Developable Mesh Segmentation,
D. Julius,V. Kraevoy, A. Sheffer, EUROGRAPHICS 2005

Segmentation

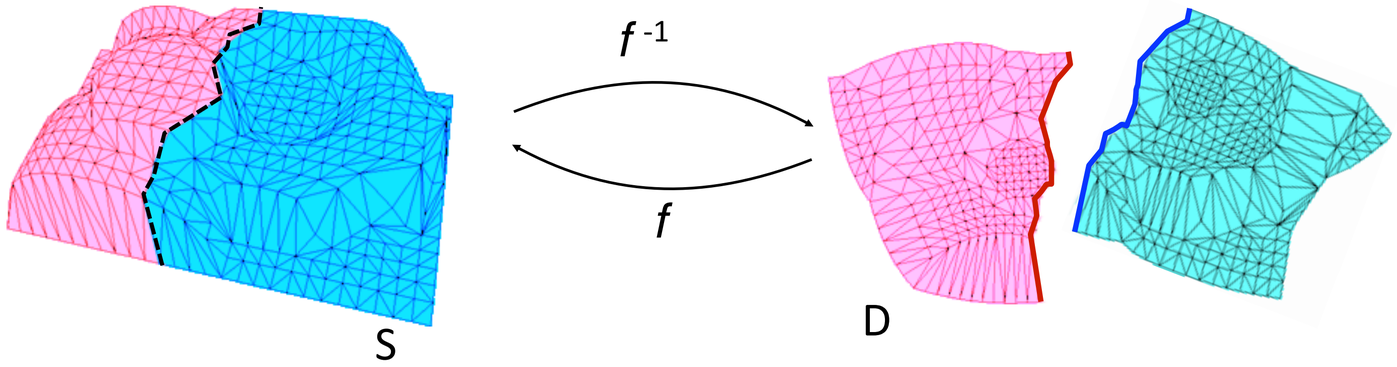


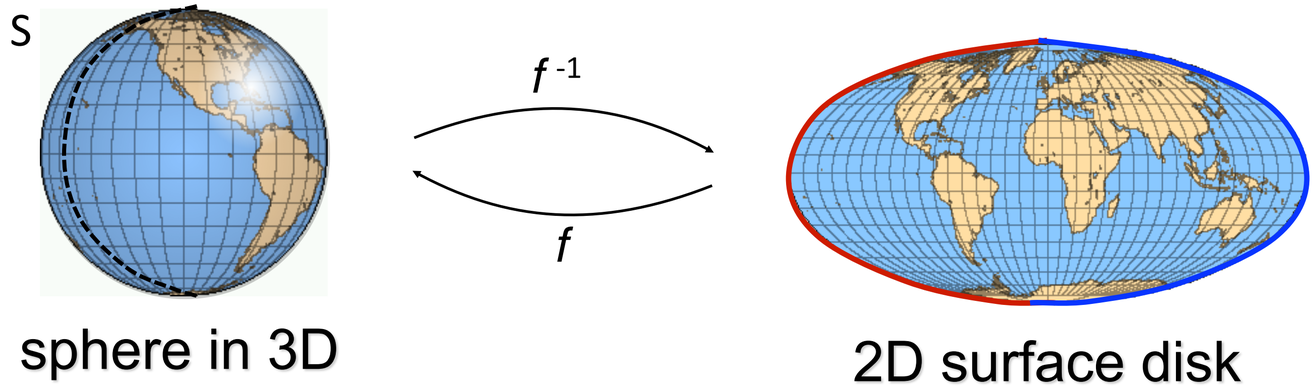
http://sophiehoulden.com/tutorials/blender/unwrapTut.html

Segmentation



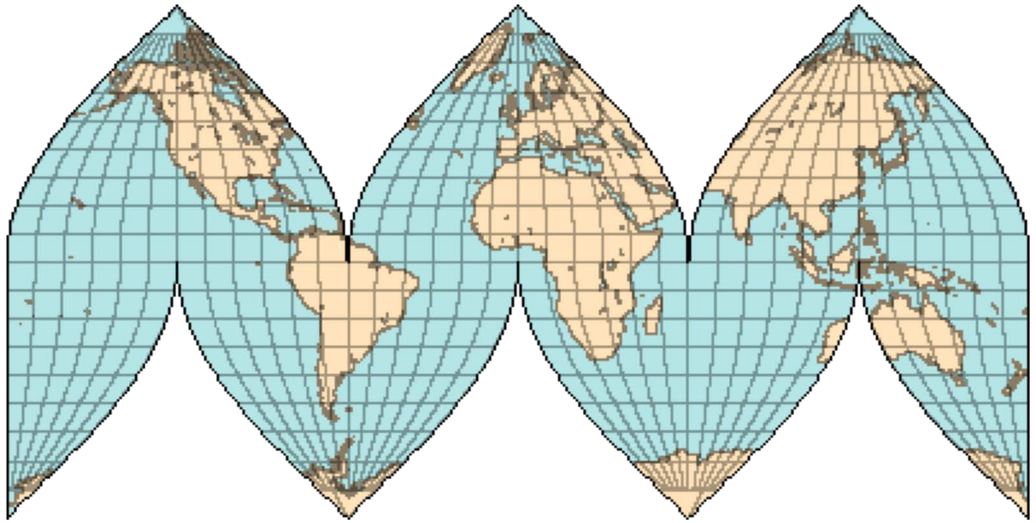
Good = “fewer cuts”?

