

# Welcome to INF 426!

# Generative Artificial Intelligence



Instructor:  
Evangelos Kalogerakis

# Welcome to INF 426!

# Generative Artificial Intelligence

“a halfway submerged car”



ShapeWords, arxiv 2025  
<https://lodurality.github.io/shapewords/>

Instructor:  
Evangelos Kalogerakis

# Generate language

**Prompt:** What should I do to obtain 10/10 in the Generative AI course offered at the Technical University of Crete?

To achieve a perfect score in the Generative AI course at the Technical University of Crete, consider the following strategies:

1. **Understand the Course Structure and Requirements:** Familiarize yourself with the course syllabus, which outlines the topics covered, assessment methods, and grading criteria. This will help you identify key areas to focus on. 
2. **Engage with Course Materials:** Actively participate in lectures, complete all assigned readings, and engage with supplementary resources provided by the instructor. This comprehensive approach will deepen your understanding of generative AI concepts.
3. **Complete Assignments Diligently:** Ensure that all assignments are submitted on time and meet the specified requirements. Pay attention to detail and strive for accuracy in your work.
4. **Participate in Discussions and Activities:** Engage in class discussions, group projects, and any practical activities. Active participation demonstrates your commitment and enhances your learning experience.
5. **Seek Feedback and Clarification:** If you have questions or need clarification on course material, don't hesitate to ask your instructor. Constructive feedback can guide your improvement.
6. **Prepare Thoroughly for Exams:** Review all course materials, practice problem-solving, and understand the exam format. Effective preparation is key to performing well in assessments.
7. **Adhere to Academic Integrity:** Maintain honesty in all your academic endeavors. Avoid plagiarism and ensure that all work submitted is your own.  Academic integrity is crucial for achieving a perfect score.

# Generate images

**Prompt:**

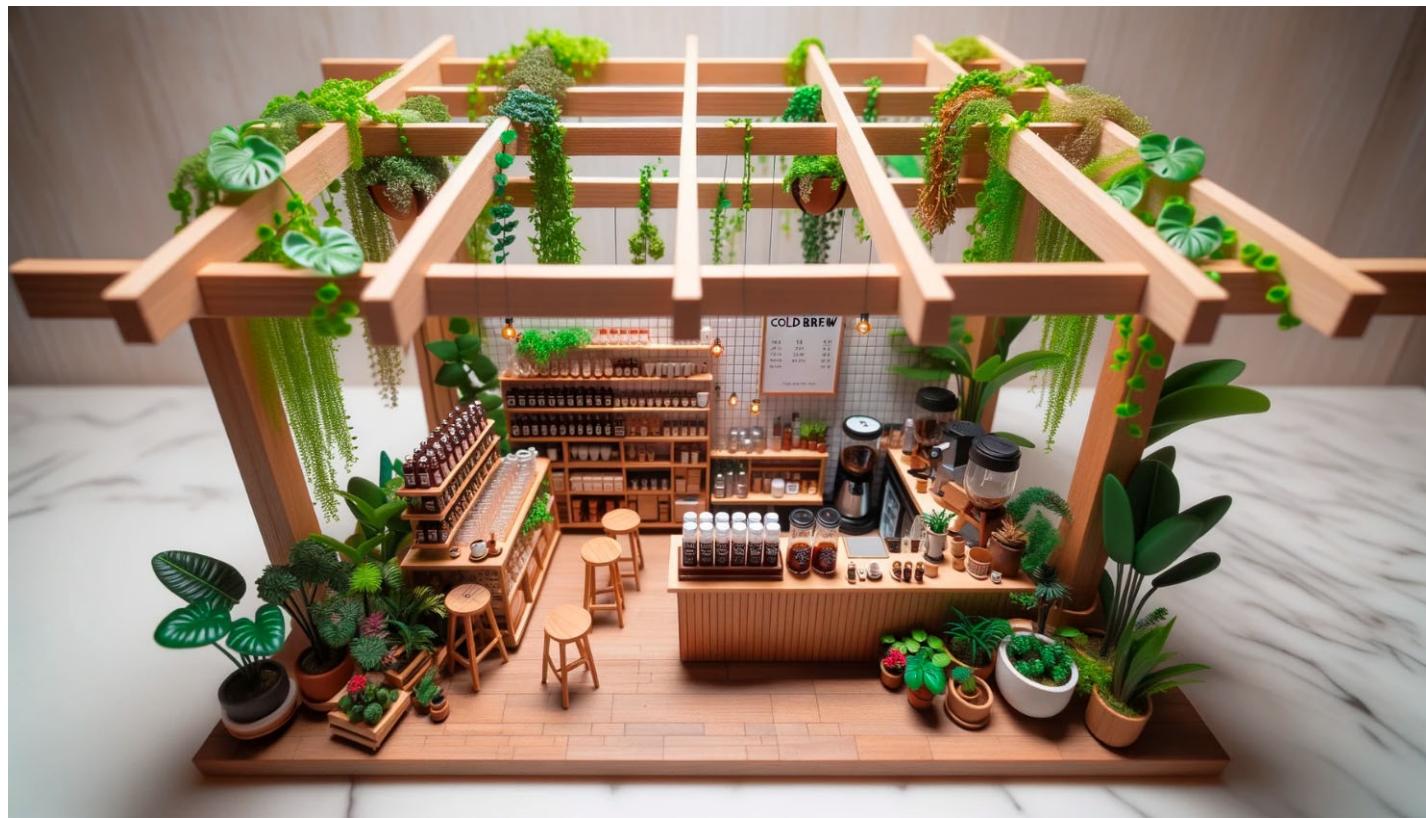
“An astronaut riding a horse”



# Generate more images

## Prompt:

“A minimap diorama of a cafe adorned with indoor plants. Wooden beams crisscross above, and a cold brew station stands out with tiny bottles and glasses”



# Generate videos



[https://www.youtube.com/watch?v=HK6y8DAPN\\_0](https://www.youtube.com/watch?v=HK6y8DAPN_0)  
SORA

# Generate 3D shapes

“chairs”



“lamps”



