Shape Representations (cont'd)

Instructor:
Evangelos Kalogerakis

So far we have discussed:

Part I: Image representations

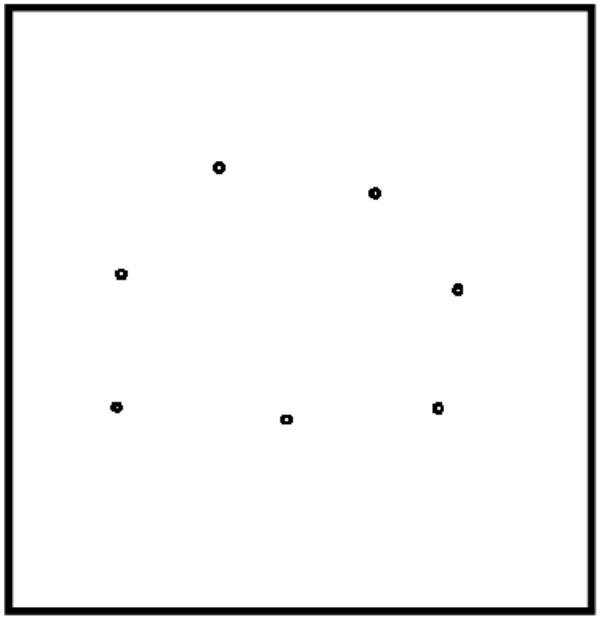
Part II: Shape representations

- Polygon Meshes
- Parametric Surfaces
- Voxel grids
- Point clouds
- **Implicit functions**

Reconstruction in general

Two family of methods:

Explicit meshing

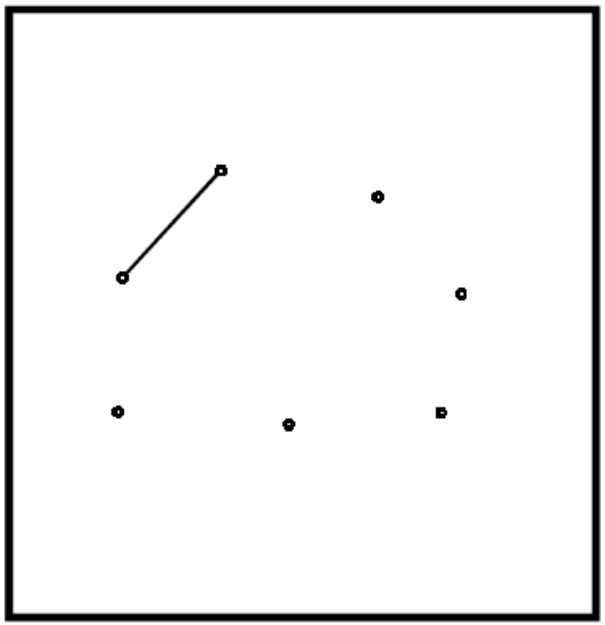


Implicit Reconstruction

Reconstruction in general

Two family of methods:

Explicit meshing

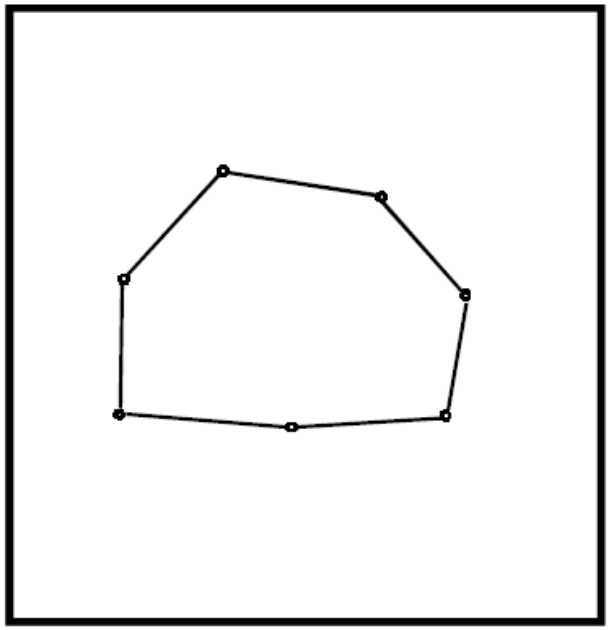


Implicit Reconstruction

Reconstruction in general

Two family of methods:

Explicit meshing

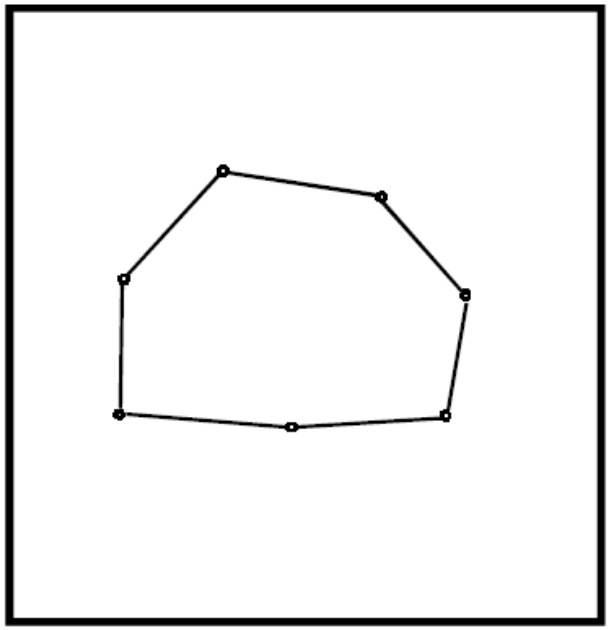


Implicit Reconstruction

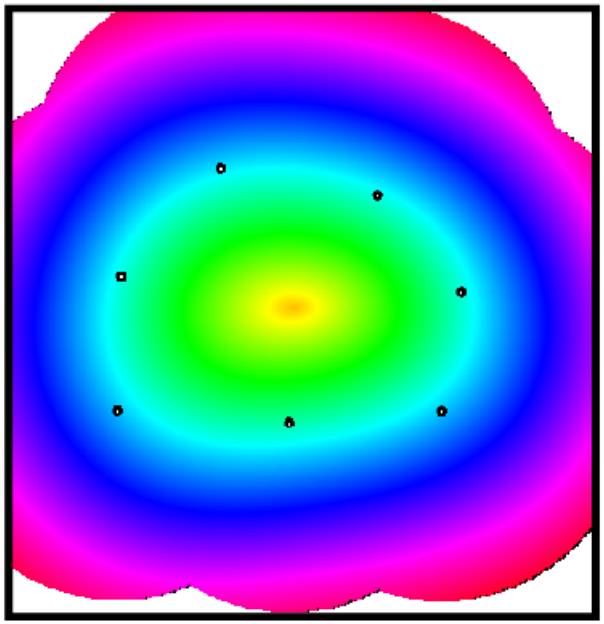
Reconstruction in general

Two family of methods:

Explicit meshing



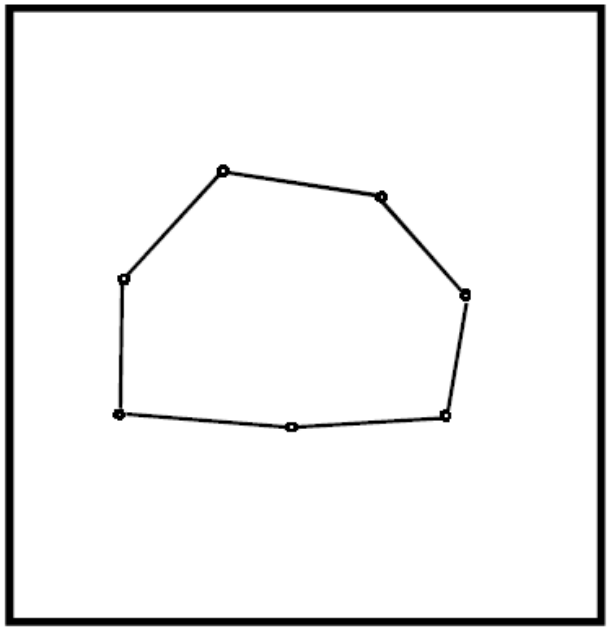
Implicit Reconstruction



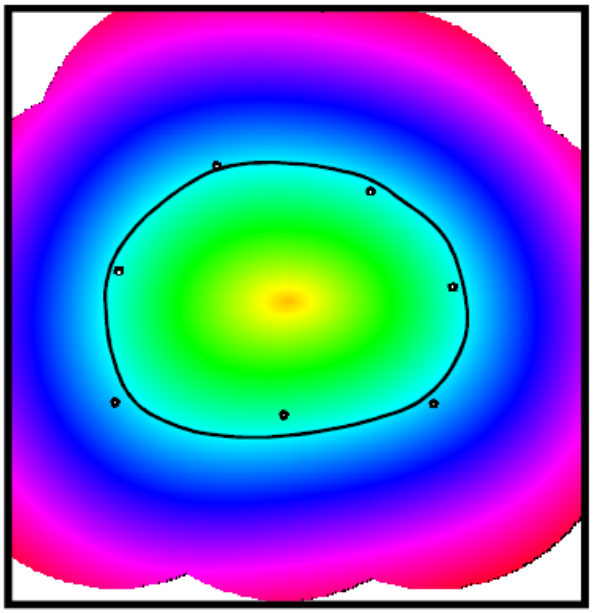
Reconstruction in general

Two family of methods:

Explicit meshing

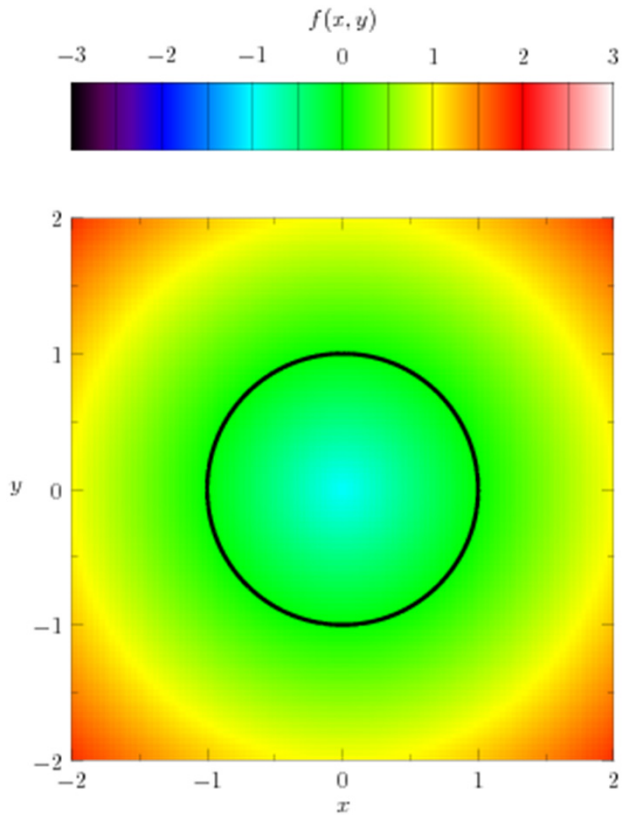


Implicit Reconstruction



Example: Implicit curve

$$f(x, y) = x^2 + y^2 - R^2$$

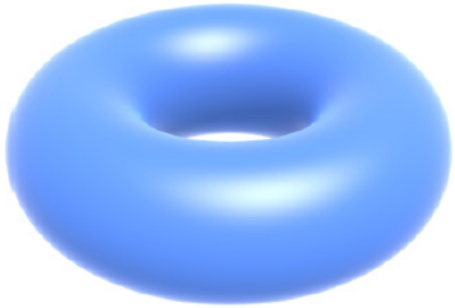


Example: Implicit surfaces

Surface is the zero set of a function in x,y,z:

$f(x,y,z) = 0$

$x^2+y^2+z^2=1$



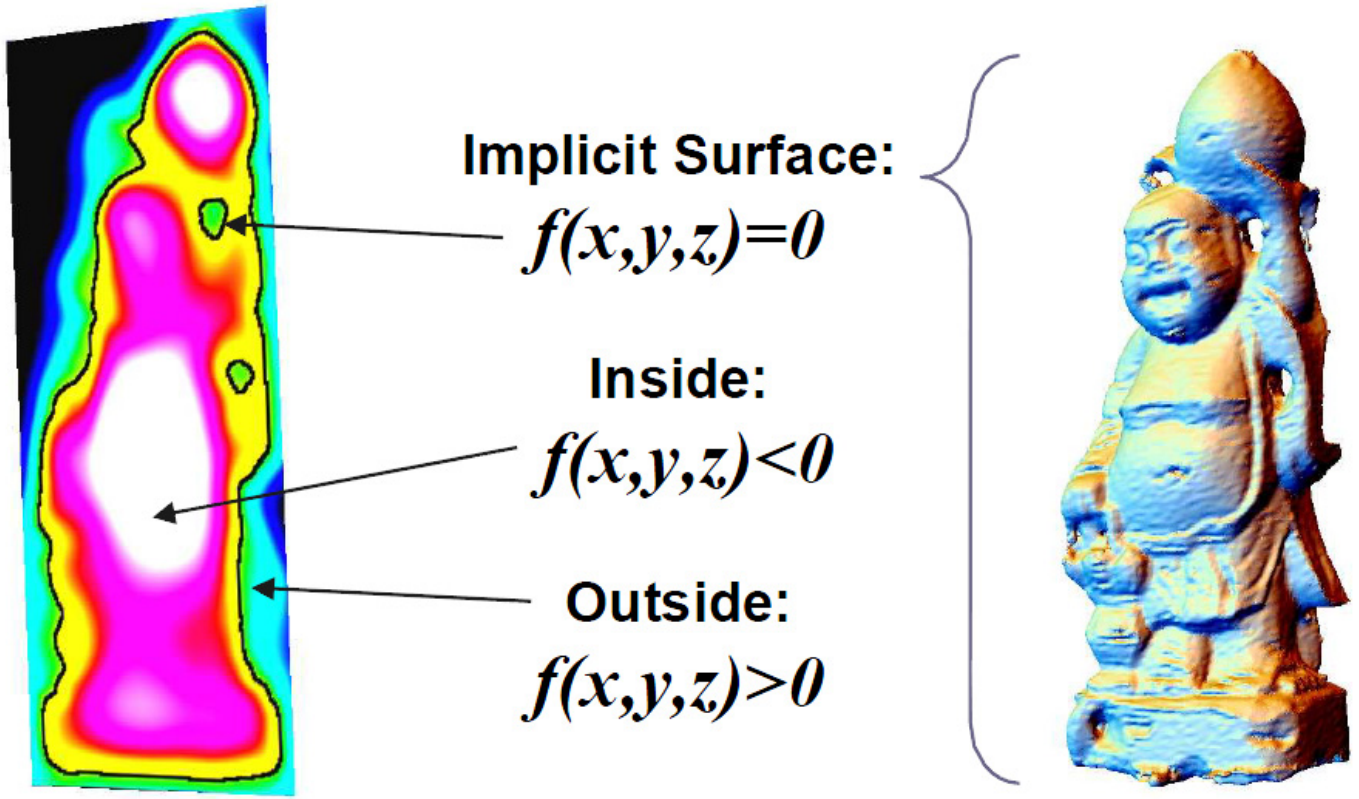
$(R-\sqrt{x^2+y^2})^2+z^2=r^2$

$(x^2+\frac{9y^2}{4}+z^2-1)^3 = x^2z^3+\frac{9y^2z^3}{80}$

Implicit surfaces

Surface is defined by **implicit** function f(x,y,z) = 0. In other words, implicit surfaces are isosurfaces through some scalar field in 3D.

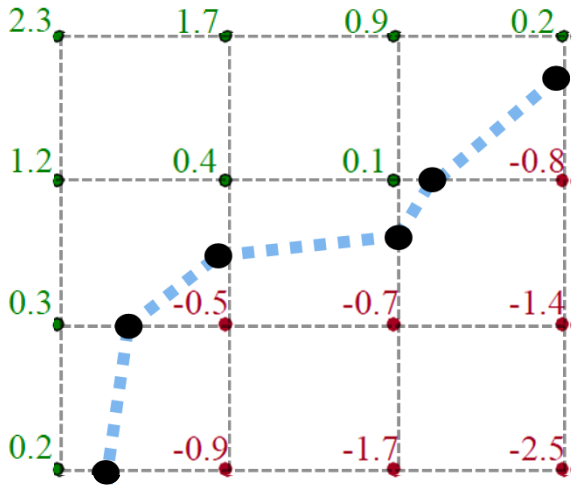
3-D example



Implicits sampled in a grid

Curve is the zero set of a function in x, y:

$f(x, y) = 0$

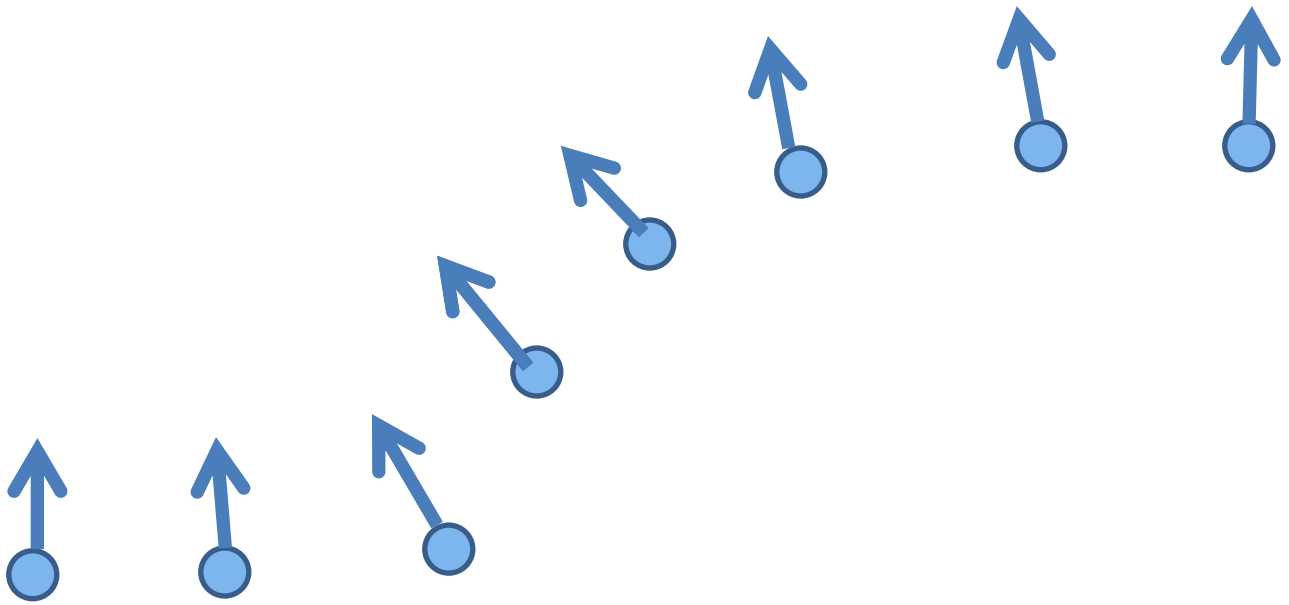


Store a grid of values approximating the function

Surface is defined where interpolated values of this function is 0.

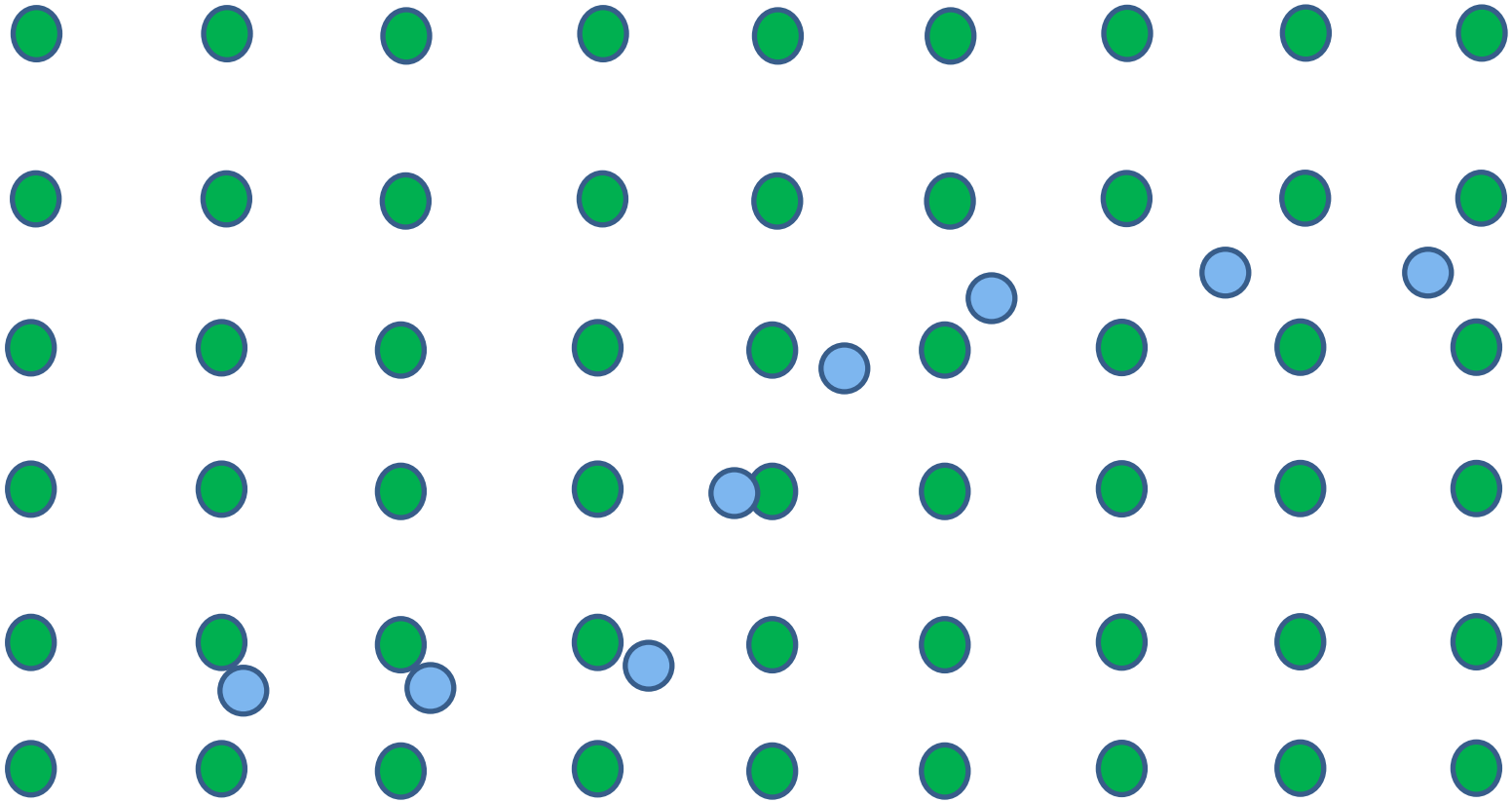
How to define implicit surface for reconstruction?