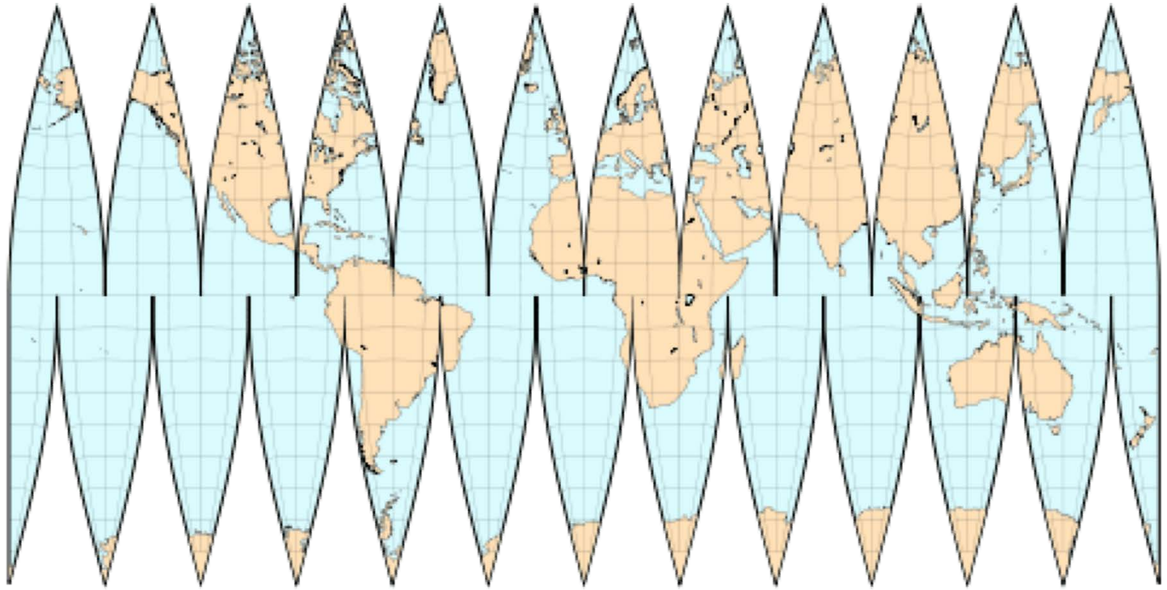




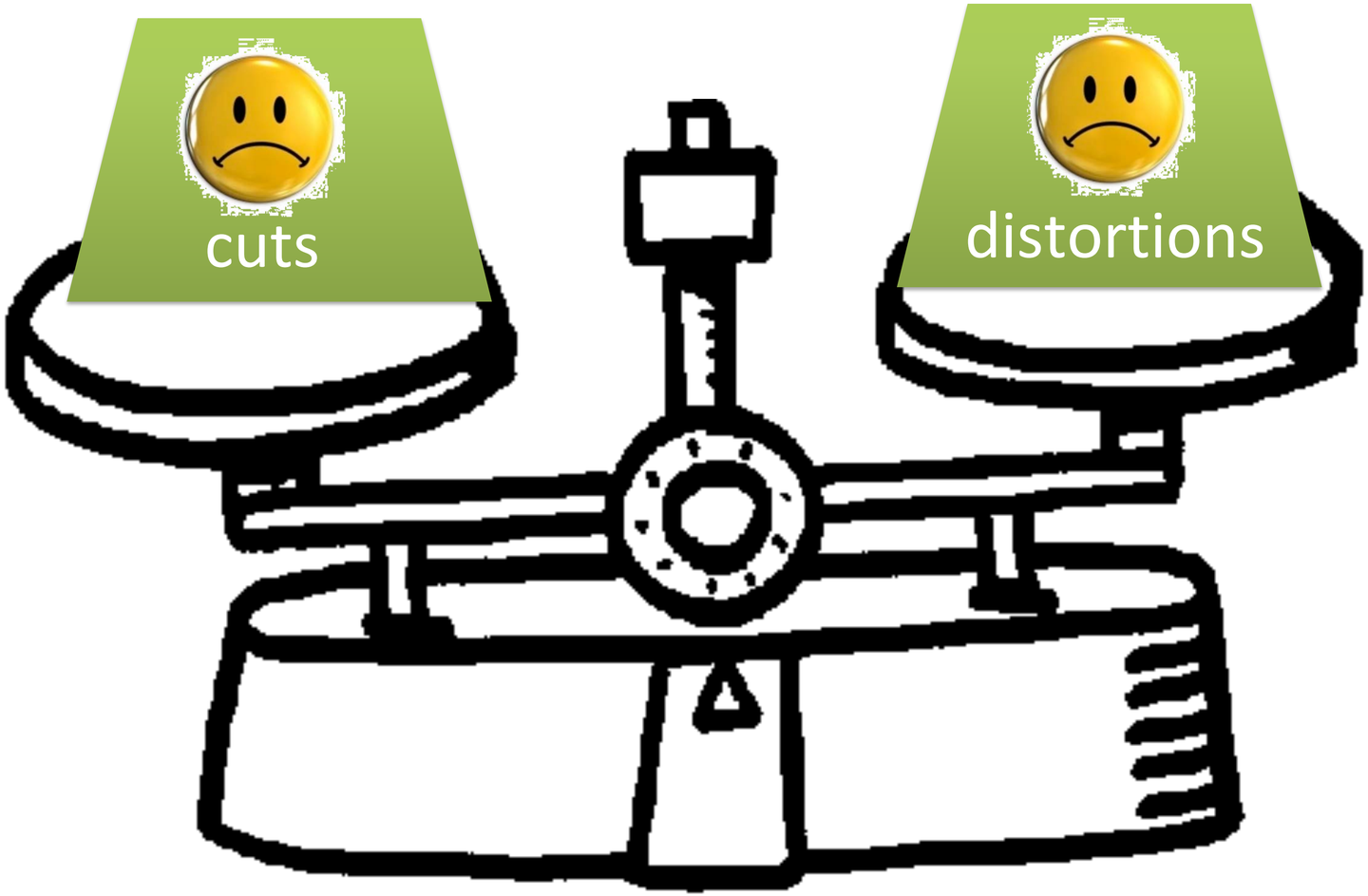
Good = “fewer cuts”?

but… more cuts => less distortion