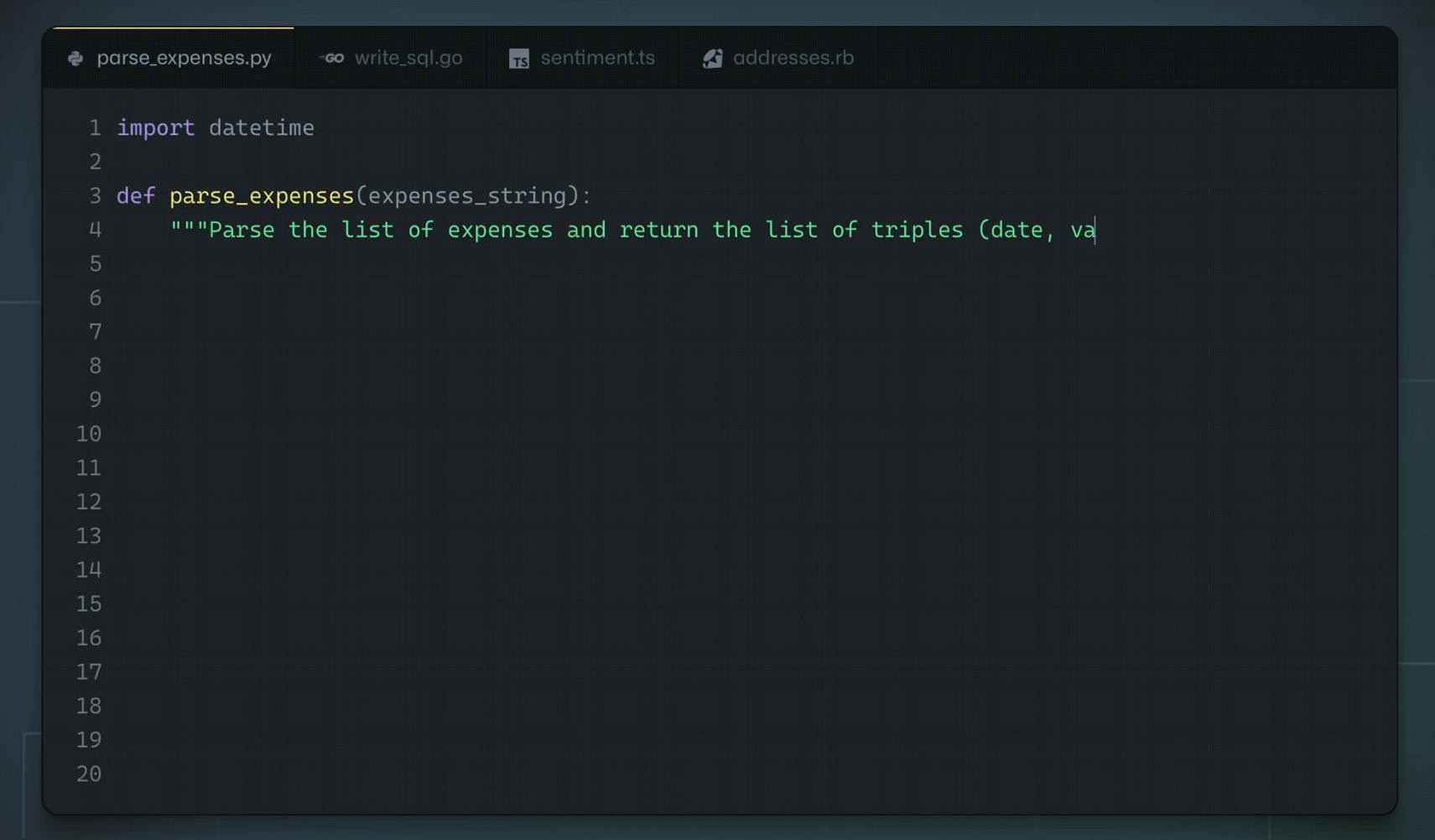
<https://lodurality.github.io/GEM3D/>

GEM3D

# Generate textured 3D models



# Generate code



A screenshot of a dark-themed code editor window. The title bar shows five tabs: 'parse\_expenses.py' (active), 'write\_sql.go', 'sentiment.ts', and 'addresses.rb'. The main area contains the following Python code:

```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, va
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

# Generate speech



# DeepFakes

Which image is real?



User  
 @StefanoErmon



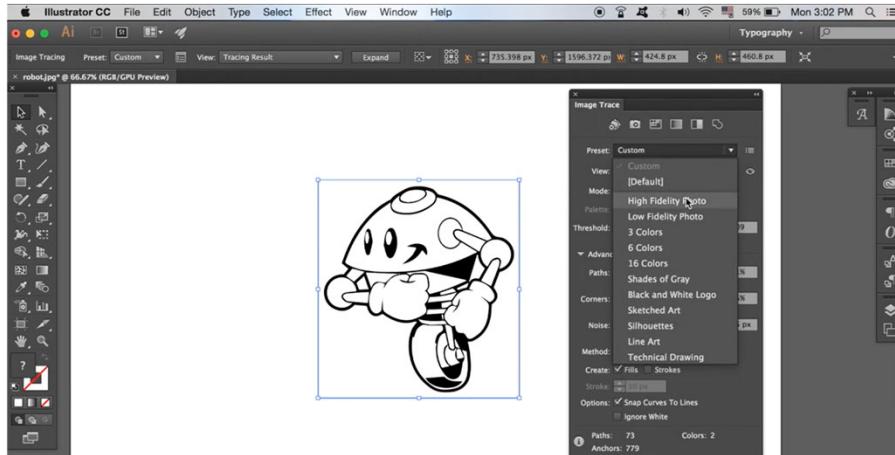
Output

# DeepFakes

Which image is real?

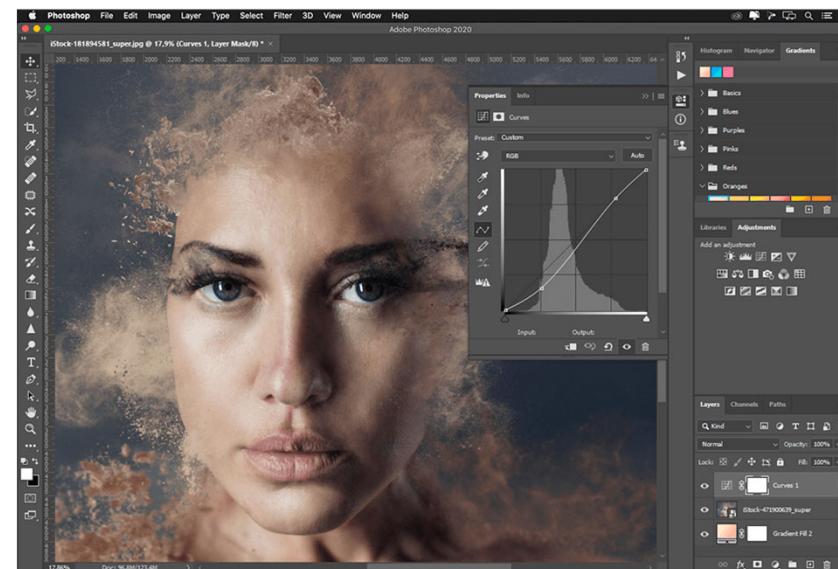


# Traditional (non-AI) Approaches for Image Modeling



Illustrator

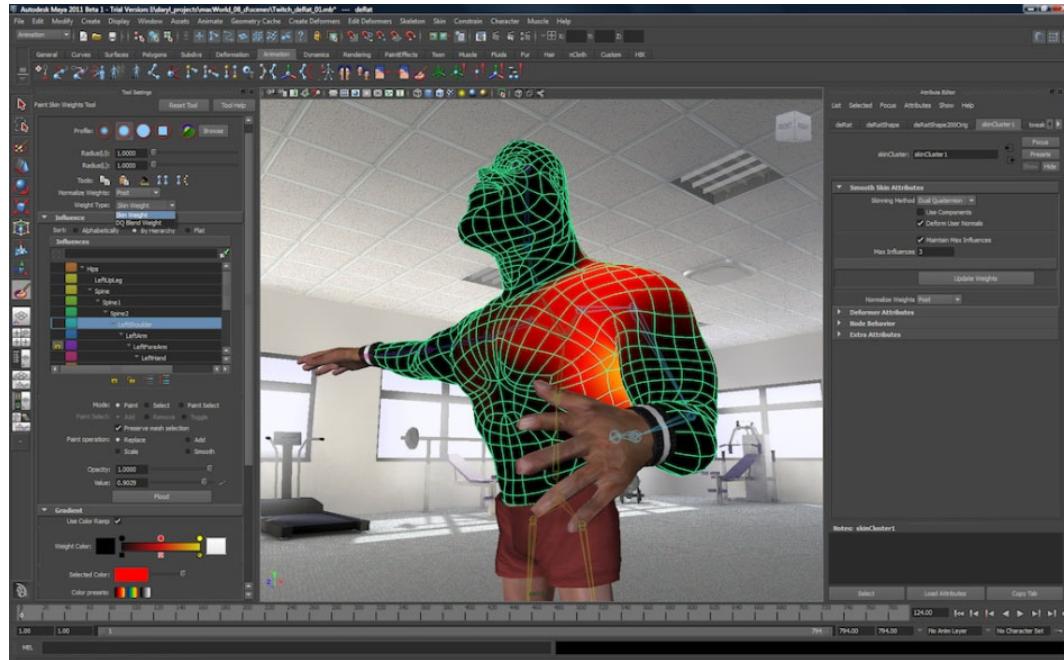
<https://www.youtube.com/watch?v=VPQ8i5BQaDE>



Photoshop

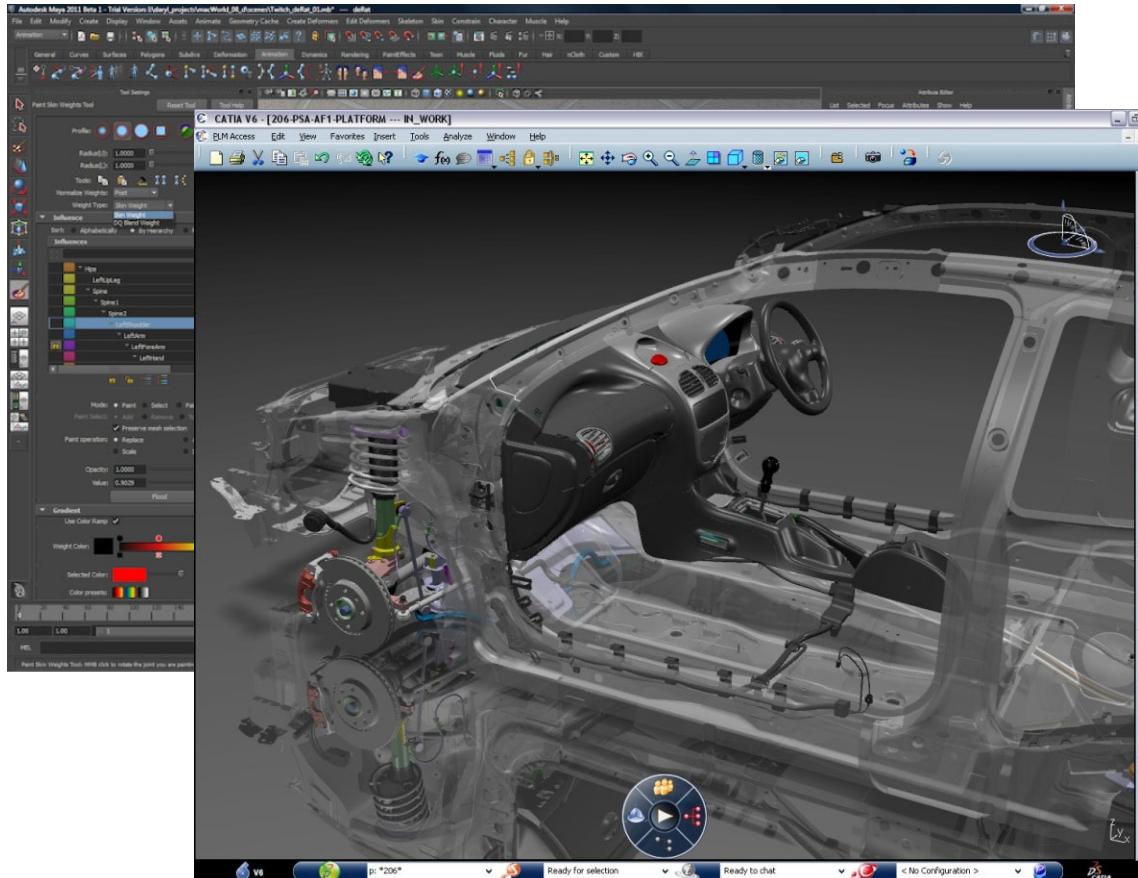
<https://cioafrica.co/adobe-considering-a-free-web-based-version-of-photoshop/>

# Traditional (non-AI) Approaches for 3D Modeling



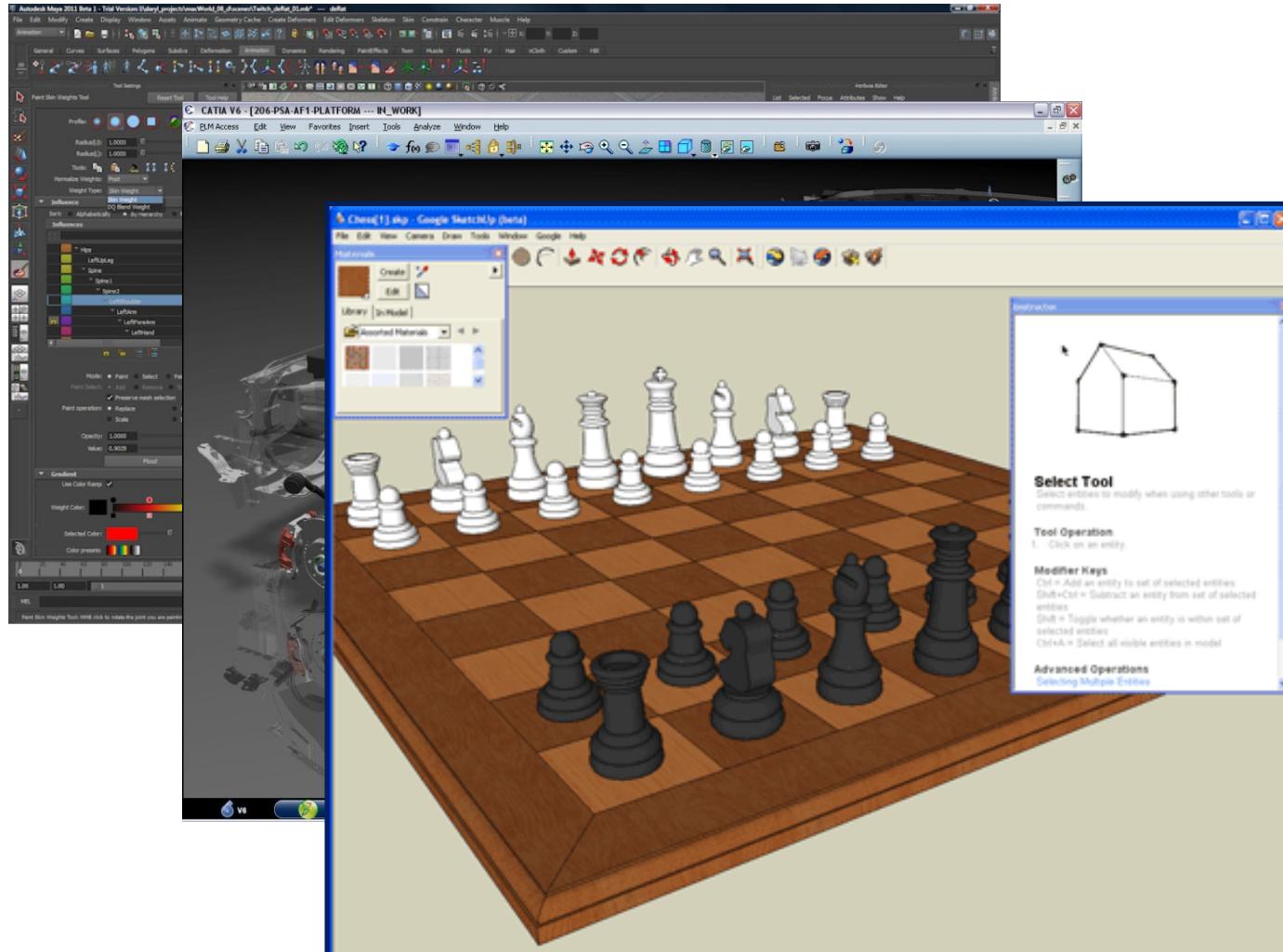
Professional 3D modeling tools [Maya]

# Traditional (non-AI) Approaches for 3D Modeling



Computer-Aided Design tools [Catia]

# Traditional (non-AI) Approaches for 3D Modeling



**General-Purpose Modeling tools [Sketch-up]**

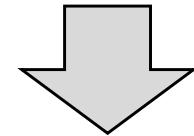
# Generation vs Inference

How to generate natural images with a computer?

High level  
description

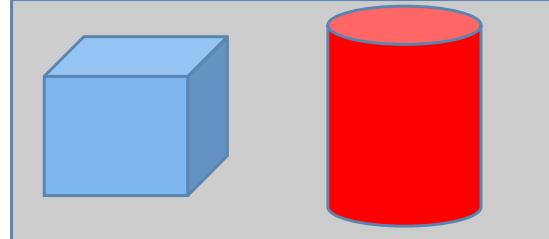
Cube(color=blue, position=(x,y,z), size=...)  
Cylinder(color=red, position=(x',y',z'), size=..)

**Generation** (graphics)



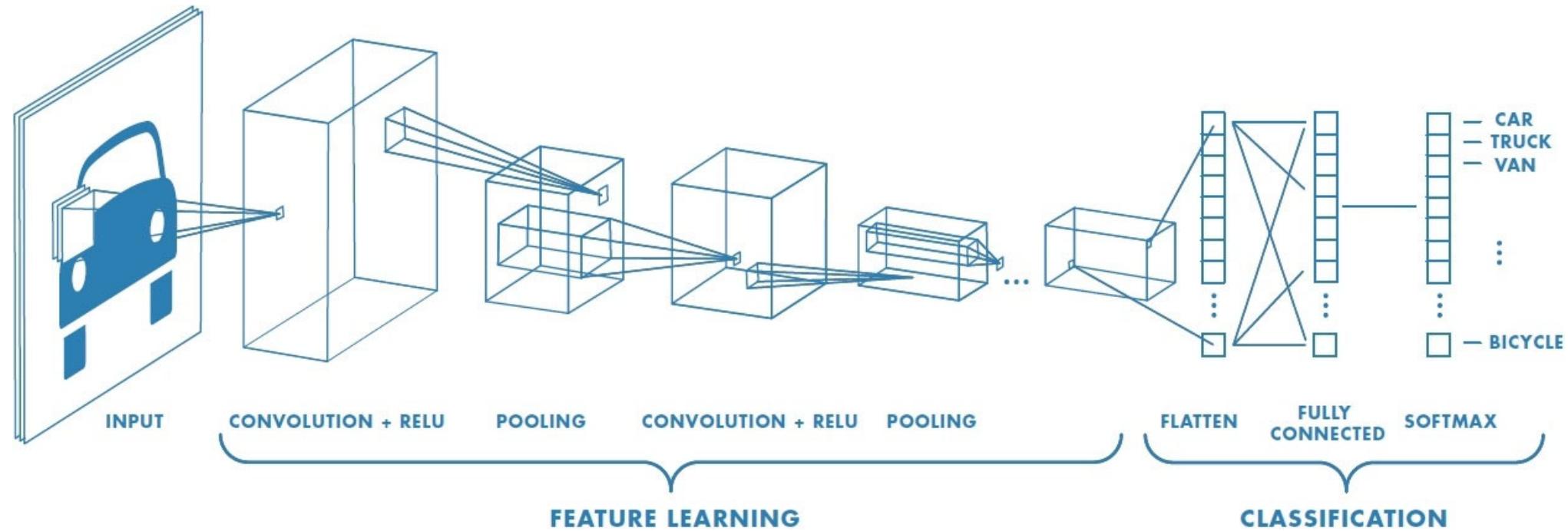
**Inference** (vision as  
inverse graphics)

Raw outputs



# Modern Computer Vision

Use **neural networks**: convolutional neural networks, transformers etc



Source: Mathworks

# Statistical Generative Models

Statistical generative models are learned from data



...



Data  
(e.g., images of bedrooms)



Prior Knowledge  
(e.g., physics, materials, ...)

# Statistical Generative Models

Statistical generative models are learned from data



...

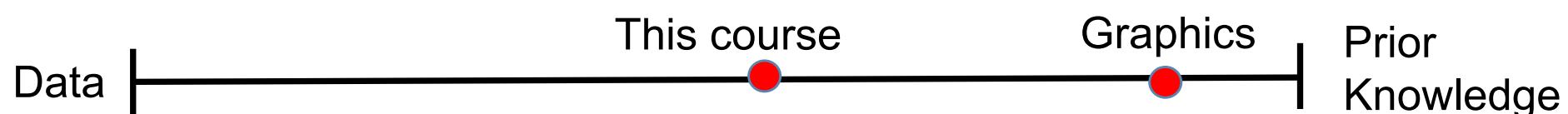


Data  
(e.g., images of bedrooms)



Prior Knowledge  
(e.g., physics, materials, ...)

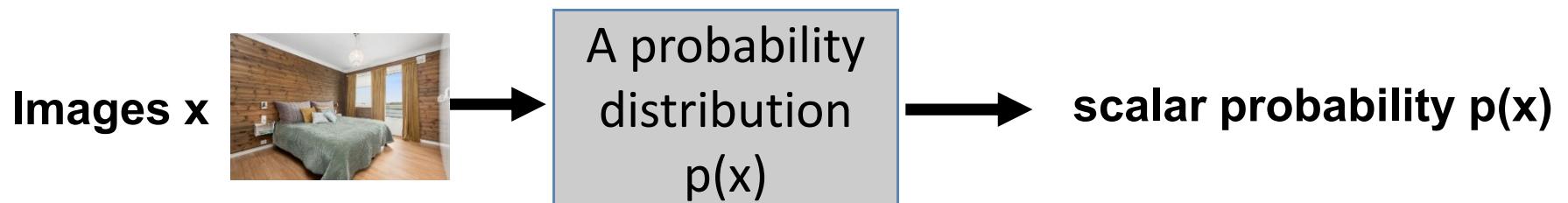
Priors are always necessary, but there is a spectrum



# Statistical Generative Models

**A statistical generative model is a probability distribution  $p(x)$**

- **Data:** samples (e.g., images of bedrooms)
- **Prior knowledge:** parametric form (e.g., Gaussian?), loss function (e.g., maximum likelihood?), optimization algorithm, etc.