One way is to represent **signed** distance to the unknown surface represented by points (in blue), their outward-pointing normals are shown as vectors



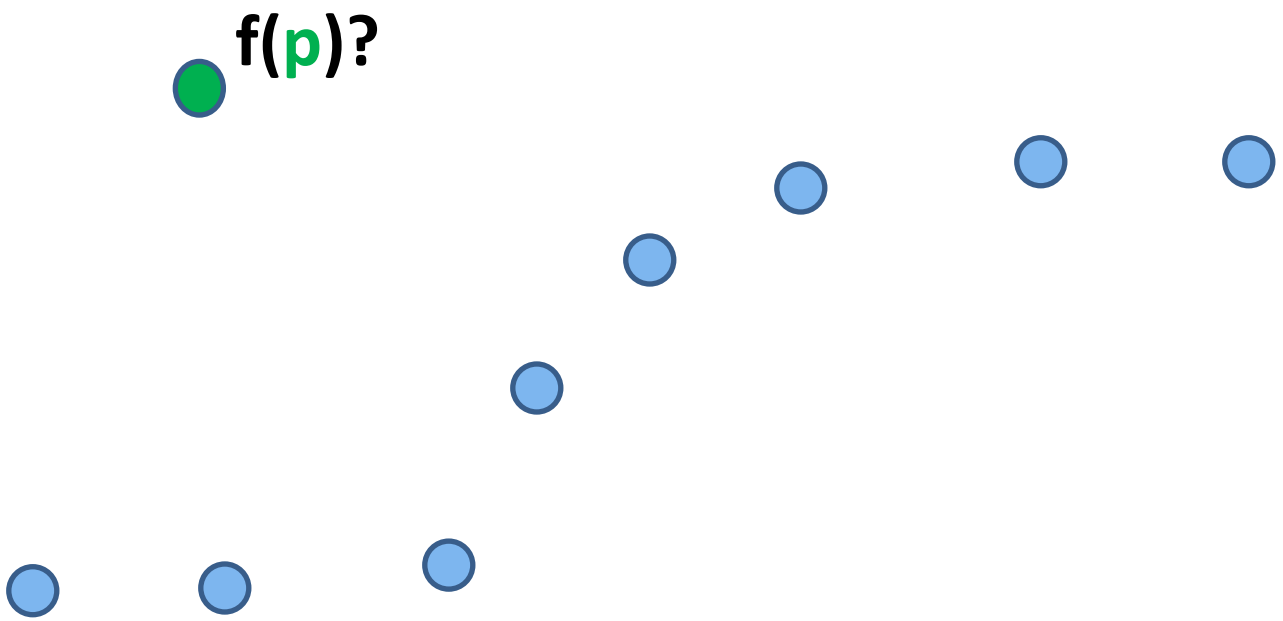
How to define implicit surface for reconstruction?

We will measure the **signed** distance to the unknown surface points in a uniform grid of points (green points)



How to define implicit surface for reconstruction?

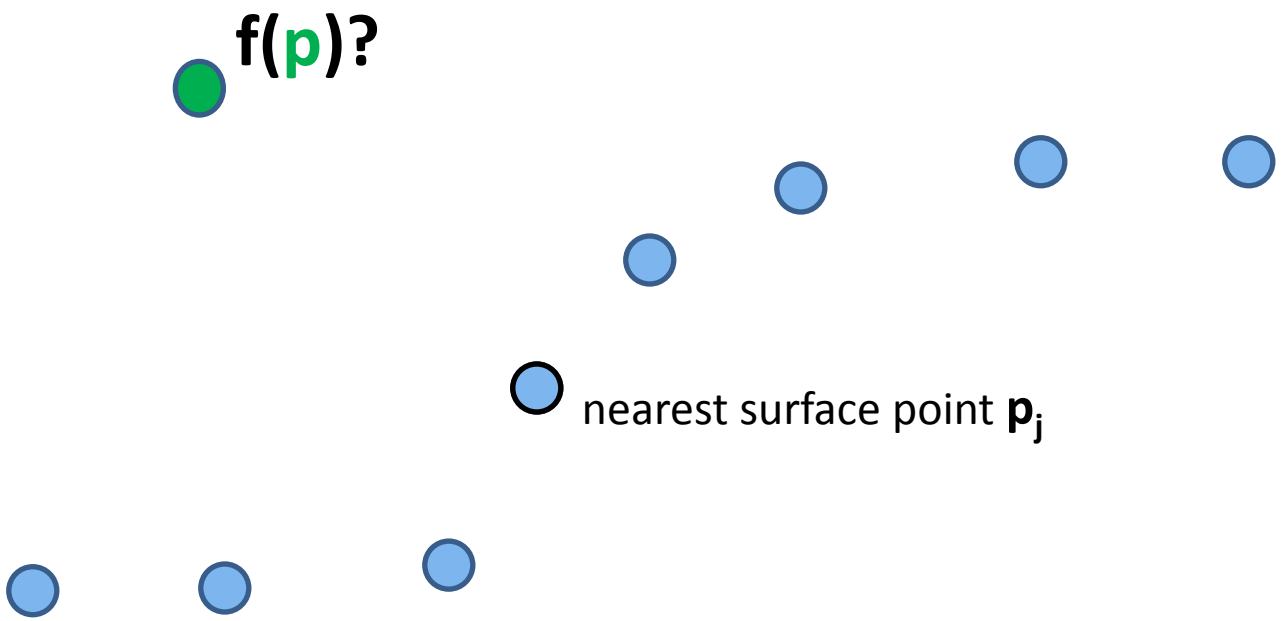
Let's focus on computing the **signed** distance f(x,y,z) for a single grid point **p={x,y,z}**:



How to define implicit surface for reconstruction?

One way is to find the nearest surface point to the grid point.

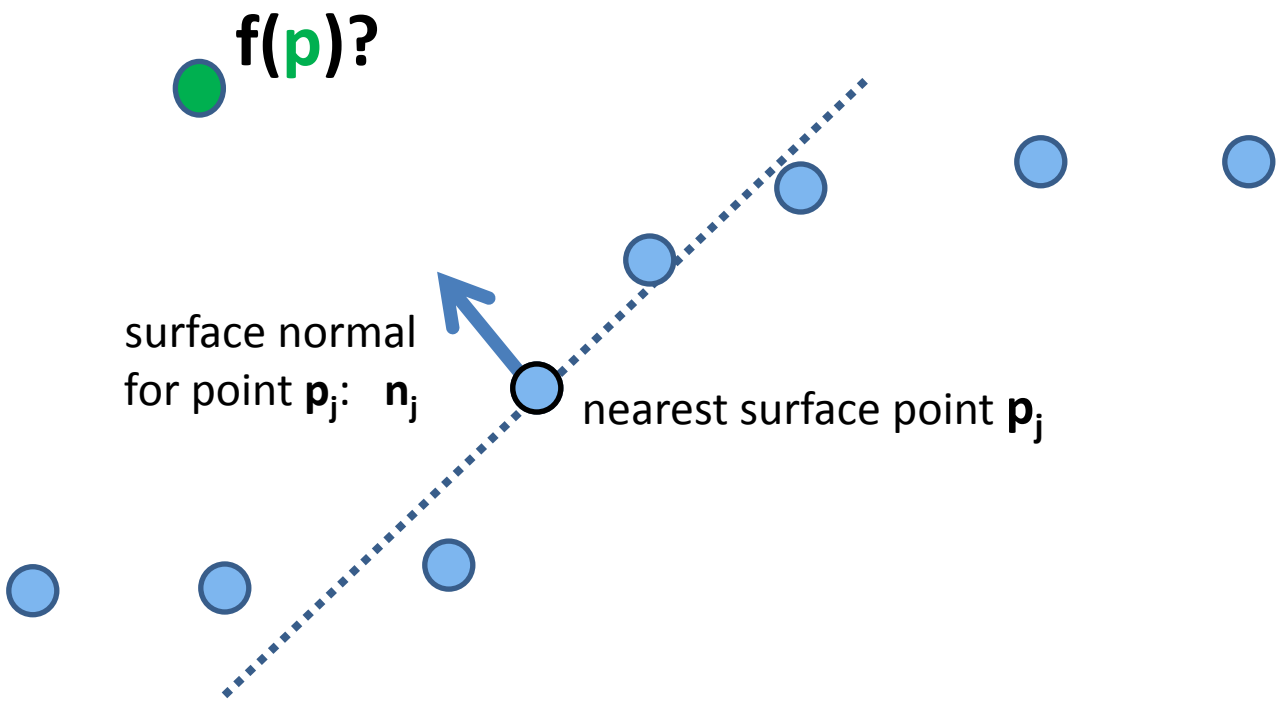
Then compute distance of the grid point to the tangent plane of that surface point



How to define implicit surface for reconstruction?

One way is to find the nearest surface point to the grid point.

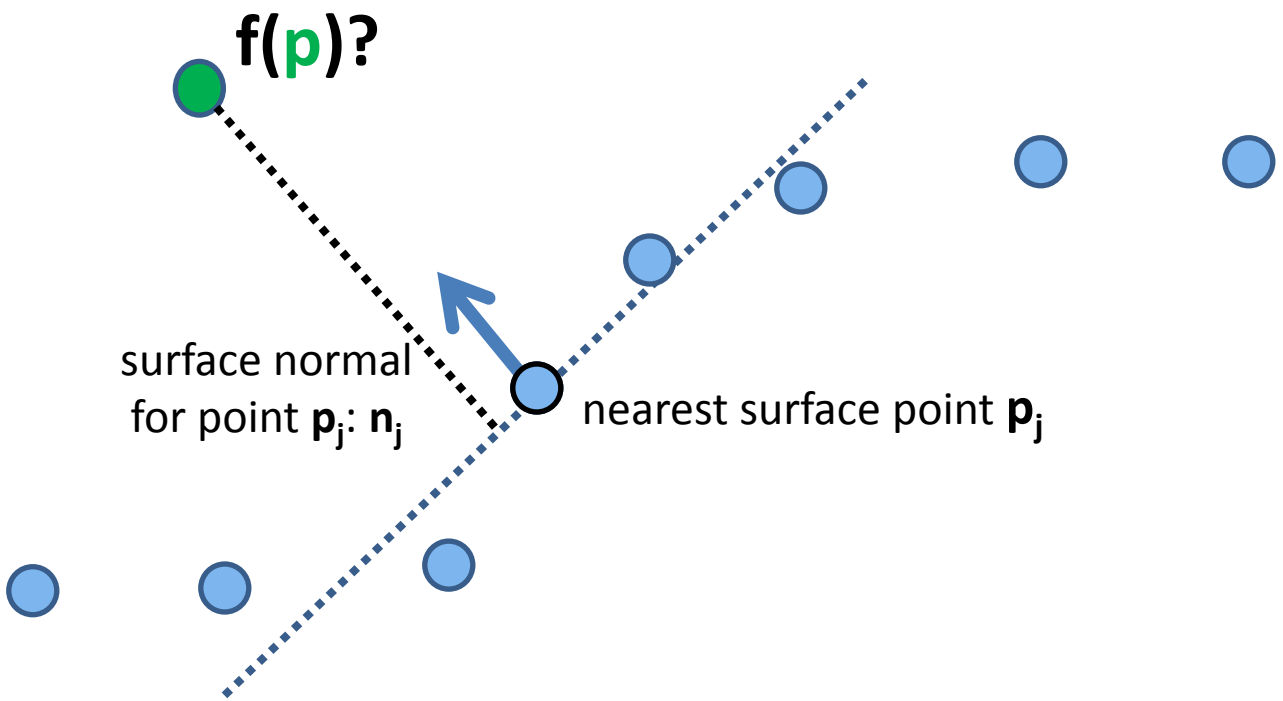
Then compute distance of the grid point to the tangent plane of that surface point



How to define implicit surface for reconstruction?

One way is to find the nearest surface point to the grid point.

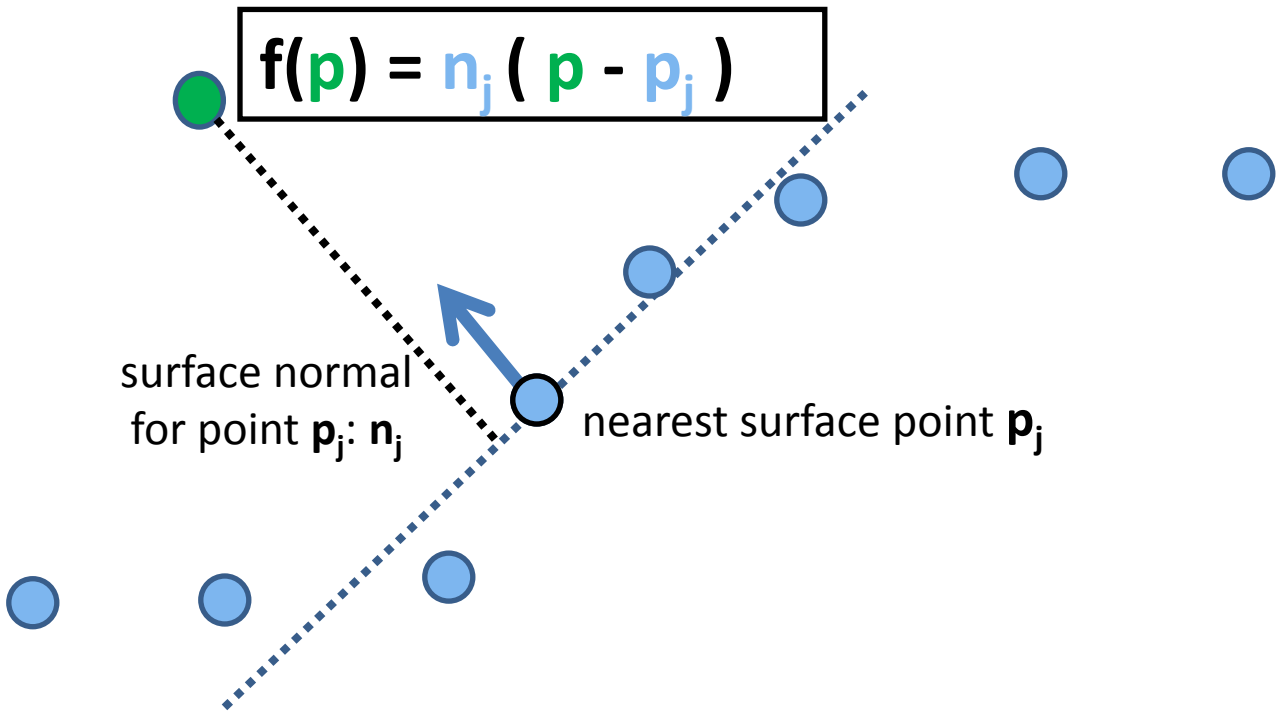
Then compute distance of the grid point to the tangent plane of that surface point



How to define implicit surface for reconstruction?

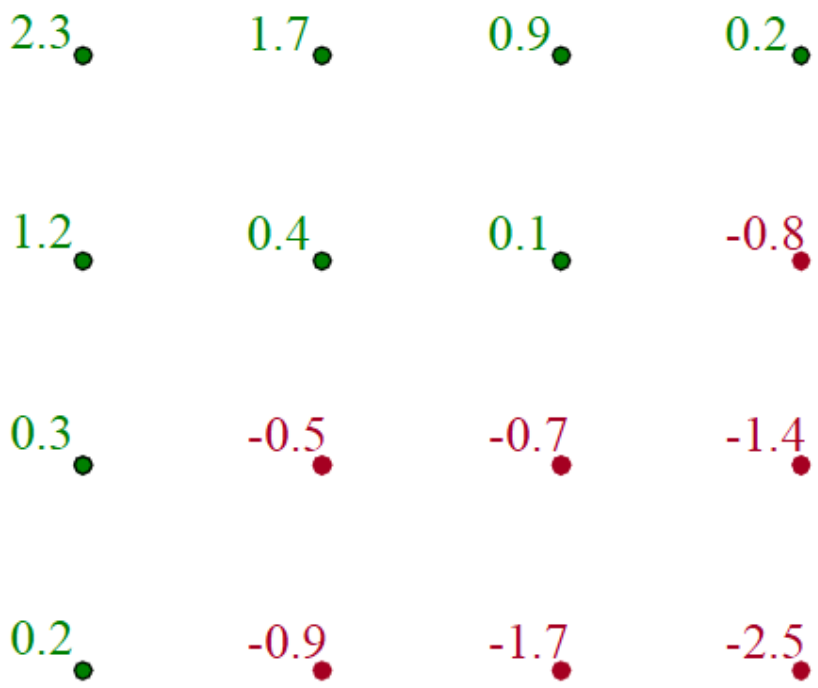
One way is to find the nearest surface point to the grid point.

Then compute distance of the grid point to the tangent plane of that surface point



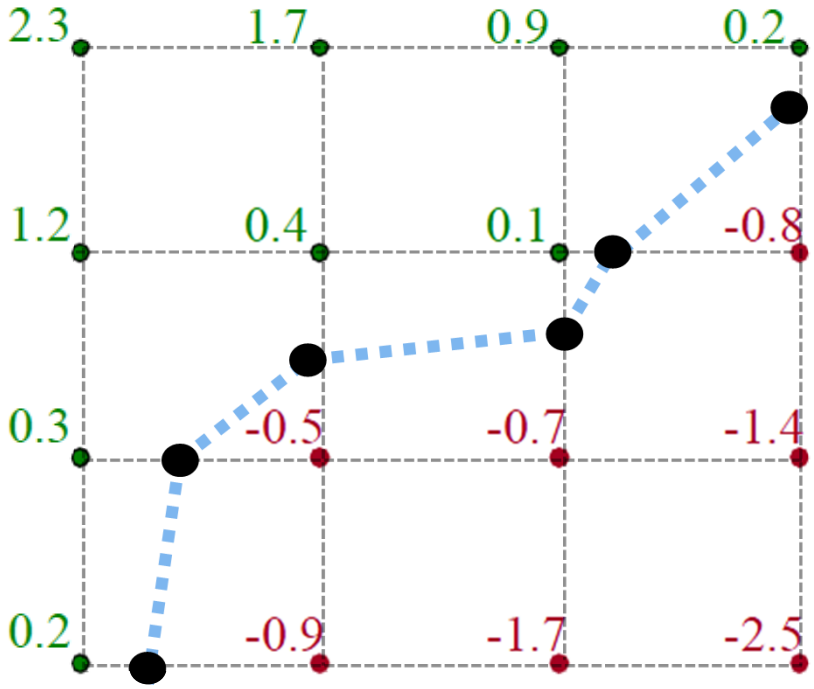
How to define implicit surface for reconstruction?

As a result, we get the implicit function measuring distance at a uniform grid



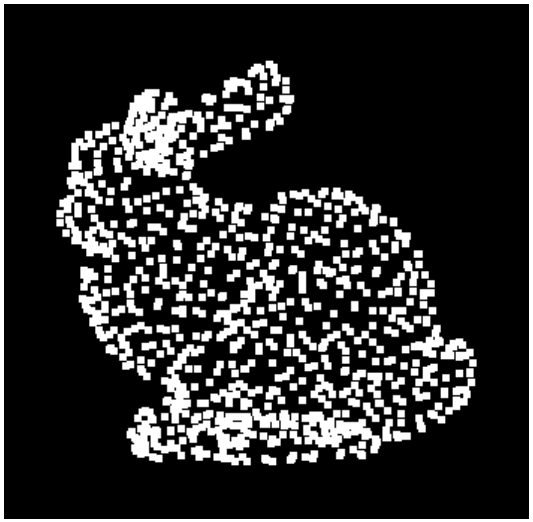
How to define implicit surface for reconstruction?

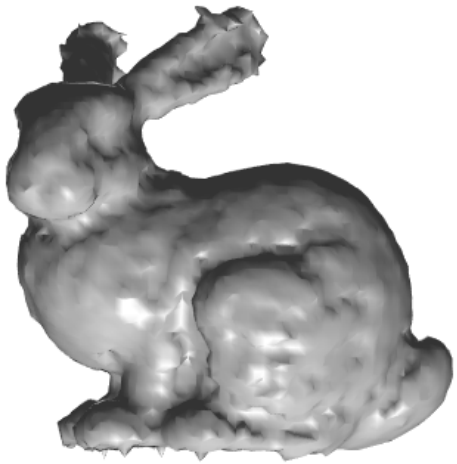
Linear interpolation to get a piecewise linear approximation of the surface



How to define implicit surface for reconstruction?