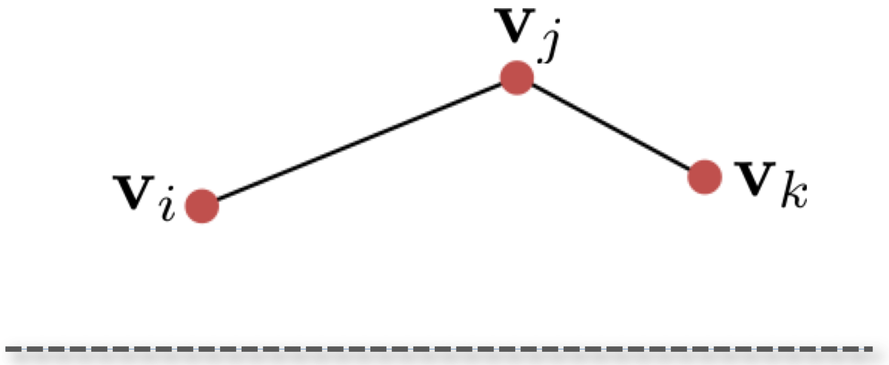
A difficult balance



Harmonic Mapping

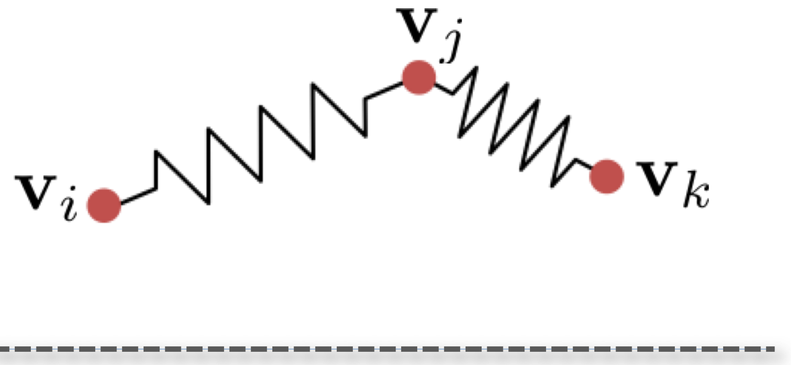
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