**It is generative because sampling from  $p(x)$  generates new images...**



# What is the course going to cover

**Combines computer graphics, vision, NLP, and deep learning!**

**Tentative schedule:**

**Weeks 1-2:** Common 2D/3D/Text Representations & Operations

**Weeks 3-4:** Introduction to Neural Networks & Deep Learning

**Week 5:** Basics of Statistical Models

**Week 6:** Variational Autoencoders

**Week 7:** Generative Adversarial Networks

**Week 8:** Autoregressive models & Transformers

**Weeks 9-10:** Diffusion Models

**Week 11:** Controlling Generative Models

**Week 12:** LLMs & VLMs

**Week 13:** 3D/4D generative models

# Course links

**Moodle (lecture notes, assignments, forum):**

<https://moodle.tuc.gr/course/view.php?id=276>

**Zoom (same for office hours & attending remotely):**

<https://tuc-gr.zoom.us/j/94858453313?pwd=q7wZRwLRM9tuOe5oumCPQTEKO55EtL.1>



# Teaching plan

**The same lectures** are given in-person (**except for emergencies**) and transmitted online in real-time (**when zoom is ready**)

**Lectures are recorded** and are made available on Moodle later the same week.

You have the following options:

- Attend the Wed-Thu live meetings **in-person or remotely**
- **Watch the recorded lectures offline** and do not attend

Lectures start at 5:05-5:50, then 6:05-6:50 on Wed, then  
4:05-4:50, then 5:05-5:50 on Thu

# Marking Scheme (tentative)

- **25 pts Programming Assignment 1**
- **25 pts Programming Assignment 2**
- **25 pts Midterm**
- **25 pts Oral Paper Presentation or Project Presentation**

**Assignments are done individually!**

# Important Note

The course is quite **experimental**.

**...I am new faculty here trying to adapt in the academic environment of Greece ... which is far from easy.**

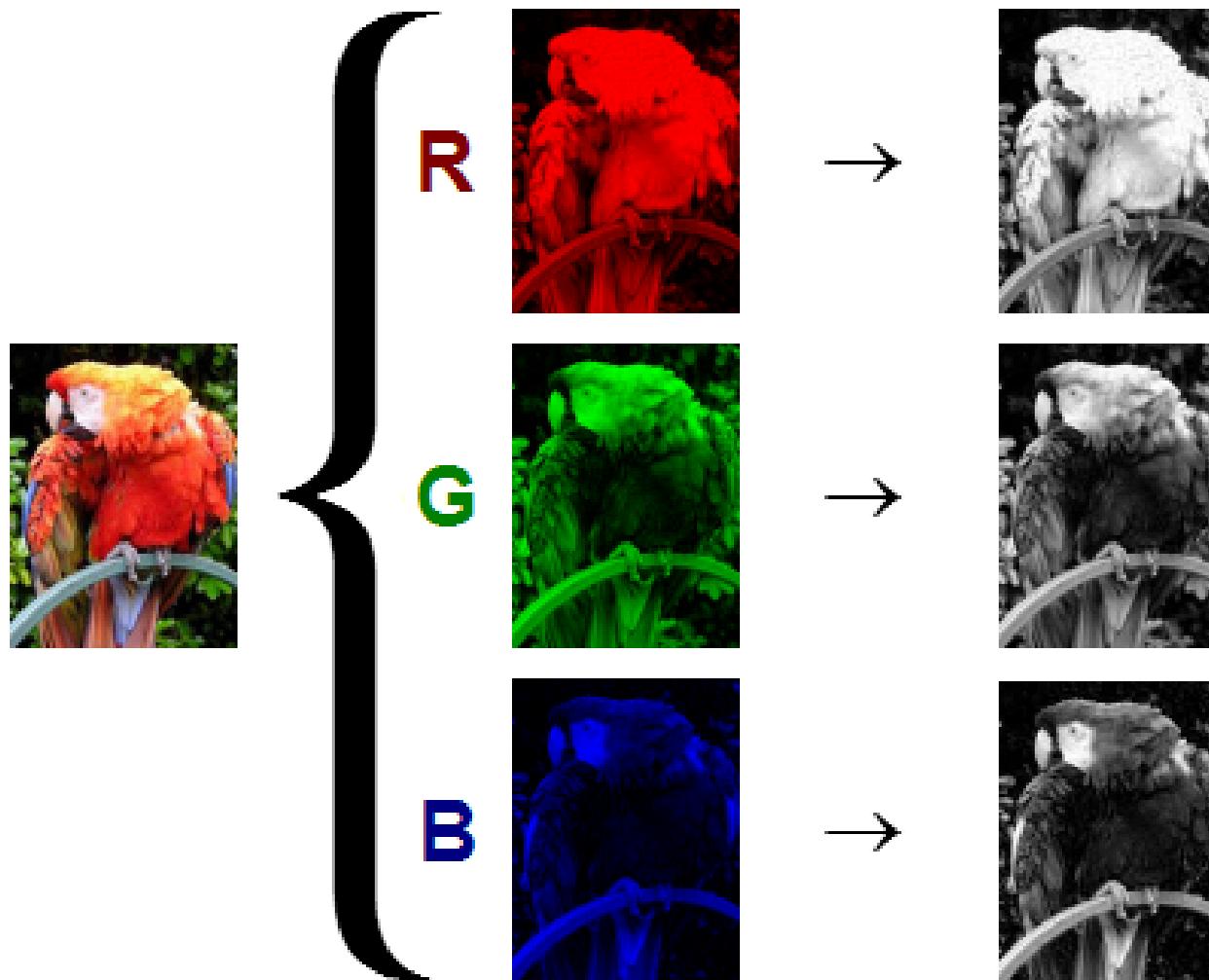
**Things may need to be adapted along the way!**

# Introduction to image and shape representations

## **Part I: Image representations**

# How do we represent images?

Images are often represented as arrays of pixels with **RGB** color.



# How do we represent images?

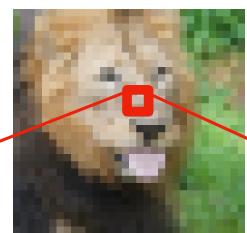
We can think of an **image** as a function,  $f$ , from  $\mathbb{R}^2$  to  $\mathbb{R}$ :

- $f(x, y)$  gives the **intensity** at position  $(x, y)$
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f: [a,b] \times [c,d] \rightarrow [0,1]$

A color image is just three functions pasted together. We can write this as a “vector-valued” function:

$$f(x, y) = \begin{bmatrix} red(x, y) \\ green(x, y) \\ blue(x, y) \end{bmatrix}$$

$(x, y)$

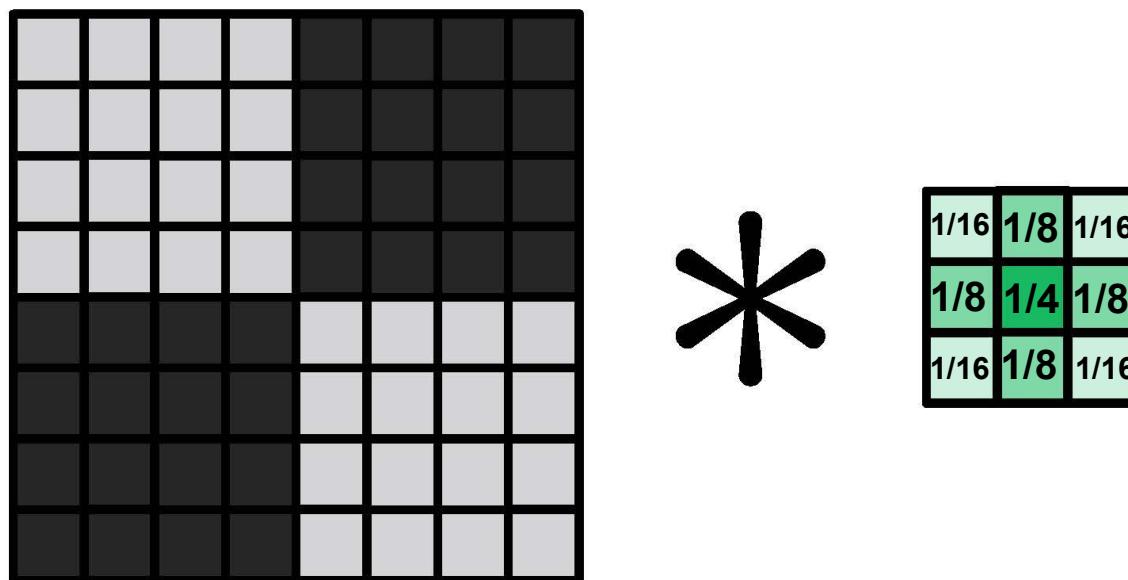


$(r, g, b)$

# Review: 2D Convolution

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l) I(i + k, j + l)$$

Note: formally, this is called cross-correlation.

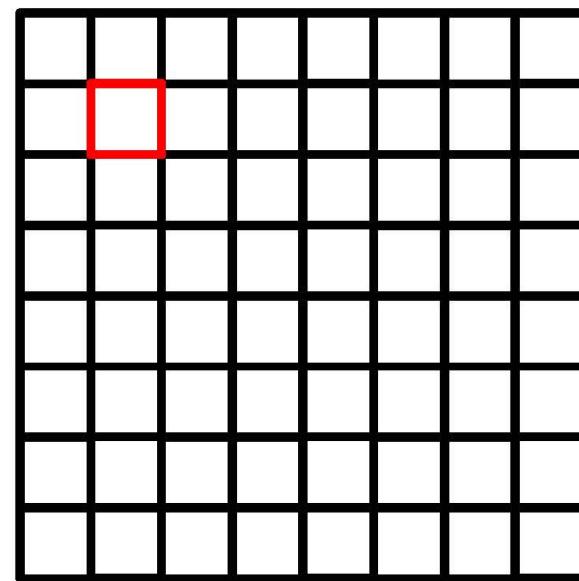
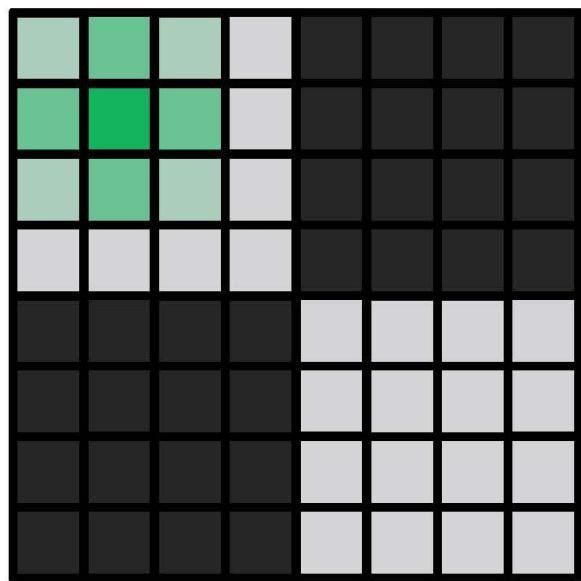


$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l) I(i - k, j - l)$$

Note: formally, this is convolution. The minus difference (i.e., flipping) is not important for neural networks since the filters are learned.

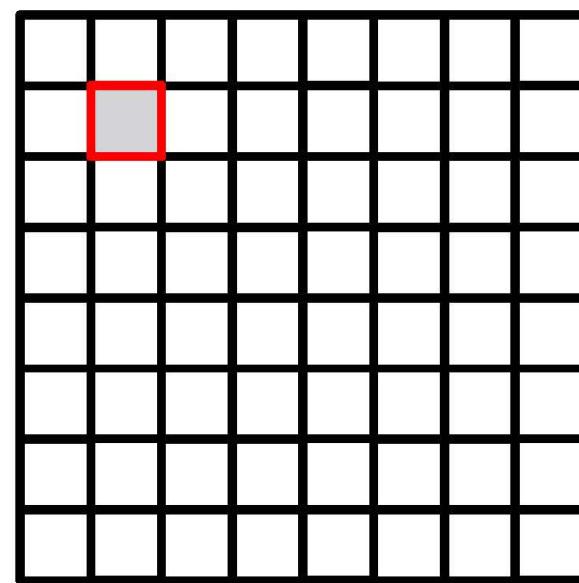