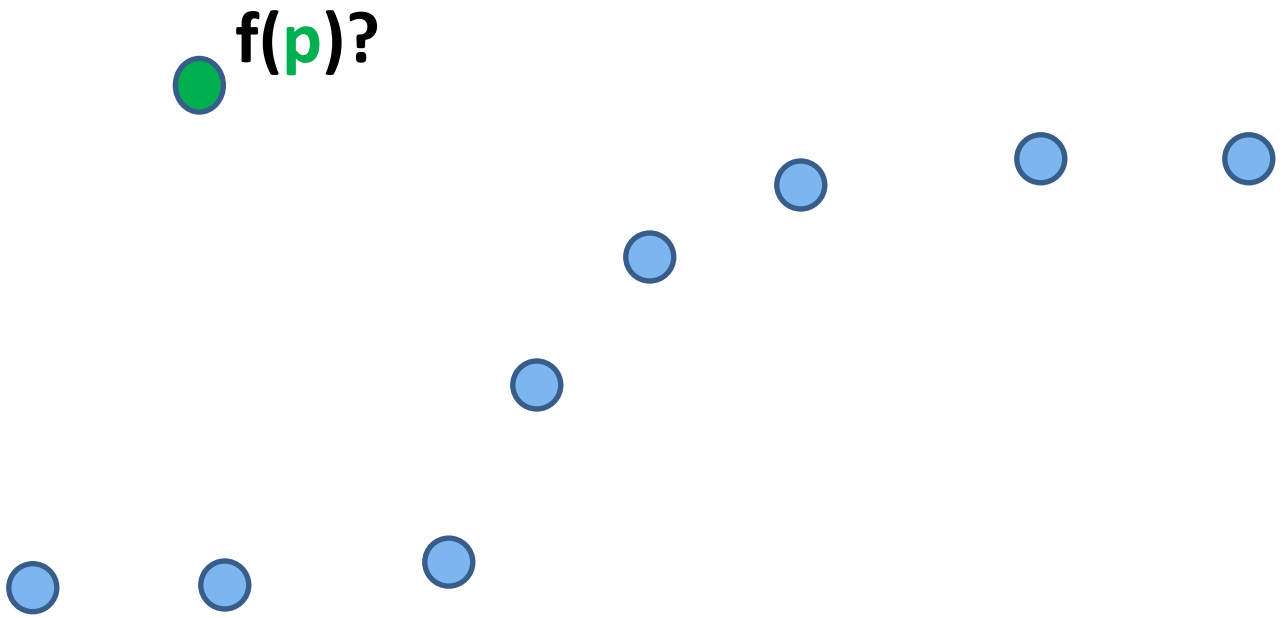
However, this implicit function definition is not good – the function changes abruptly along the query points of the grid (when their nearest points change)





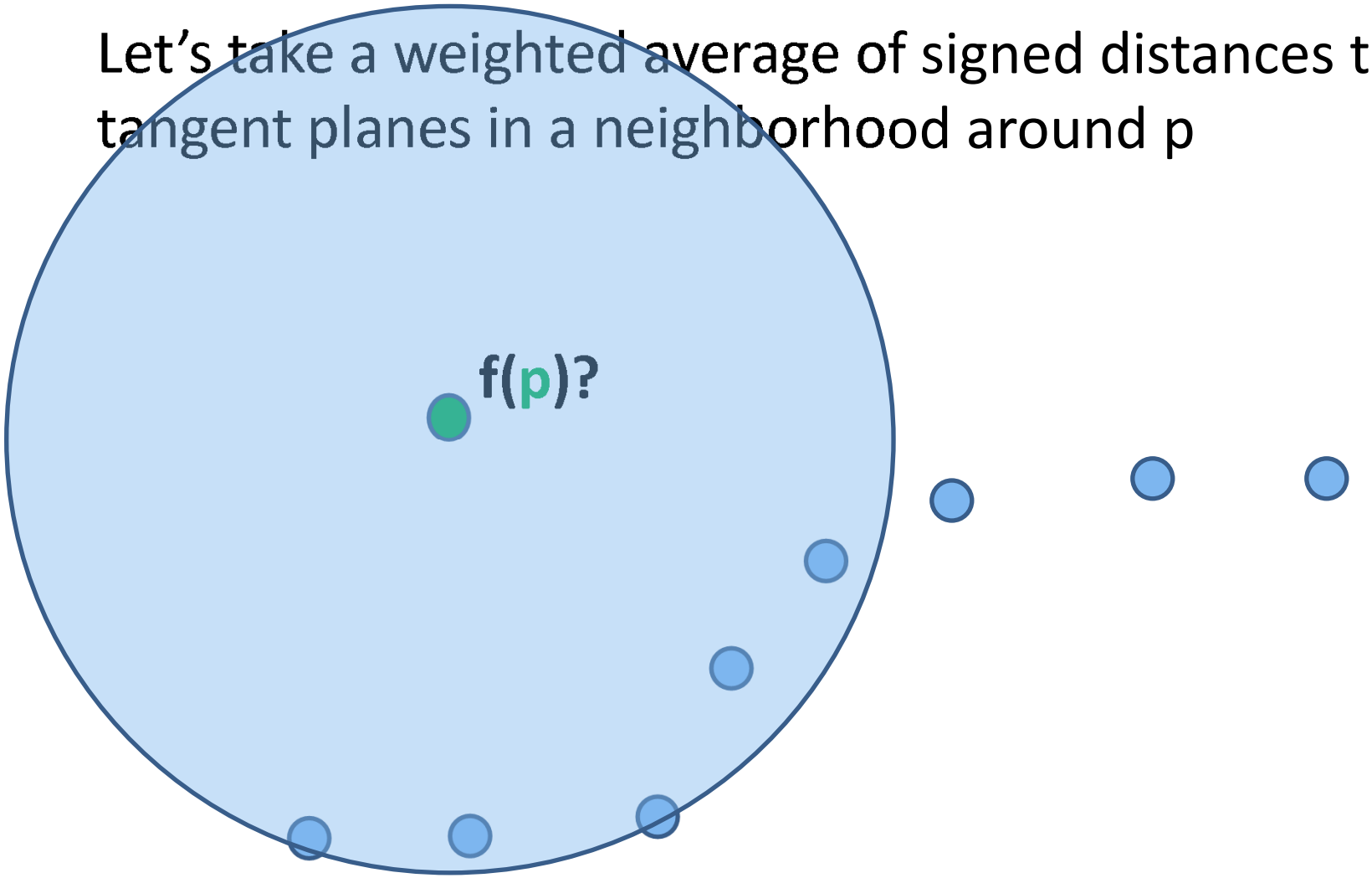
Moving Least Squares

Let’s take a weighted average of signed distances to tangent planes in a neighborhood around p



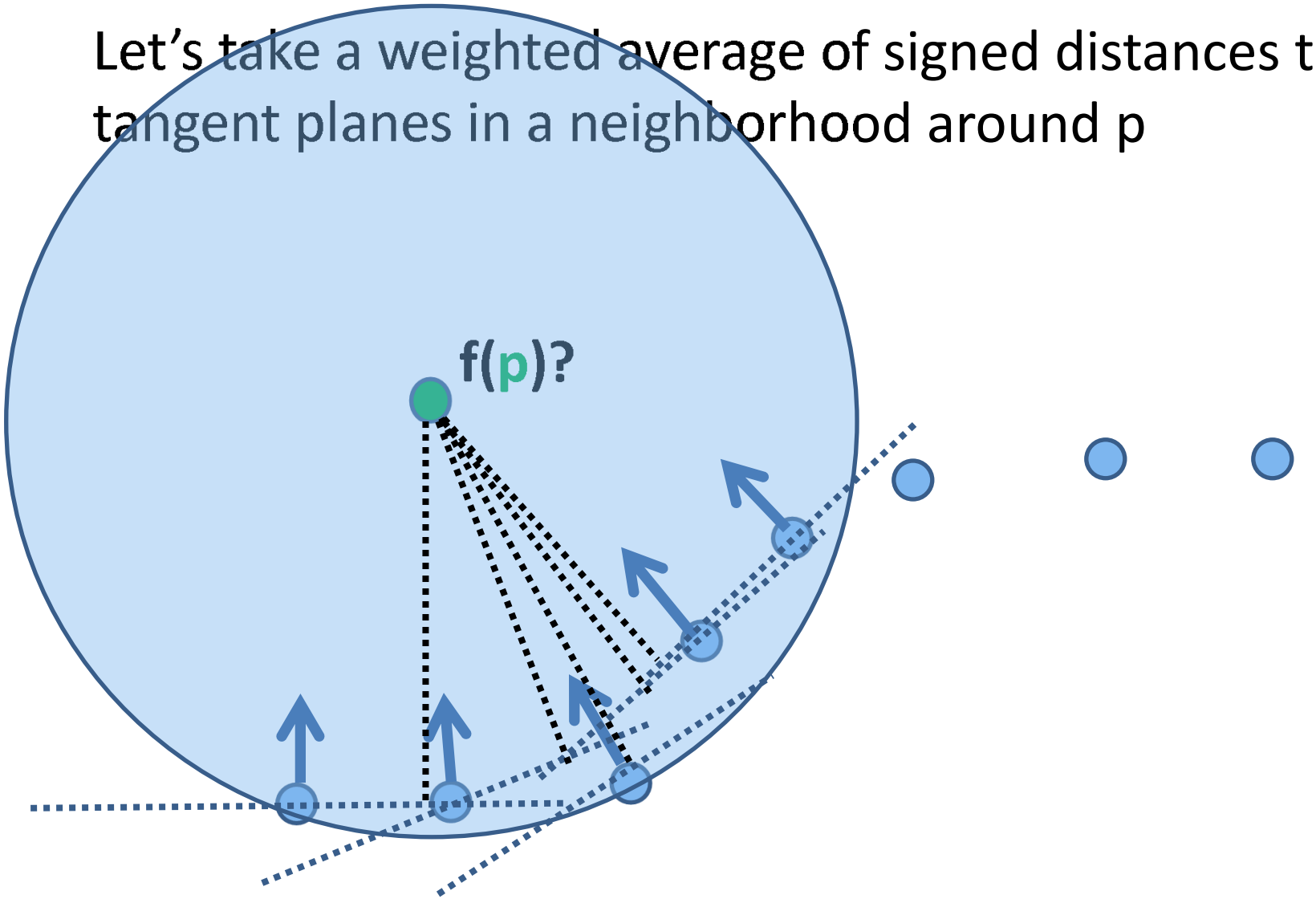
Moving Least Squares

Let's take a weighted average of signed distances to tangent planes in a neighborhood around p



Moving Least Squares

Let's take a weighted average of signed distances to tangent planes in a neighborhood around p



Moving Least Squares

Take a weighted average of all signed distances to tangent planes

$$f(\mathbf{p}) = \frac{\sum_i \mathbf{n_i}(\mathbf{p} - \mathbf{p_i})\varphi(\| \mathbf{p} - \mathbf{p_i} \|)}{\sum_i \varphi(\| \mathbf{p} - \mathbf{p_i} \|)}$$

$$\varphi(\| \mathbf{p} - \mathbf{p_i} \|) = \exp(- \| \mathbf{p} - \mathbf{p_i} \|^2 / \sigma^2)$$

Mesh-independent surface interpolation, 2003

Moving Least Squares

Take a weighted average of all signed distances to tangent planes

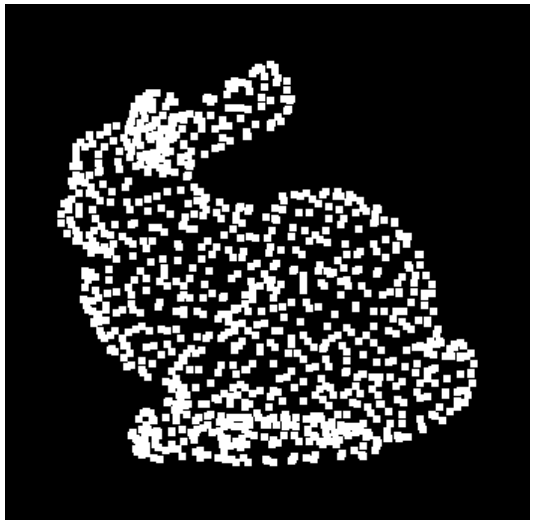
$$f(\mathbf{p}) = \frac{\sum_i \mathbf{n_i}(\mathbf{p} - \mathbf{p_i})\varphi(\| \mathbf{p} - \mathbf{p_i} \|)}{\sum_i \varphi(\| \mathbf{p} - \mathbf{p_i} \|)}$$

$$\varphi(\| \mathbf{p} - \mathbf{p_i} \|) = \exp(- \| \mathbf{p} - \mathbf{p_i} \|^2 / \sigma^2)$$

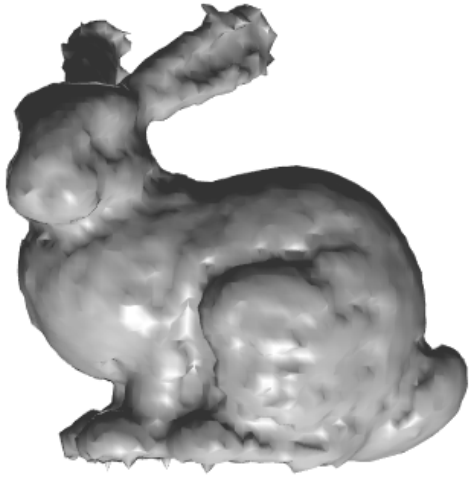
controls weight decay:
Set σ to average distance of each surface point to its nearest surface point multiplied by a small factor (like 2.0)

Moving Least Squares

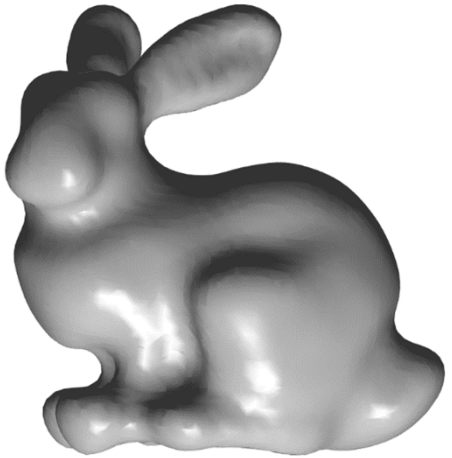
Point cloud (with normals)



SDF based on tangent plane of nearest point

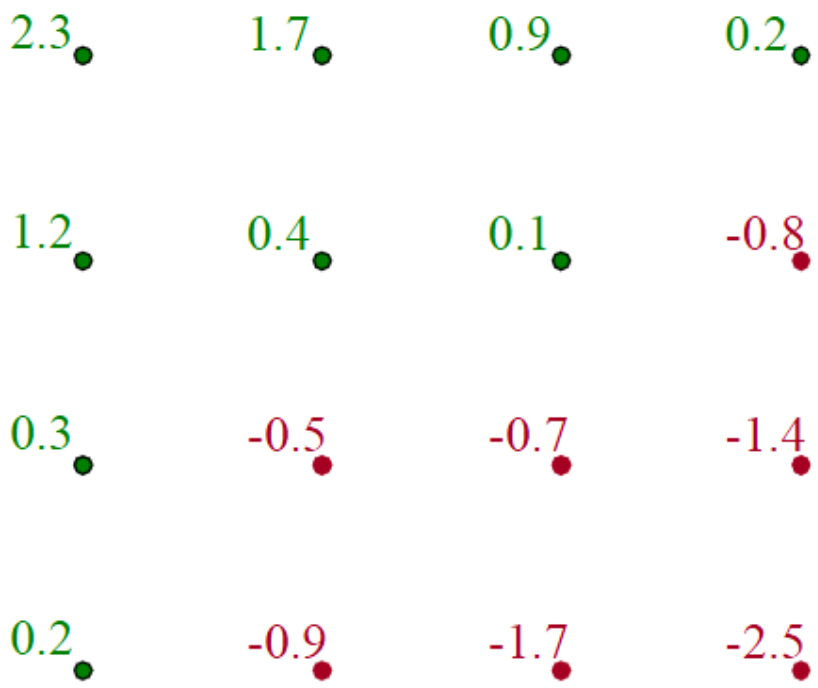


SDF based on MLS



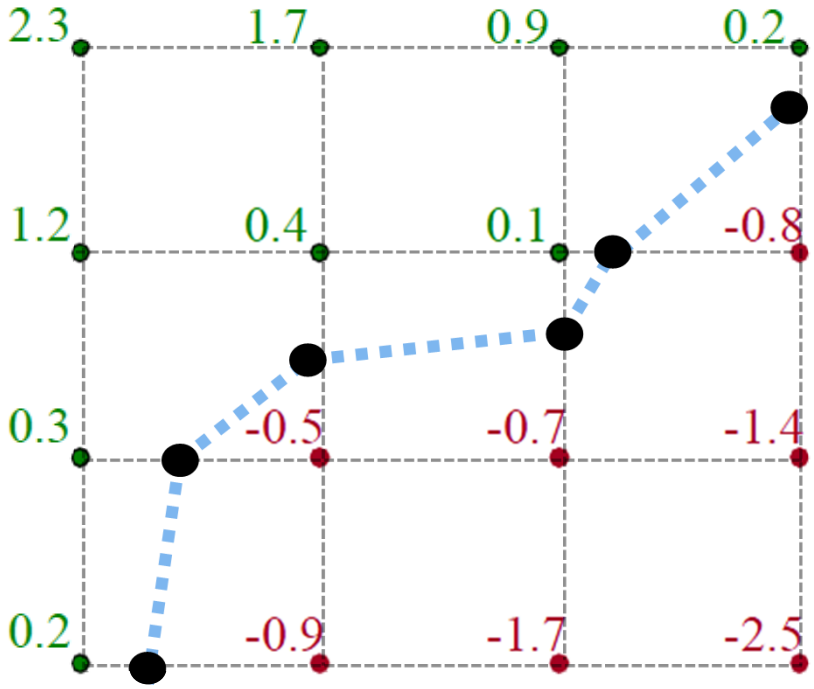
Isosurface extraction

Let’s start with 2D case



Isosurface extraction

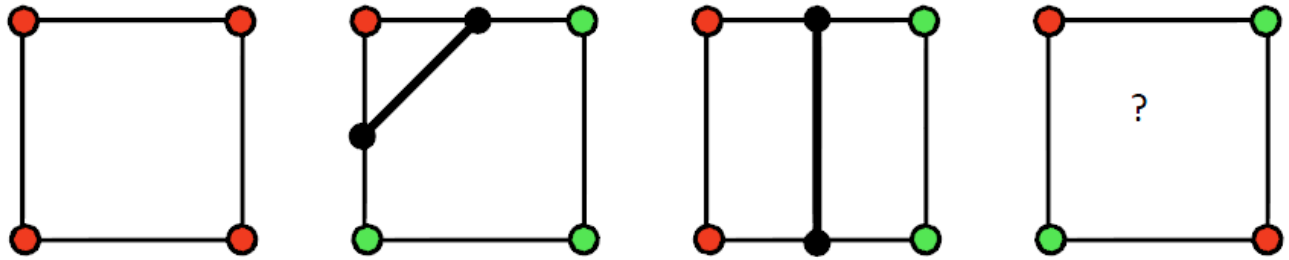
Let’s start with 2D case



Marching cubes – 2D

2^4 = 16 possible combinations of positive/negative (green/red) values on the vertices of the square

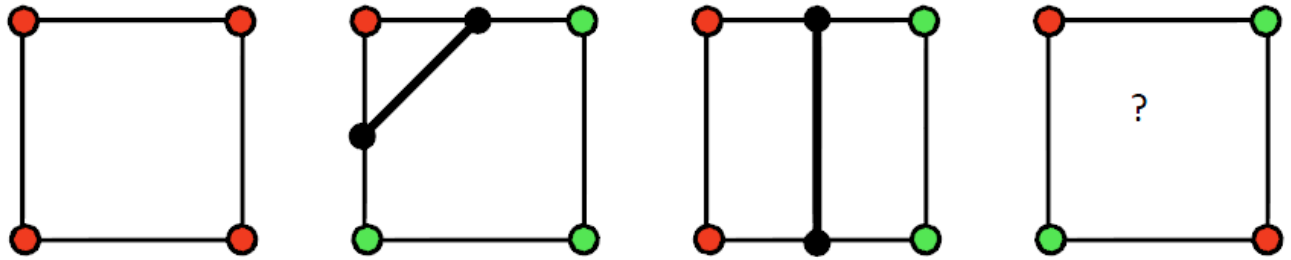
Due to symmetry, 4 unique configurations:



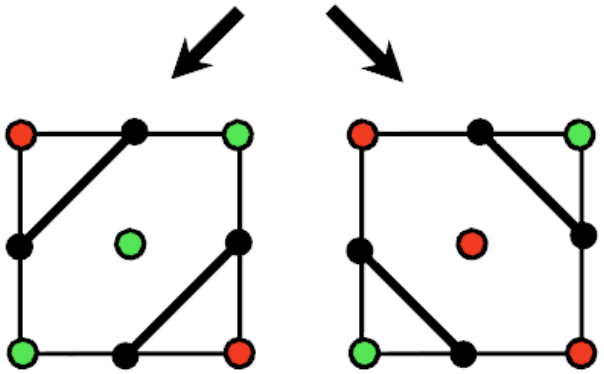
Marching cubes – 2D

2^4 = 16 possible combinations of positive/negative (green/red) values on the vertices of the square

Due to symmetry, 4 unique configurations:



Check value at the center point.



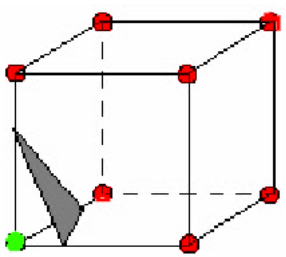
Marching cubes - 3D

Classify grid vertices as inside/outside

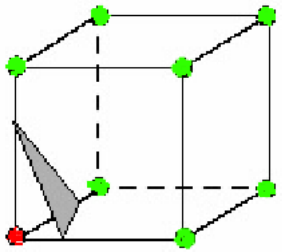
$2^8 = 256$ cases in 3D

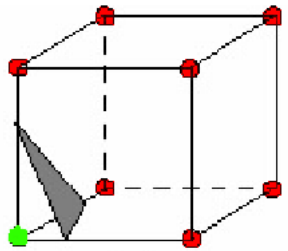
Look-up table for each case - each entry defines the edge configuration to be created

Due to symmetries (see below), 15 unique configurations

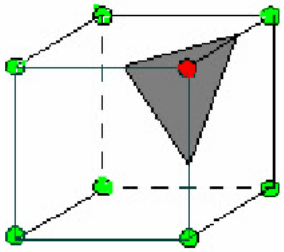


reverse case:

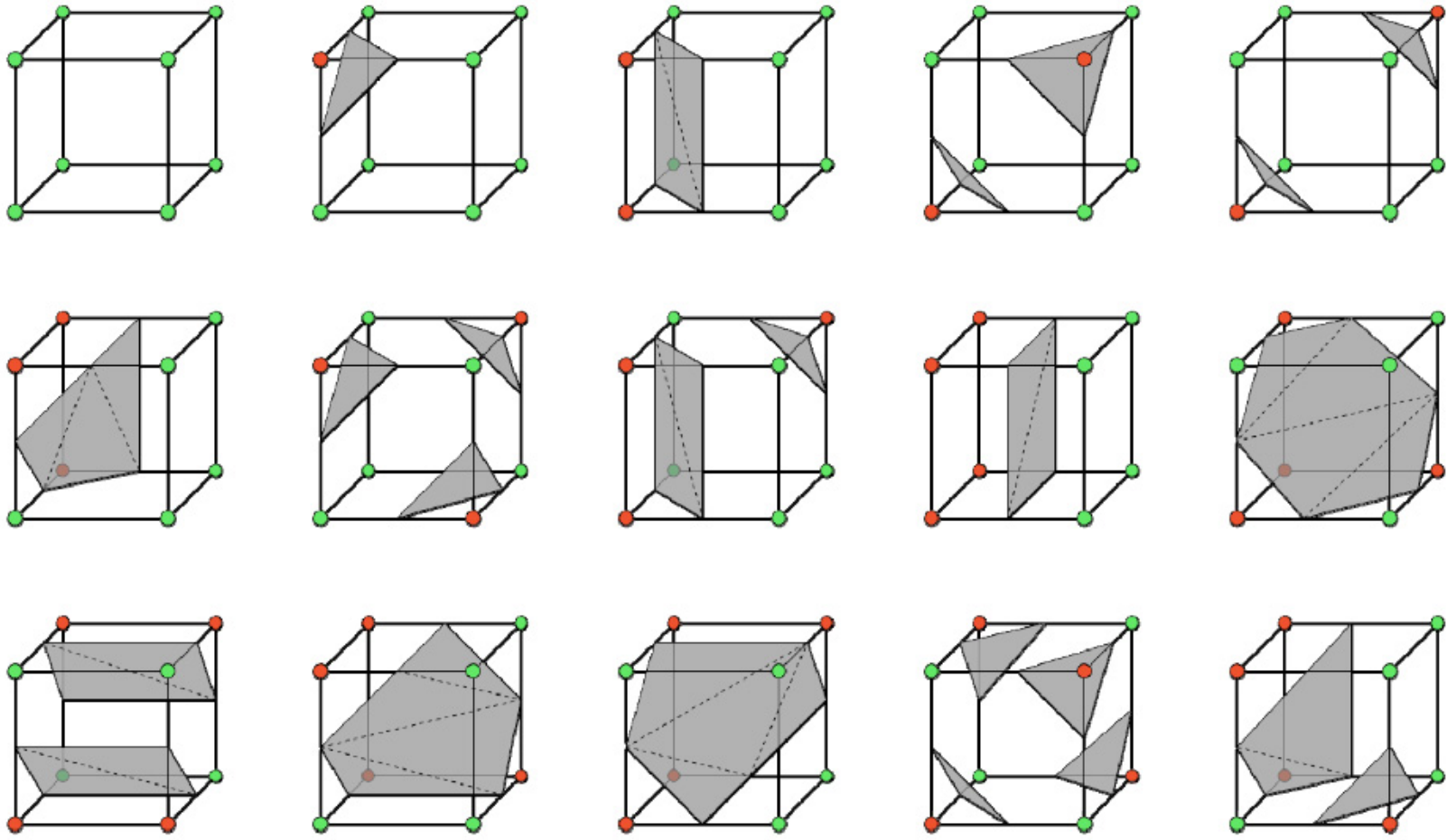




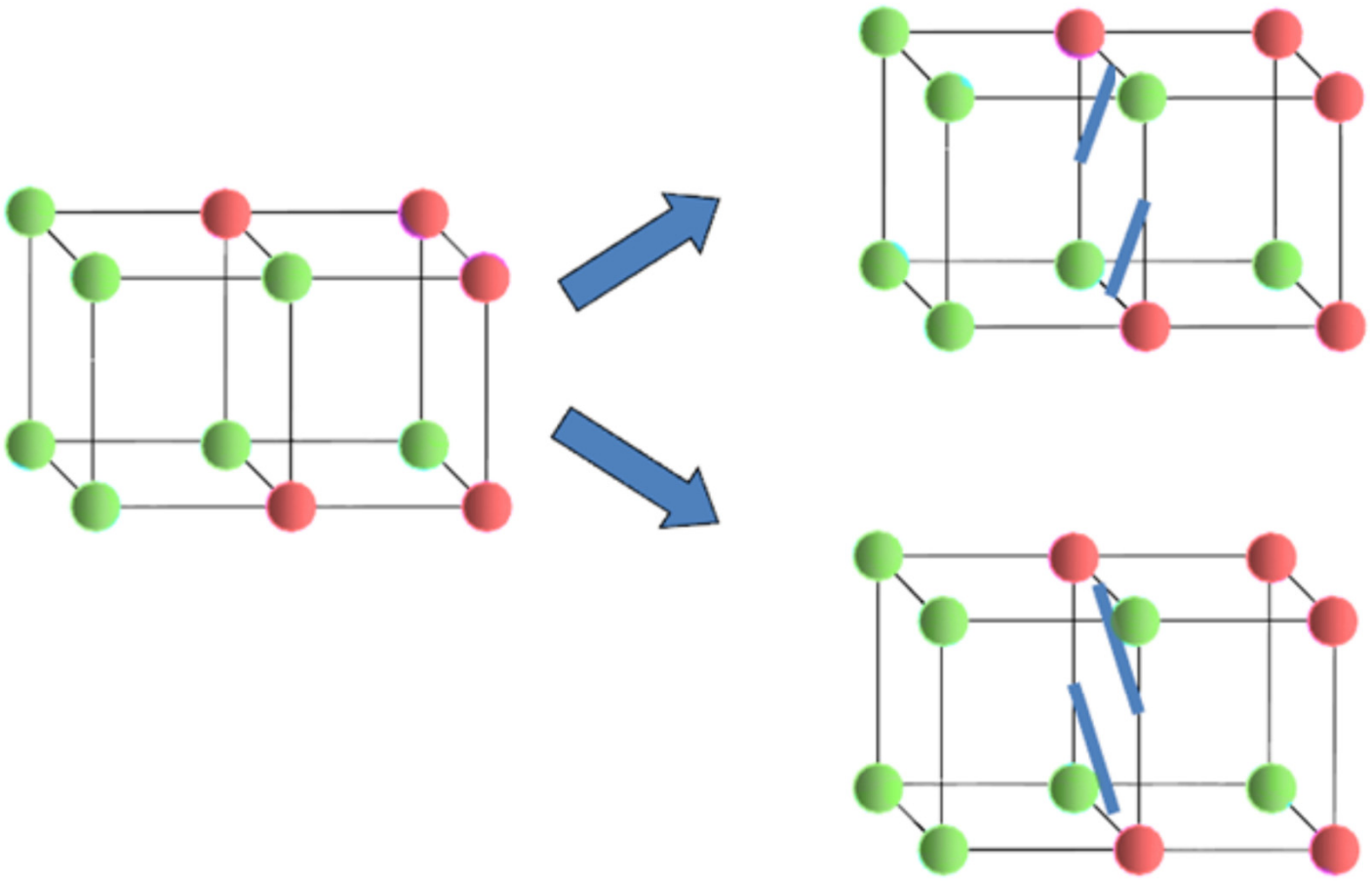
symmetric case:



Marching cubes - 3D

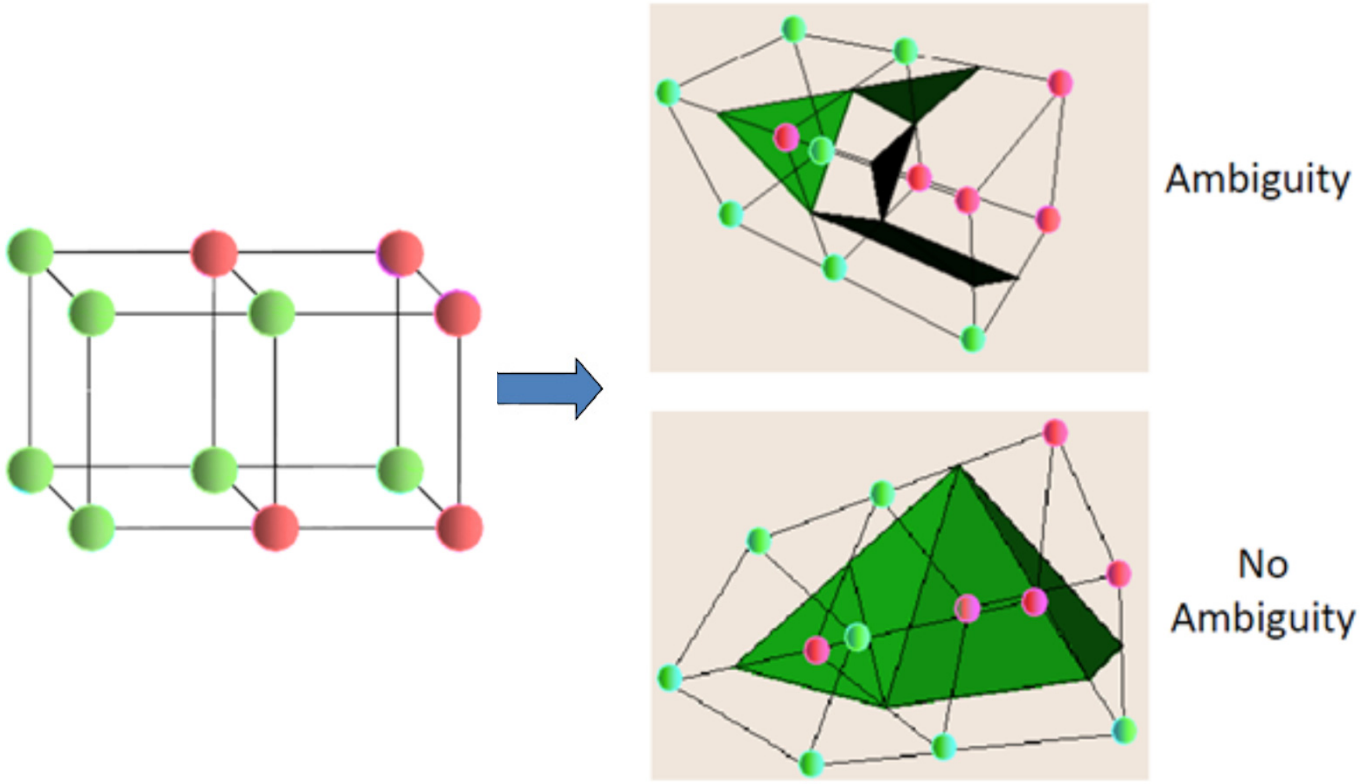


Ambiguity also in 3D



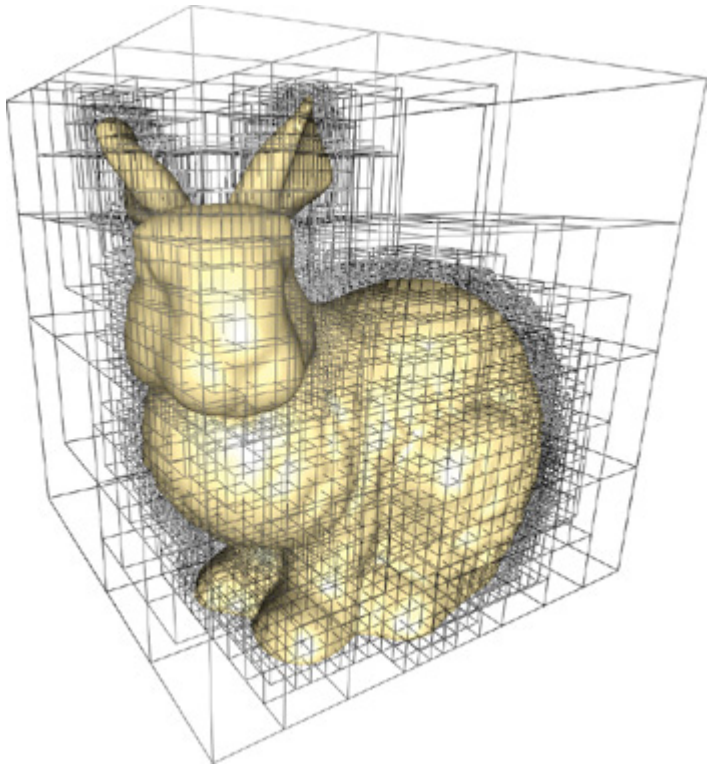
Resolve ambiguity

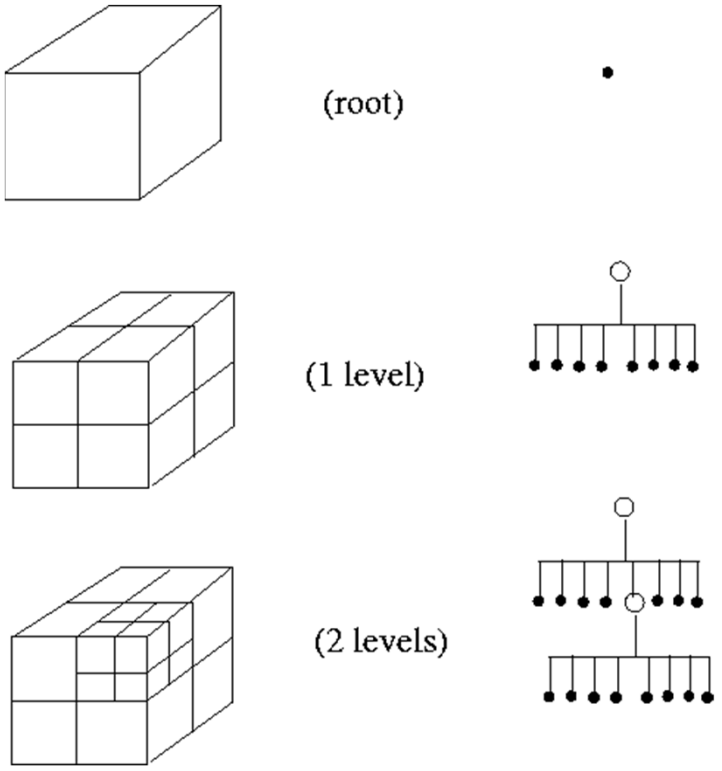
Look at neighboring cube, and select configuration that avoids holes



Adaptive grids (octrees)

Use adaptive grid (**octree** data structure)





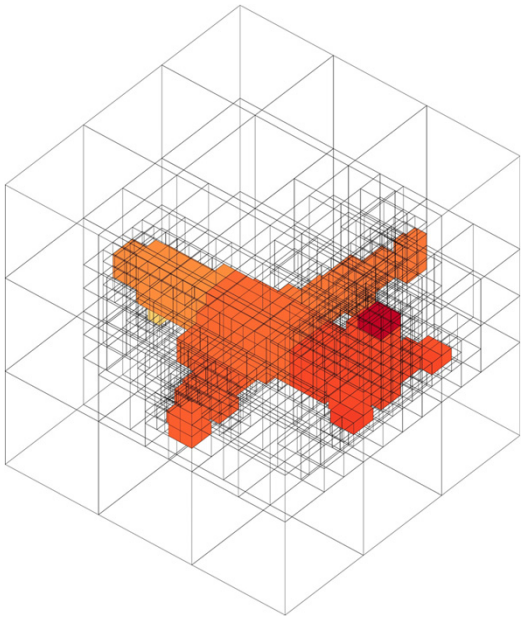
So far we have discussed:

Part I: Image representations

Part II: Shape representations

- Polygon Meshes
- Parametric Surfaces
- Voxel grids … ***or adaptive grids (octrees)***
- Point clouds
- Implicit functions

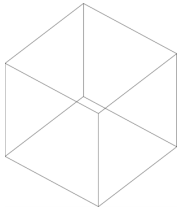
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

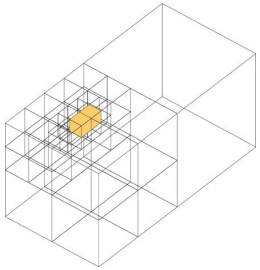
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

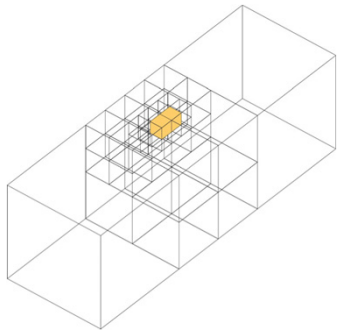
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

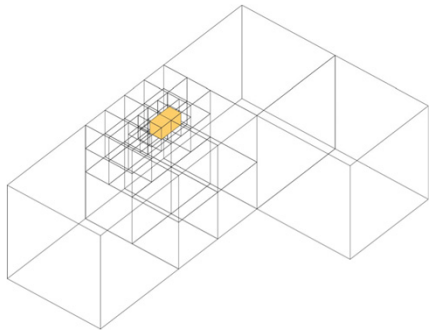
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

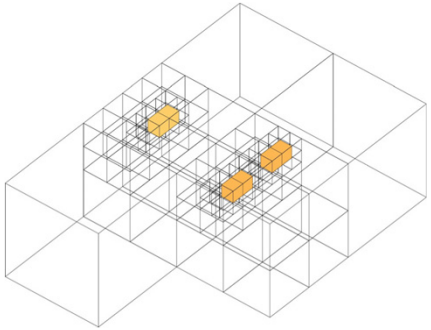
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

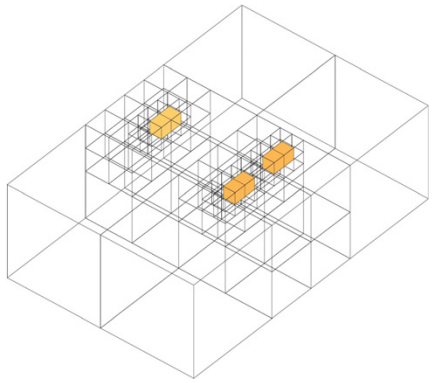
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

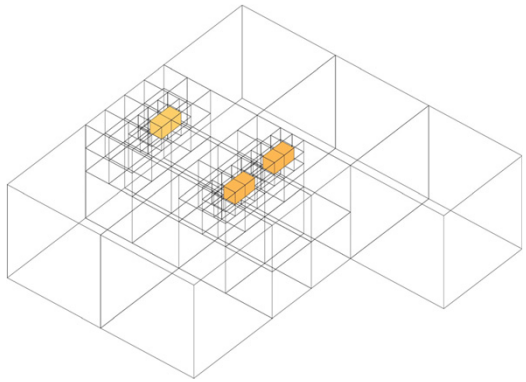
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

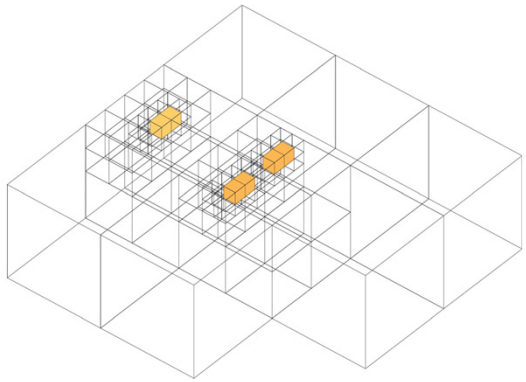
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

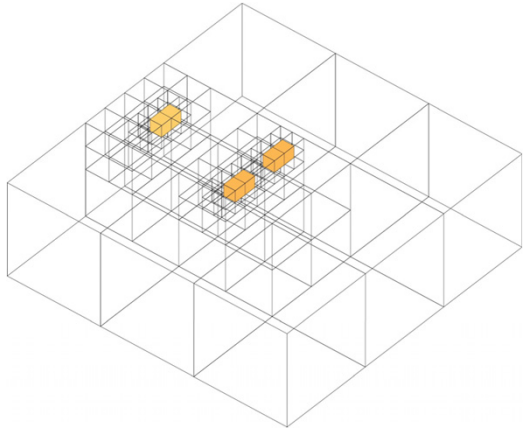
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

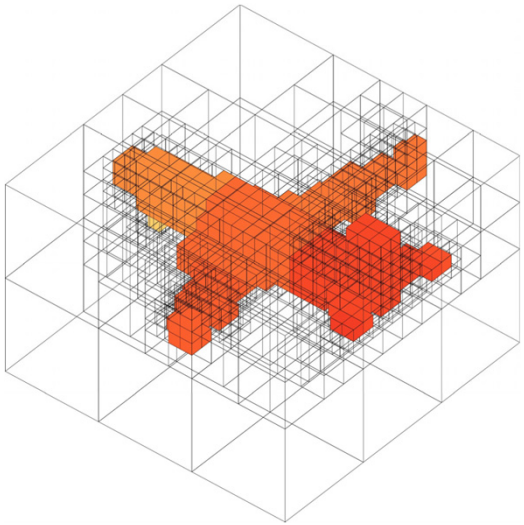
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

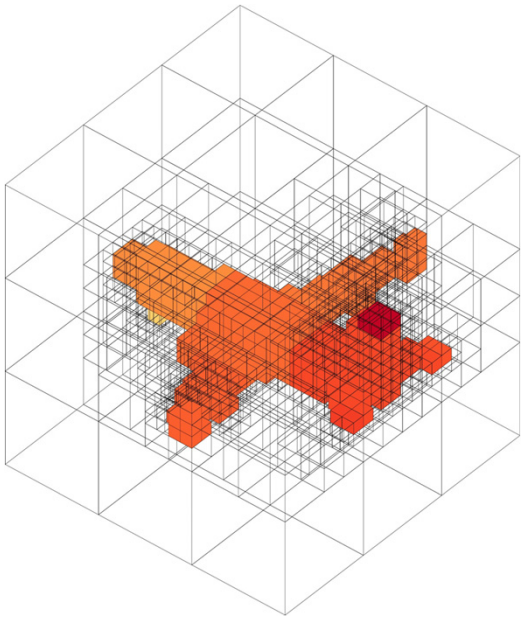
Octrees



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

Octrees



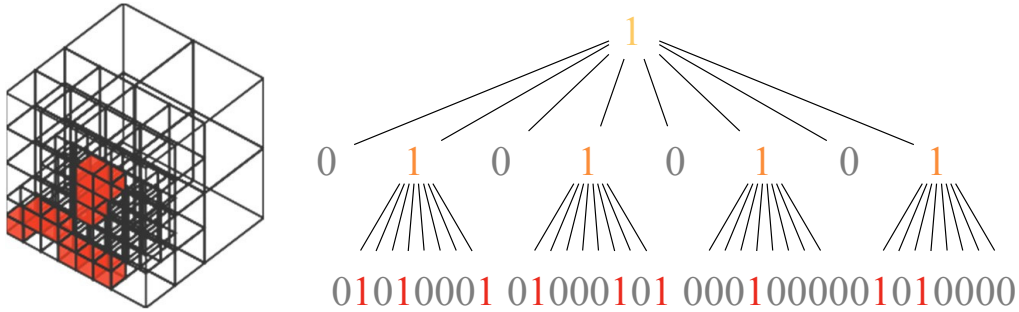
[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

Octrees

Octrees are efficiently encoded as bit-strings

1010101010000000001010001000000000100010100000000000010000
0000000001010000



[Slides from Riegler et al. OctNet]

OctNet: Learning Deep 3D Representations at High Resolutions, 2017

Texture representations

Part I: Image representations

Part II: Shape representations

- Polygon Meshes
- Parametric Surfaces
- Voxel grids *or adaptive grids (octrees)*
- Point clouds
- Implicit functions

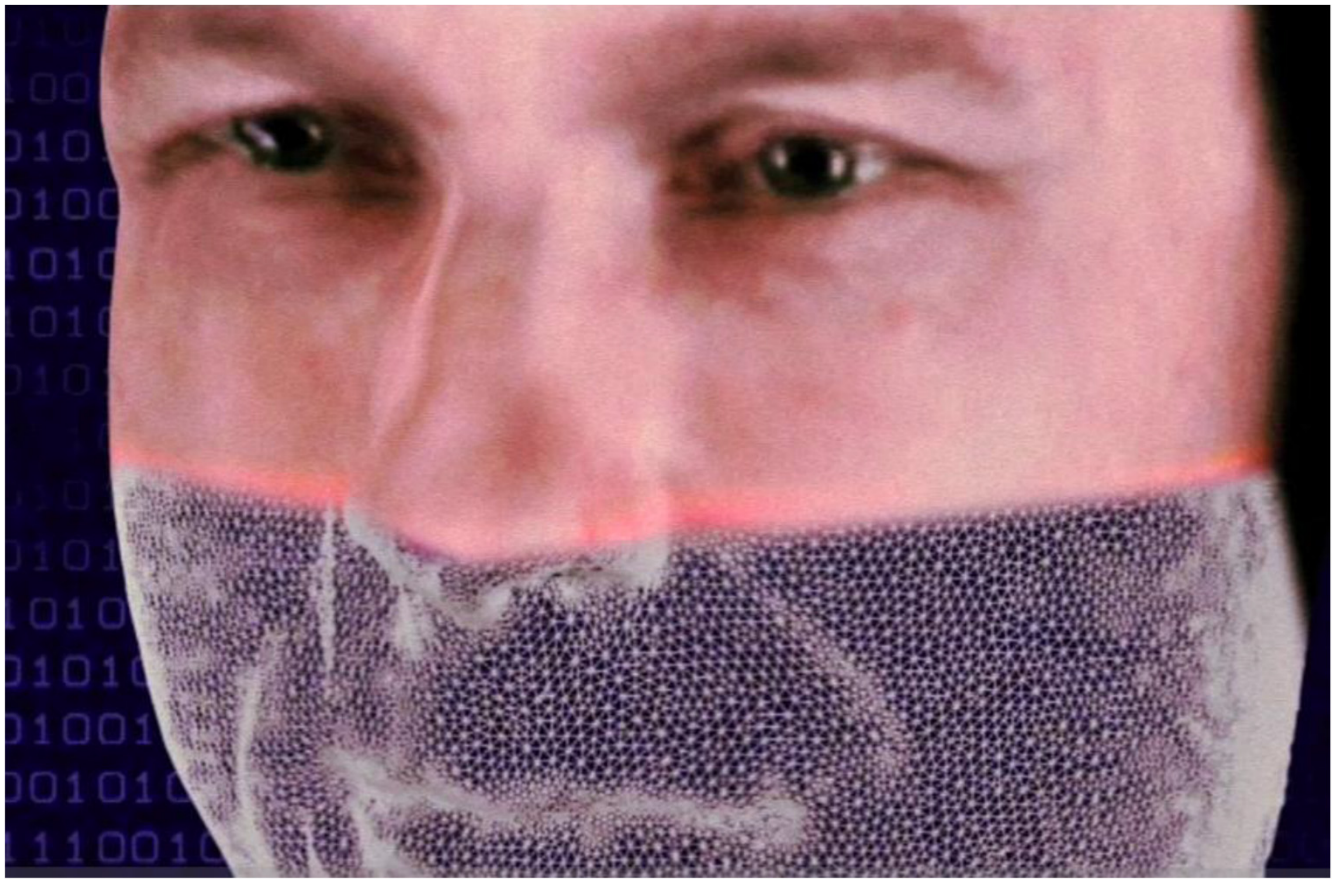
Part III: Textures

Textures: add visual details

Make a surface look **more colorful and realisti**c!