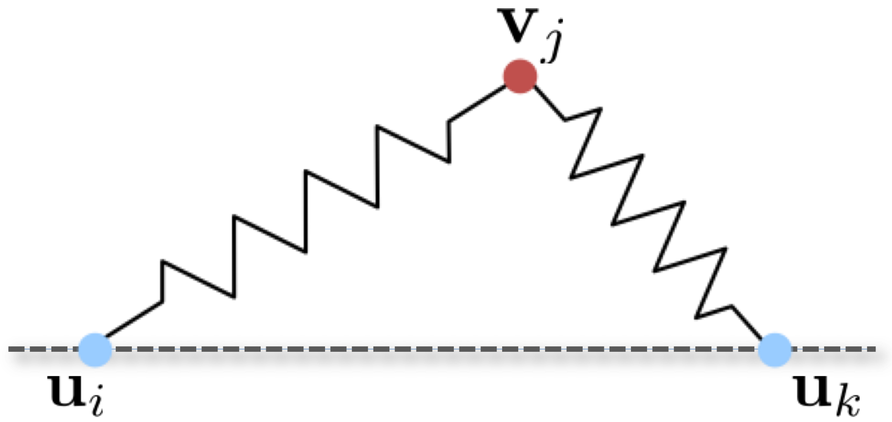
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



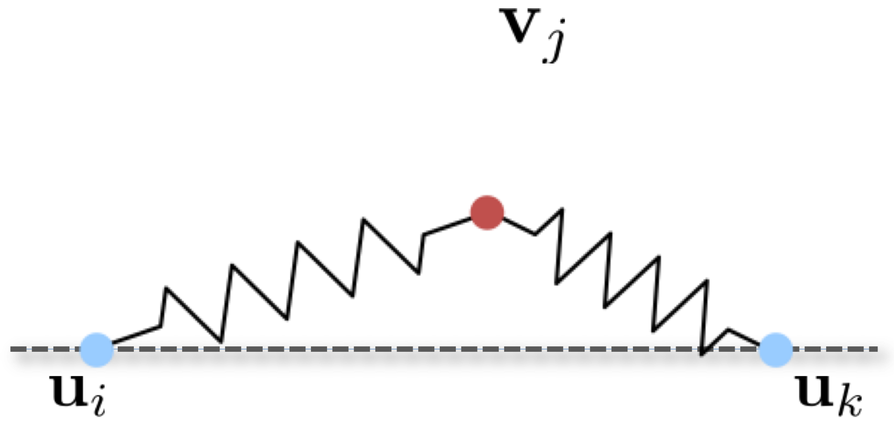
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



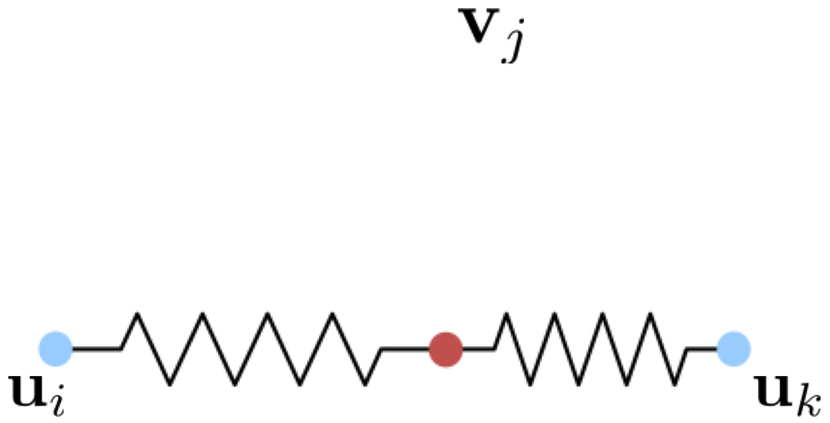
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



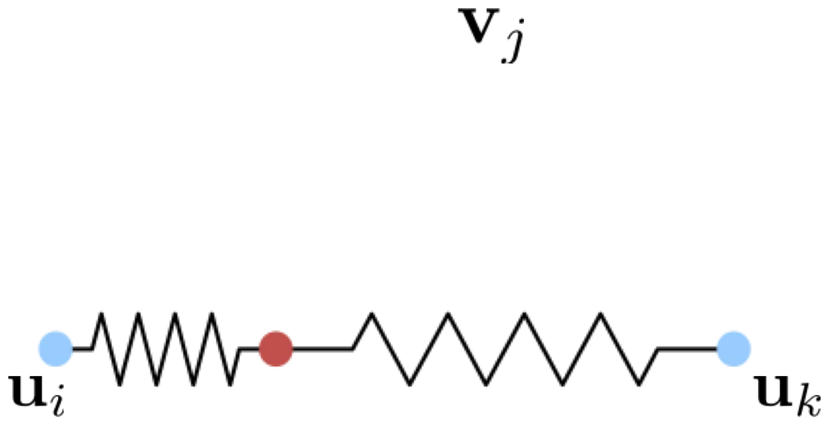
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