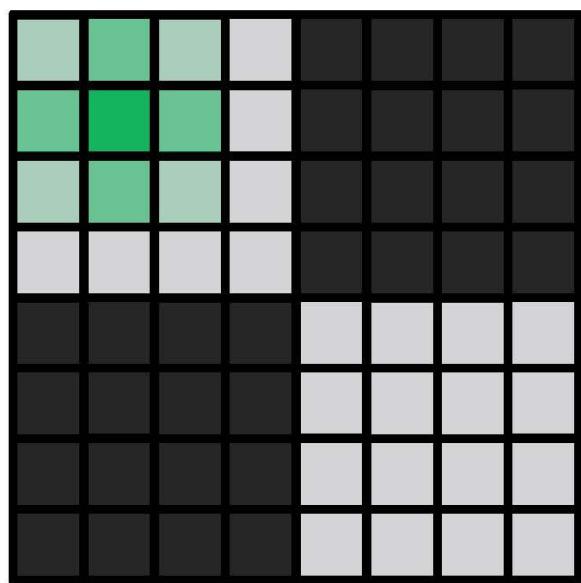
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



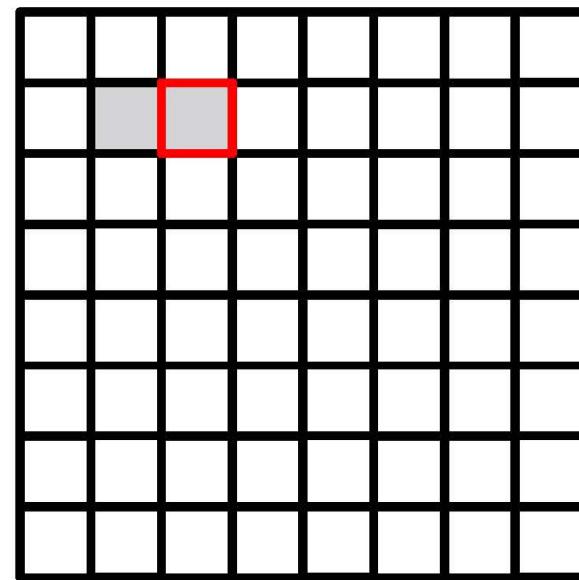
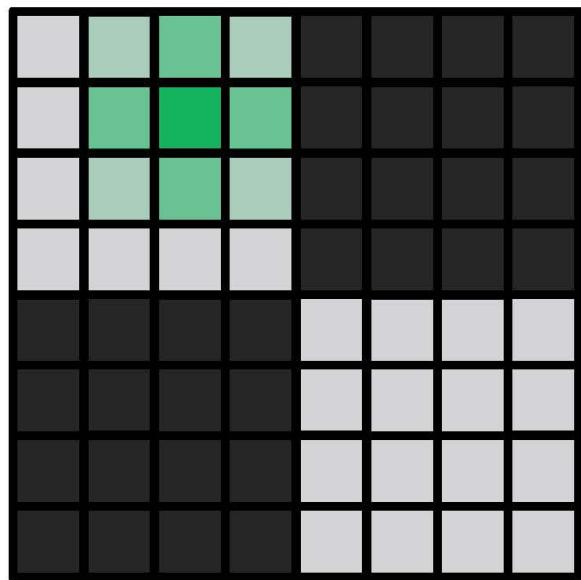
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



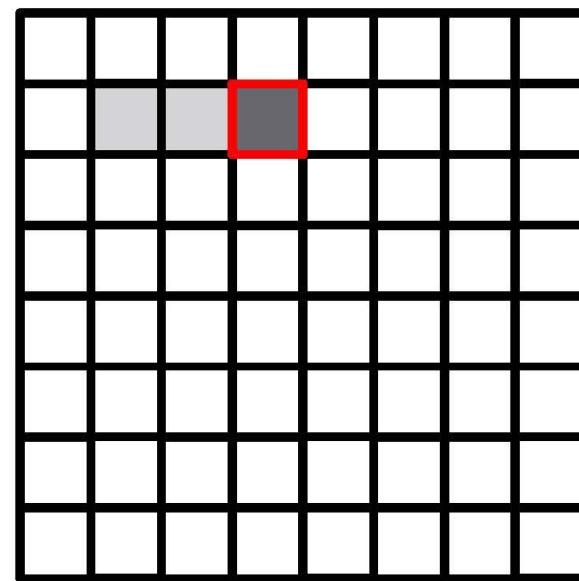
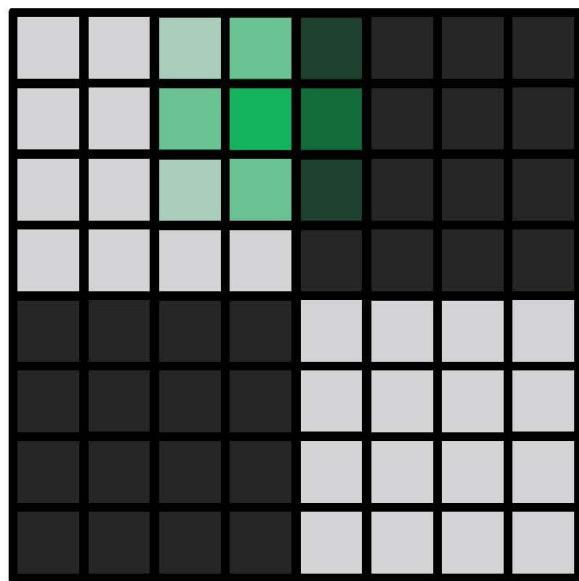
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



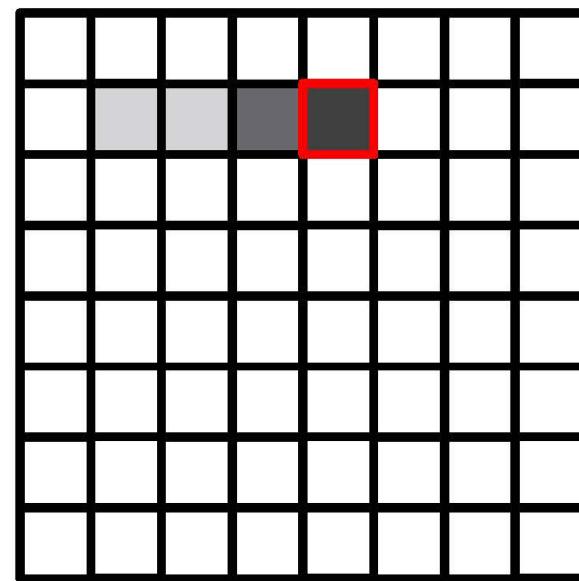
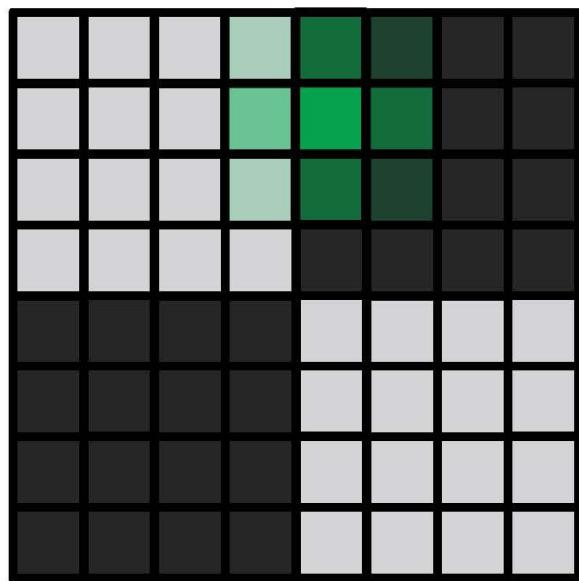
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



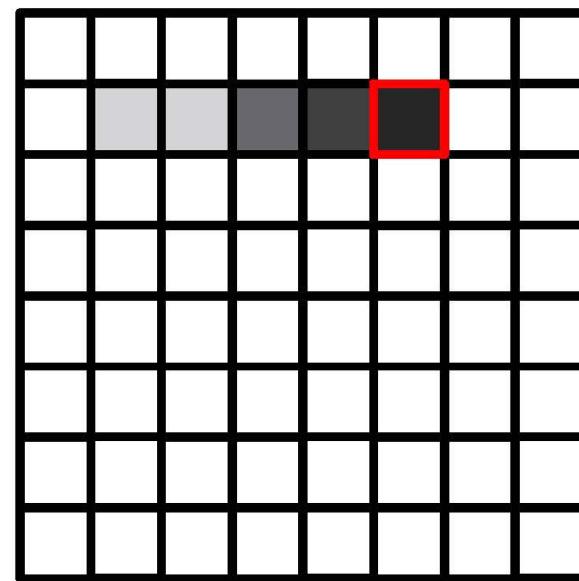
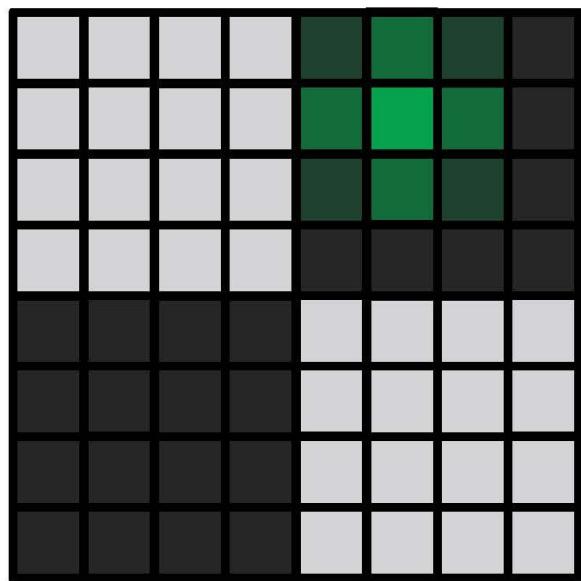
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



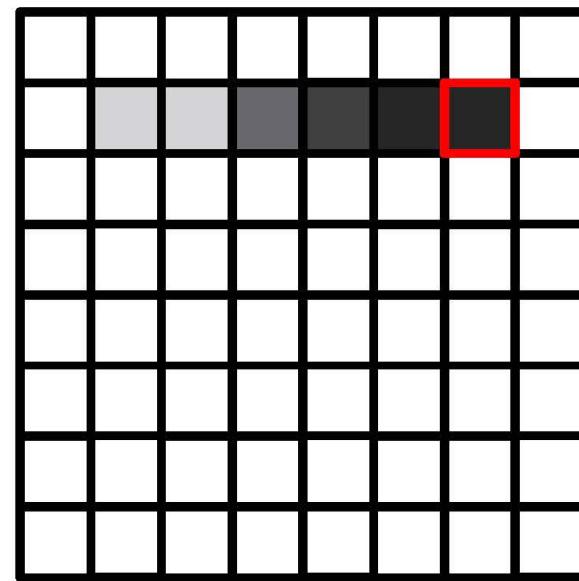
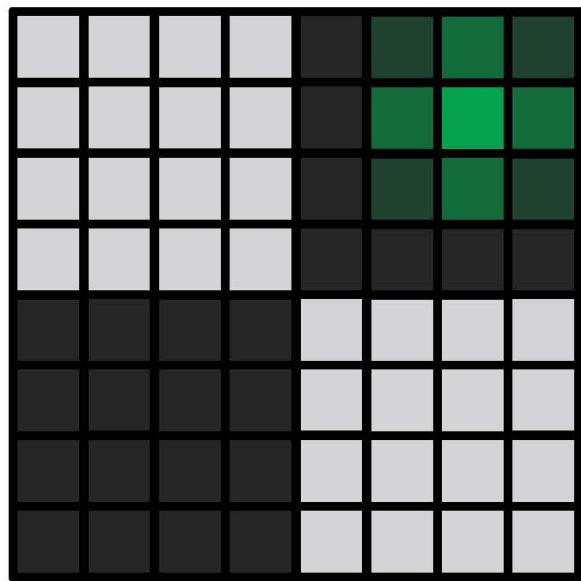
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



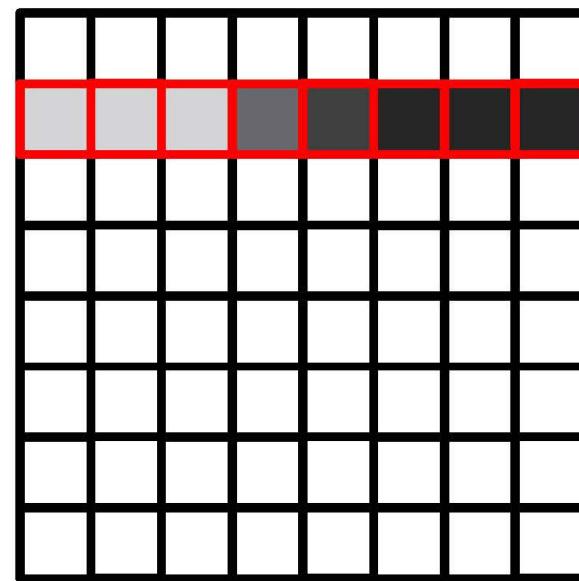
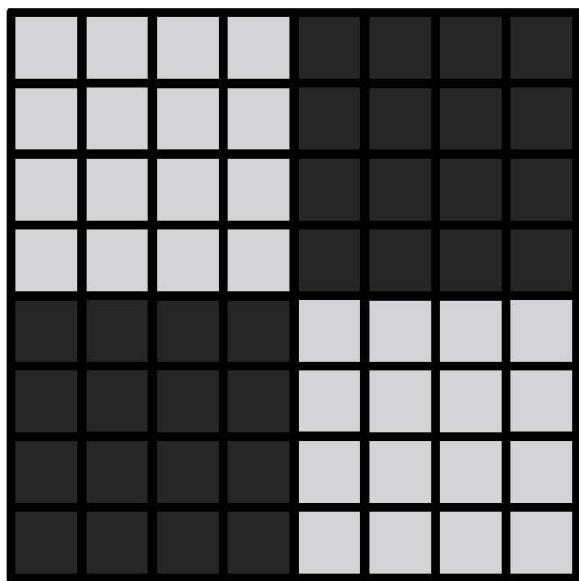
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



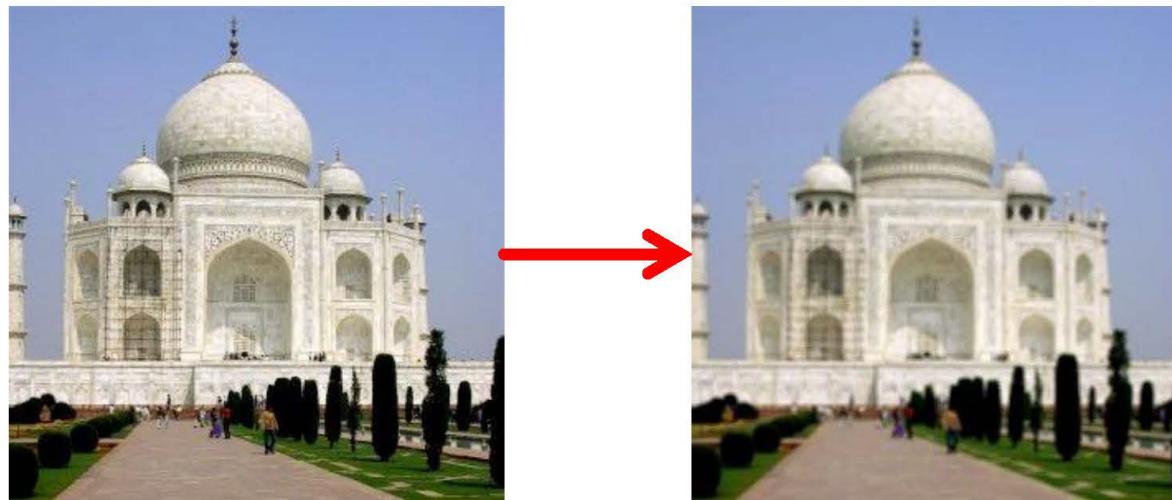
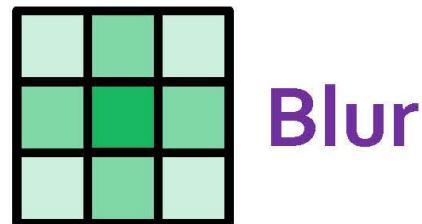
# 2D Convolution Review

$$O(i, j) = \sum_{k=-n}^{k=n} \sum_{l=-n}^{l=n} w(k, l)I(i + k, j + l)$$



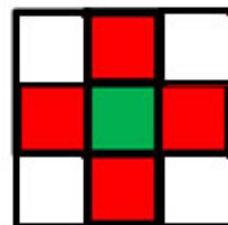
# 2D Convolution Review (Blurring)

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

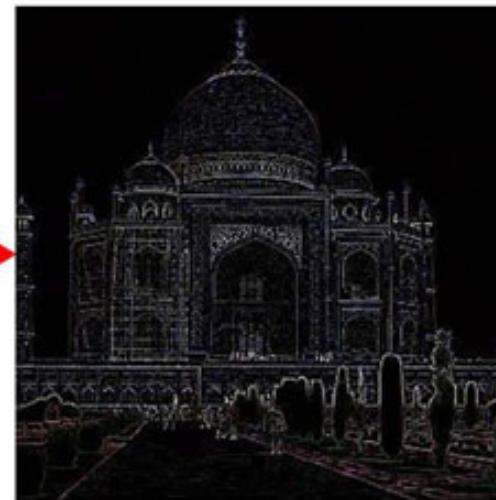


# 2D Convolution Review (Edge Detection)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Edge detect



# Part II: Shape Representations



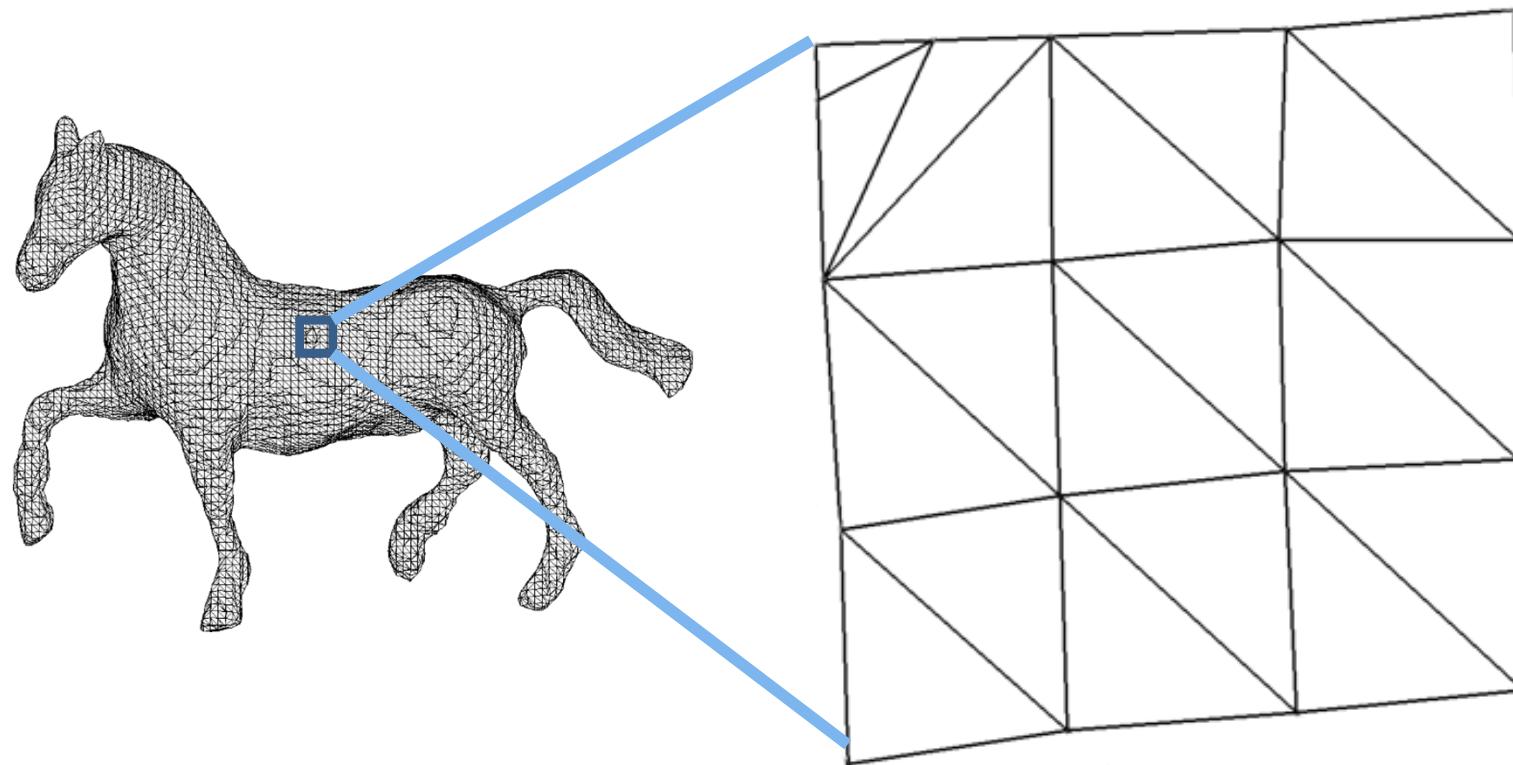
# Shape representations

- **Polygon Meshes**
- Textures
- Parametric Surfaces
- Voxel grids / octrees
- Point clouds
- Gaussian Splats
- Implicit functions

# Polygon Meshes

Most common digital shape representation in graphics

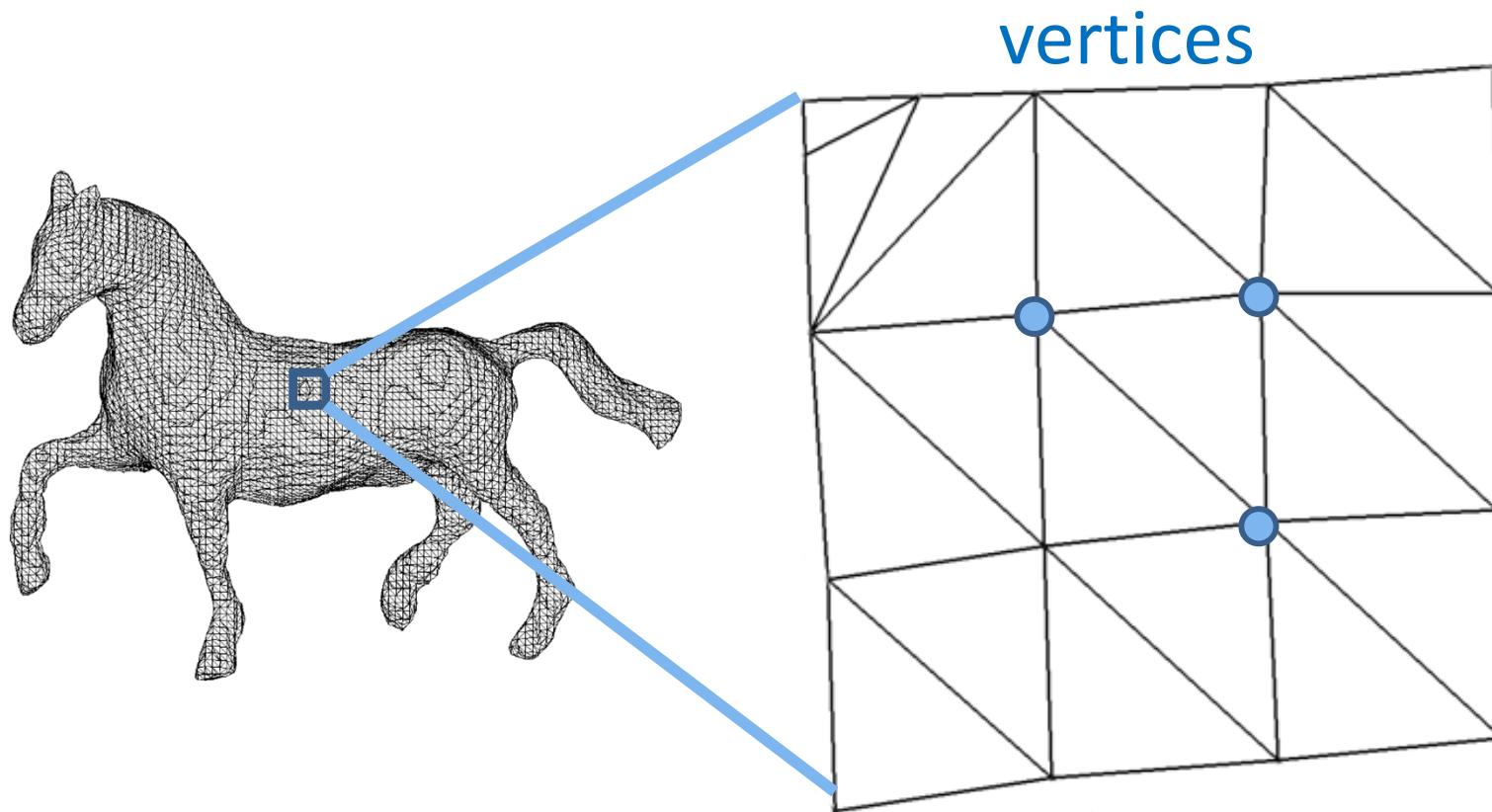
Collection of polygons (faces) that form the “skin” of an object



# Polygon Meshes

Most common digital shape representation in graphics

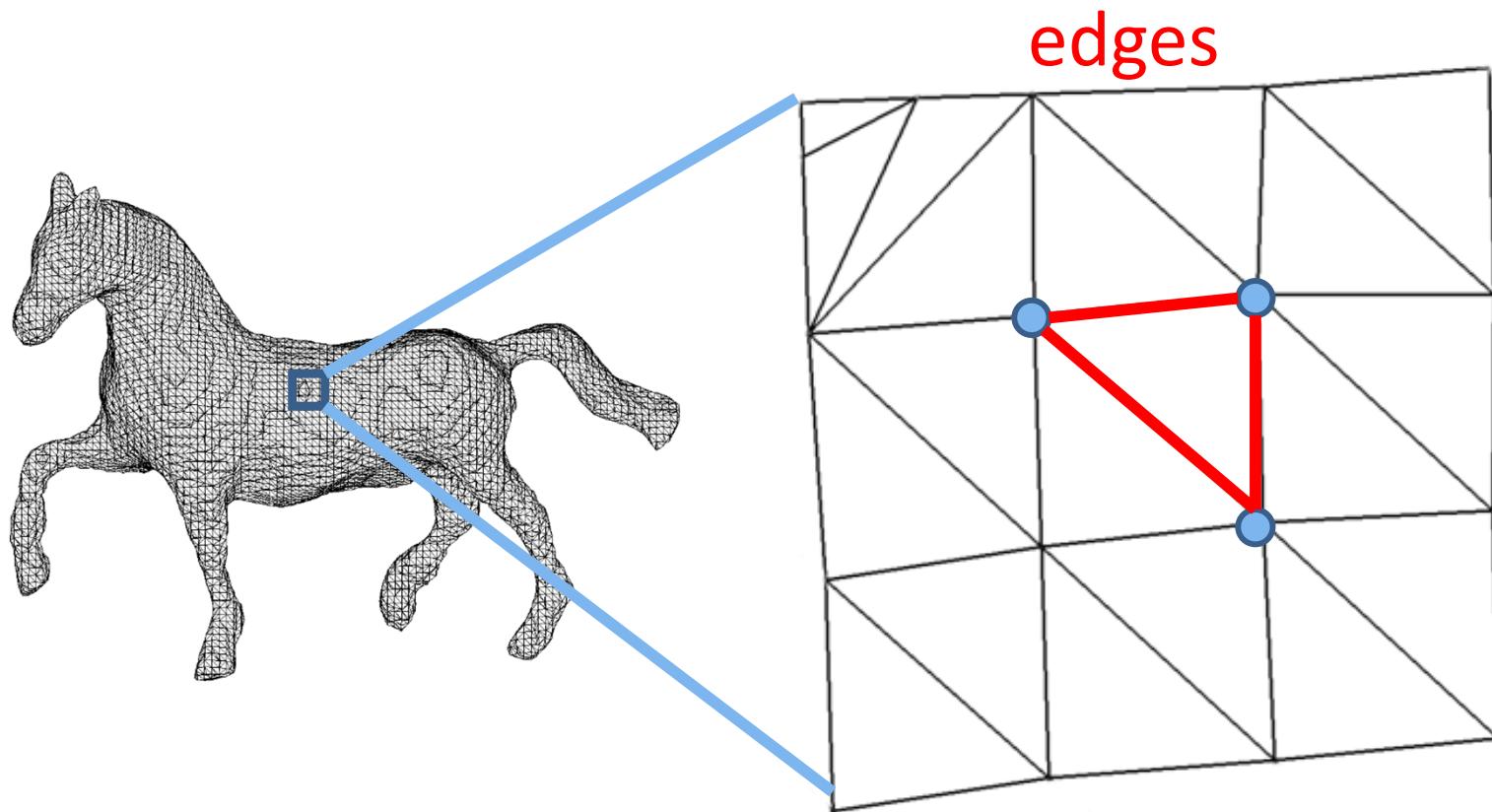
Collection of polygons (faces) that form the “skin” of an object



# Polygon Meshes

Most common digital shape representation in graphics

Collection of polygons (faces) that form the “skin” of an object

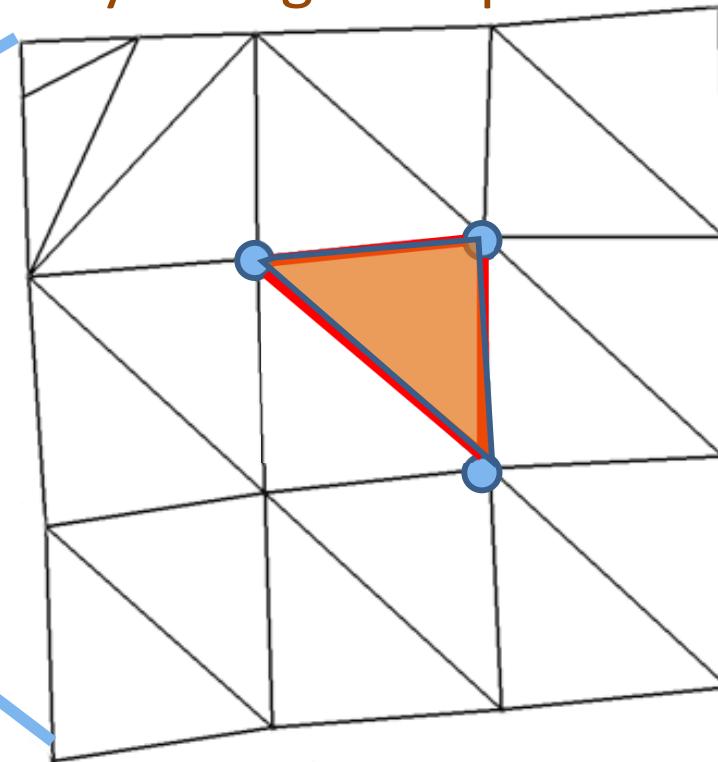
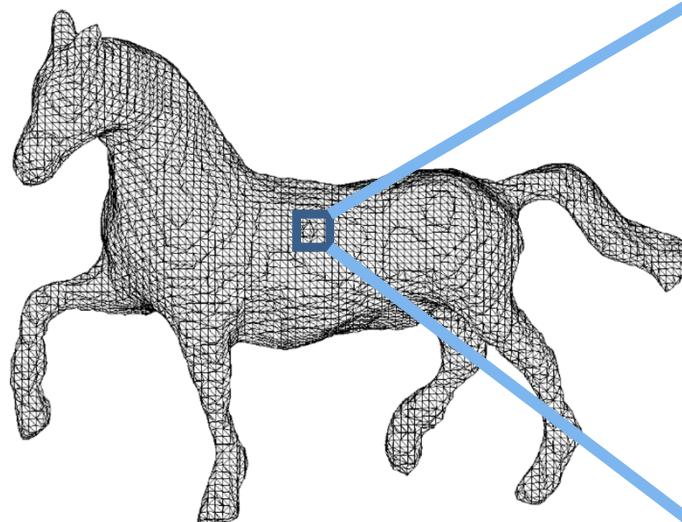


# Polygon Meshes

Most common digital shape representation in graphics

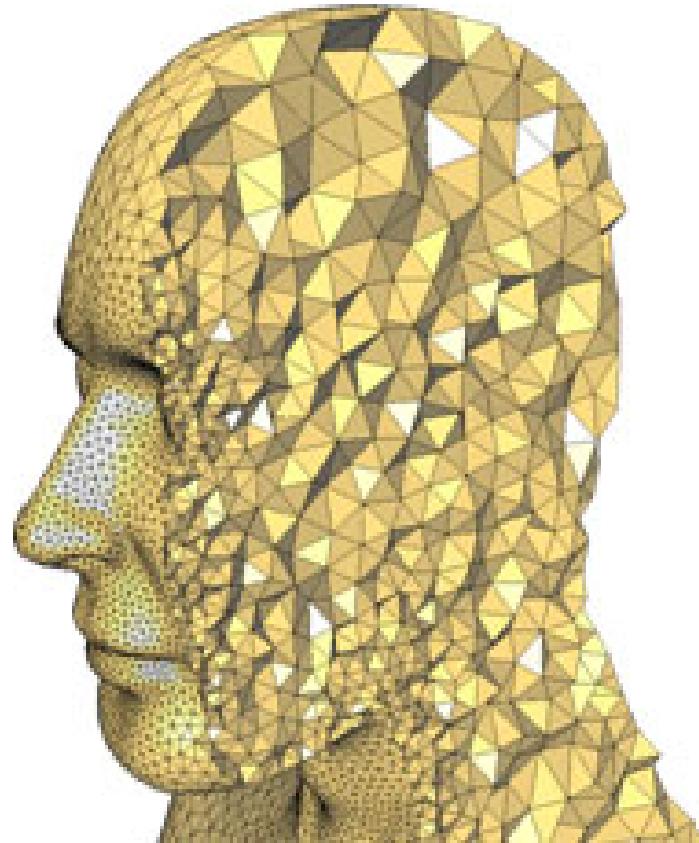
Collection of polygons (faces) that form the “skin” of an object  
face

(usually triangle or quadrilaterals)



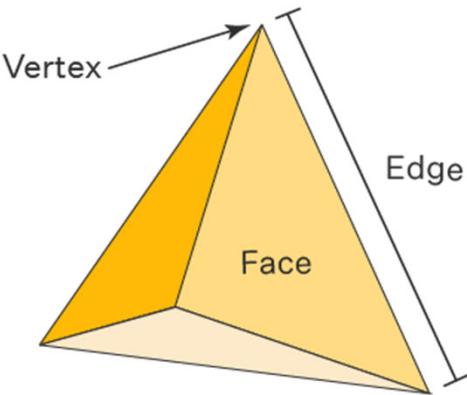
# Volumetric Meshes

Useful for physics-based simulations



Tetrahedral meshes

## Tetrahedron Properties



A tetrahedron has:

- 4 Faces
- 4 Vertices
- 6 Edges

# Data Structures for meshes

- What should we store?
  - Geometry: 3D coordinates of polygon vertices
  - Other mesh properties e.g. connectivity information, normals, colors, texture coordinates, etc.