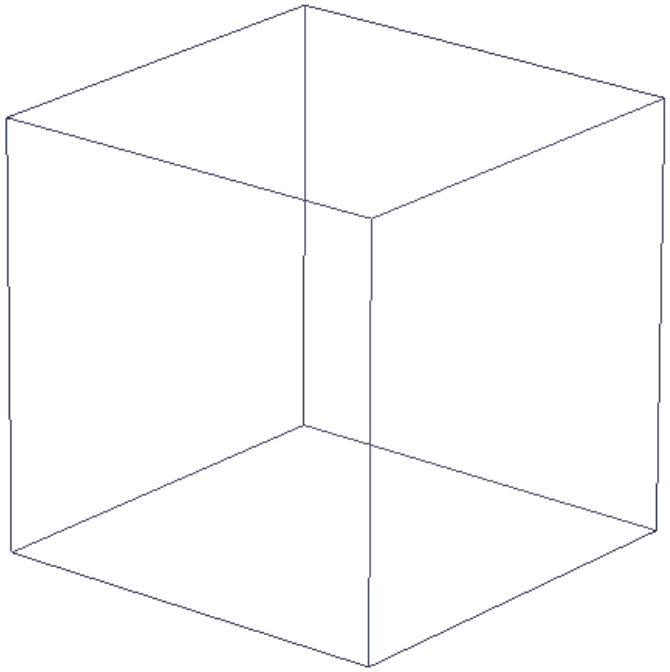
One option is to ...

Set material color per vertex for meshes
(or per voxel in voxel grids, or per point in point clouds...)



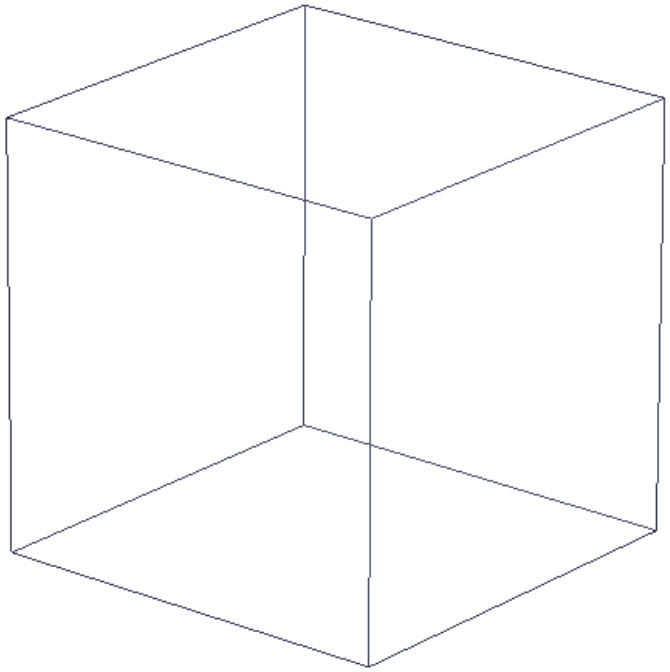
High-res texture, low-res geometry

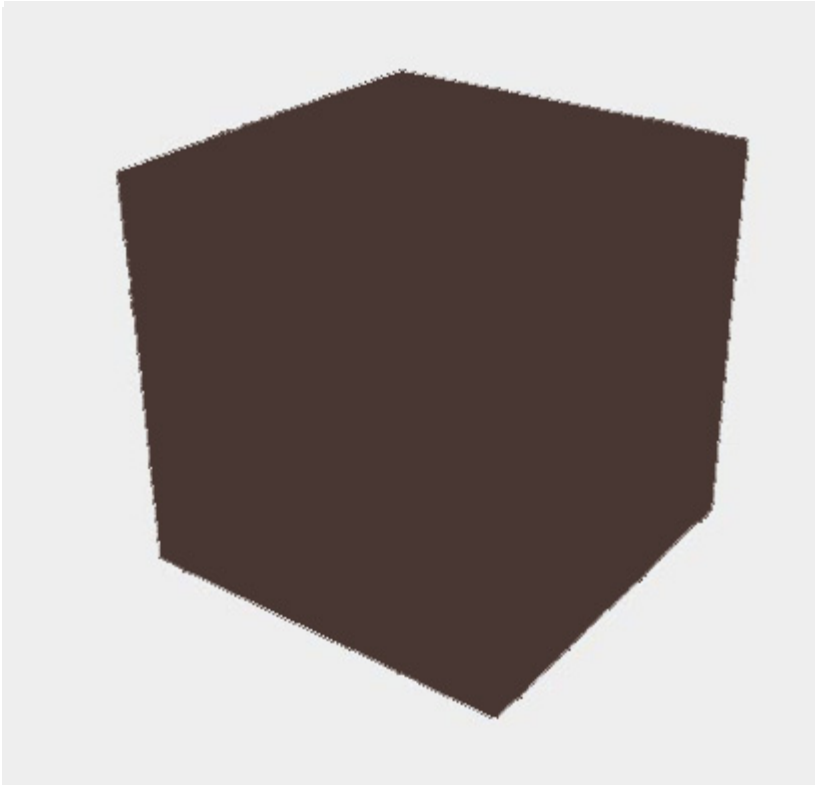
Let’s say you want to design a crate



High-res texture, low-res geometry

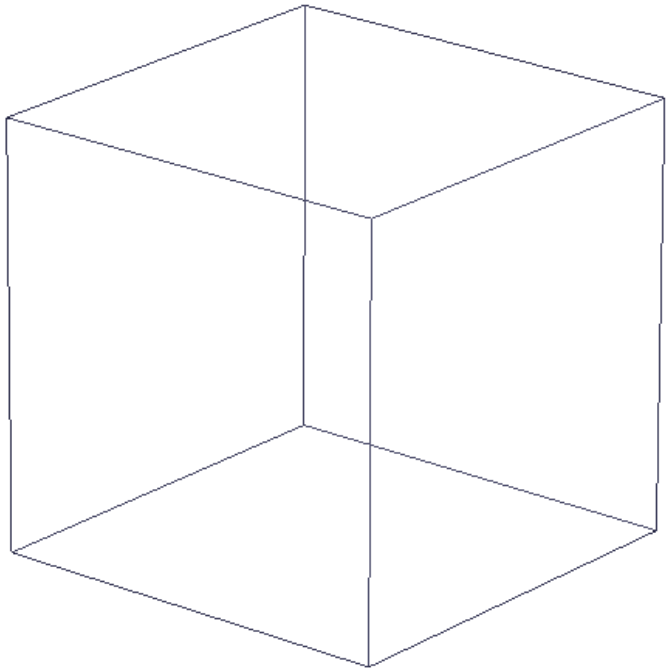
If you set a material color per vertex, you'll get this:





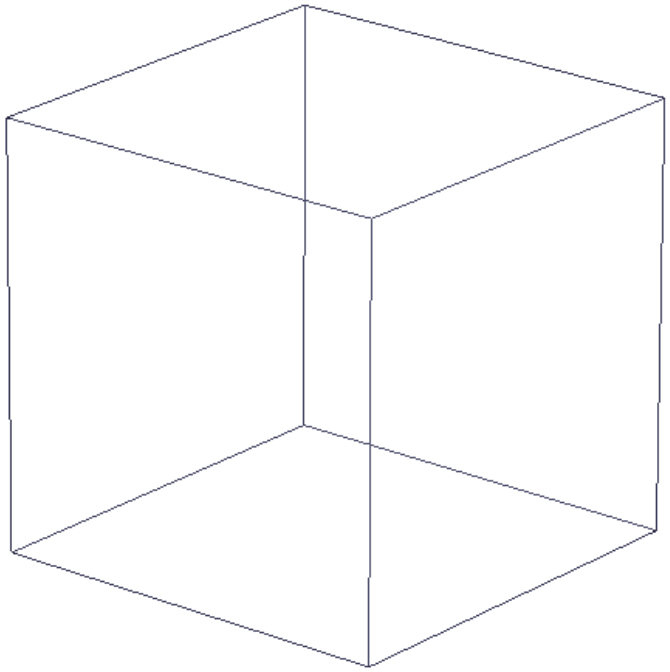
High-res texture, low-res geometry

Texture and geometry resolution do not necessarily coincide.



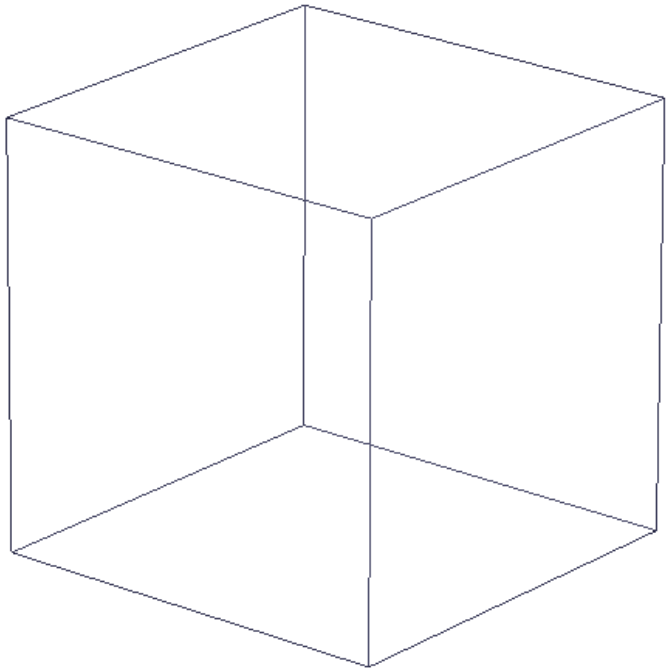
High-res texture, low-res geometry

In fact**, it is NOT necessary to have a highly detailed polygon mesh**. Designing accurate geometry is hard!



High-res texture, low-res geometry

BUT you can have high-res textures based on photos or image editing. This is the key to success in many games.



Where do textures come from?

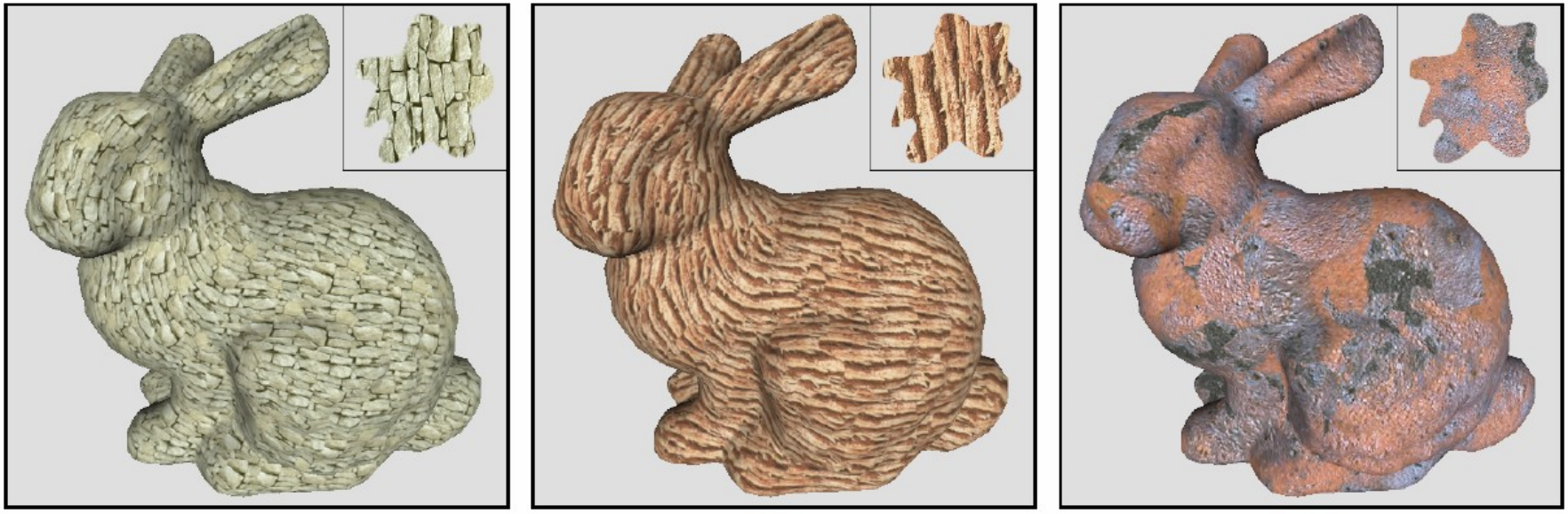
Usually **photographs or synthesized images** e.g.:

wood

metal

Advantages of using 2D images as textures...

Easy to find/store/edit 2D images, easily change surface appearance



Advantages of using 2D images as textures...

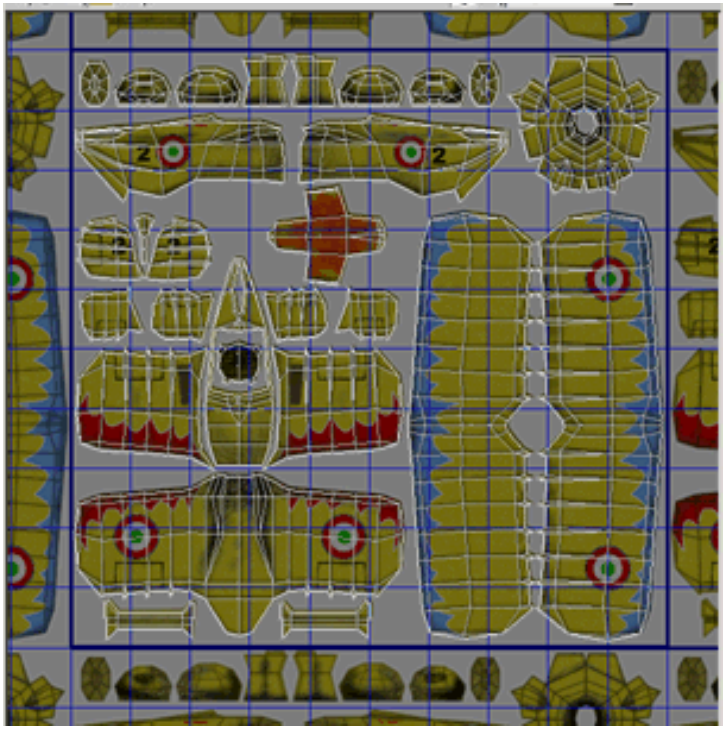
Also **easily animate surface appearance.**

A fountain in the game “World of Warcraft”

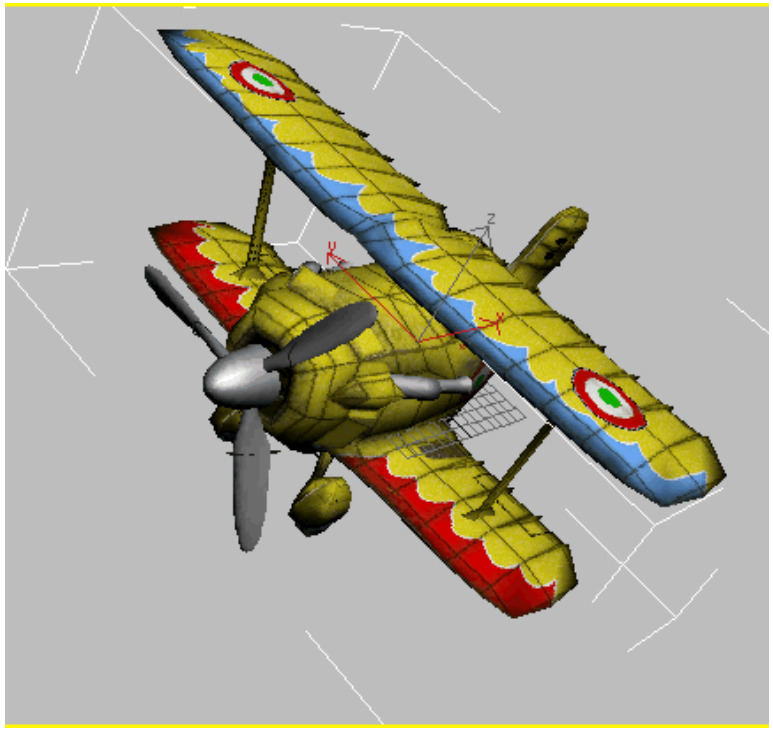
Problem: mapping between 2D and 3D…

However, images are defined in **2D**. Our shapes are **3D**.

2D textures



3D model

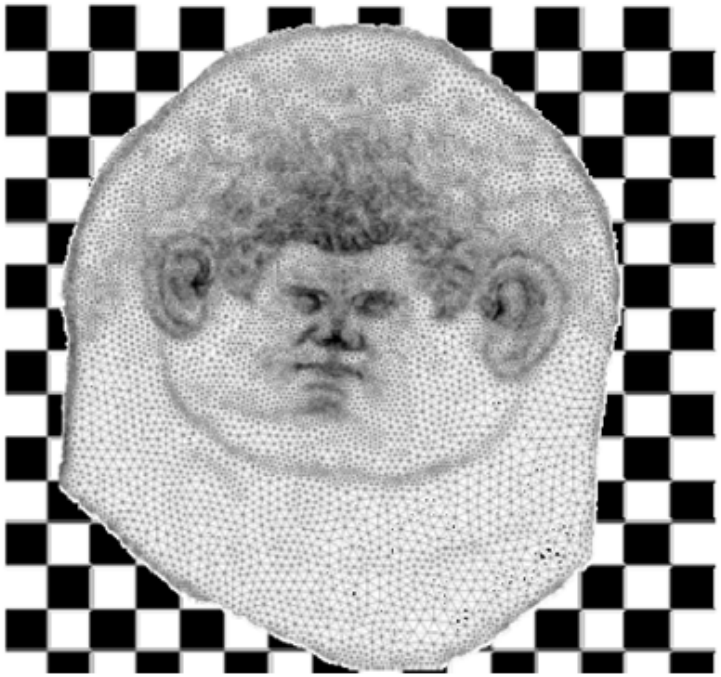


We need a way to flatten a 3D surface...

Polygon Mesh

We need a way to flatten a 3D surface...

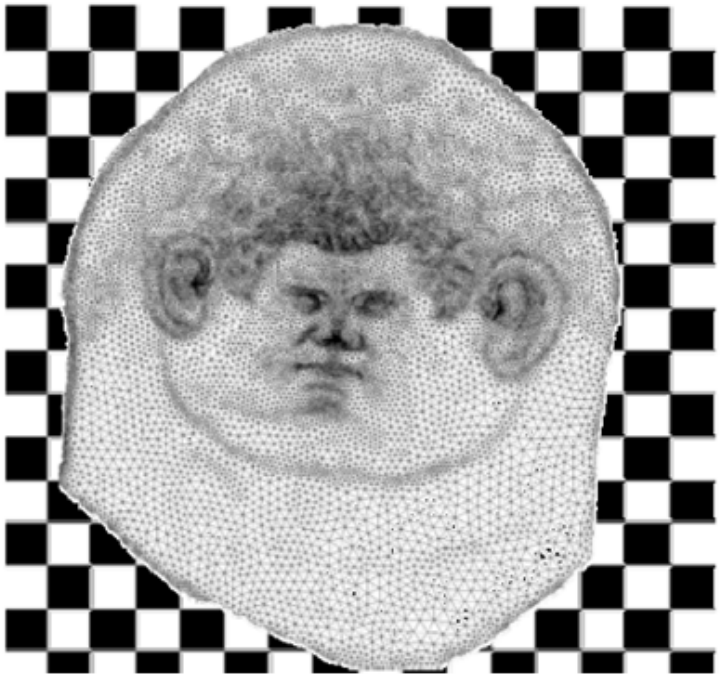
Polygon Mesh



Flattened mesh
every vertex now is associated with a 2D coordinate in image (texture space)

We need a way to flatten a 3D surface...

Polygon Mesh



Flattened mesh
every vertex now is associated with a 2D coordinate in image (texture space)

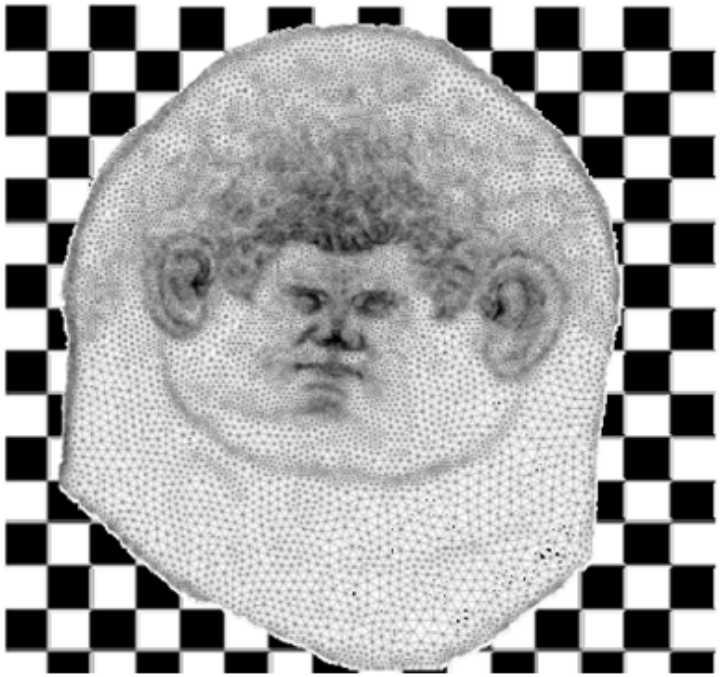
Mr Checkerboard

The reason I showed Mr Checkerboard

... is that **flattening a mesh causes distortions**!