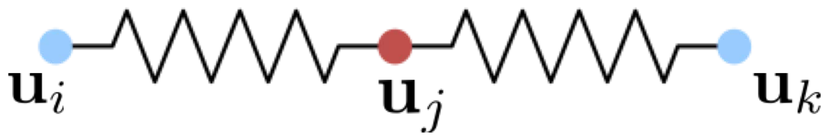
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



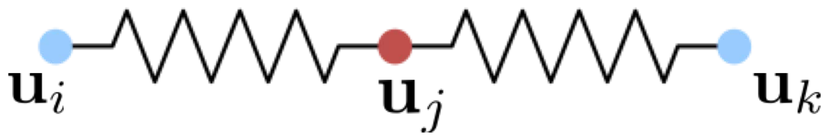
Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane



Harmonic Mapping

- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane
- Spring energy:

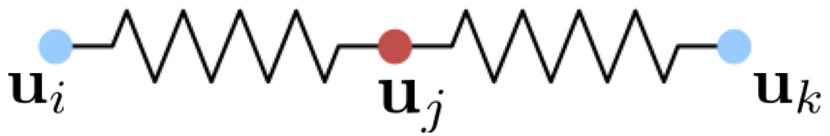


$$\frac{1}{2}k_{i,j}\|\mathbf{u}_i - \mathbf{u}_j\|^2$$

$$\mathbf{u}_i, \mathbf{u}_j \in \mathbb{R}^2$$

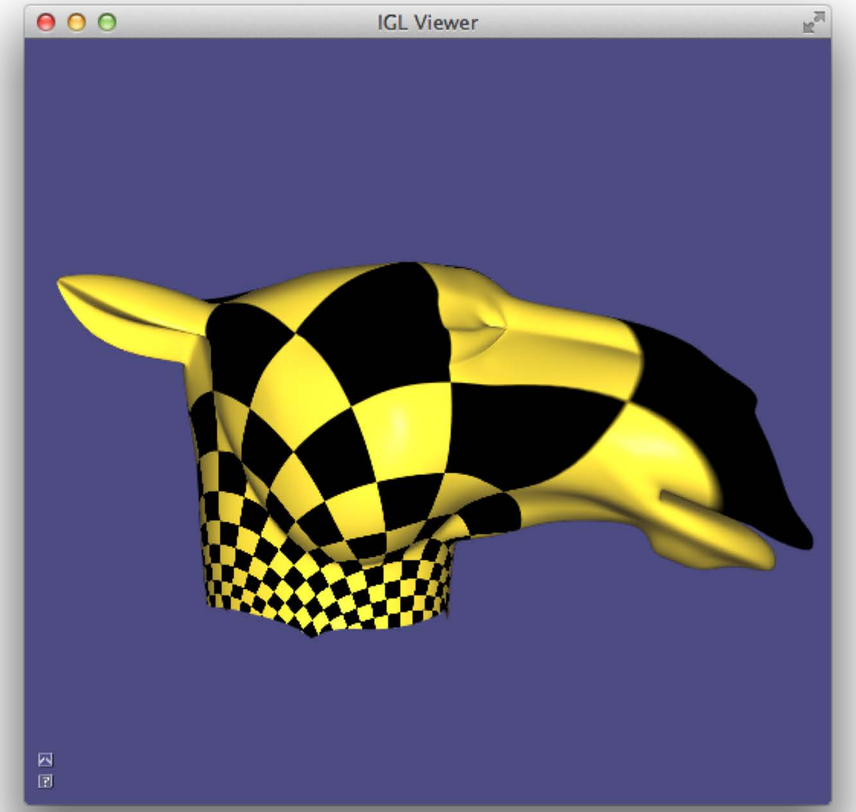
Harmonic Mapping

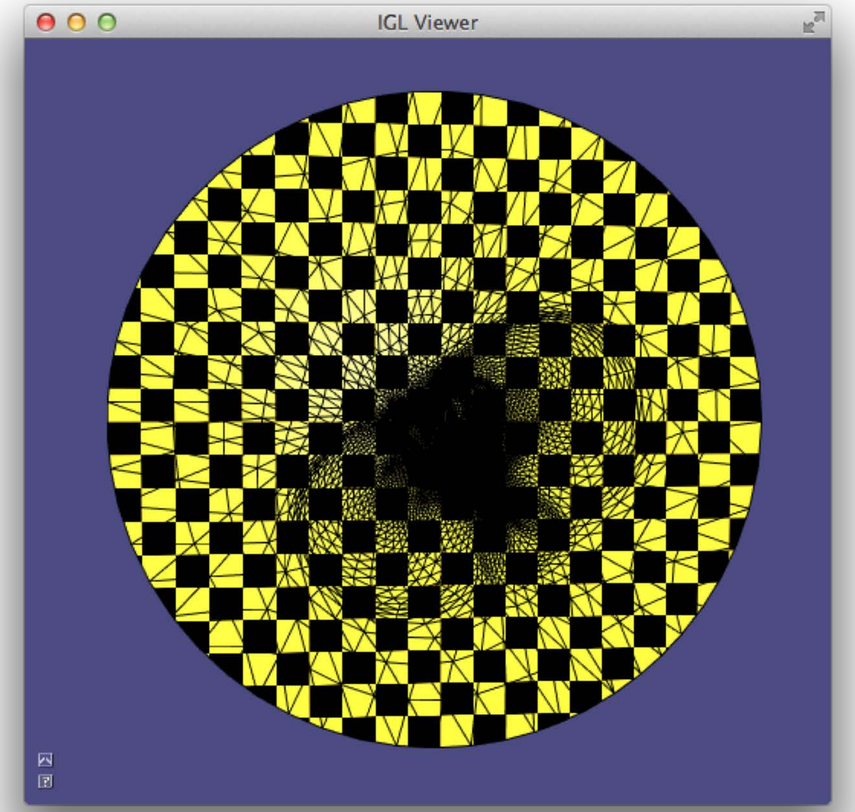
- Inner mesh edges as springs
- Find minimum-energy state where all vertices lie in the 2D plane
- Total spring energy of the flattened mesh:



$$E(\mathbf{u}_1, \ldots, \mathbf{u}_n) = \sum_{(i,j)\in\mathcal{E}} \frac{1}{2} k_{i,j} \|\mathbf{u}_i - \mathbf{u}_j\|^2$$

Demo





https://libigl.github.io/libigl/tutorial/tutorial.html#harmonicparametrization

Minimizing Spring Energy

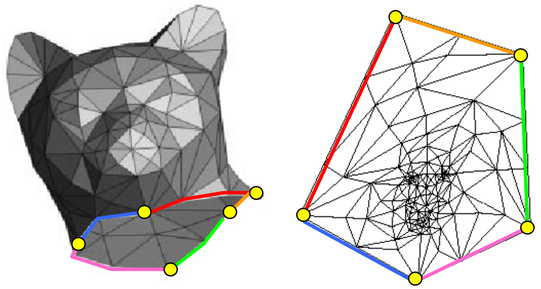
$$E(\mathbf{u}_1, \ldots, \mathbf{u}_n) = \sum_{(i,j)\in\mathcal{E}} \frac{1}{2} k_{i,j} \|\mathbf{u}_i - \mathbf{u}_j\|^2$$

$$\frac{\partial E(\mathbf{u}_1, \ldots, \mathbf{u}_n)}{\partial \mathbf{u}_i} = \sum_{j\in\mathcal{N}(i)} k_{i,j}(\mathbf{u}_i - \mathbf{u}_j) = 0$$

$$\sum_{j\in\mathcal{N}(i)\cap\mathcal{B}} k_{i,j}\mathbf{u}_i + \sum_{j\in\mathcal{N}(i)\setminus\mathcal{B}} k_{i,j}(\mathbf{u}_i - \mathbf{u}_j) = \sum_{j\in\mathcal{N}(i)\cap\mathcal{B}} k_{i,j}\mathbf{u}_j$$

unknown flat vertex positions

known fixed boundary positions



$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$- inner vertices

$\mathbf{v}_{n+1}, \ldots, \mathbf{v}_N$ - boundary vertices

Minimizing Spring Energy

- Sparse linear system of n equations to solve!

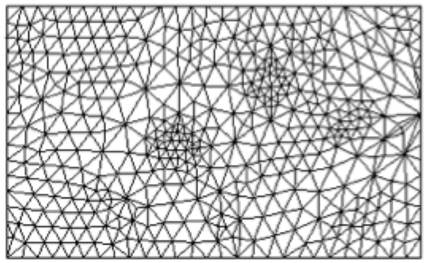
$$\sum_{j \in \mathcal{N}(i) \cap \mathcal{B}} k_{i,j}\mathbf{u}_i + \sum_{j \in \mathcal{N}(i) \setminus \mathcal{B}} k_{i,j}(\mathbf{u}_i - \mathbf{u}_j) = \sum_{j \in \mathcal{N}(i) \cap \mathcal{B}} k_{i,j}\mathbf{u}_j$$

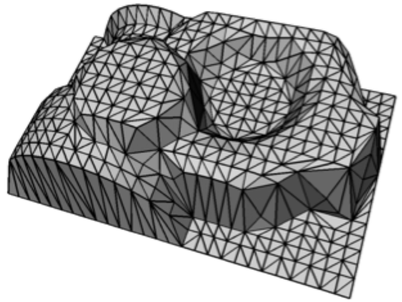
$$\begin{pmatrix} \sum_j k_{i,j} & * & \cdots & -k_{i,j} \\ * & \sum_j k_{i,j} & * & \vdots \\ \vdots & * & \ddots & * \\ -k_{j,i} & \cdots & * & \sum_j k_{i,j} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_n \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{u}}_1 \\ \bar{\mathbf{u}}_2 \\ \vdots \\ \bar{\mathbf{u}}_n \end{pmatrix}$$