- Common mesh representation: **vertex and face list**

# OBJ file format

# Comments start with '#'

# List of Vertices, with (x,y,z) coordinates

v 0.123 0.234 0.345

v 0.433 0.348 0.438

...

# Then list of faces (**vertex index starts from 1**)

f 1 2 3

f 3 4 5

...

# Polygonal modeling

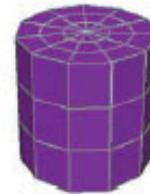
Start with predefined primitives...



sphere



cube



cylinder



cone



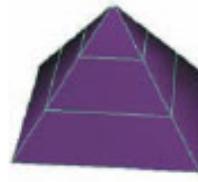
plane



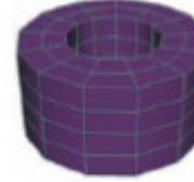
torus



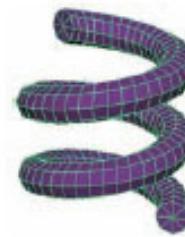
prism



pyramid



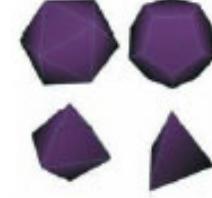
pipe



helix



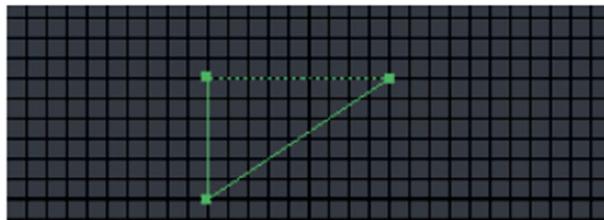
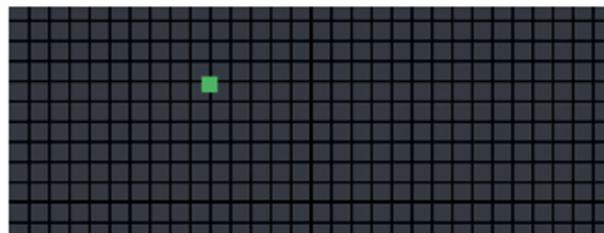
soccer ball



platonic

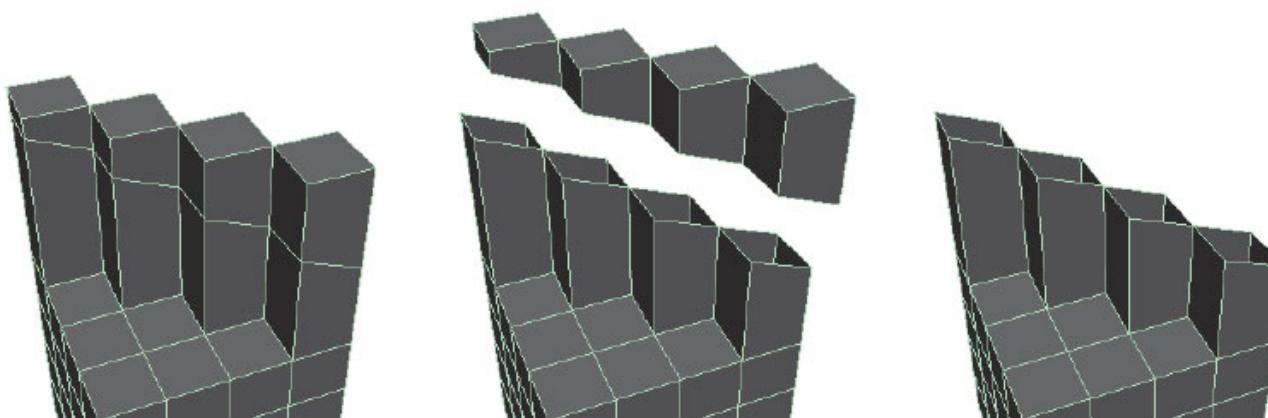
# Polygonal modeling

Click and add/delete/move vertices...



# Polygonal modeling

Specify cutting line / plane to cut polygon mesh



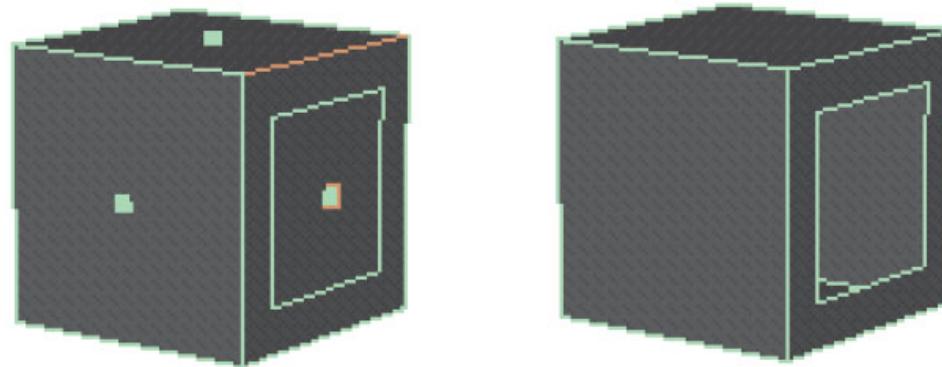
Use Cut Faces to create  
edges...

...to extract faces, or...

...to delete faces.

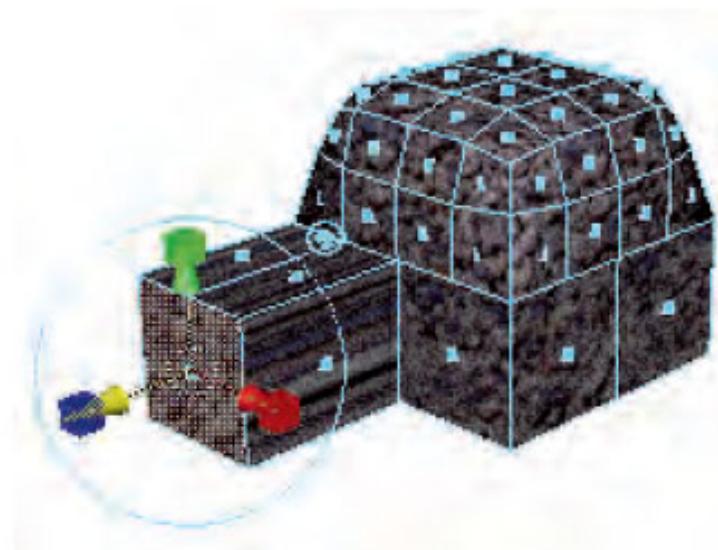
# Polygonal modeling

Specify a polygon to create a hole in another polygon

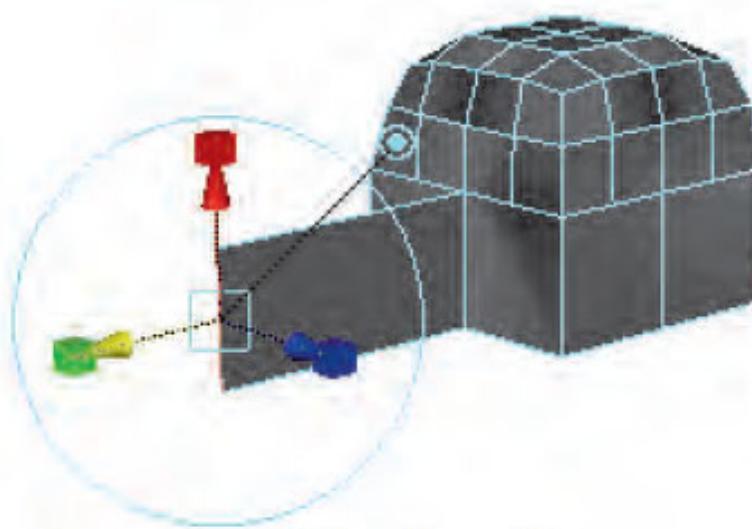


# Polygonal modeling

Extrude polygon faces, edges or vertices along some specified direction

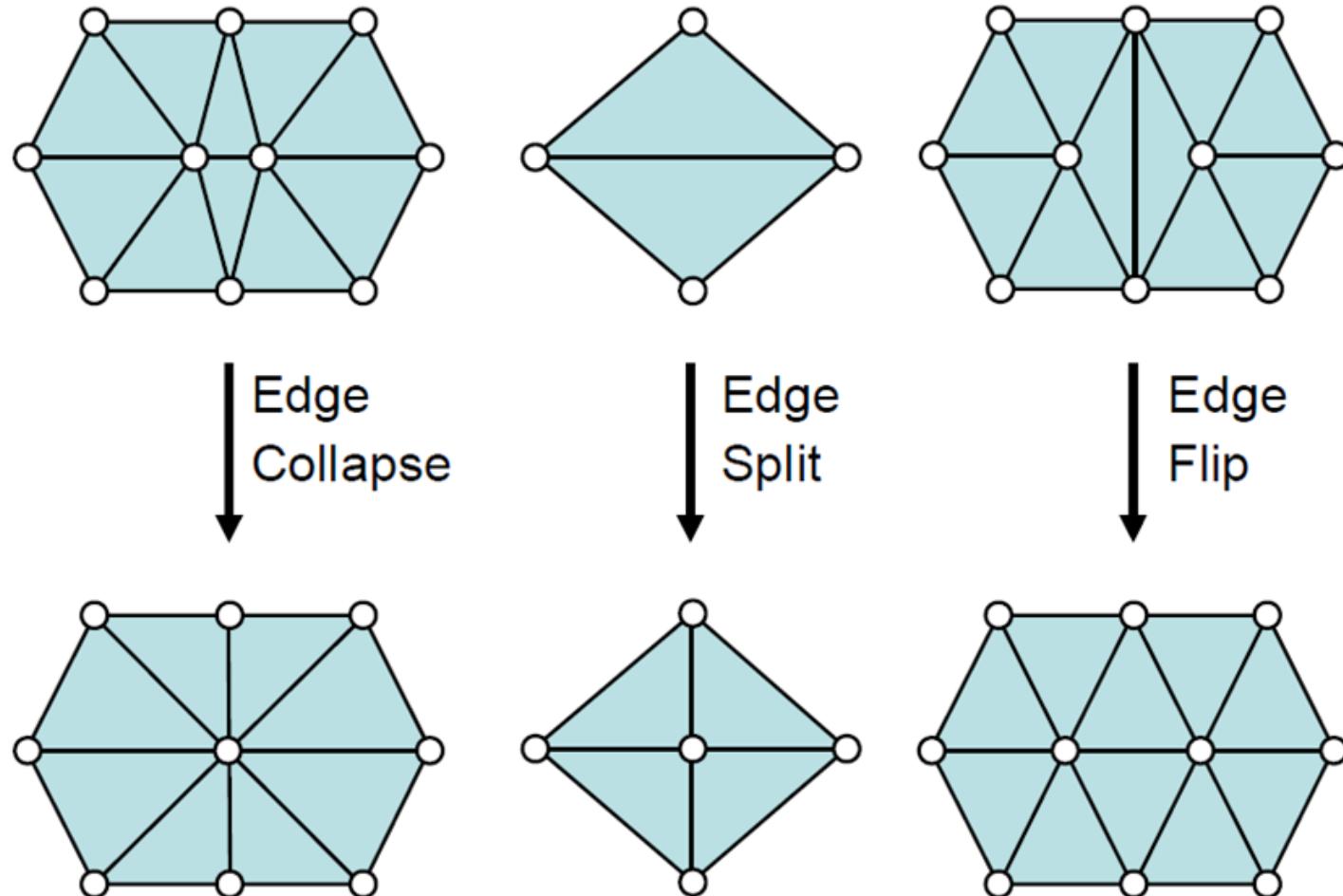


Extrude face

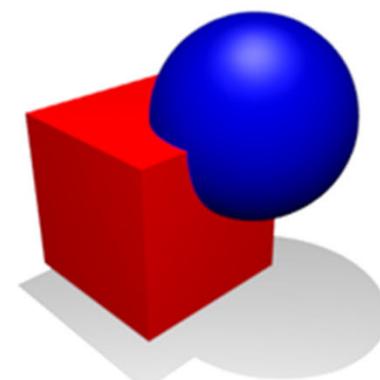


Extrude edge

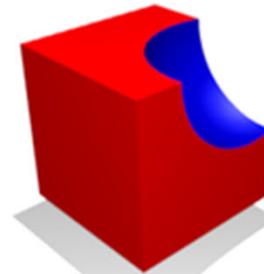
# Polygonal modeling



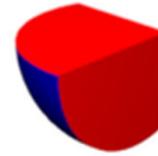
# Boolean Operations (Constructive Solid Geometry)



A union B

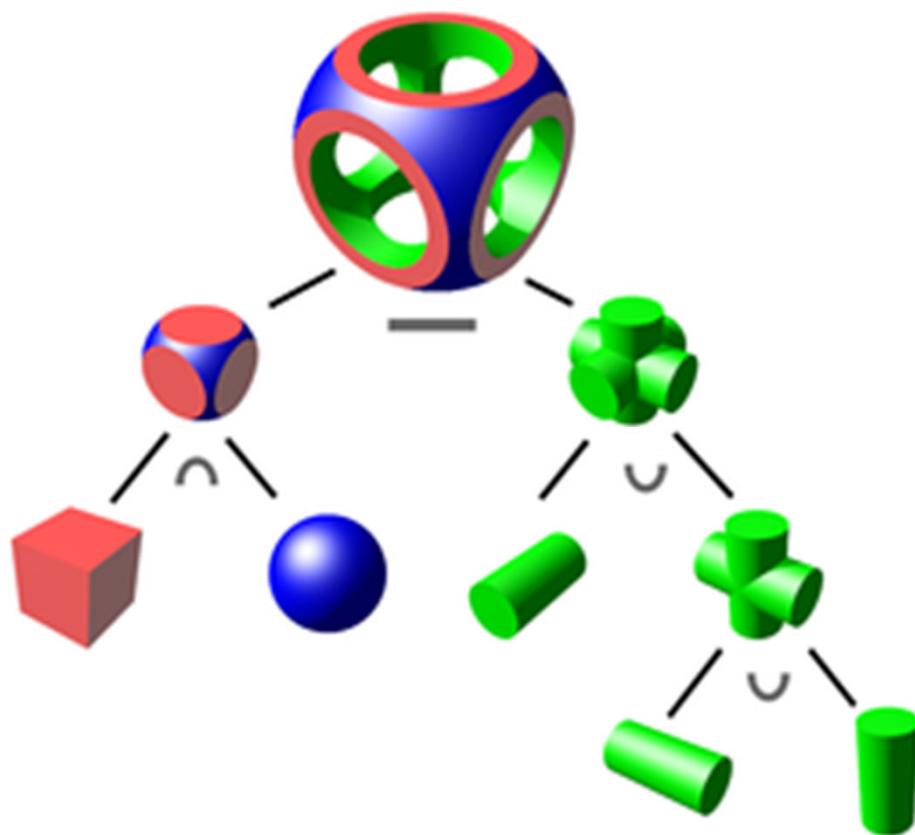


A diff B

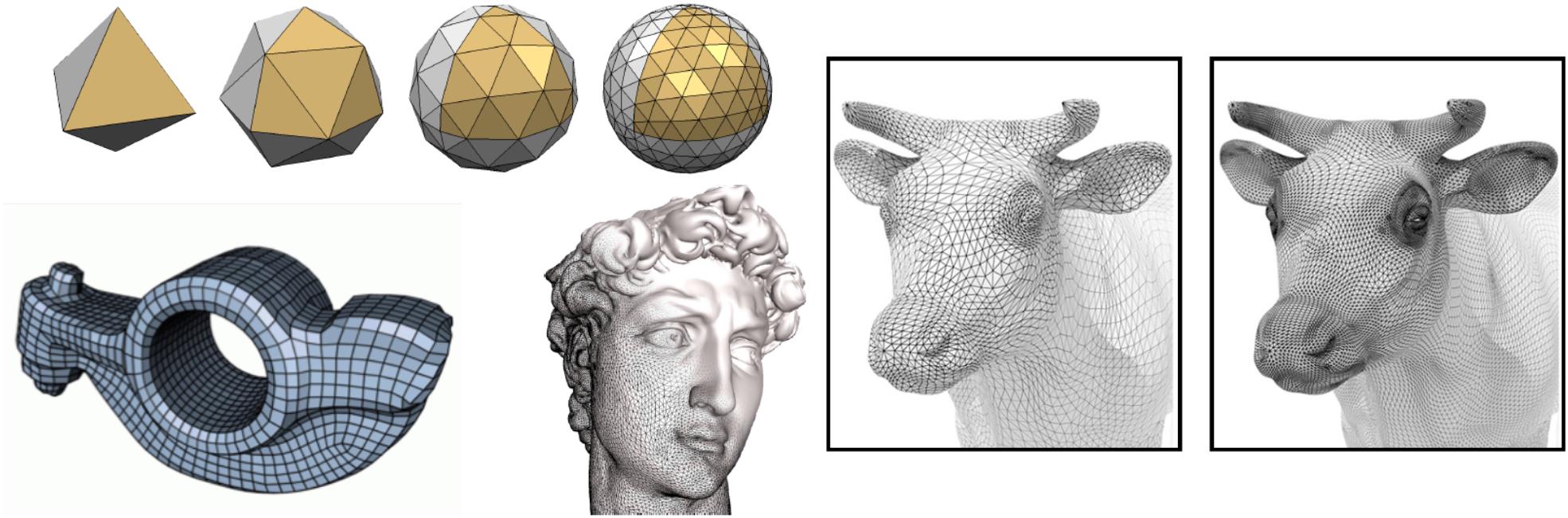


A intersection B

# Boolean Operations (Constructive Solid Geometry)



# Polygon Mesh Advantages/Disadvantages



- Rendering is plausible and fast
- Many techniques for mesh deformation, animation, subdivision...
- Piecewise linear approximation of the underlying surface
- Unordered representation: vertex order is arbitrary, no fixed number of vertices, different #neighbors per vertex...

# Shape representations

- Polygon Meshes
- **Textures**
- Parametric Surfaces
- Voxel grids / octrees
- Point clouds
- Gaussian Splats
- Implicit functions



# What about color (texture)?

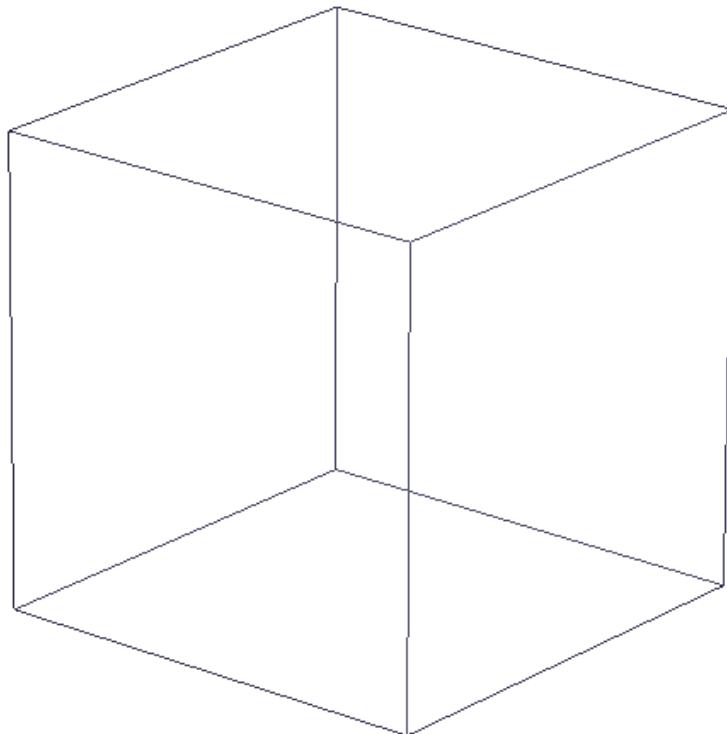
**Set material color per vertex for meshes**

(or per voxel in voxel grids, or per point in point clouds...)



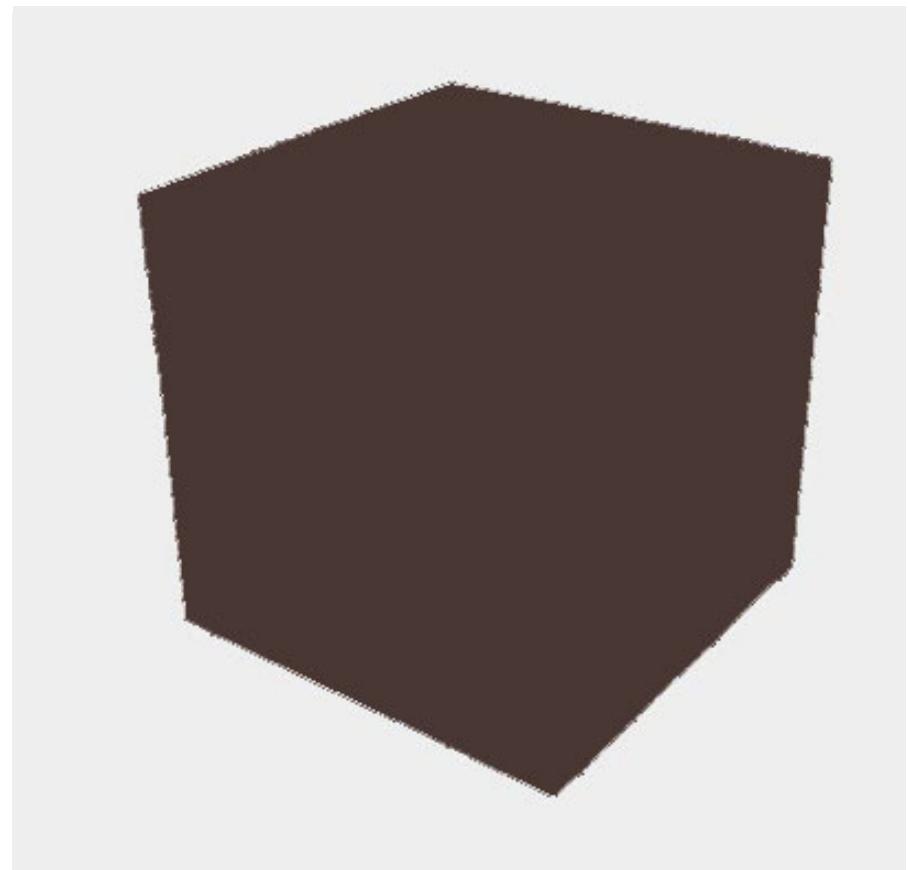
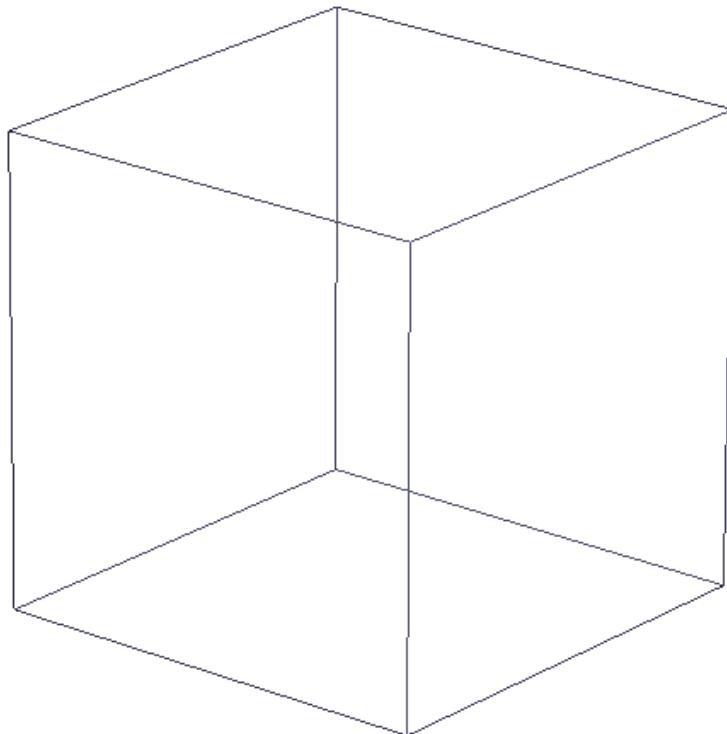
# High-res texture, low-res geometry

Let's say you want to design a crate



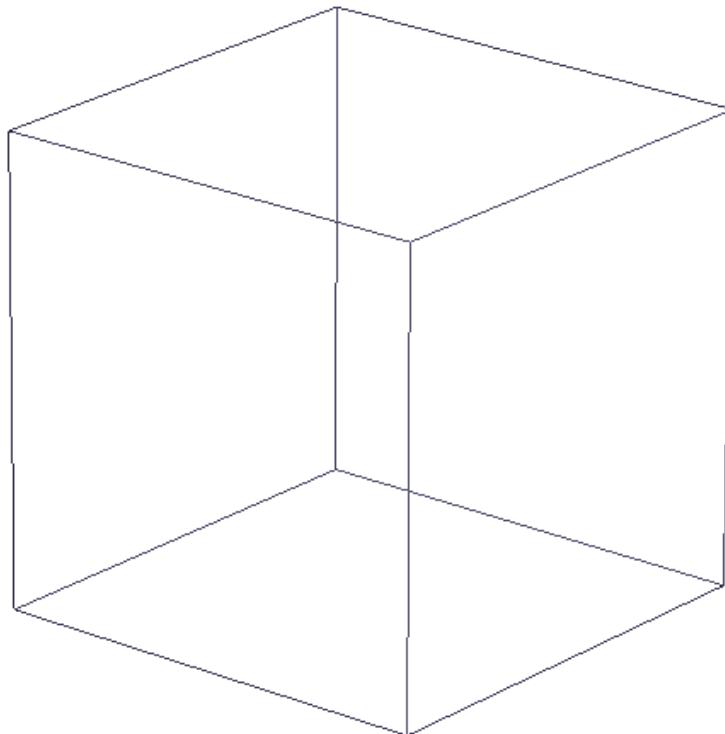
# High-res texture, low-res geometry

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



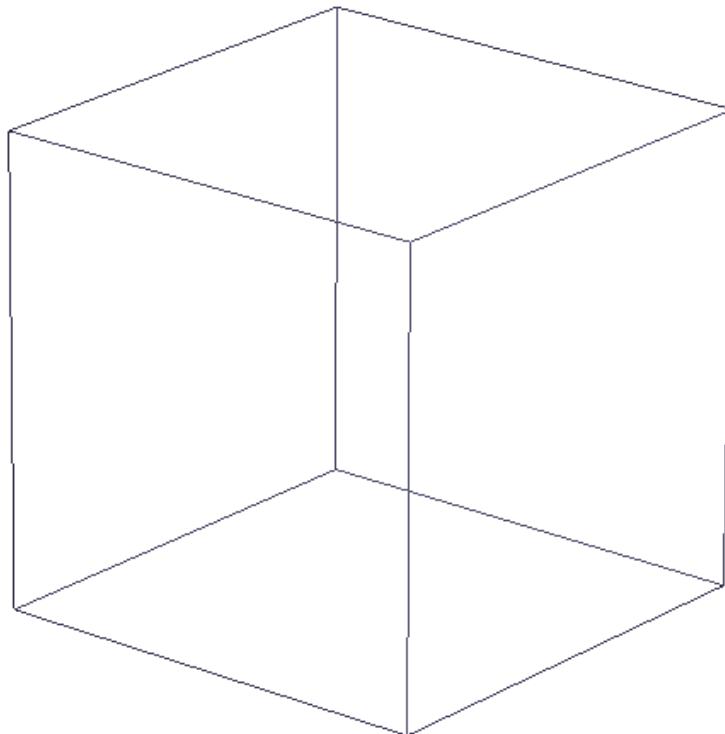
# High-res texture, low-res geometry

Texture and geometry resolution do not necessarily coincide.



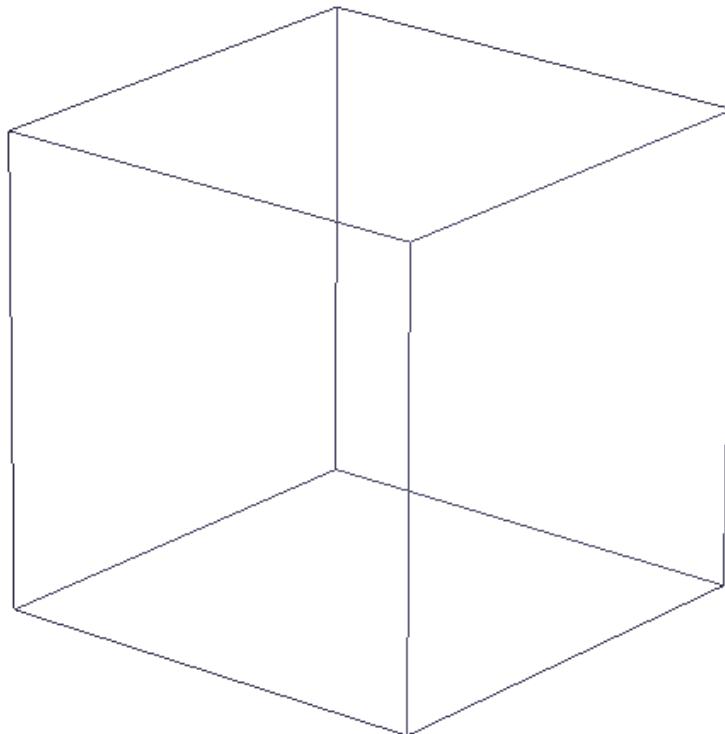