(note: unless your mesh is a developable surface)

Polygon Mesh



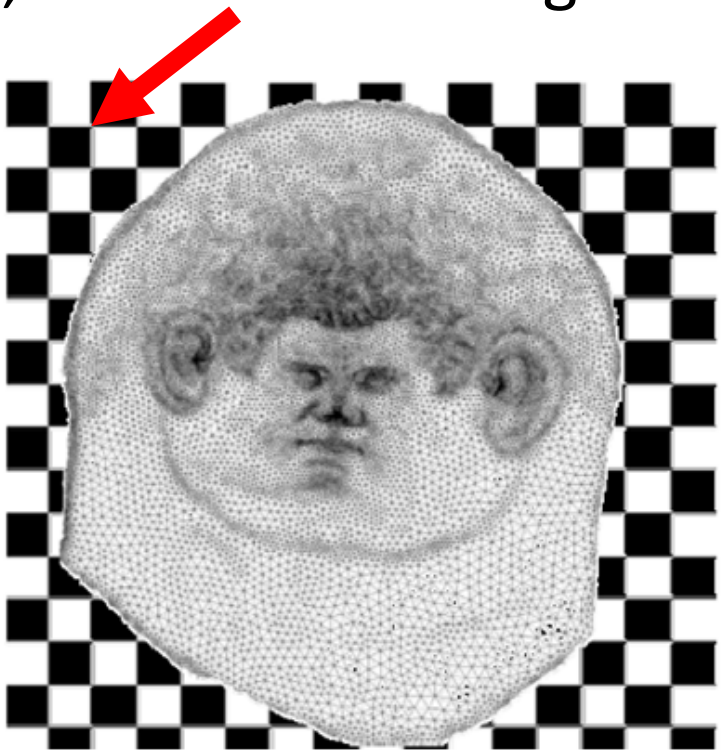
Flattened mesh
every vertex now is associated with a color from the 2D checkerboard image

Mr Checkerboard

Angle Distortions

In texture space, all corners had right angles...

Polygon Mesh



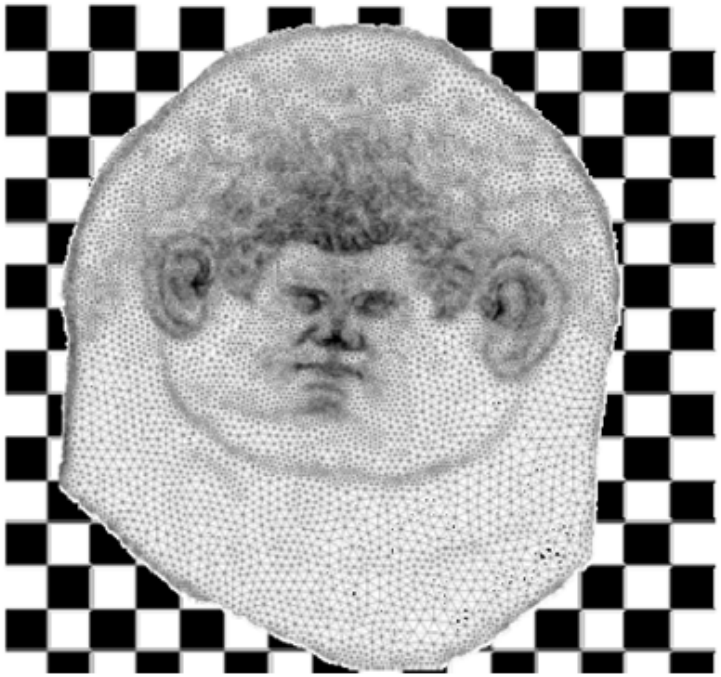
Flattened mesh
every vertex now is associated with a color from the 2D checkerboard image

Mr Checkerboard

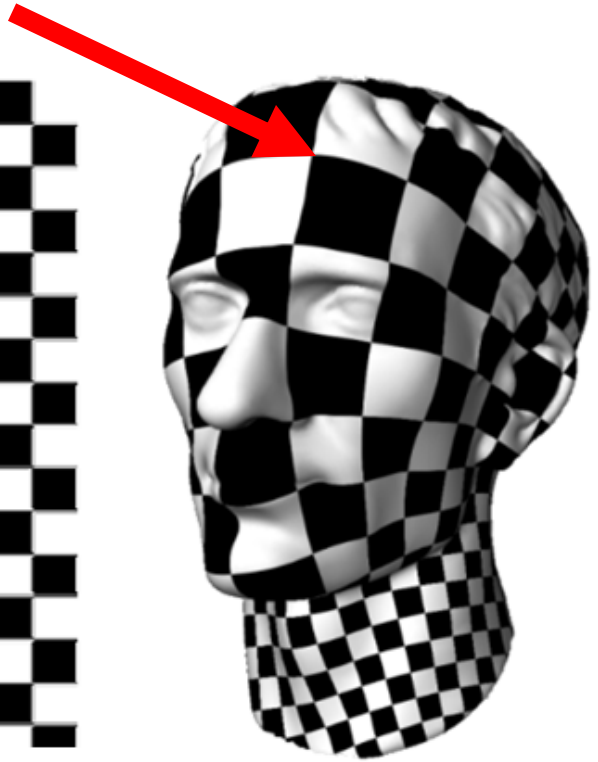
Angle Distortions

Is the angle here 90 degrees?

Polygon Mesh



Flattened mesh
every vertex now is associated with a color from the 2D checkerboard image

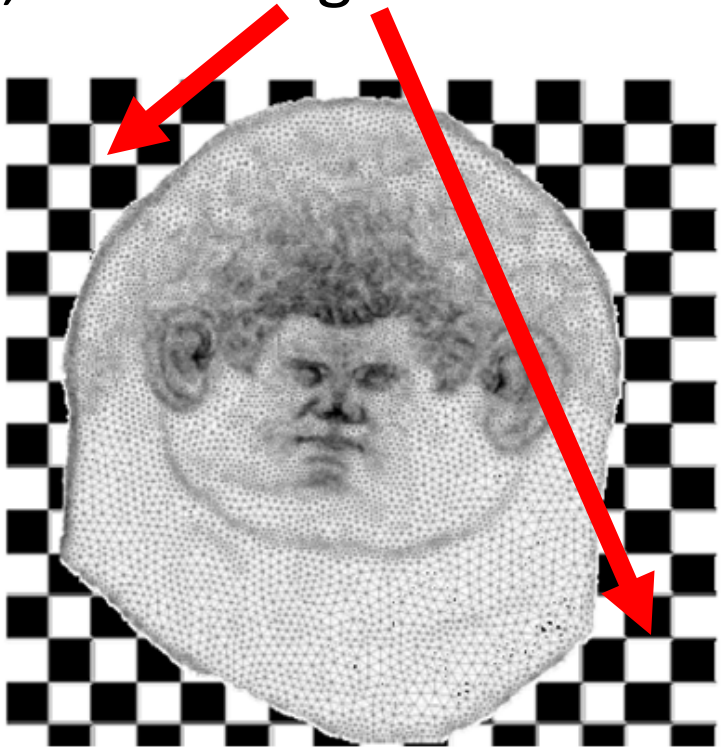


Mr Checkerboard

Area Distortions

In texture space, all rectangles were of the same size.

Polygon Mesh



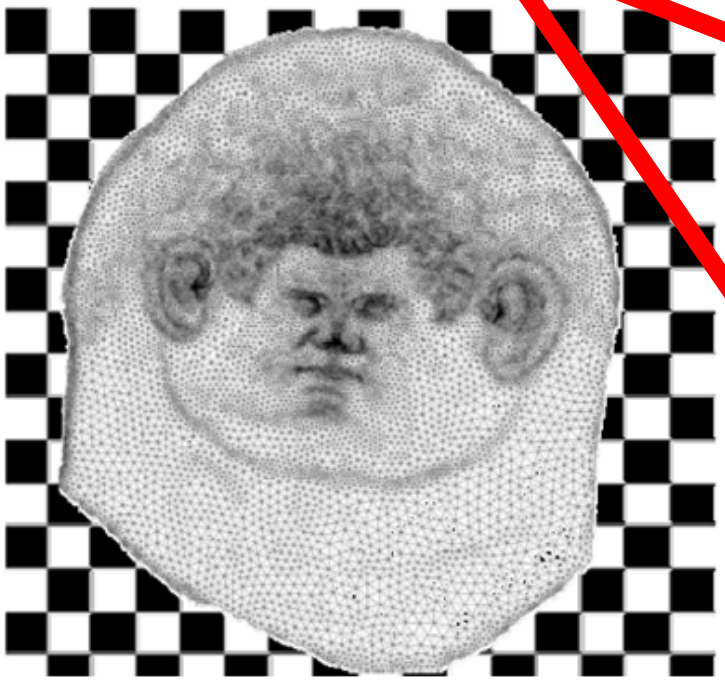
Flattened mesh
every vertex now is associated with a color from the 2D checkerboard image

Mr Checkerboard

Area Distortions

Are the areas of rectangles the same?

Polygon Mesh

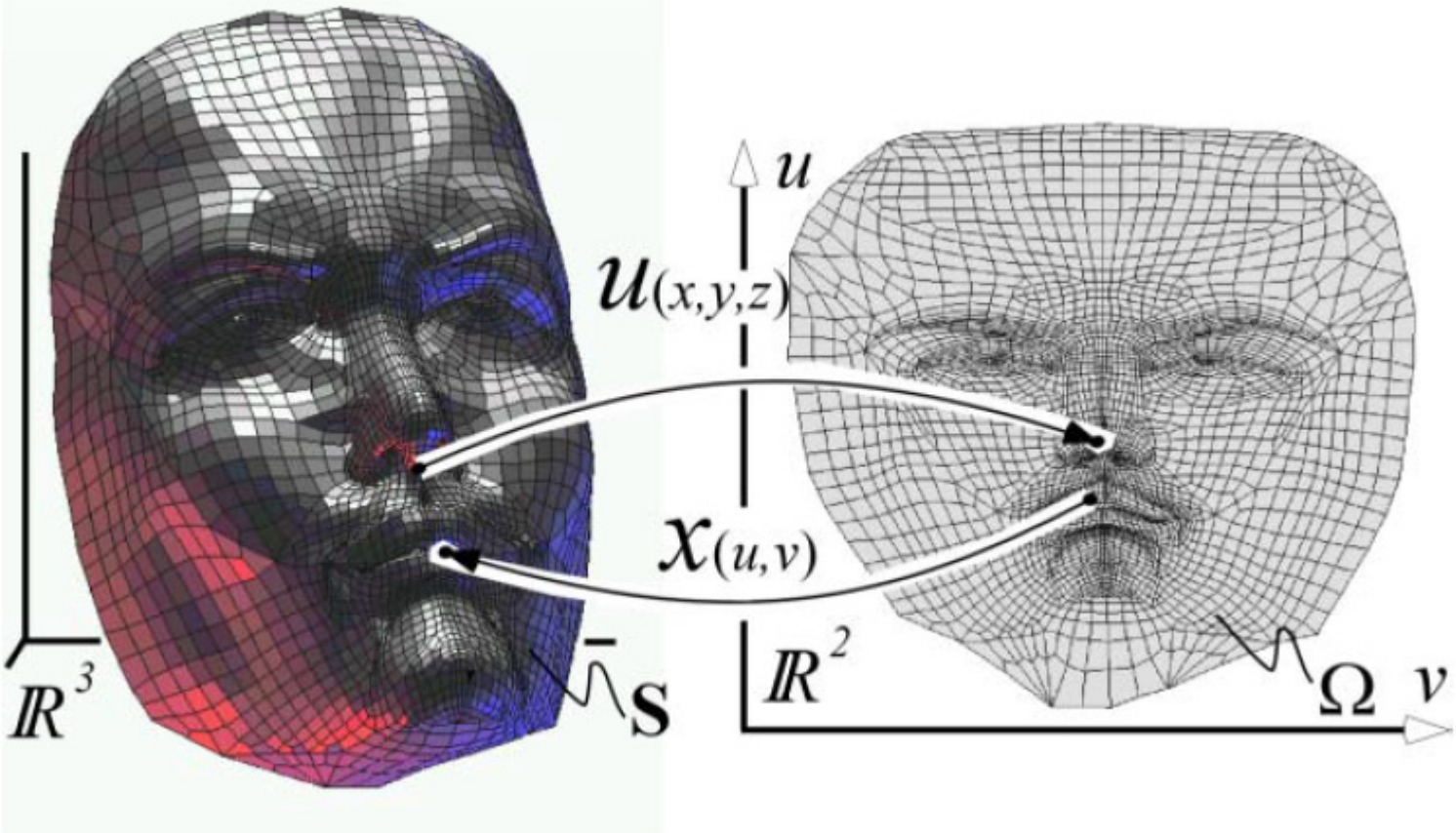


Flattened mesh
every vertex now is associated with a color from the 2D checkerboard image

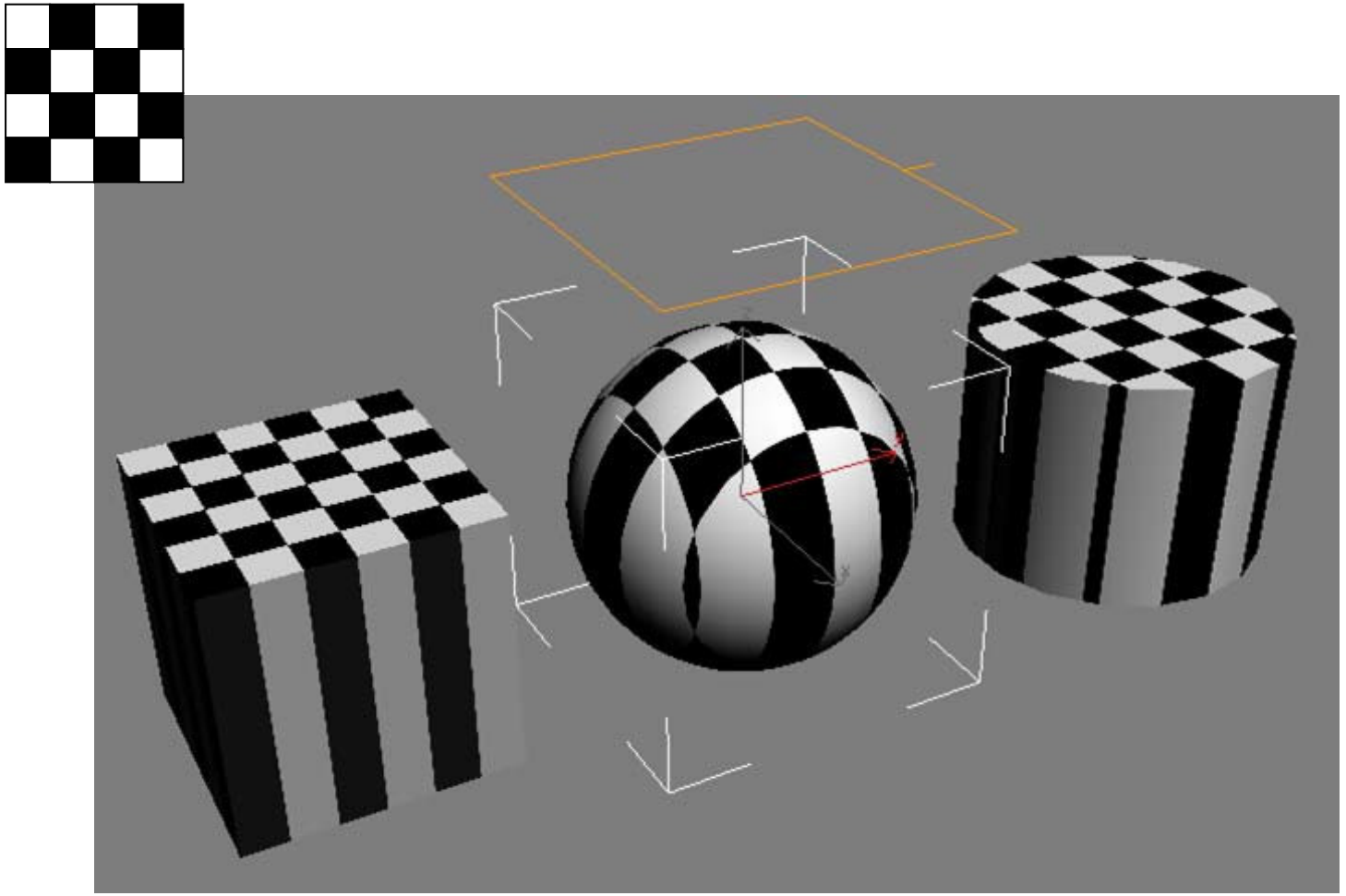
Mr Checkerboard

Surface parameterization

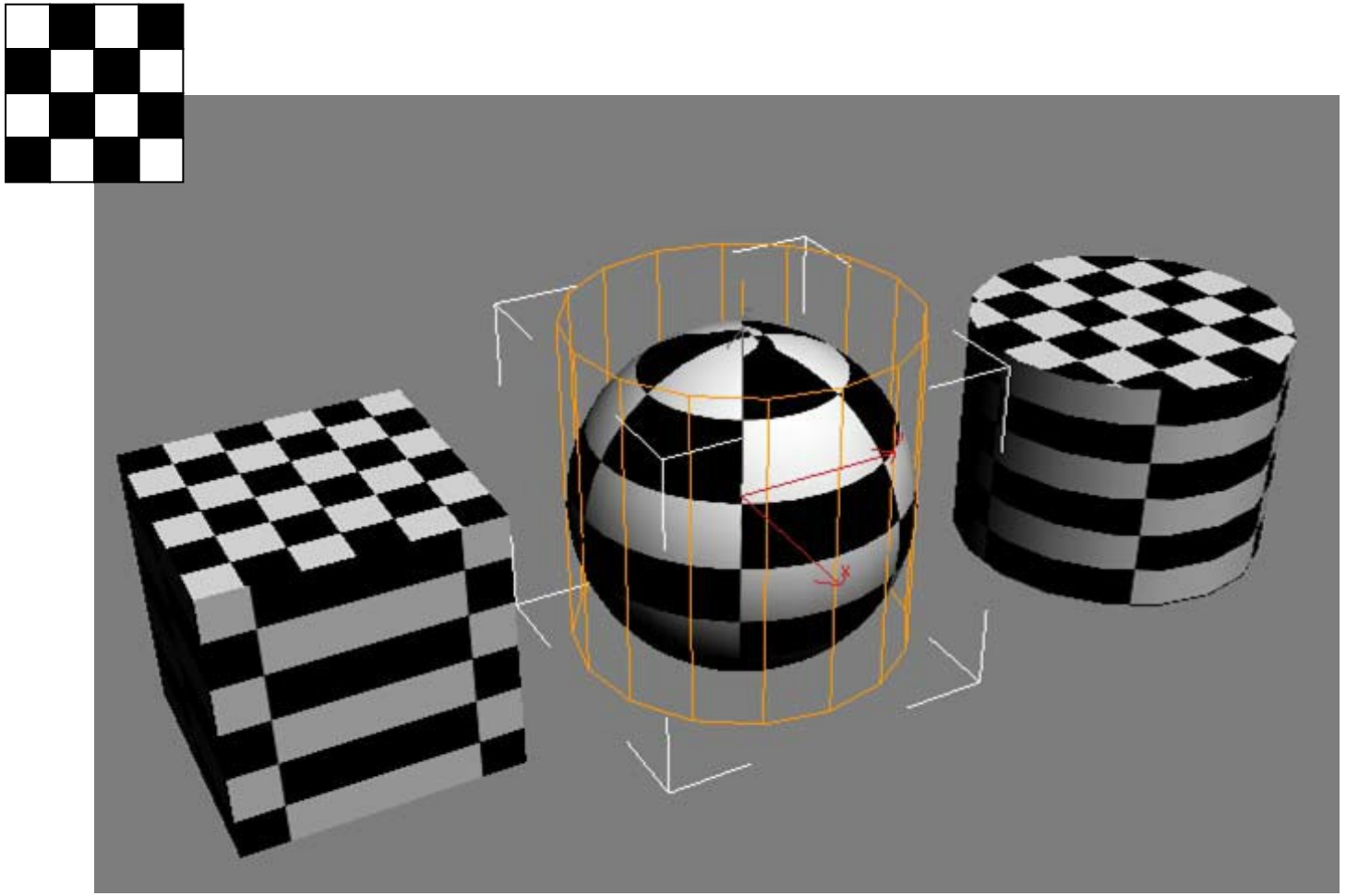
For polygon meshes, there **is no predefined way to flatten the surface**, thus we have to compute it!