Choice of spring constants $k_{i,j}$

- Uniform

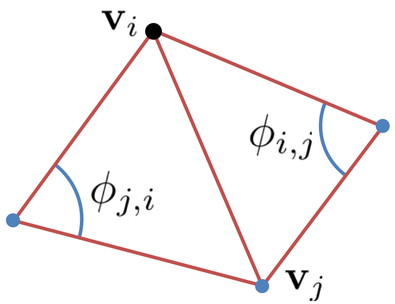
$$k_{i,j} = 1$$

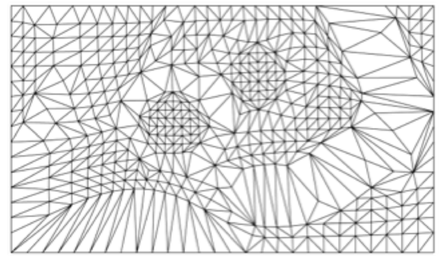




- Cotangent

$$k_{i,j} = \cot \phi_{i,j} + \cot \phi_{j,i}$$





Tutte's Theorem

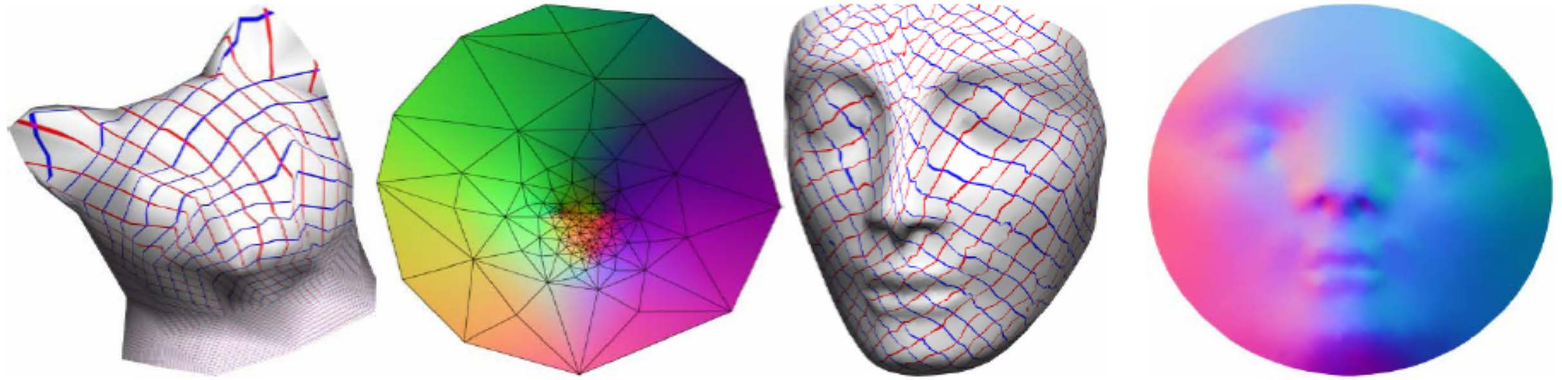
- If the weights are **nonnegative**, and the boundary is fixed to a **convex** polygon, the parameterization is **bijective**

- (Tutte'63 proved for uniform weights, Floater'97 extended to arbitrary nonnegative weights)

- W.T. Tutte. "How to draw a graph". Proceedings of the London Mathematical Society, 13(3):743-768, 1963.