# High-res texture, low-res geometry

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.:



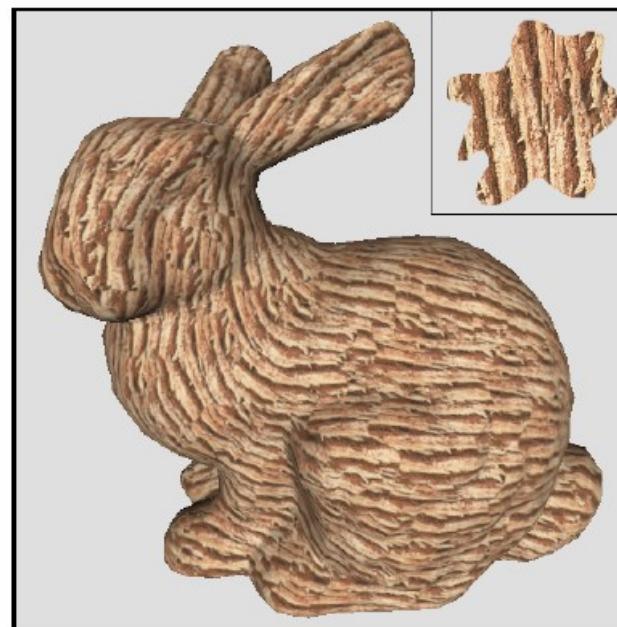
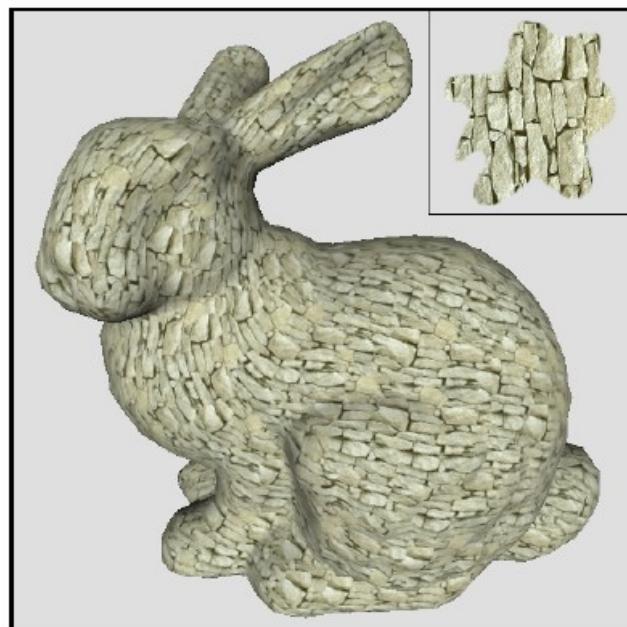
**wood**



**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”

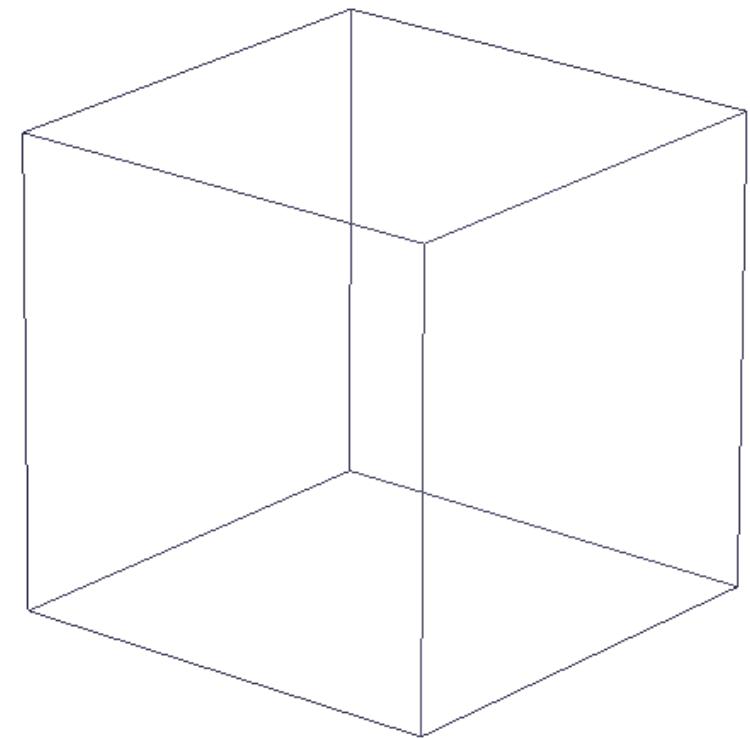
# Problem: mapping between 2D and 3D...

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

**2D texture**



**3D model**

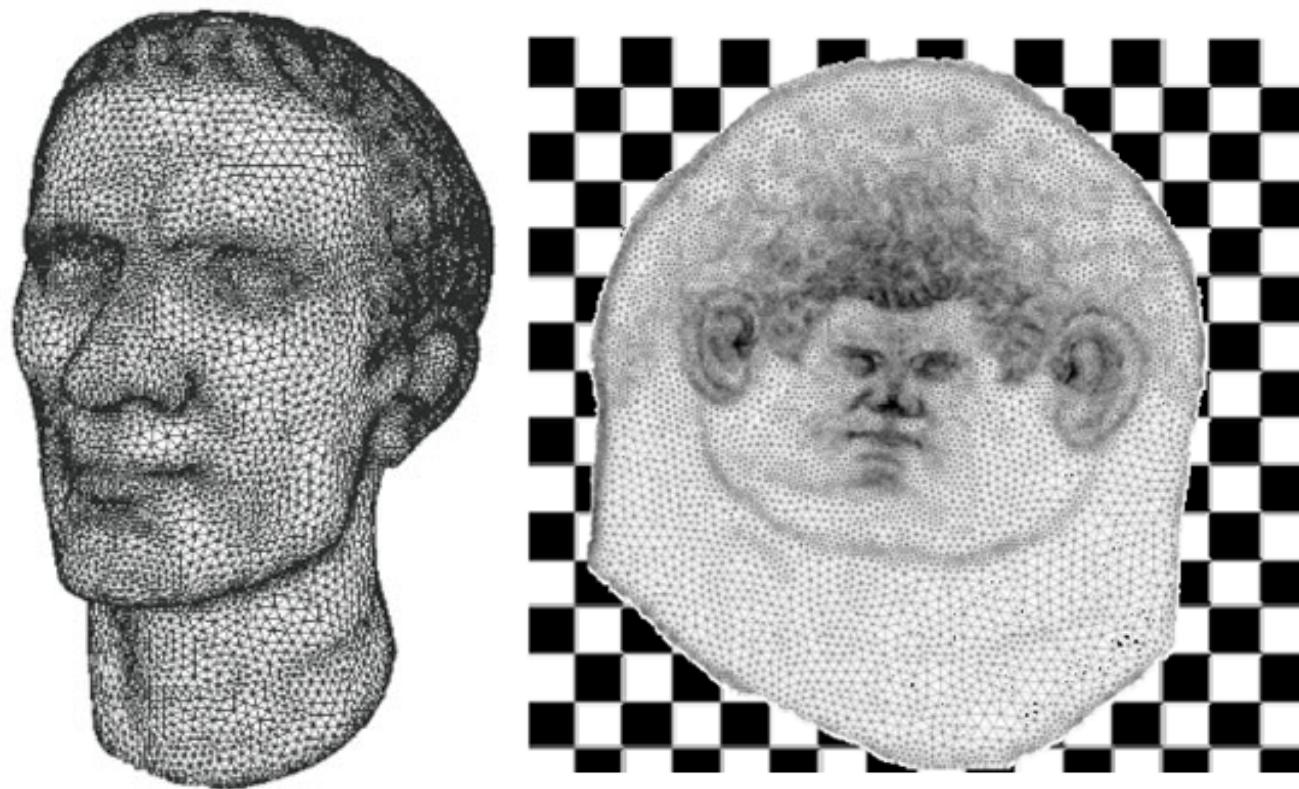


We need a way to flatten a 3D surface...



**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



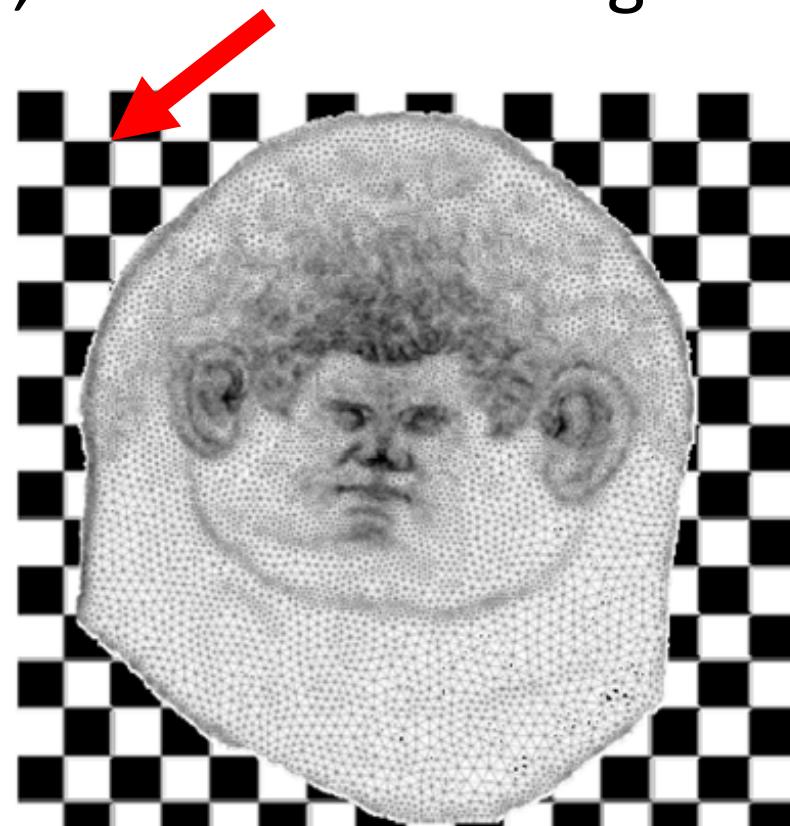
Mr  
Checkerboard

# 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

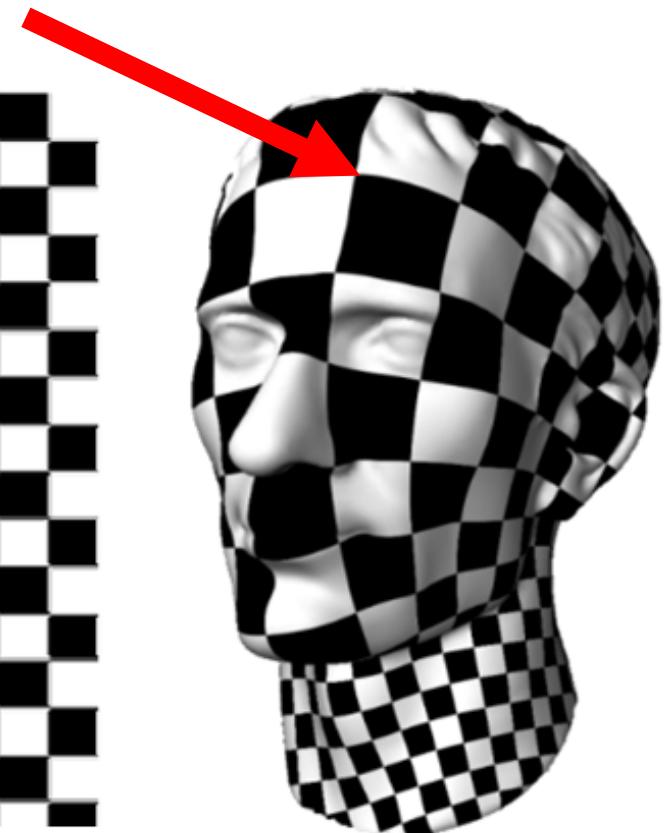
Is the angle here 90 degrees?



**Polygon Mesh**



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



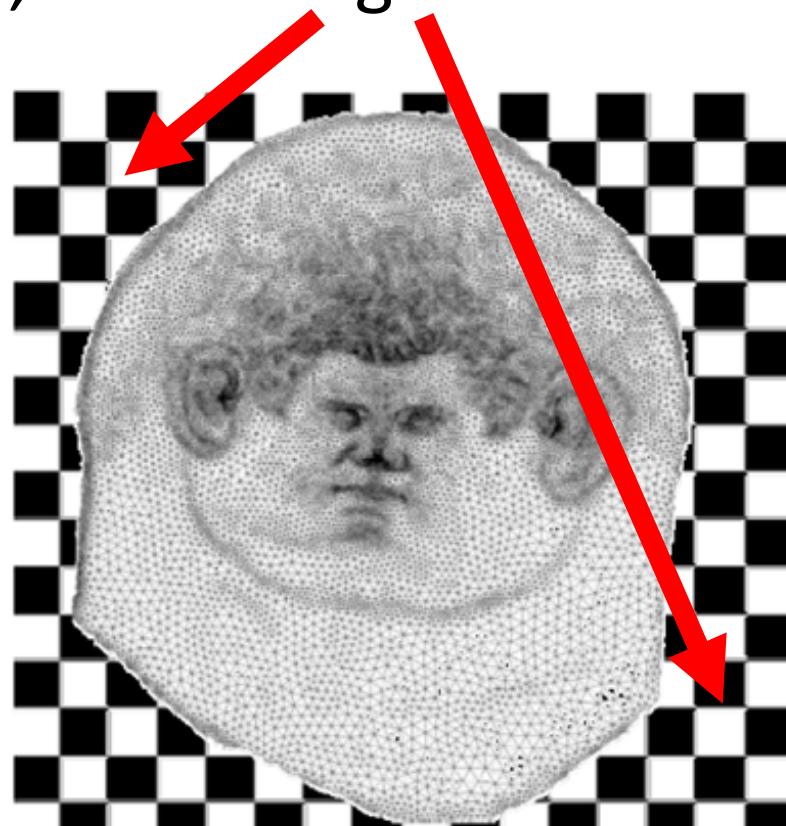
**Mr  
Checkerboard**

# 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**



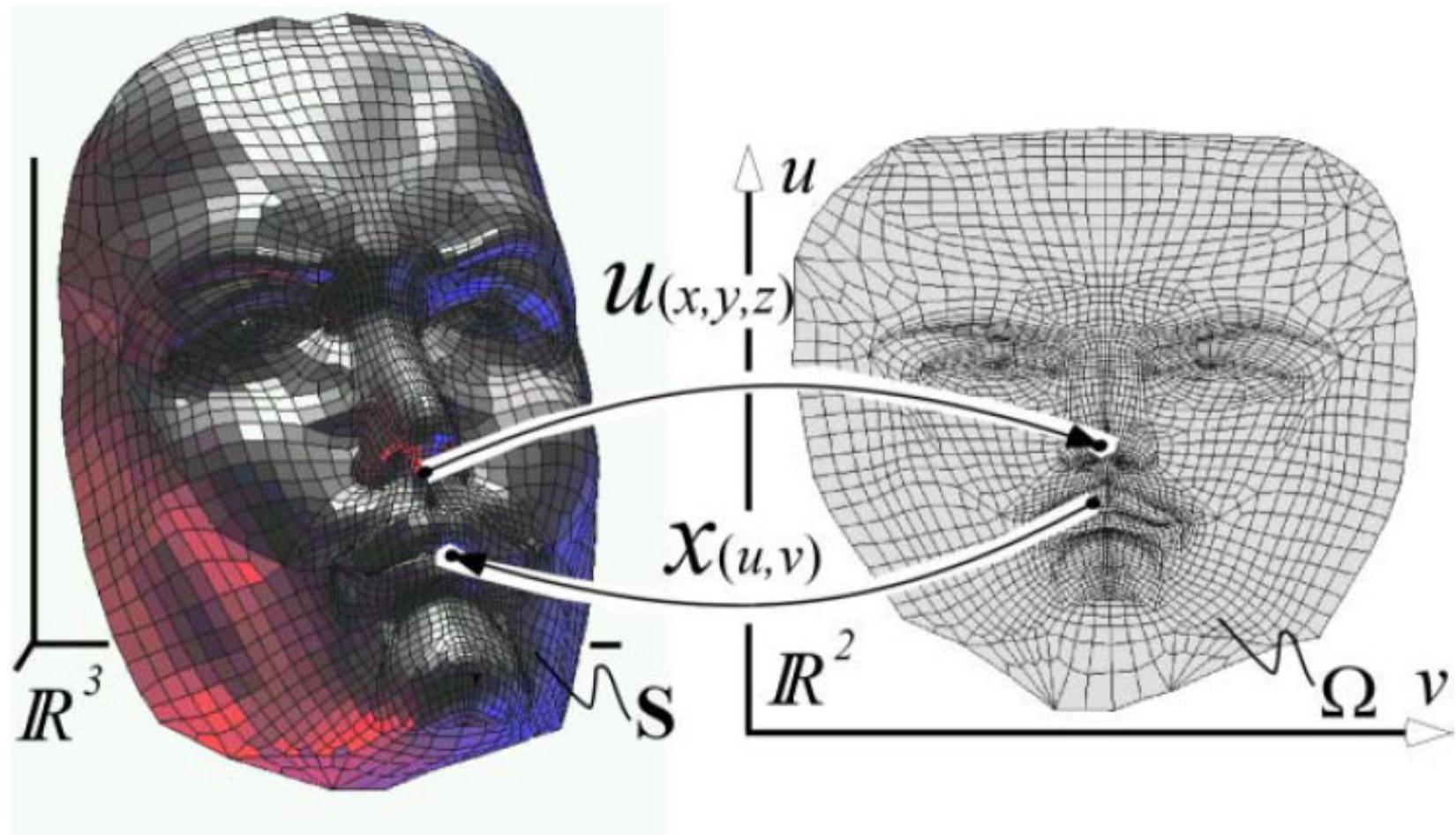
**Flattened mesh**  
every vertex now is associated  
with a color from the 2D  
checkerboard image



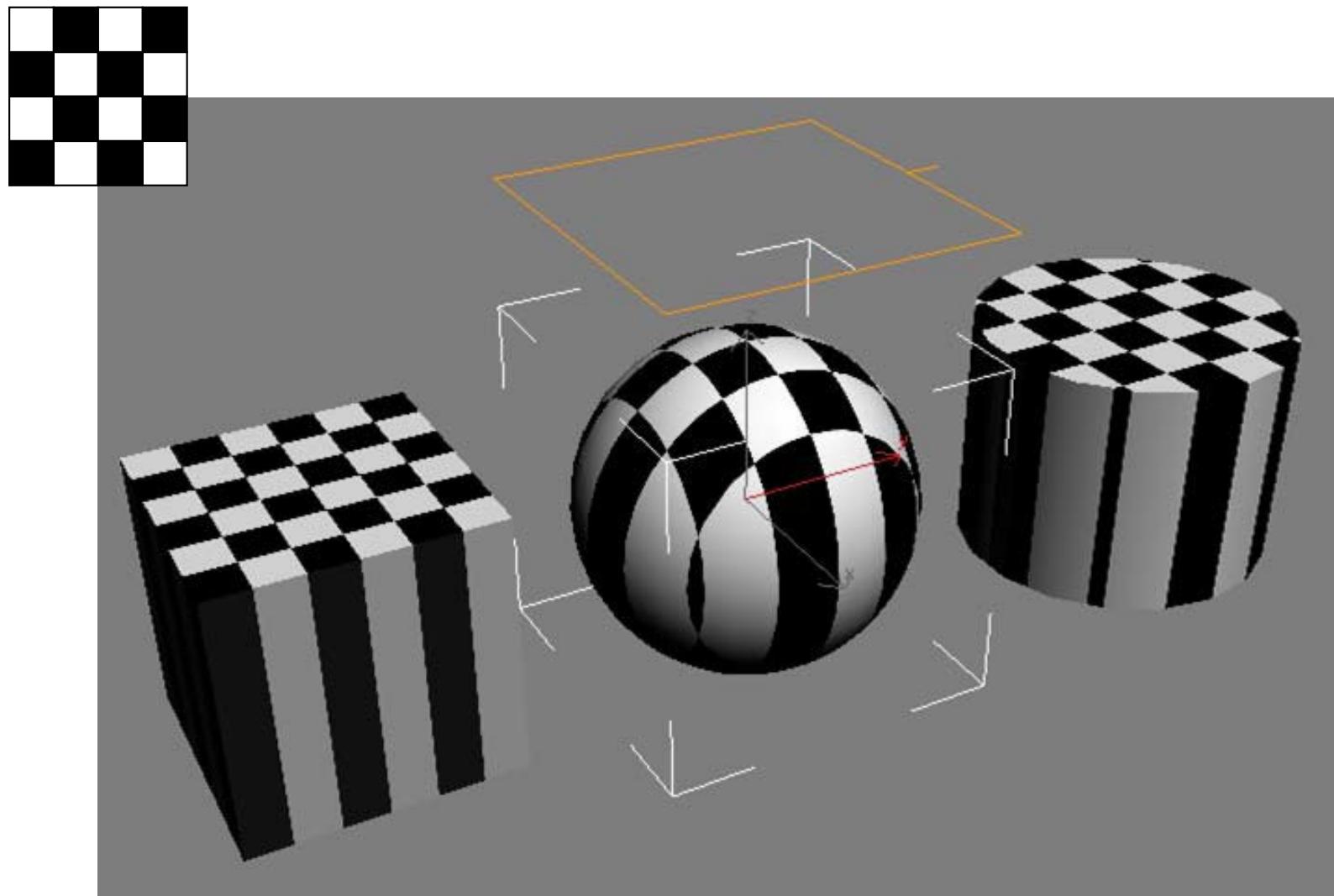
**Mr  
Checkerboard**

# Surface parameterization

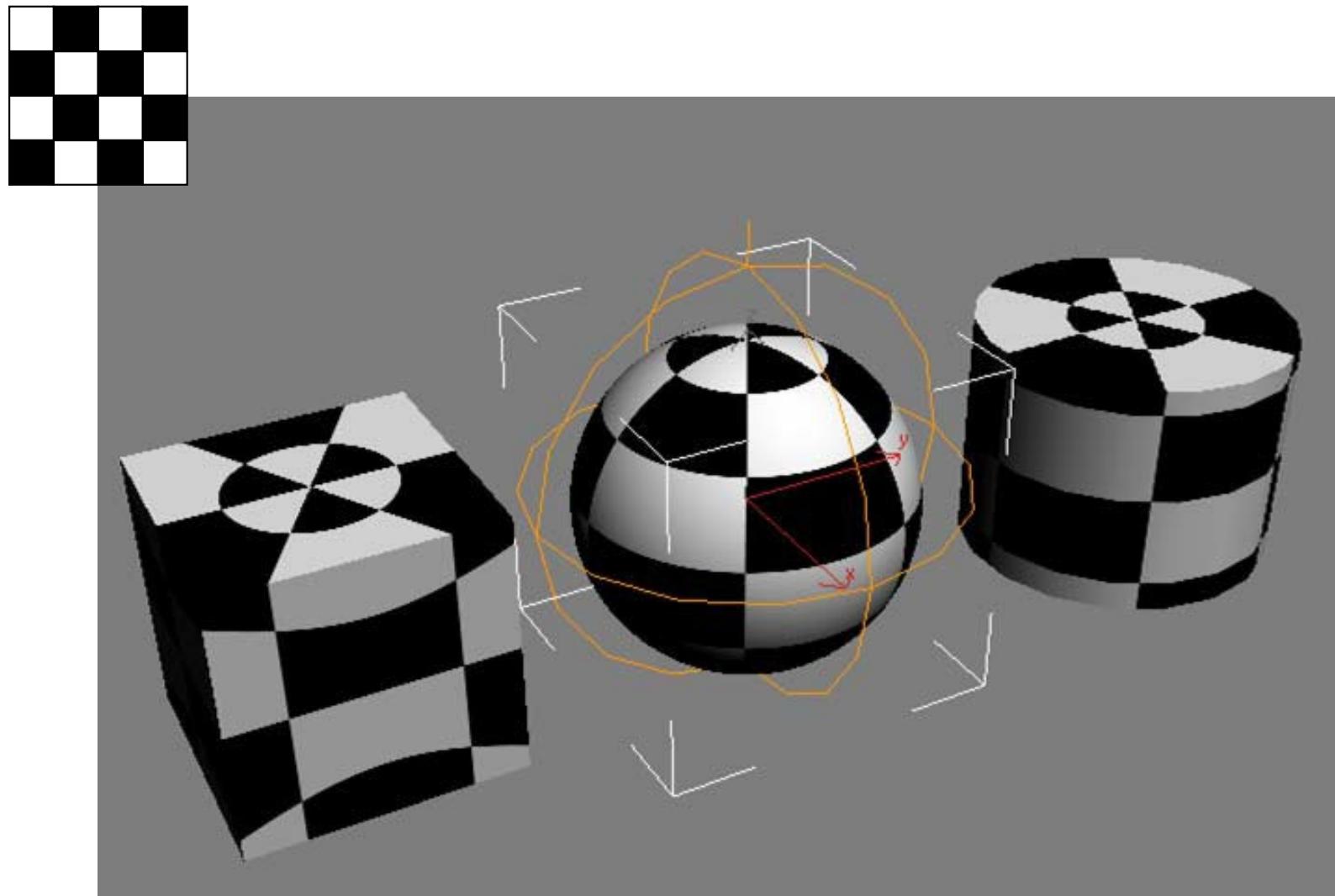
For polygon meshes, there **is no predefined way to flatten the surface**, thus we have to compute it!



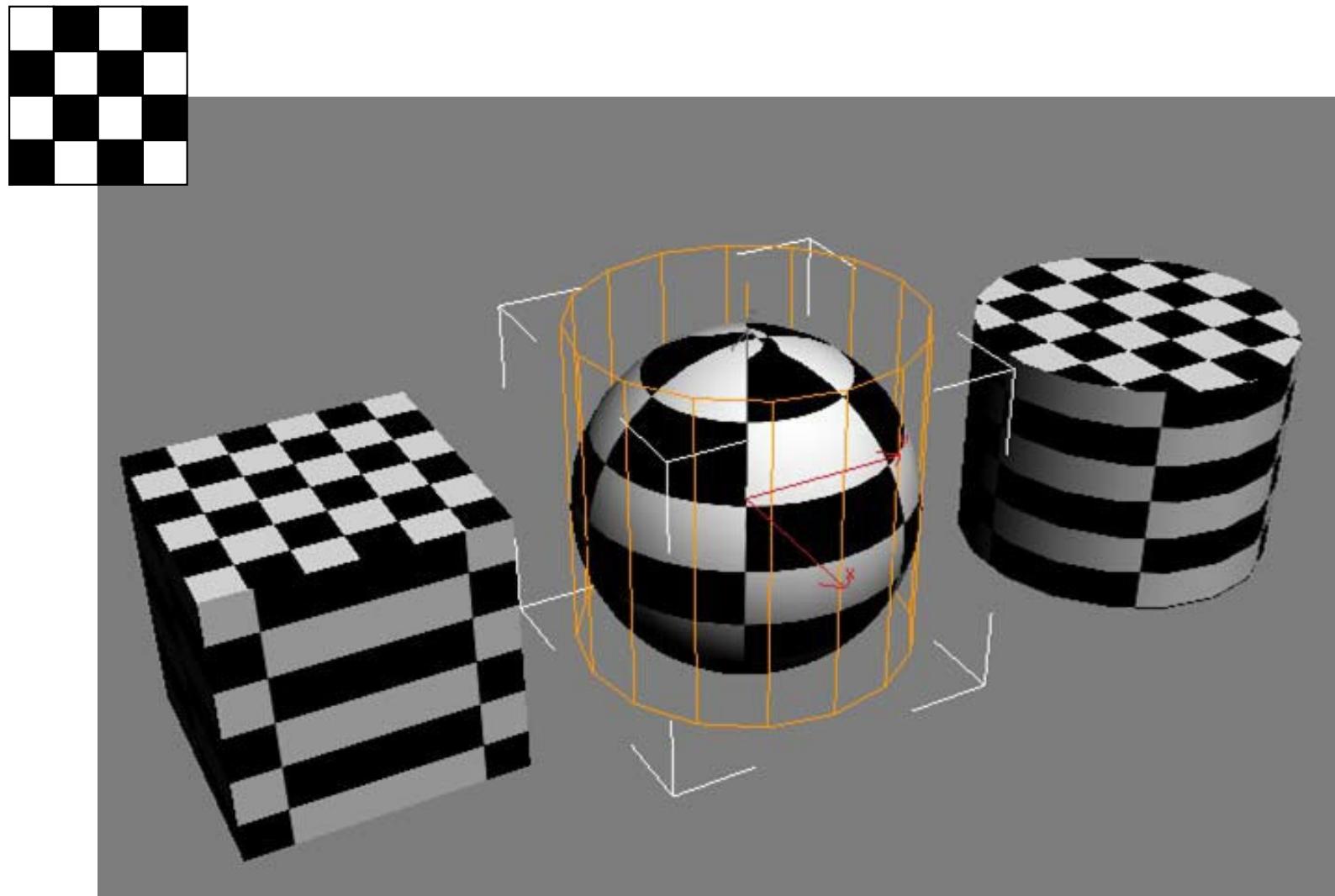
# Simple approach – planar projection



# Simple approach – spherical projection



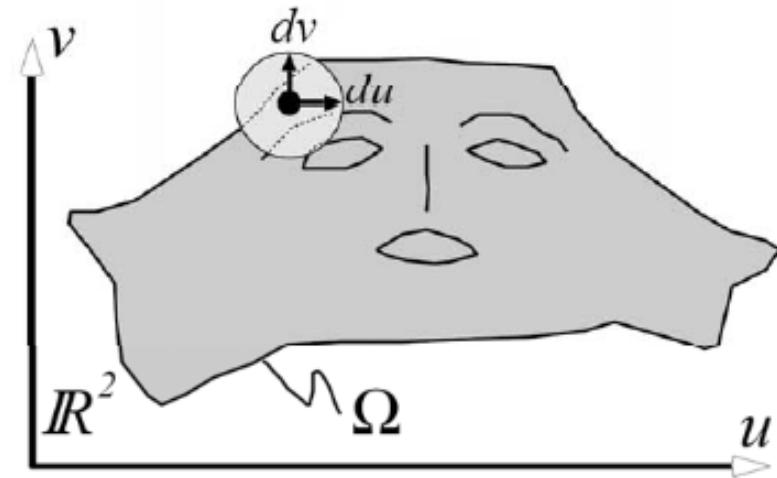
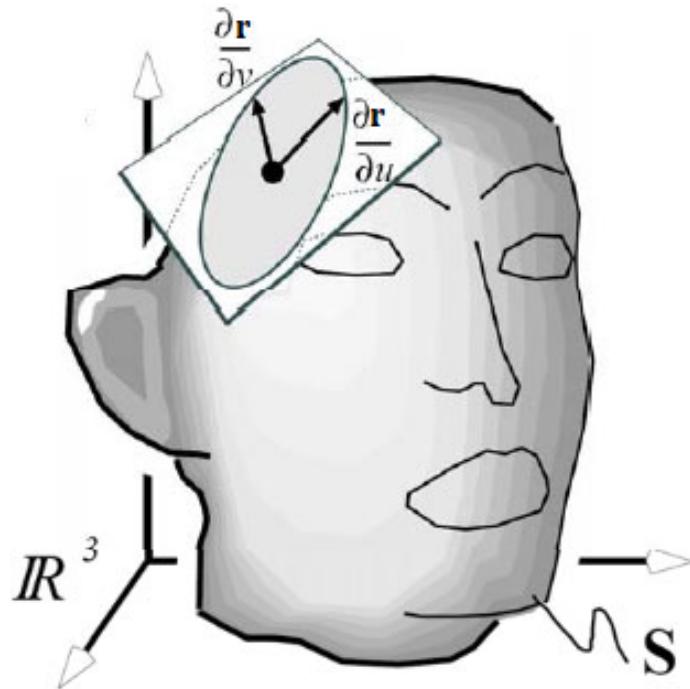
# Simple approach – cylindrical projection



# More complex approaches

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

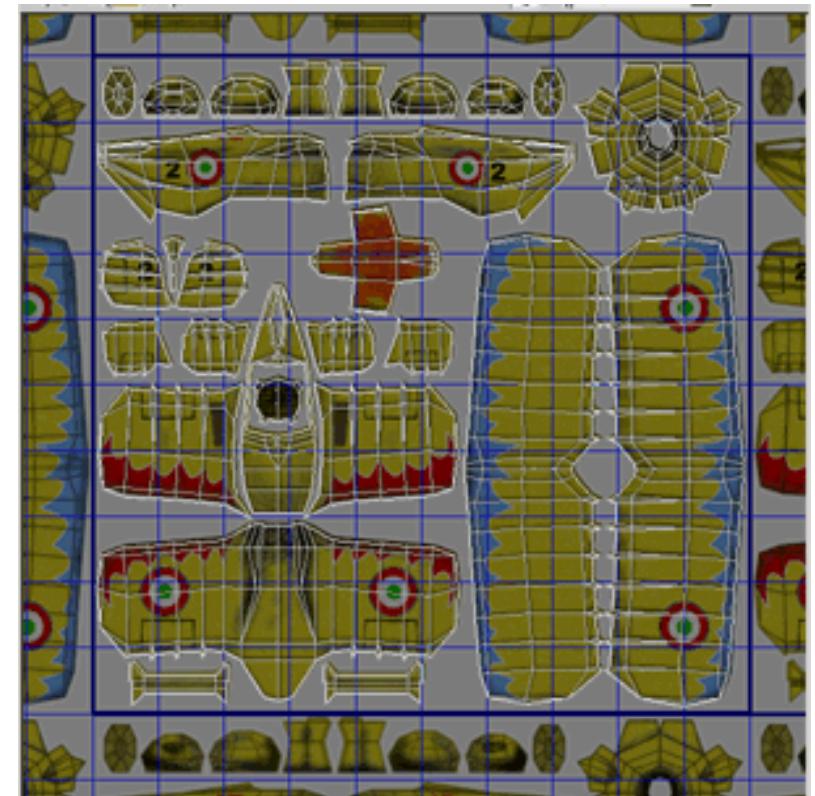
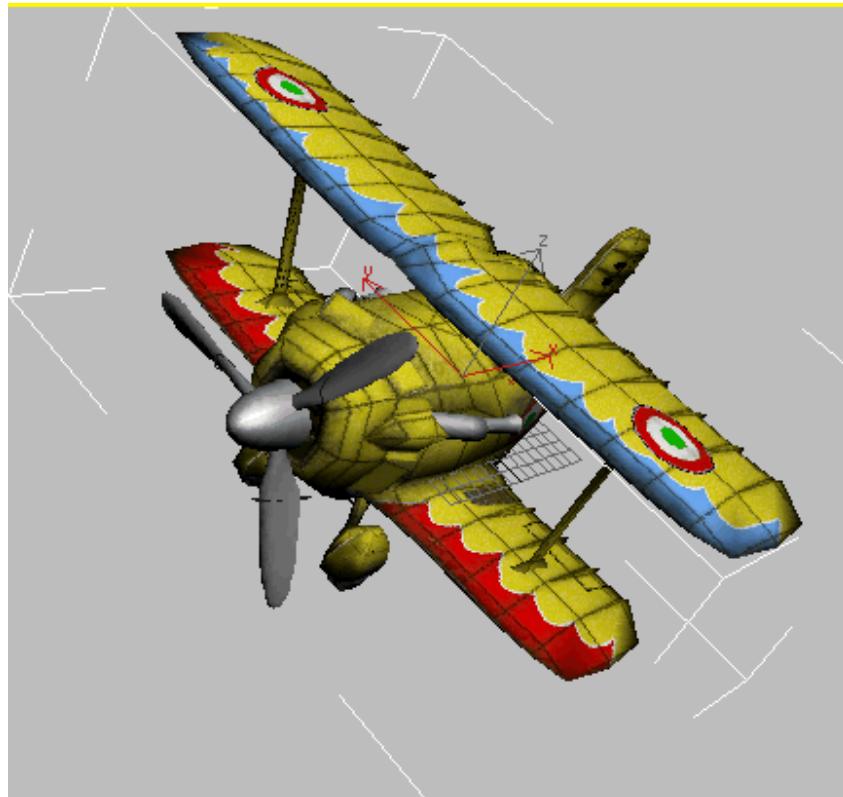
**Energy( $u_1, v_1, \dots, u_p, v_p$ ) = some measure of distortion**



# 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!**



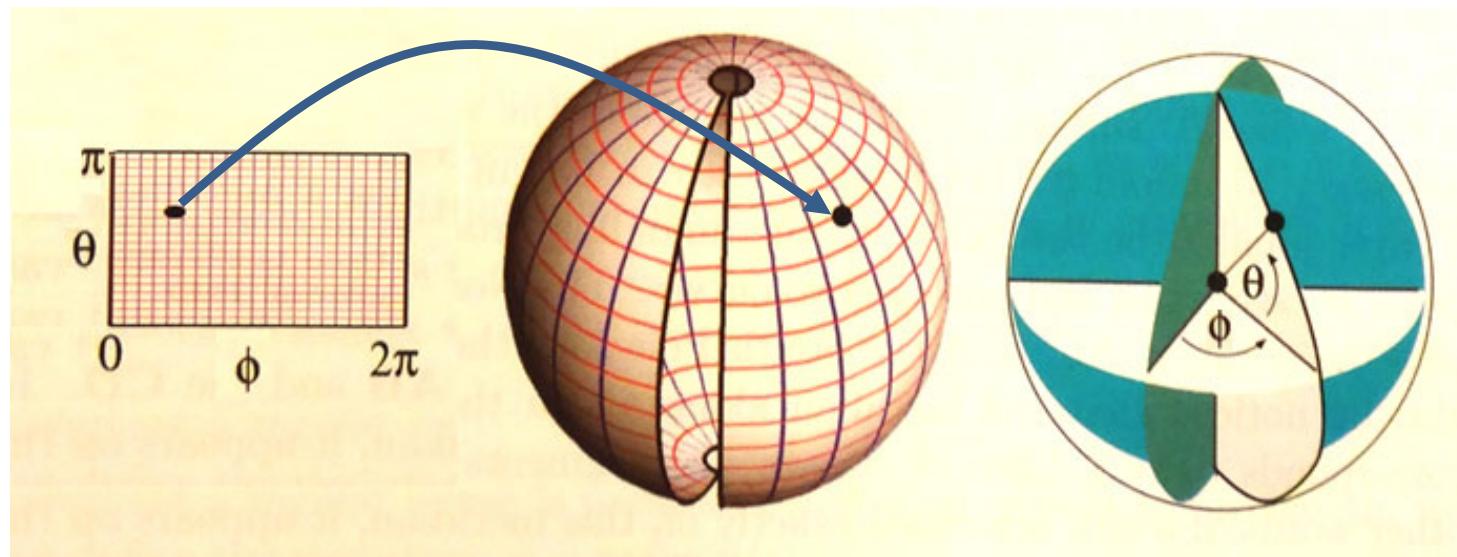
**Texture atlas**

# Shape representations

- Polygon Meshes
- Textures
- **Parametric Surfaces**
- Voxel grids / octrees
- Point clouds
- Gaussian Splats
- Implicit functions

# Parametric Representation of Surfaces

A smooth surface in 3D can be thought of as a map from a 2D set of parameters to 3D space



$$x(\theta, \varphi) = R \sin \theta \cos \varphi$$

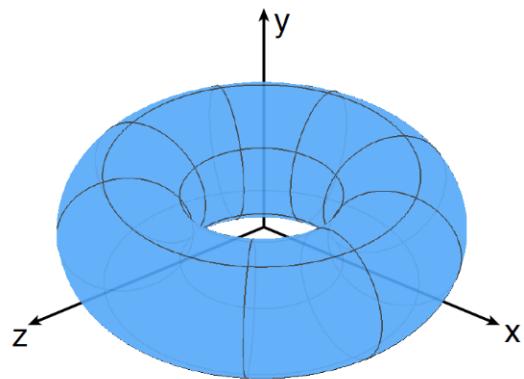
$$y(\theta, \varphi) = R \sin \theta \sin \varphi$$

$$z(\theta, \varphi) = R \cos \theta$$

$$\theta \in (0, \pi), \varphi \in (0, 2\pi)$$

# Parametric surface representation

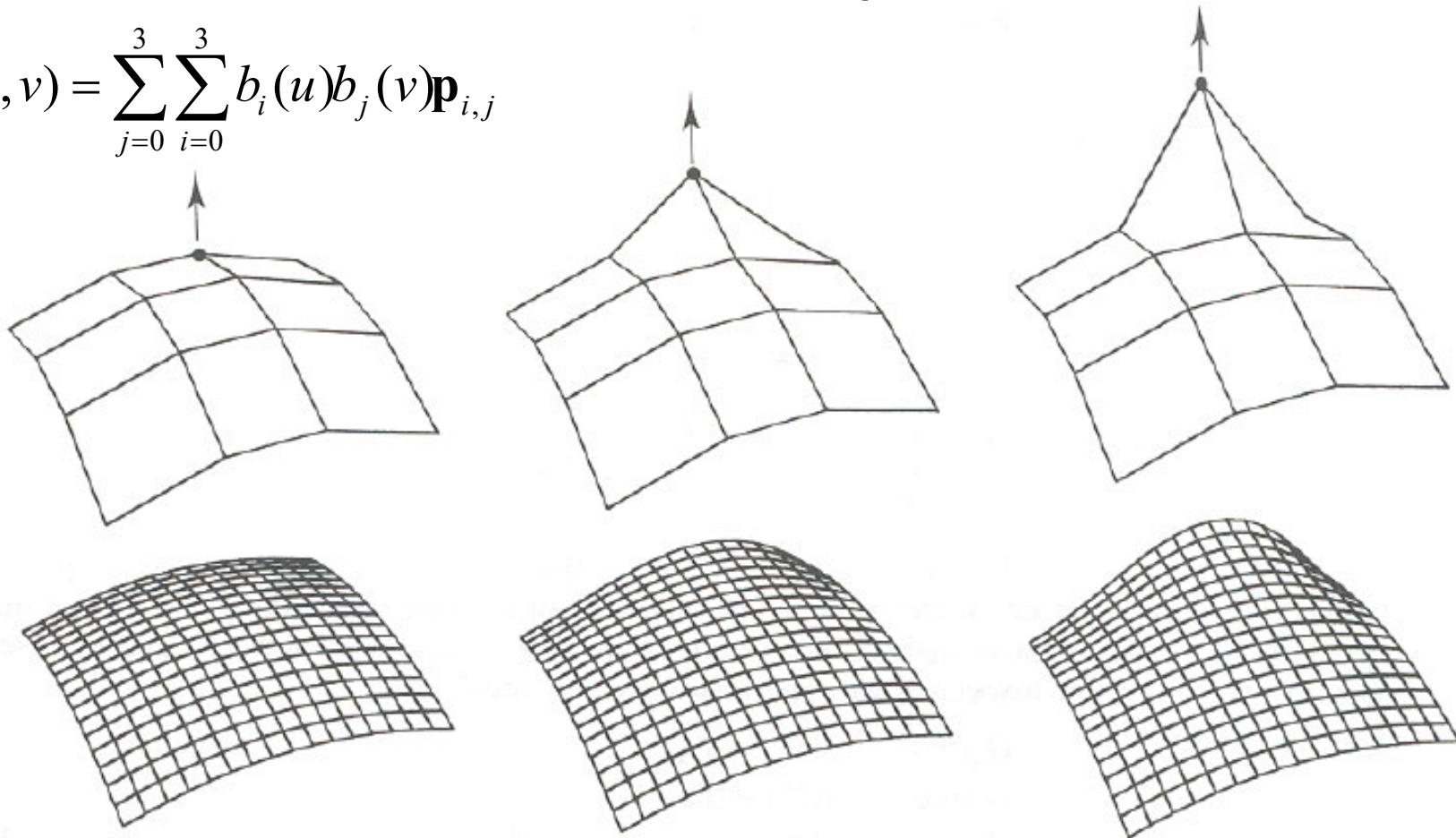
A smooth surface in 3D can be thought of as a map from a 2D set of parameters to 3D space



$$f(u, v) = ((2 + \cos u) \cos v, (2 + \cos u) \sin v, \sin u)$$

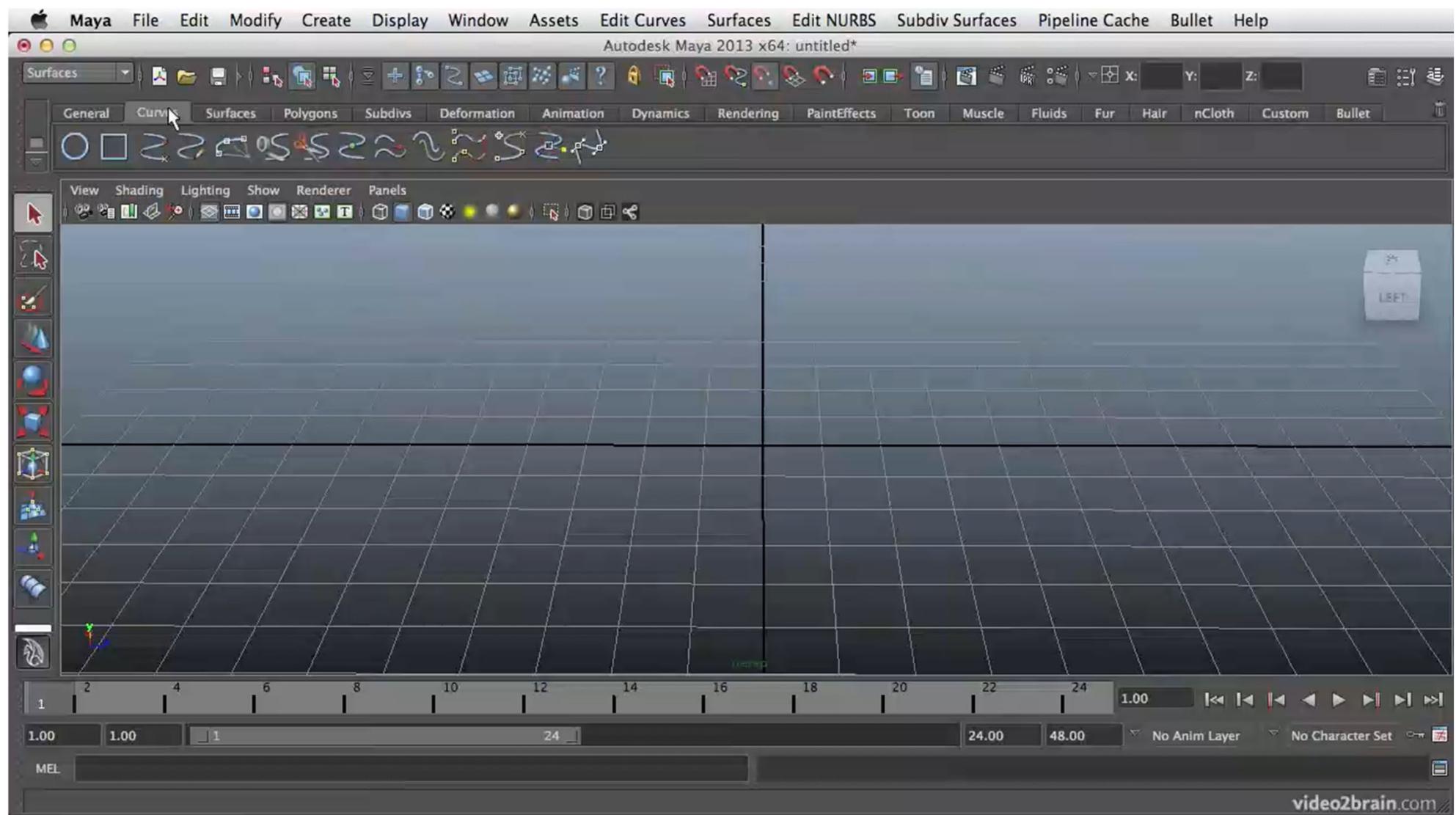
# Parametric patches

$$\mathbf{f}(u, v) = \sum_{j=0}^3 \sum_{i=0}^3 b_i(u) b_j(v) \mathbf{p}_{i,j}$$



- Surface is continuous