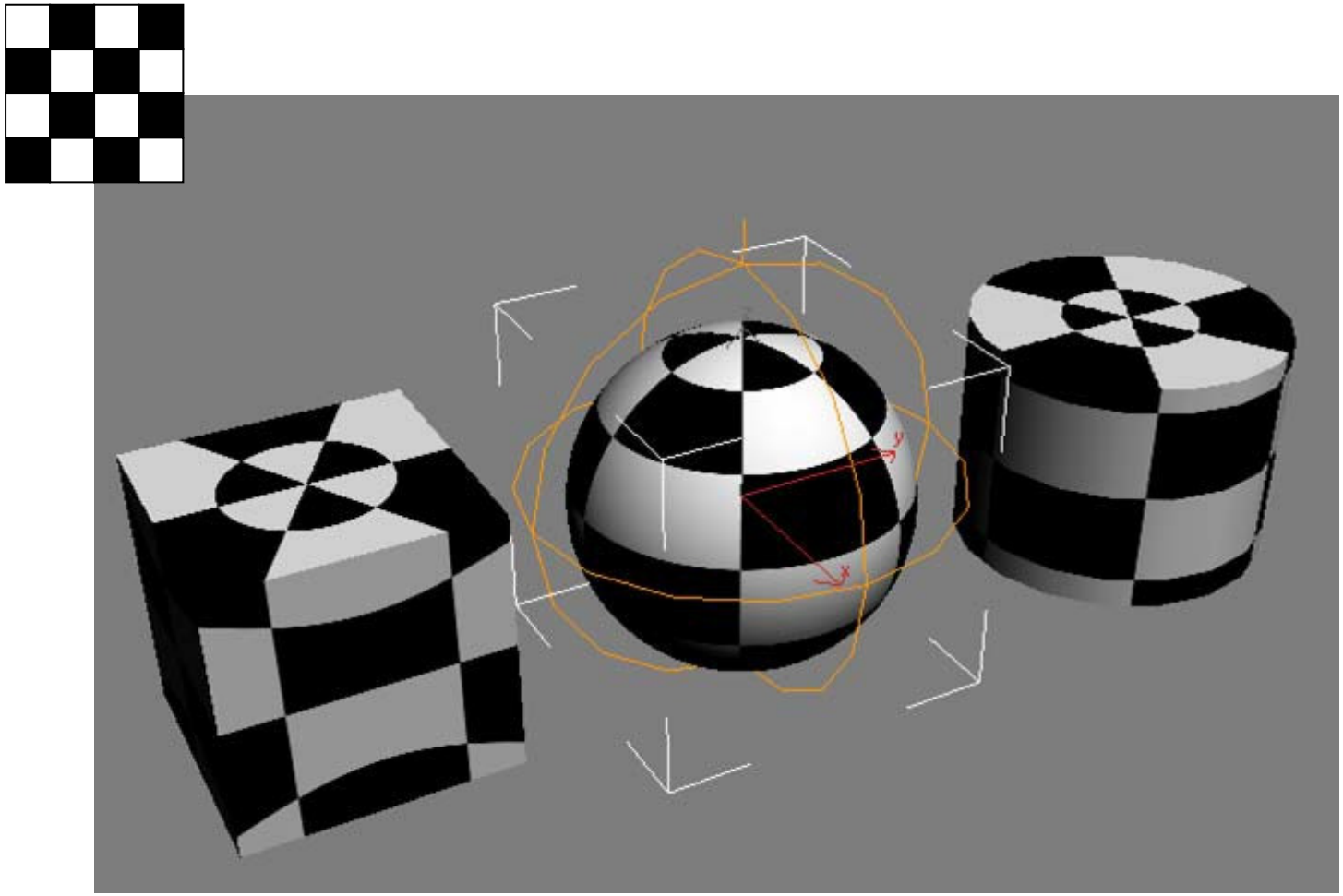
Simple approach – planar projection



Simple approach – cylindrical projection



Simple approach – spherical projection

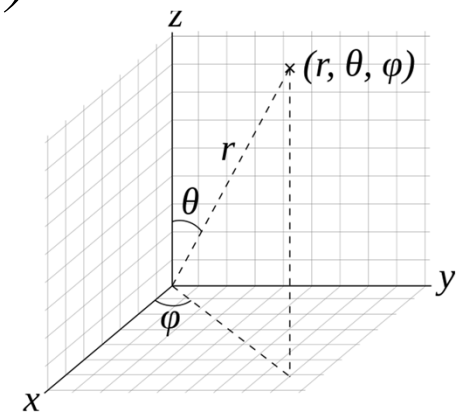


Simple approaches

For every 3D point *p*:

- **Planar projection** along xy plane of size (w,h)

$$[u_{p,}\ v_p] \ = \ (p_x / w, p_y / h)$$

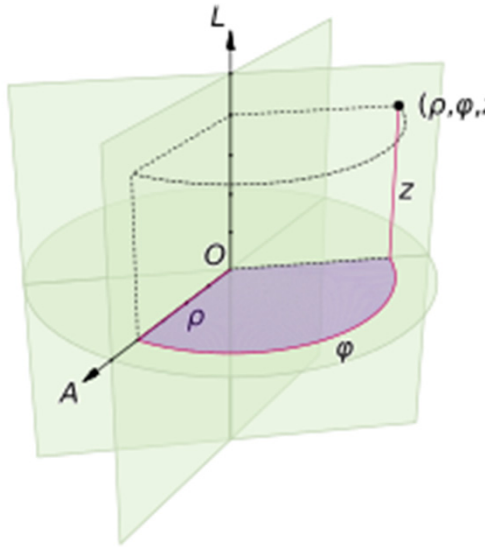


- **Spherical projection** on unit sphere
  (transform cartesian to spherical coordinates)

$$[u_{p,}\ v_p] \ = \ (\varphi, \theta)$$

- **Cylindrical projection** on unit cylinder with height h & its axis is aligned with the z-axis
  (transform cartesian to cylindrical coordinates)

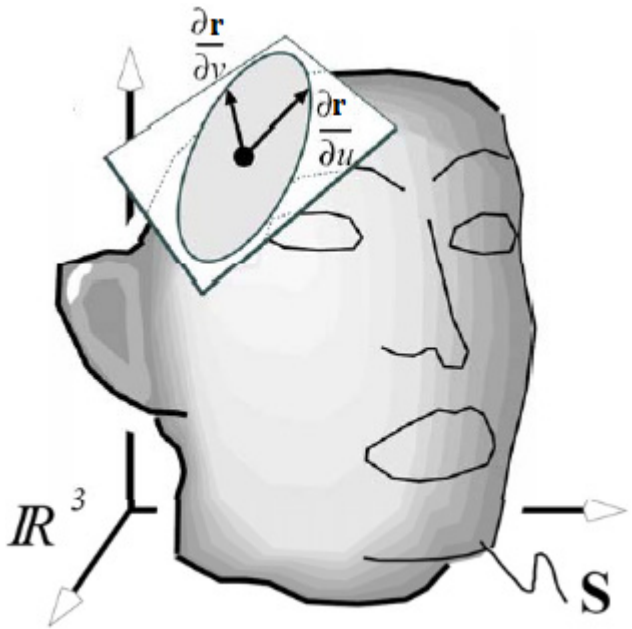
$$[u_p,\ v_p] = (\varphi, p_z \ / h)$$

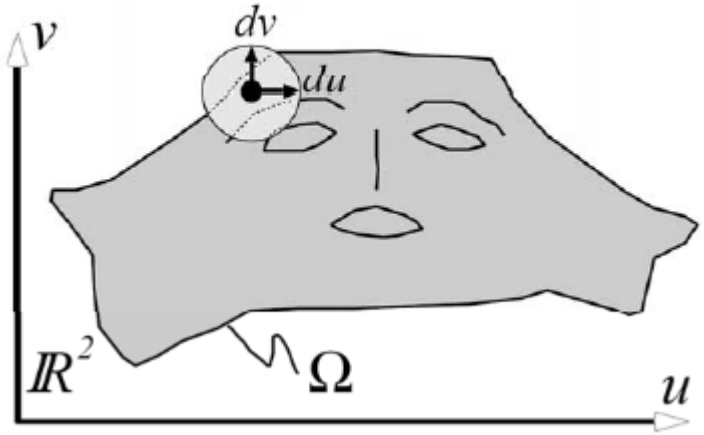


More complex approaches

Optimization problem… solve for u-v coordinates of every vertex to minimize distortion

Energy(u_1, v_1,…, u_p, v_p) = some measure of distortion

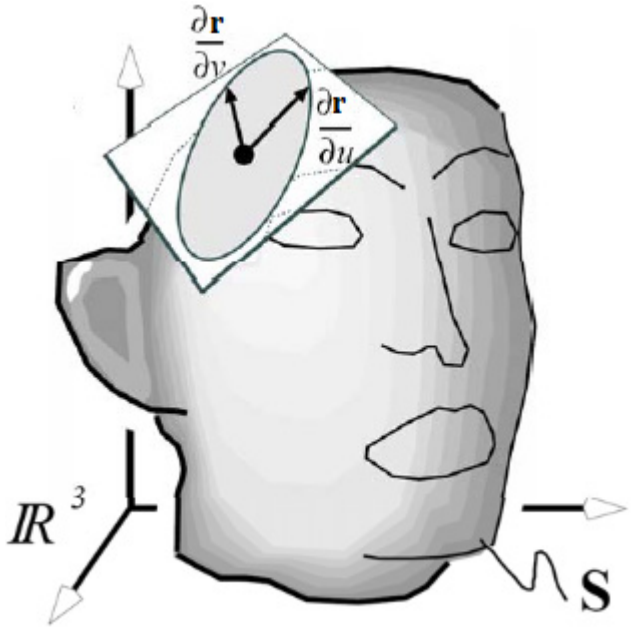


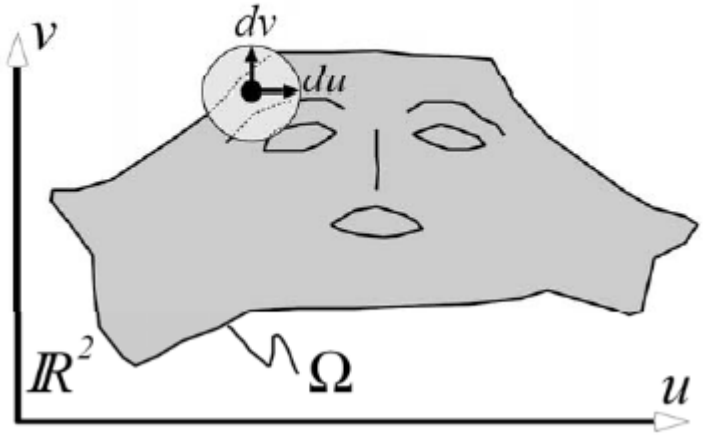


How to characterize distortion?

Every different parameterization defines a different parametric function for the surface

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

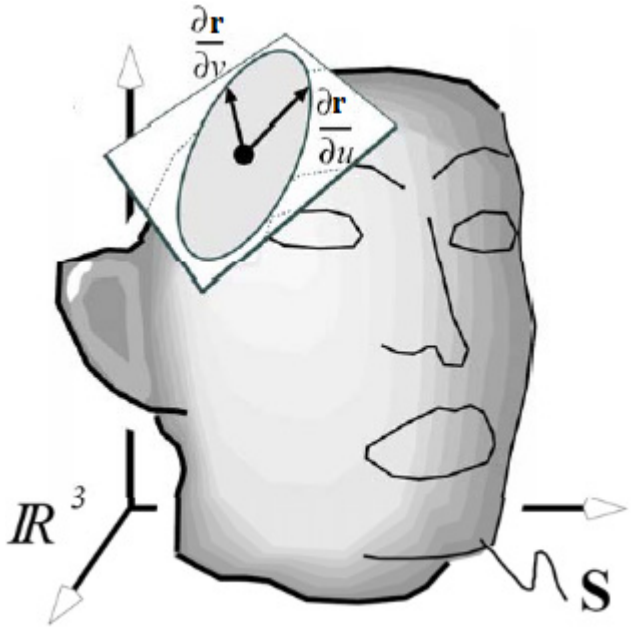


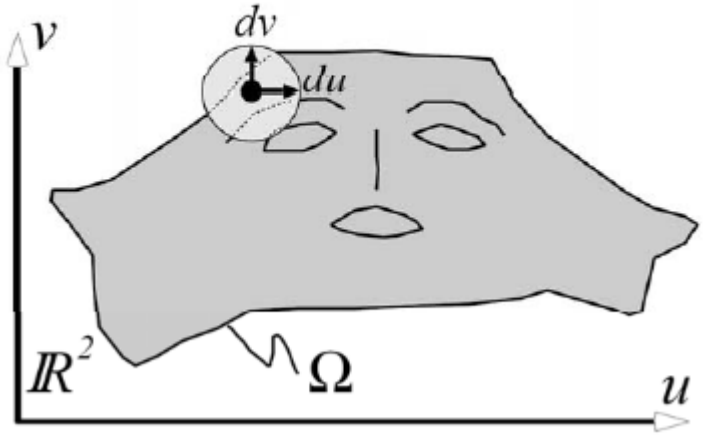


How to characterize distortion?

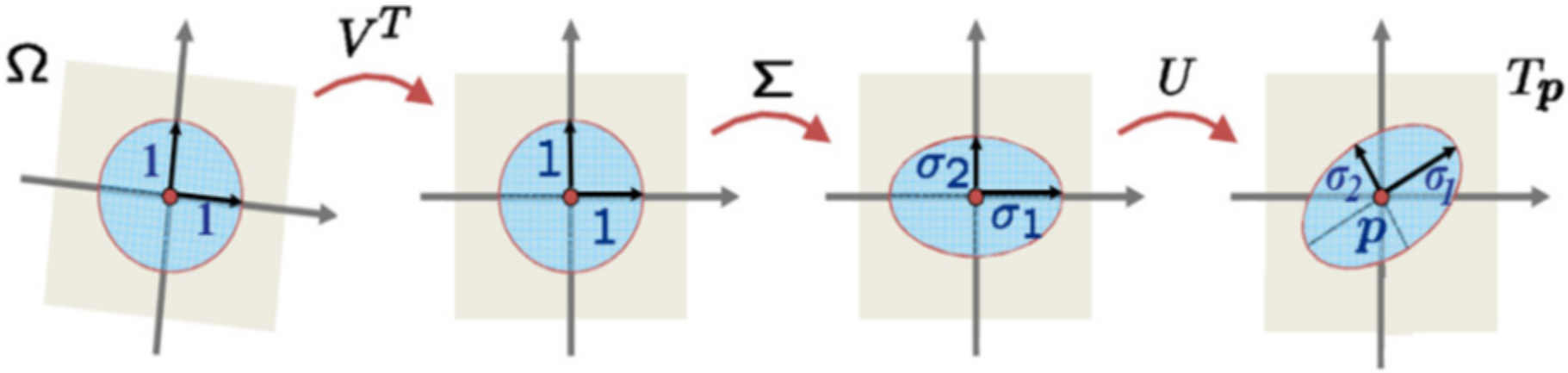
Every different parametric function has different set of first partial derivatives

$$\textbf{Jacobian } \mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} \end{bmatrix}$$





How to characterize distortion?



Singular Value Decomposition (SVD) of J

$$J = U \Sigma V^T = U \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{pmatrix} V^T$$

with **rotations** $U \in \mathbb{R}^{3\times3}$ and $V \in \mathbb{R}^{2\times2}$

and **scale factors** (singular values) $\sigma_1 \geq \sigma_2 > 0$

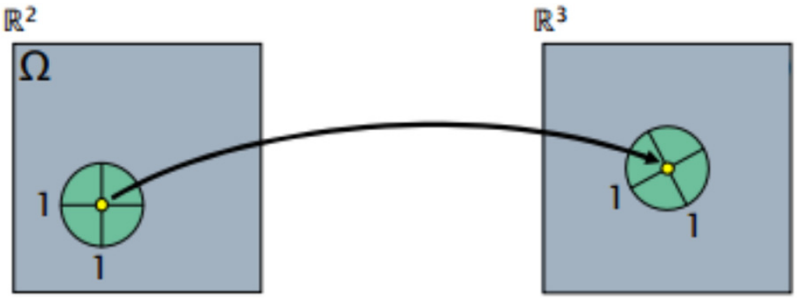
Isometric mappings

When:

$\sigma_1 = \sigma_2 = 1$

then mapping is ***isometric*** (length preserving):

the *length* of any arc on the surface S is preserved on the mapped domain Ω.



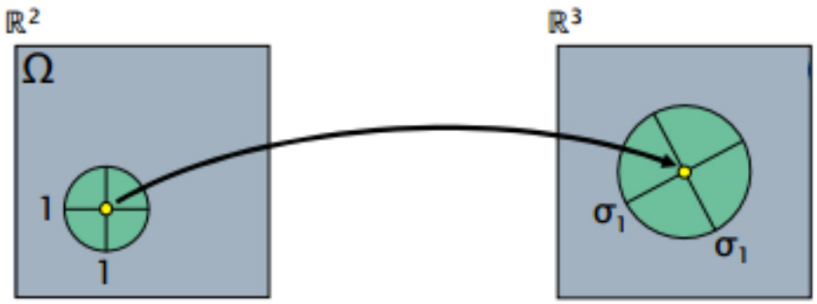
Conformal mappings

When:

$\sigma_1 = \sigma_2$ *(some constant)*

then mapping is ***conformal*** (angle preserving):

the *angle* of intersection of every pair of intersecting arcs on the surface S is preserved on the mapped domain $\boldsymbol{\Omega}$.



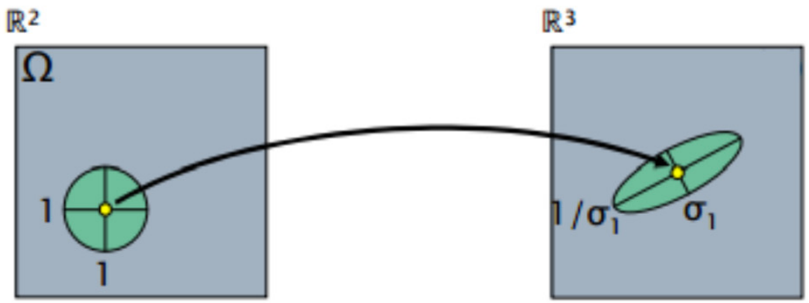
Equiareal mappings

When:

$\sigma_1 \cdot \sigma_2 = 1$

then mapping is ***equiareal*** (area preserving):

the *area* of an area element on the surface S is preserved on the mapped domain Ω.



isometric* ⇔ *conformal* + *equiareal

Theorem: every **isometric** mapping is ***conformal*** and ***equiareal*** and vice versa.

Most often we prefer **isometric** parametrizations.

However, a surface in general cannot be flattened isometrically (unless it is *developable* surface)

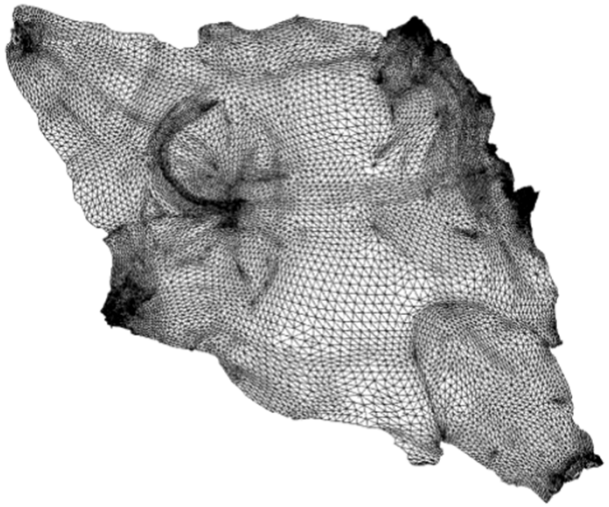
In practice, we go for conformal or equiareal, or some balance of both.

Conformal vs "near"-isometric

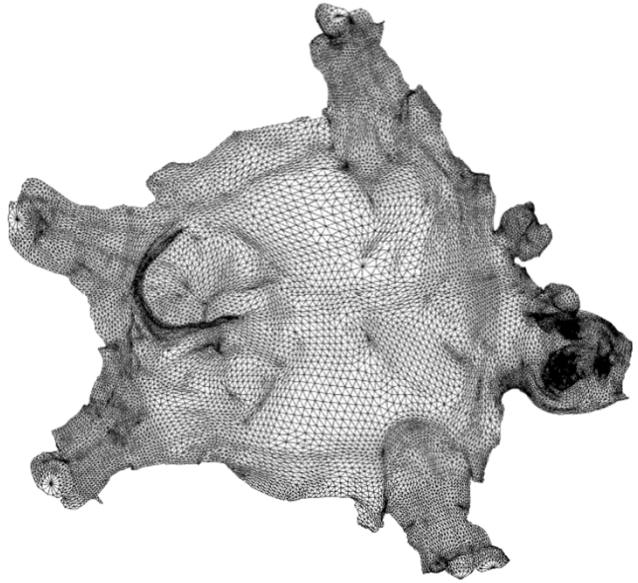
Conformal

Minimal Stretch

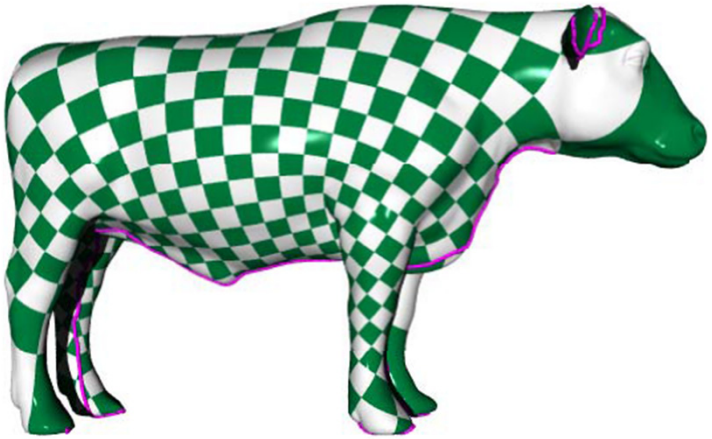
Conformal vs "near"-isometric



Conformal



Minimal Stretch



Minimize distortion for a triangle mesh

Basic idea: minimize an energy function that sums the distortions for every triangle when flattened into 2D space.

$$E(M) = \frac{\sum_{T_i \in M} w_i E(T_i)}{\sum_{T_i \in M} w_i}$$

w_i – weight for each triangle, typically area of triangle in 3D mesh M

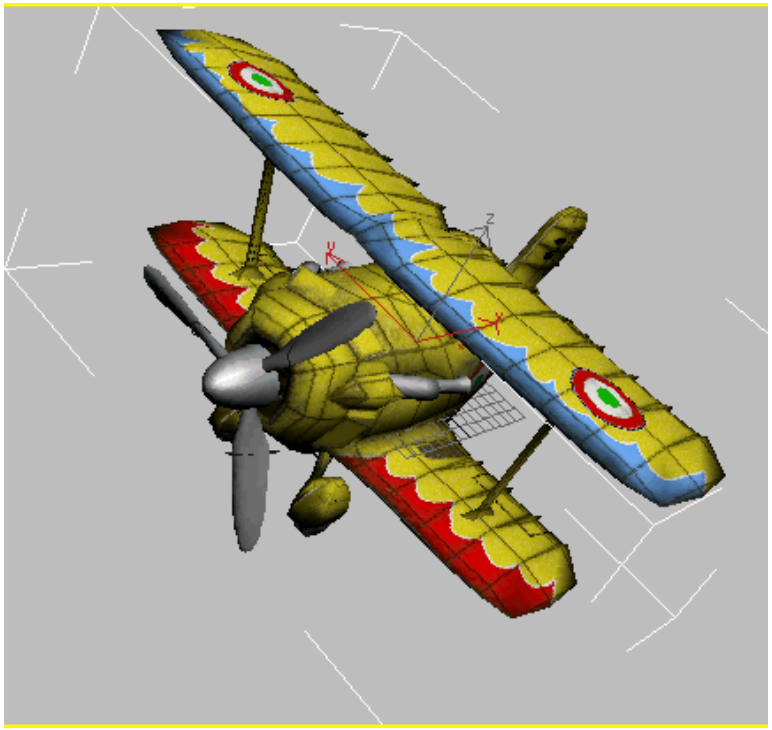
Possible Error Metrics:

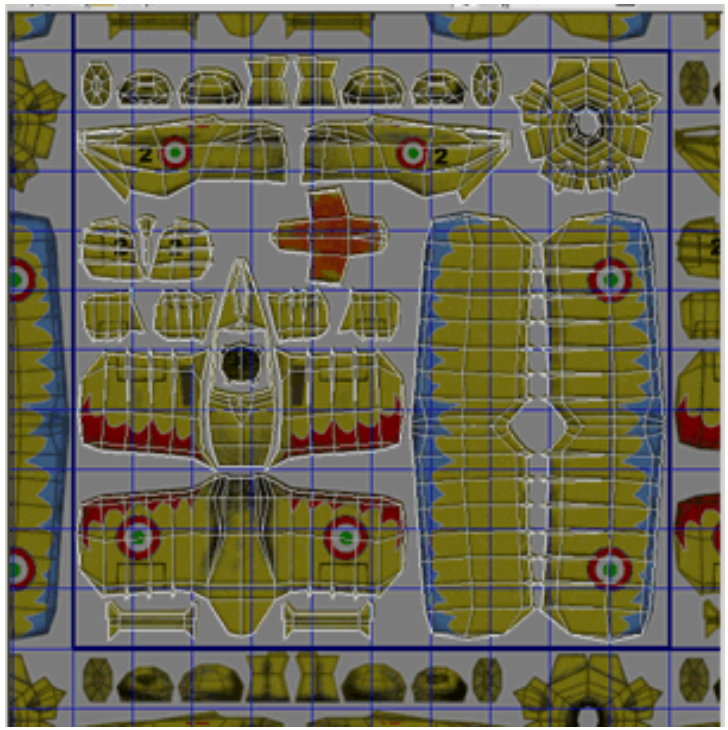
- $E_c(T_i) = (\sigma_1 - \sigma_2)^2$ conformal mapping energy
- $E_s(T_i) = (\sigma_1 - 1)^2 + (\sigma_2 - 1)^2$ stretch energy (more isometric)

Texture atlas

In general, you do not need to compute the parametrization for the whole surface.

Cut it into pieces, and flatten each piece separately!