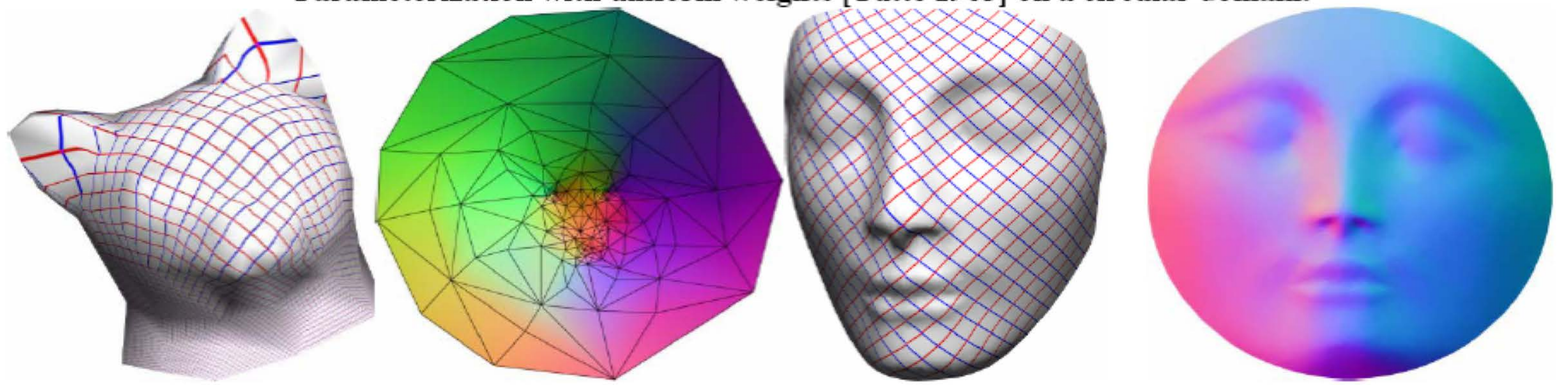
Comparison of Weights

uniform weights



Parameterization with uniform weights [Tutte 1963] on a circular domain.

cotan weights



Parameterization with harmonic weights [Eck et al. 1995] on a circular domain.

Eck et al. 1995, “Multiresolution analysis of arbitrary meshes”, SIGGRAPH 1995

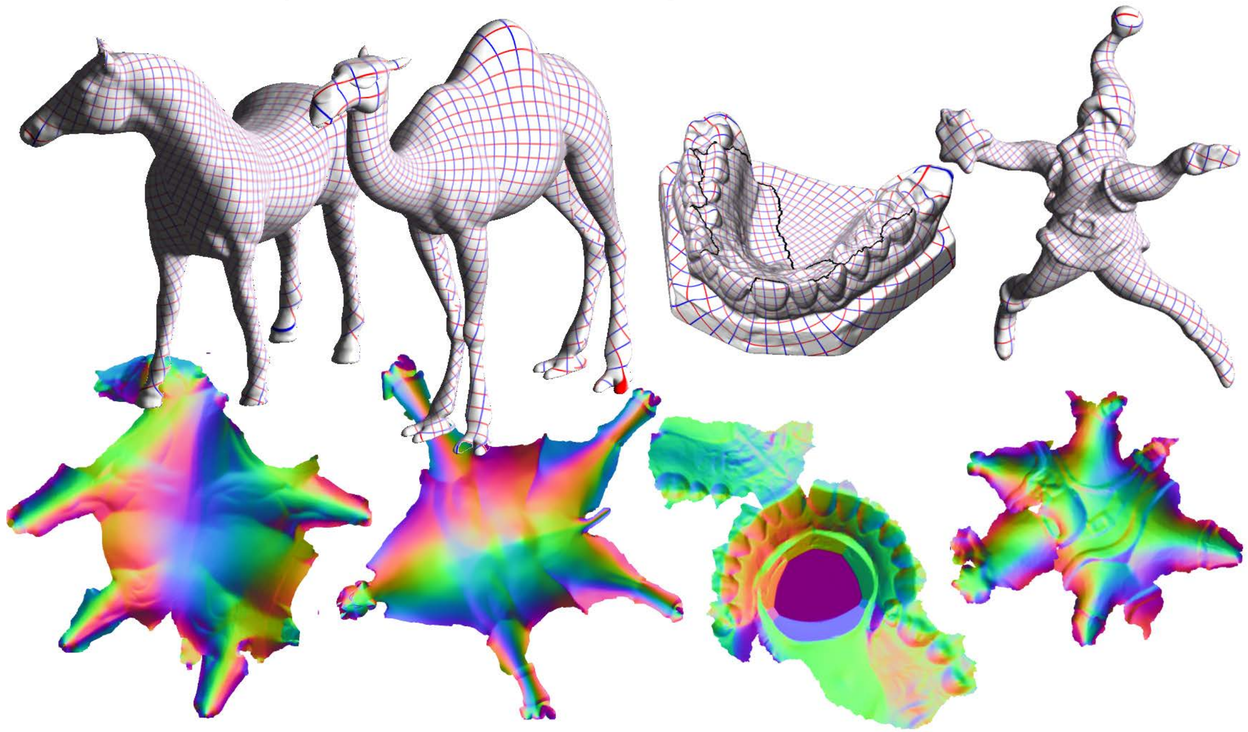
Discussion

- The results of cotan-weights mapping are ***better*** than those of uniform convex mapping (local area and angles preservation).
- But: the mapping ***is not always legal*** (the cotan weights can be negative for badly-shaped triangles…)

- In any case: sparse system to solve. Robust and efficient numerical solvers exist (Eigen Sparse LDLT)

Discussion

- Both mappings have the problem of fixed boundary – it constrains the minimization and causes distortion.
- More advanced methods do not require boundary conditions.



ABF++ method,
Sheffer et al. 2005
http://www.cs.ubc.ca/~sheffa/ABF++/abf.htm

References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by Steve Marschner, Peter Shirley**
Chapter 11

https://open.gl

Polygon Mesh Processing
Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levy