- Rendering is plausible
- Surface is defined in terms of (u,v) [no flattening needed]
- Represents specific topologies

# Parametric patches

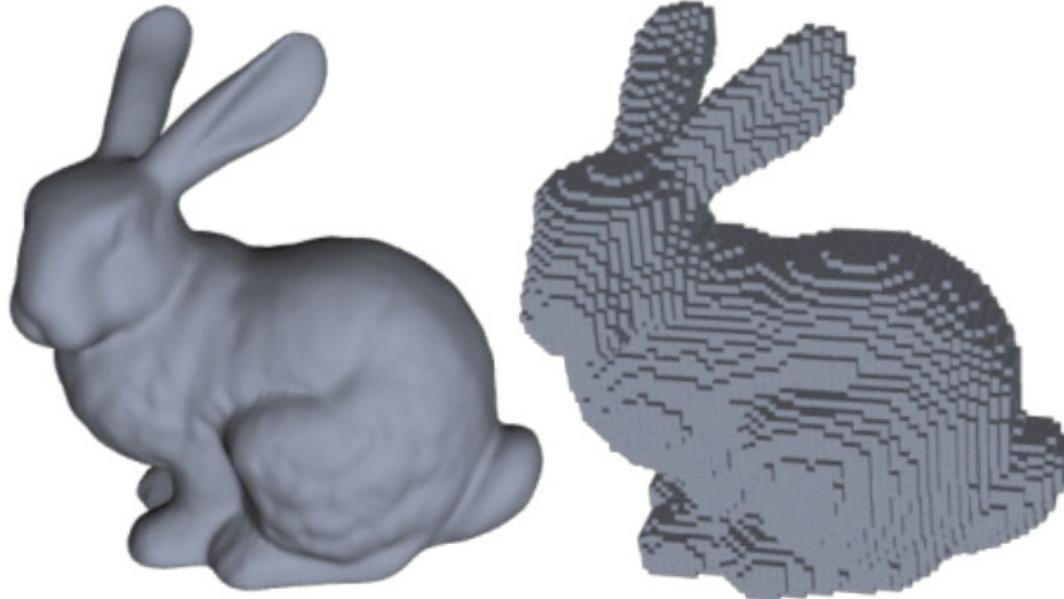


<https://www.youtube.com/watch?v=8iNAnA1mt9M>

# Shape representations

- Polygon Meshes
- Textures
- Parametric Surfaces
- **Voxel grids / octrees**
- Point clouds
- Gaussian Splats
- Implicit functions

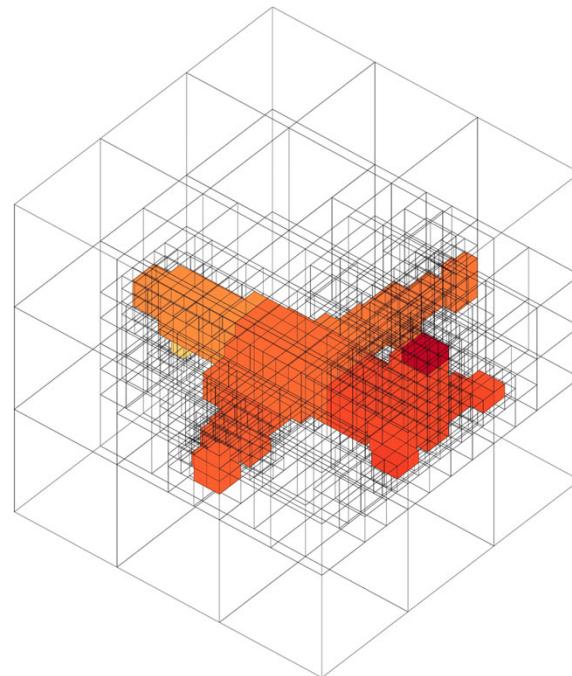
# Voxel representation



$$f(x, y, z) = \begin{cases} 0 & \text{if voxel is outside the shape (unoccupied)} \\ 1 & \text{if voxel is inside the shape / contains surface (occupied)} \end{cases}$$

- **Ordered representation (representation in a NxNxN grid)**
- **Implausible to display in low-resolutions**
- **Expensive to store at high resolutions ( $N^3$  voxels)**

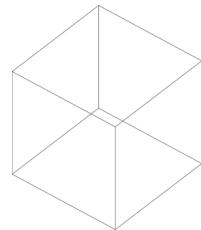
# Octrees



[Slides from Riegler et al. OctNet]

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

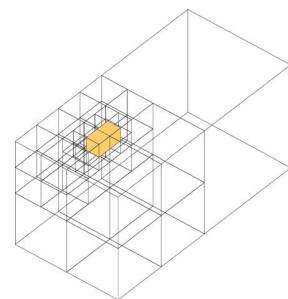
# Octrees



[Slides from Riegler et al. OctNet]

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

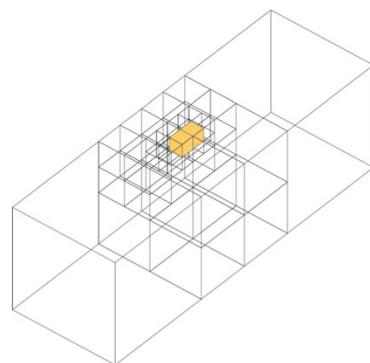
# Octrees



[Slides from Riegler et al. OctNet]

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

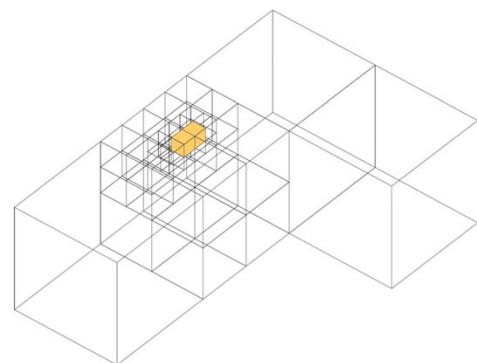
# Octrees



[Slides from Riegler et al. OctNet]

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

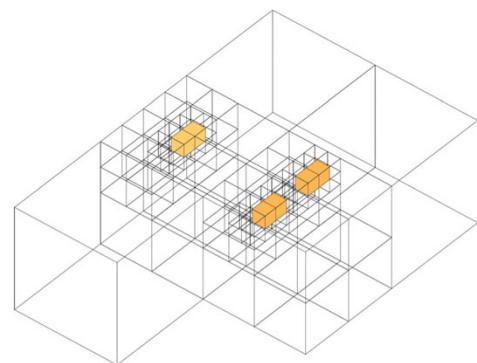
# Octrees



[Slides from Riegler et al. OctNet]

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

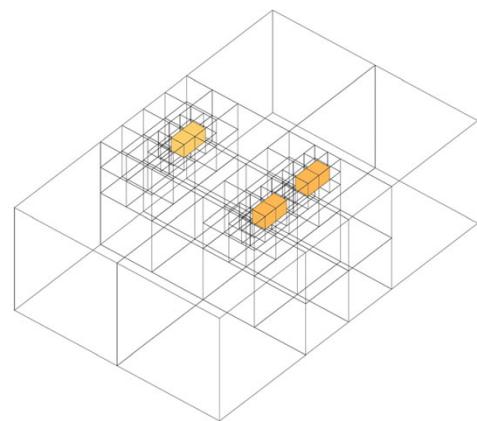
# Octrees



[Slides from Riegler et al. OctNet]

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

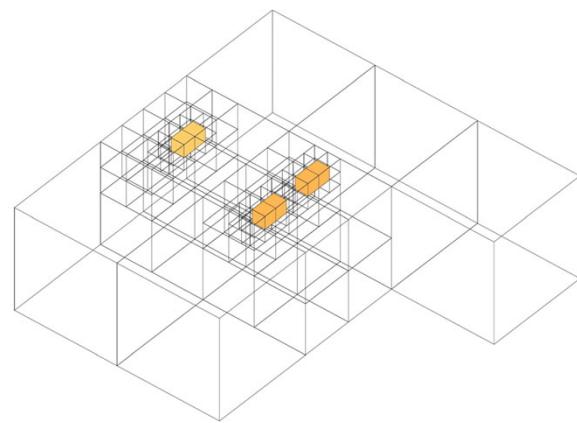
# Octrees



[Slides from Riegler et al. OctNet]

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

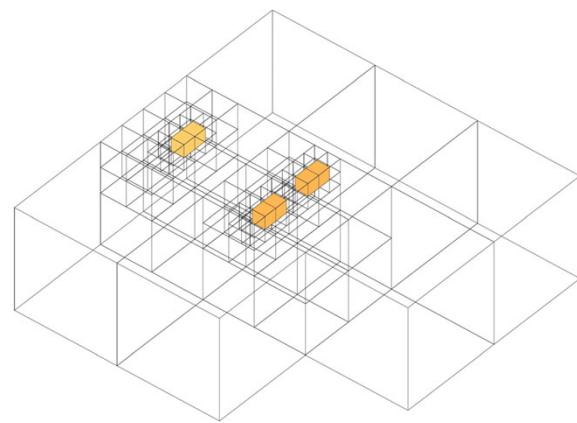
# Octrees



[Slides from Riegler et al. OctNet]

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

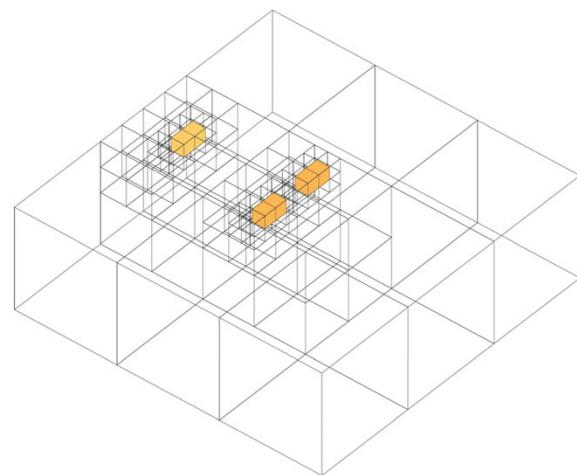
# Octrees



[Slides from Riegler et al. OctNet]

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

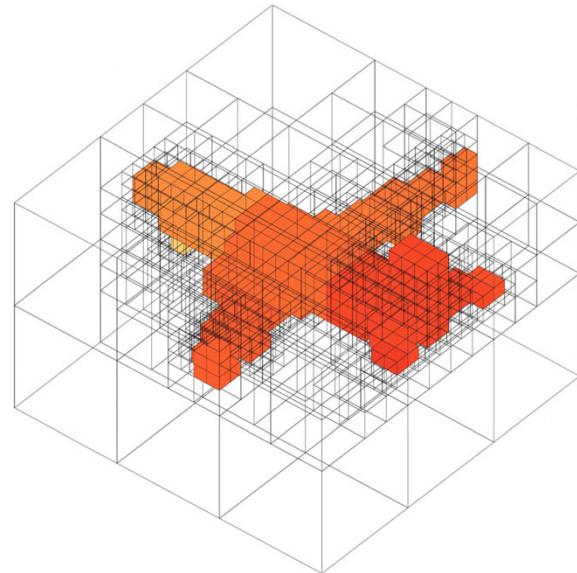
# Octrees



[Slides from Riegler et al. OctNet]

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

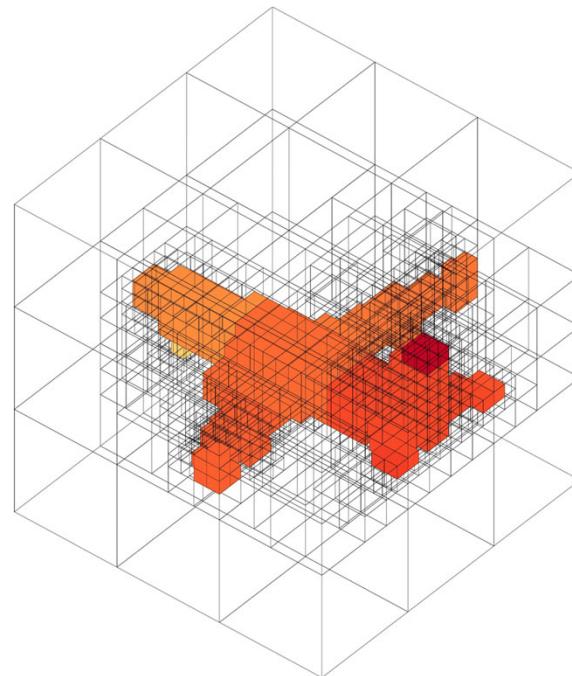
# Octrees



[Slides from Riegler et al. OctNet]

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

# Octrees

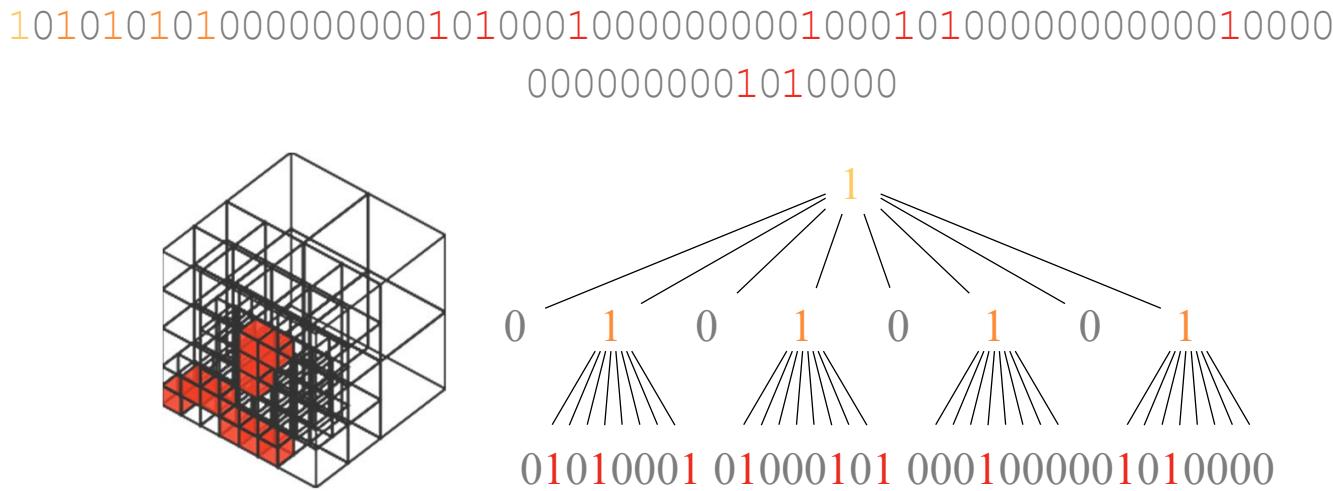


[Slides from Riegler et al. OctNet]

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

# Octrees

Octrees are efficiently encoded as bit-strings



- More accurate representation than voxels
  - Compact to store
  - Still voxelized look

[Slides from Riegler et al. OctNet]

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

# Shape representations

- Polygon Meshes
- Textures
- Parametric Surfaces