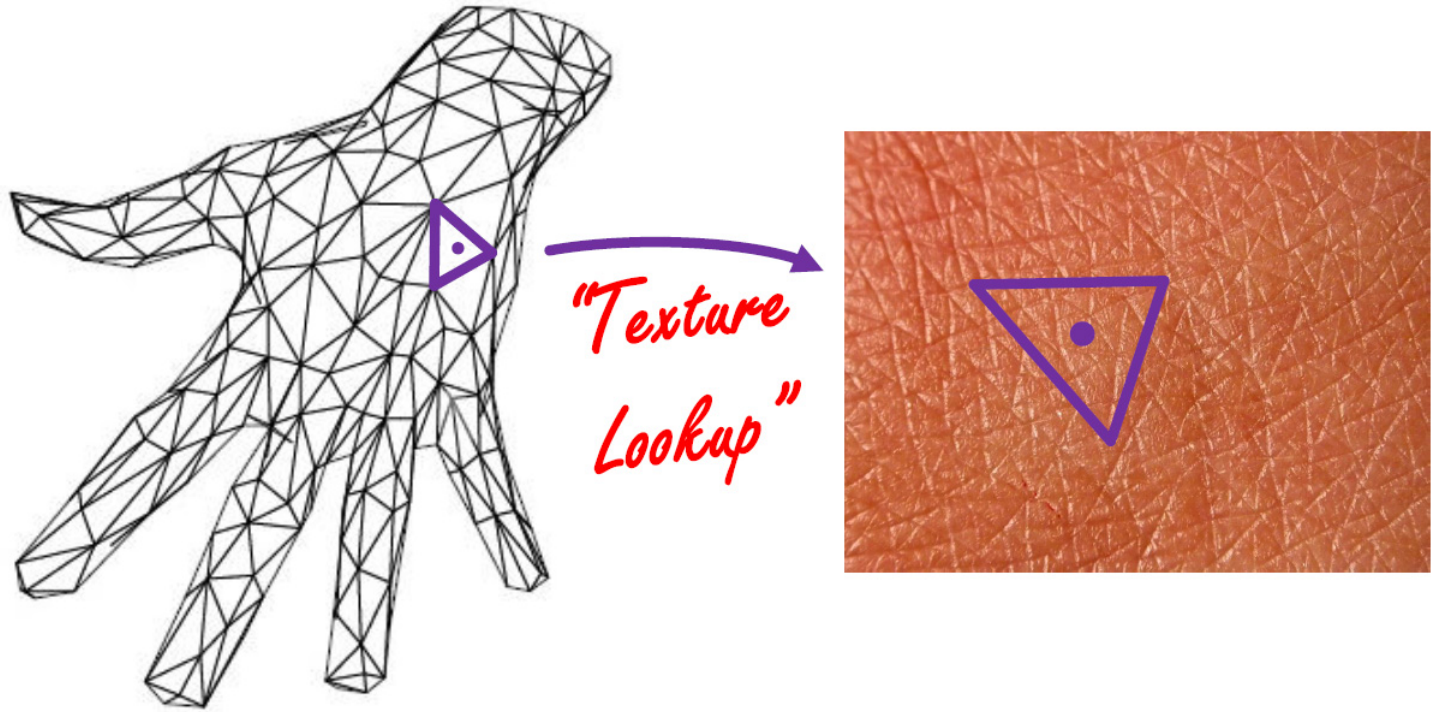




Texture atlas

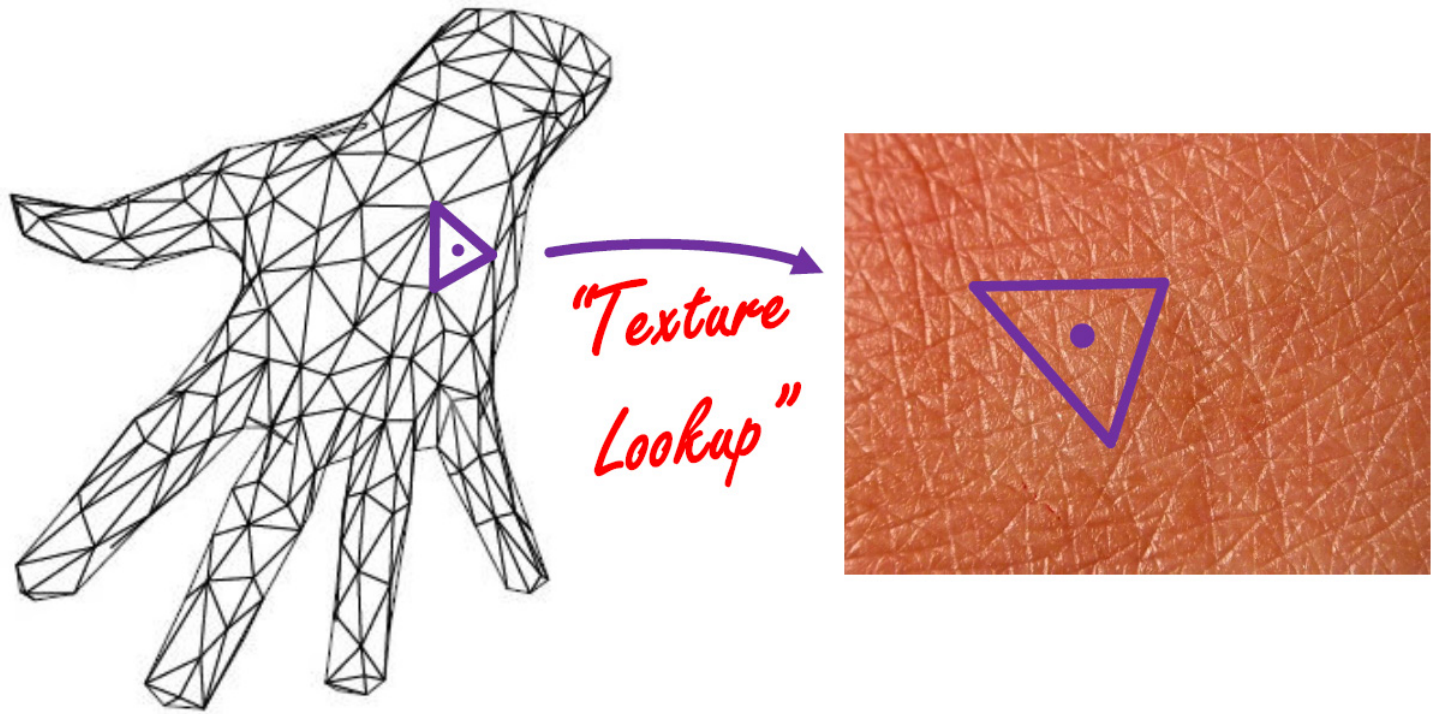
Now that we flattened the mesh or its pieces...

Every vertex is mapped somewhere in texture space i.e., **has texture coordinates** (also known as *uv-coordinates*).



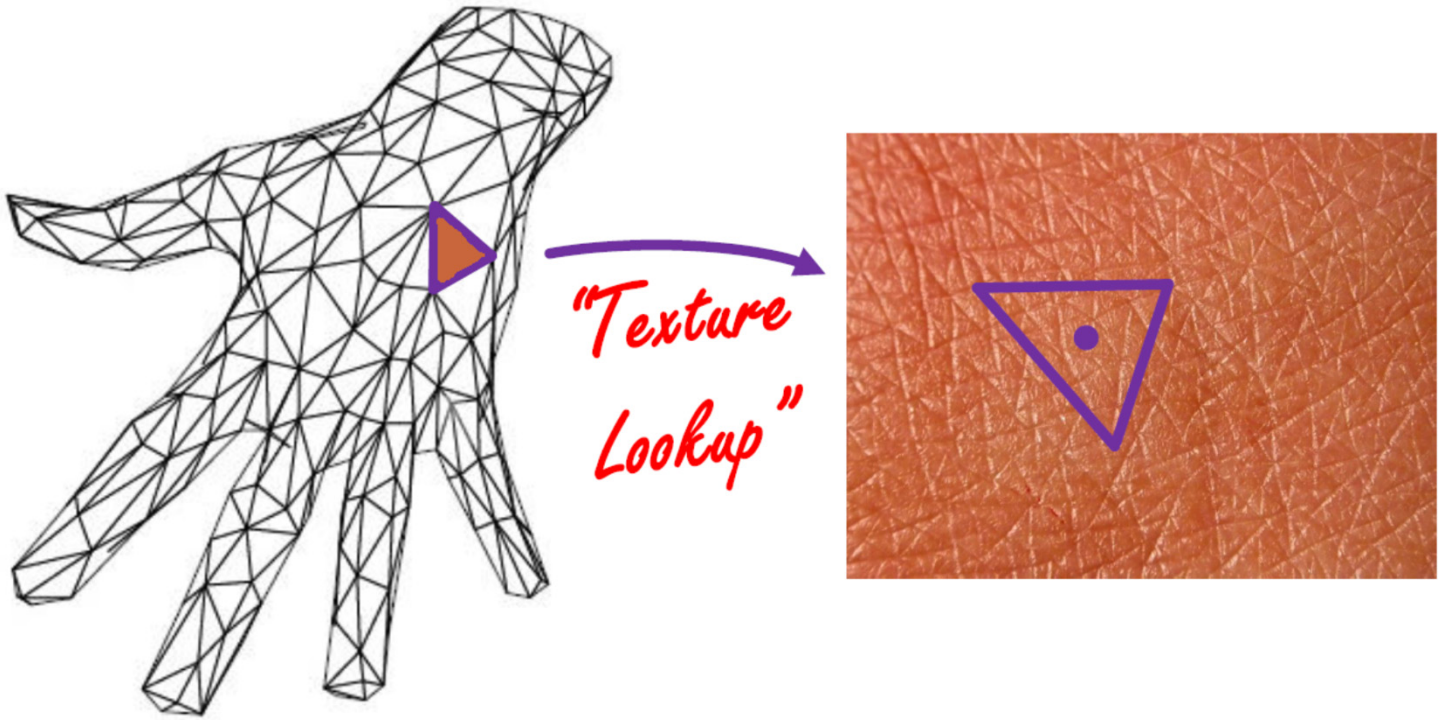
Now that we flattened the mesh or its pieces...

Do we now set a color per vertex based on the texture?



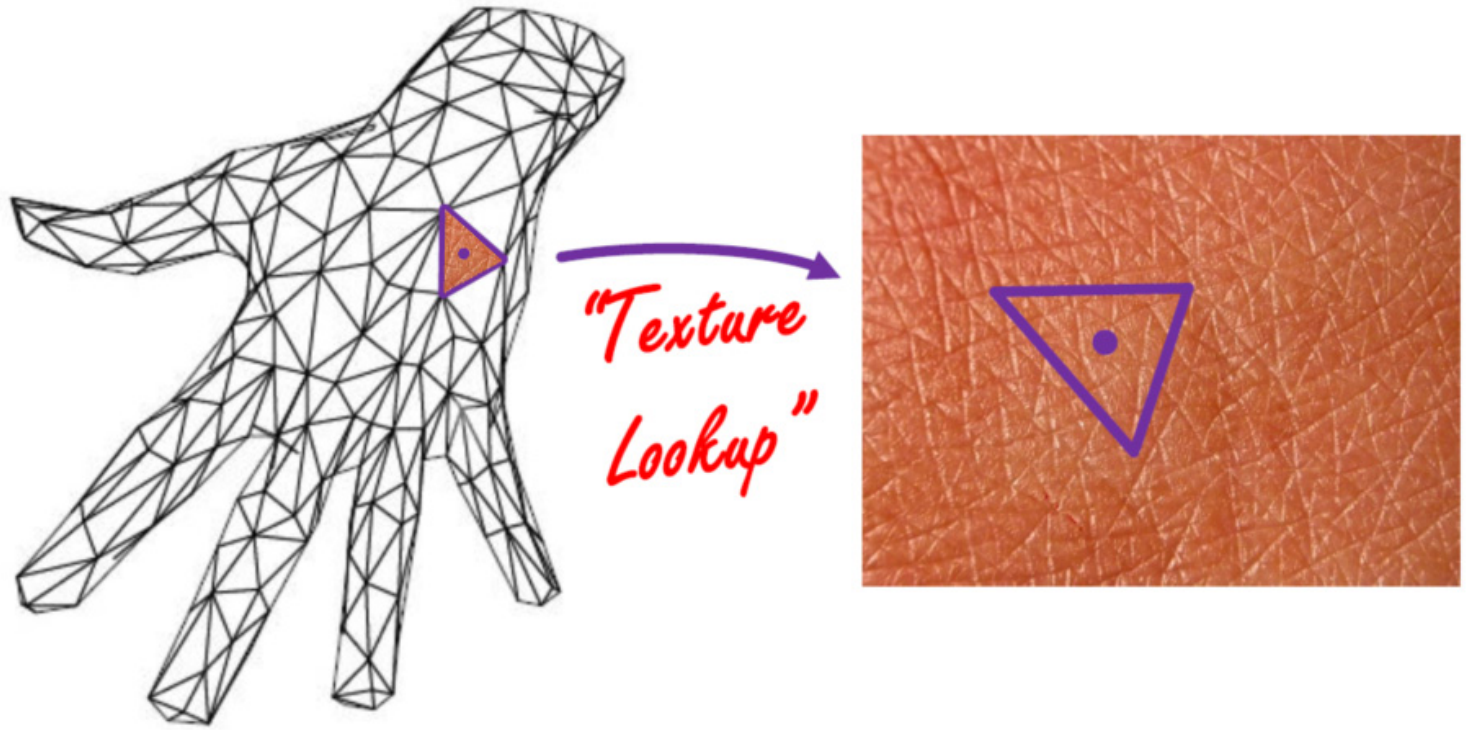
Now that we flattened the mesh or its pieces...

No, we lose all the texture detail we wanted!



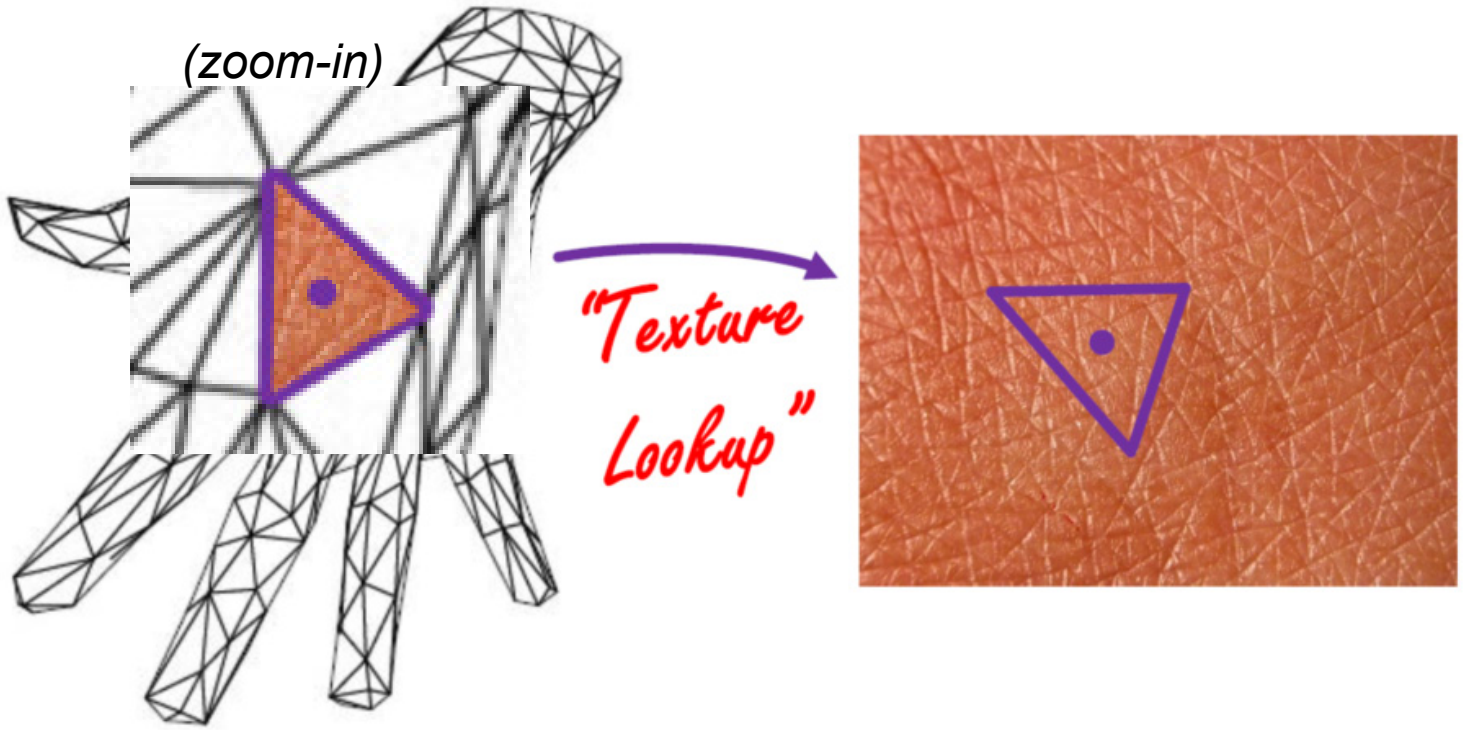
Now that we flattened the mesh or its pieces...

We **rasterize the triangle**, and **every pixel inside the triangle gets the color from the texture** [by interpolating the texture coordinates of the vertices]



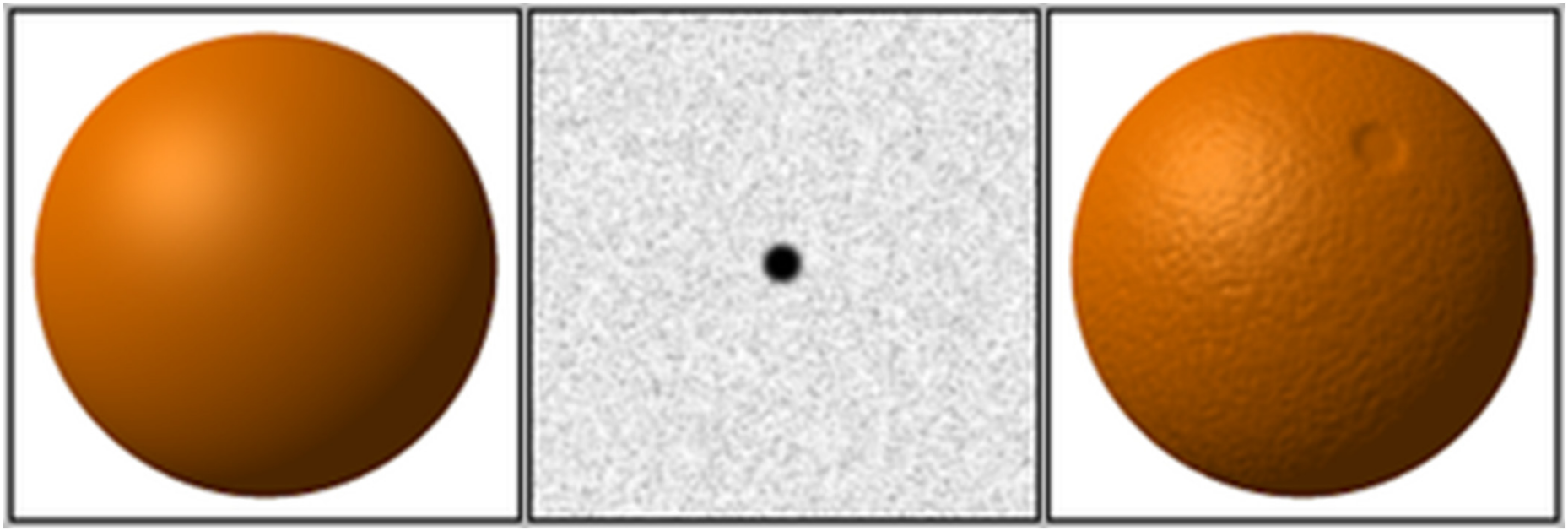
Now that we flattened the mesh or its pieces...

We **rasterize the triangle**, and **every pixel inside the triangle gets the color from the texture** [by interpolating the texture coordinates of the vertices]



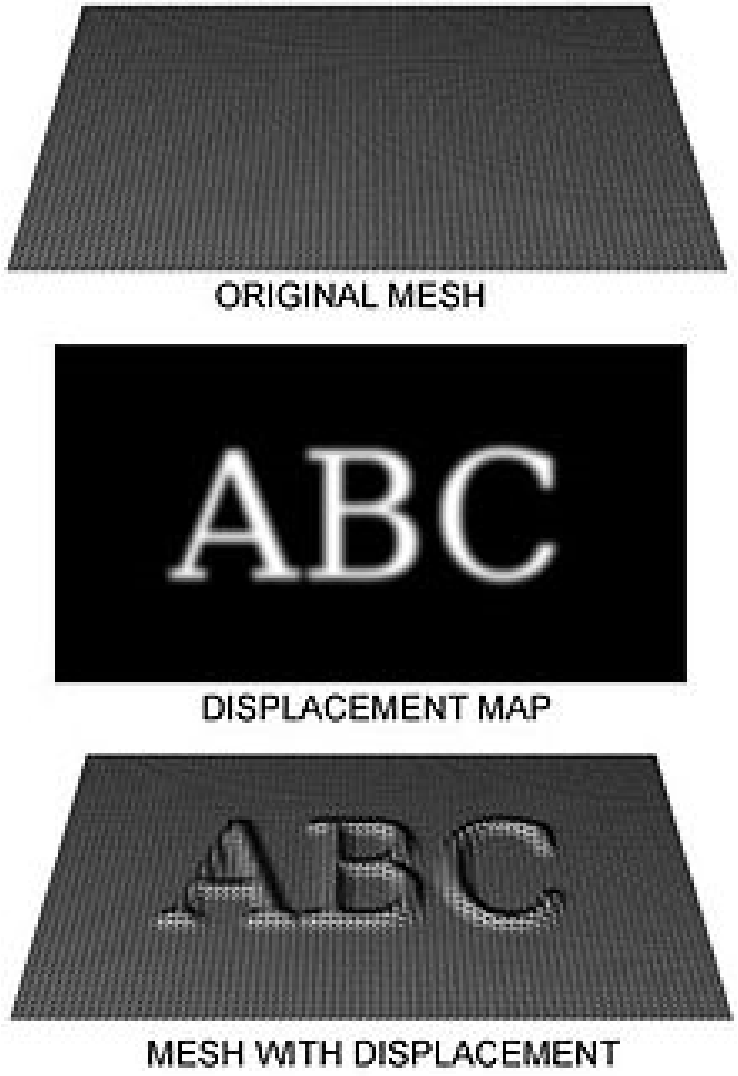
Bump mapping

Instead of modulating surface color, a texture can be defined to **simulate surface perturbations along the normal**



Displacement mapping

You can even change the geometry, by treating the texture as an offset map to the vertex position (**displacement mapping**)



Combine

texture + bump + displacement mapping

3D Textures

Instead of using a 2D image, we can instead use a 3D volumetric image: f(x, y, z) = [red green blue]

No need to flatten the mesh. **Specify 3D texture coordinates per vertex.** Lots of memory might be required.