- Voxel grids / octrees
- **Point clouds**
- Gaussian Splats
- Implicit functions

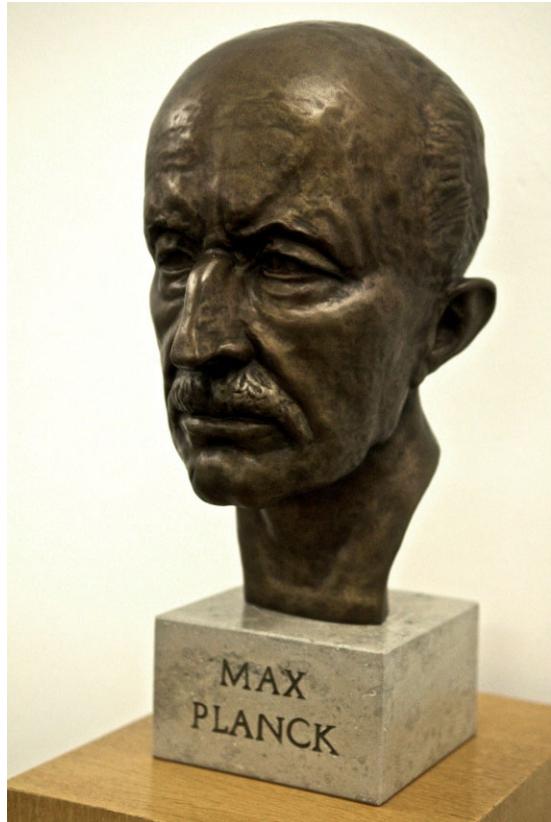
# Point Clouds

A collection of points without connectivity information...  
usually the raw output of a scan



# 3D scanning

Acquire geometry of a physical object



scanner



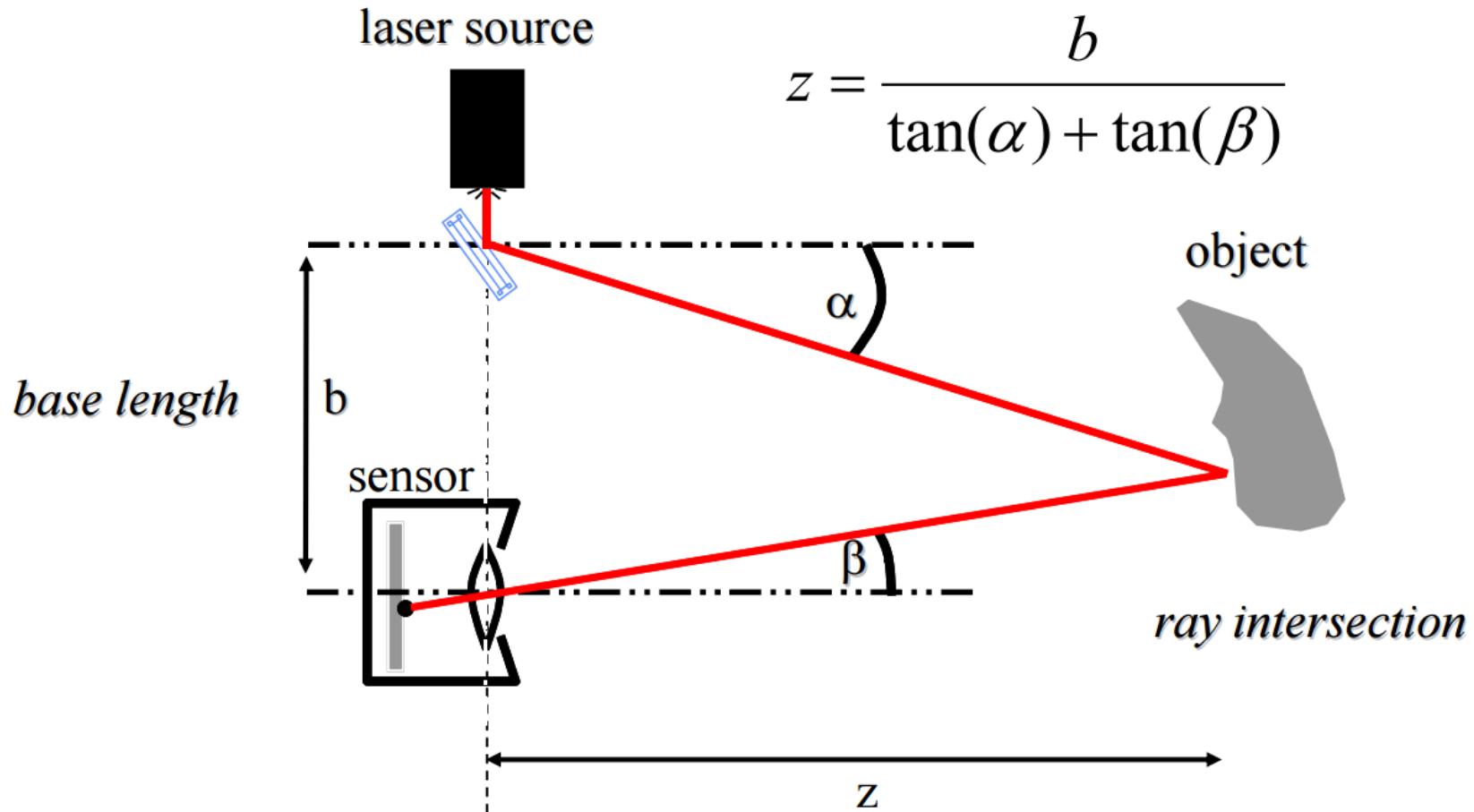
Physical object

Raw scans



# Basic Idea (laser scanning)

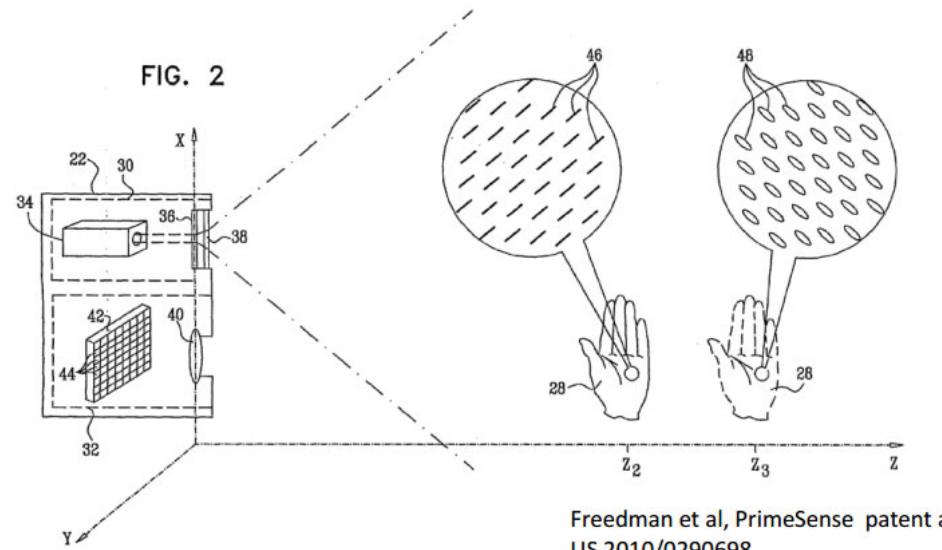
Project light onto an object, analyze reflected light



# Basic Idea (Structured light scanning)



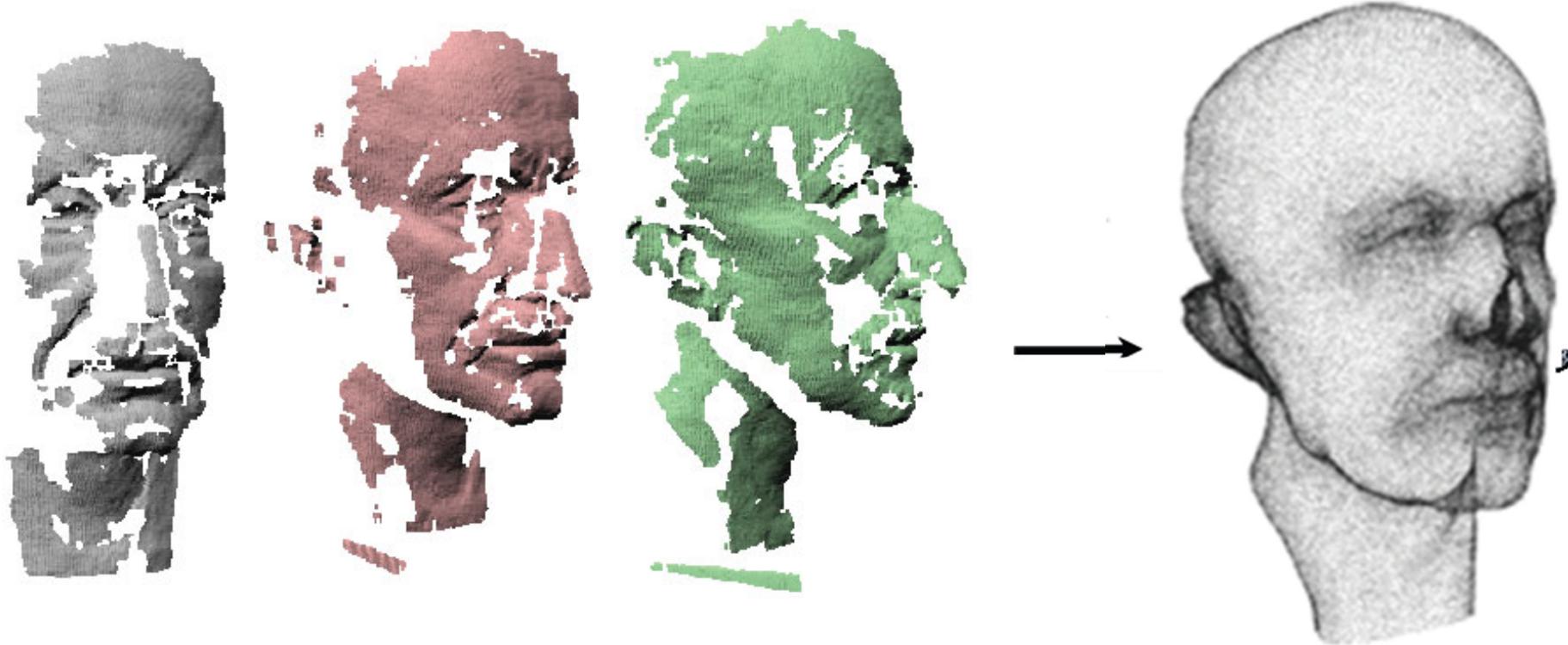
Kinect projects light patterns through a special lens ("astigmatic") with different focal length in x- & y-directions.



Freedman et al, PrimeSense patent application  
US 2010/0290698

A projected circle then becomes an ellipse. Measures depth based on deformation

# Registration



Set of raw scans

Point cloud after registration

- **Compact presentation (store only points)**
- **Point cloud rendering does not look plausible – not continuous representation**

# Shape representations

- Polygon Meshes
- Textures
- Parametric Surfaces
- Voxel grids / octrees
- Point clouds
- **Gaussian Splats**
- Implicit functions

# GSplats

[https://www.youtube.com/watch?v=Tnij\\_xHEnXc](https://www.youtube.com/watch?v=Tnij_xHEnXc)



- **Much more faithful representation than points (yet with tons of splats)**

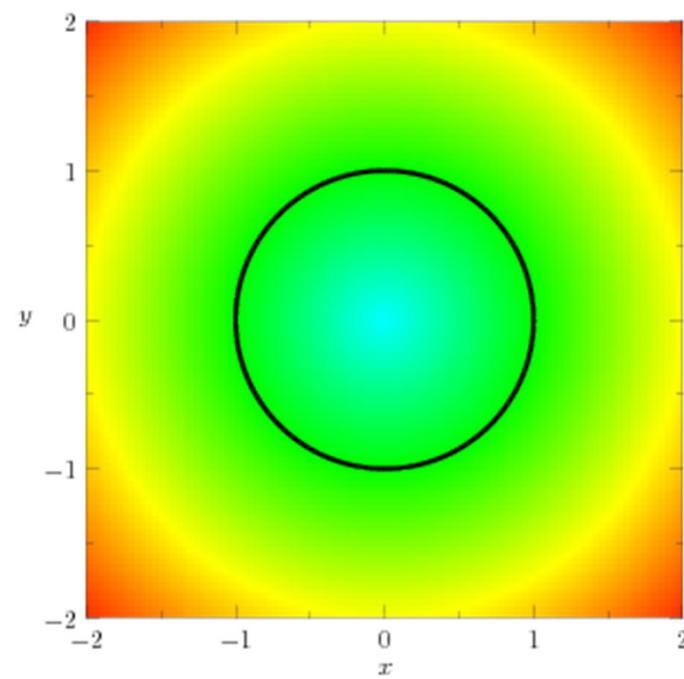
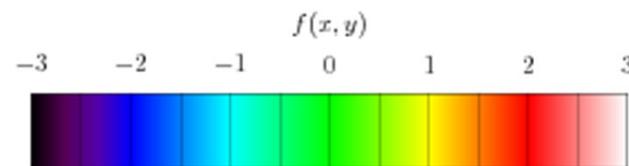
Kopanas et al. 3D Gaussian Splatting for Real-Time Radiance Field Rendering

# Shape representations

- Polygon Meshes
- Textures
- Parametric Surfaces
- Voxel grids / octrees
- Point clouds
- Gaussian Splats
- **Implicit functions**

# Example: Implicit curve

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



# Example: Implicit surfaces

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

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



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



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



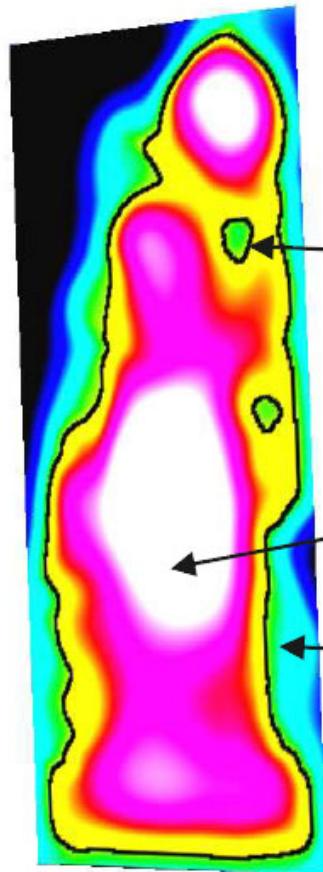
$$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 =$$

$$x^2 z^3 + \frac{9y^2 z^3}{80}$$

# 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*



**Implicit Surface:**

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

**Inside:**

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

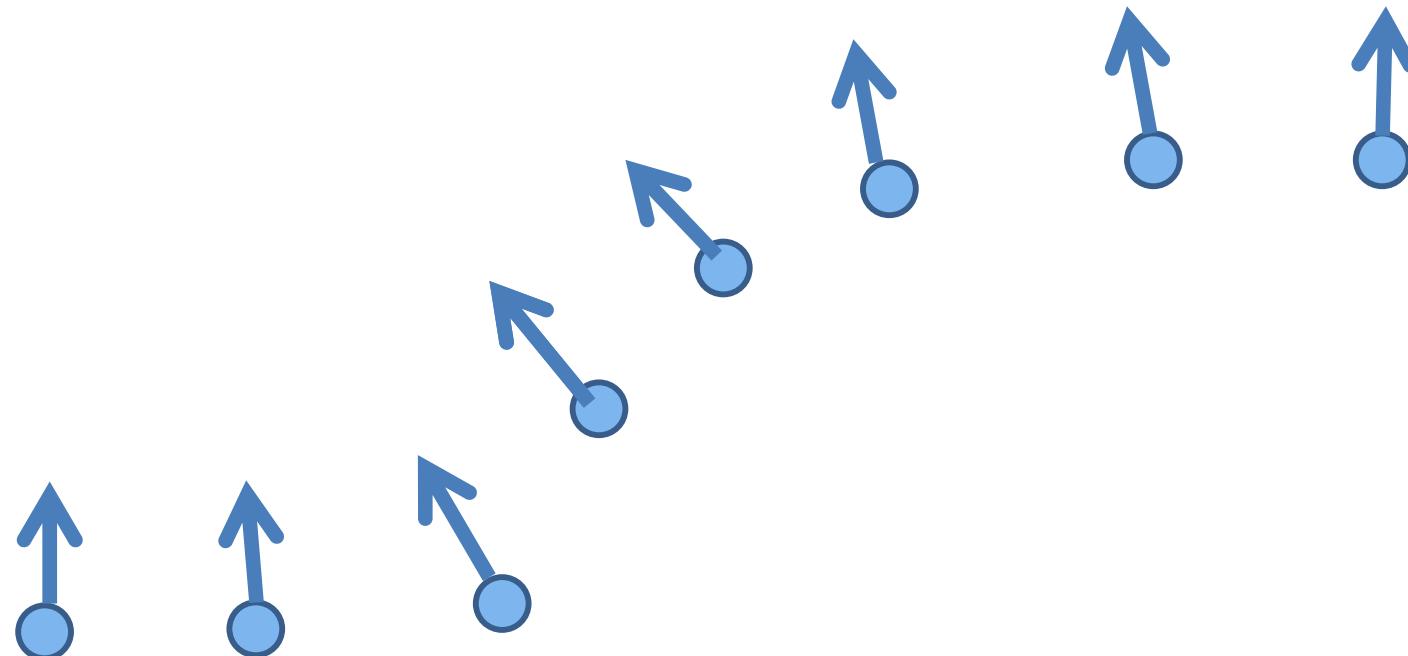
**Outside:**

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



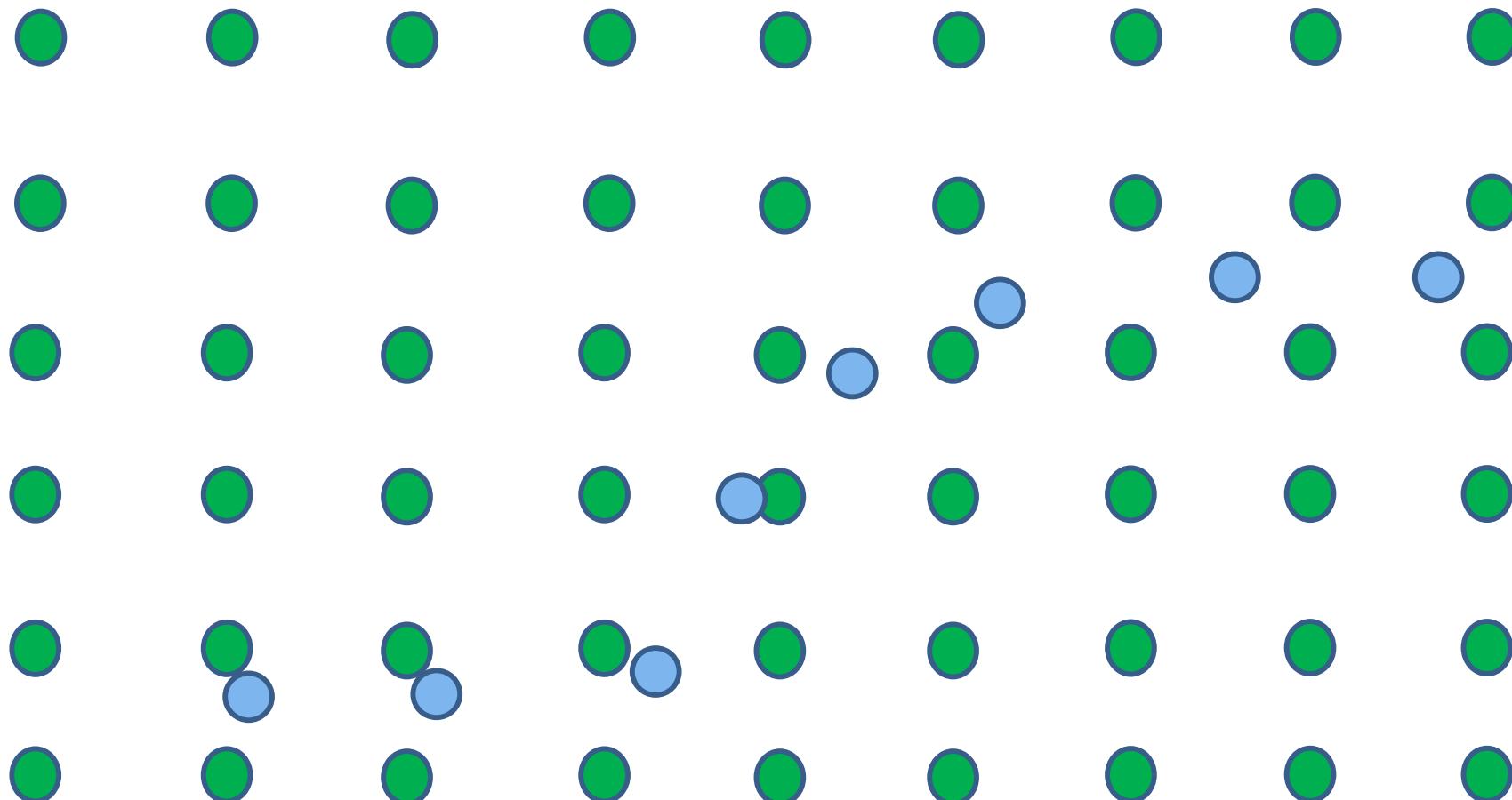
# How to define a shape implicit from a point cloud?

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



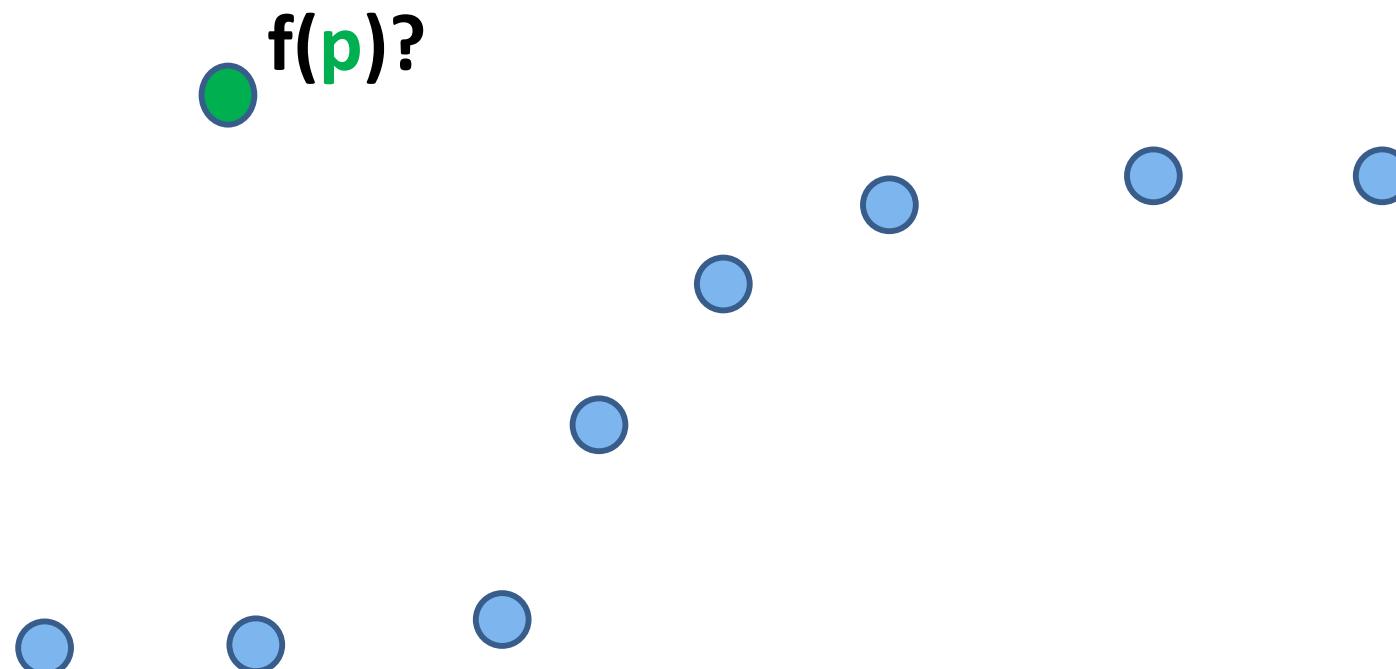
# How to define a shape implicit from a point cloud?

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



# How to define a shape implicit from a point cloud?

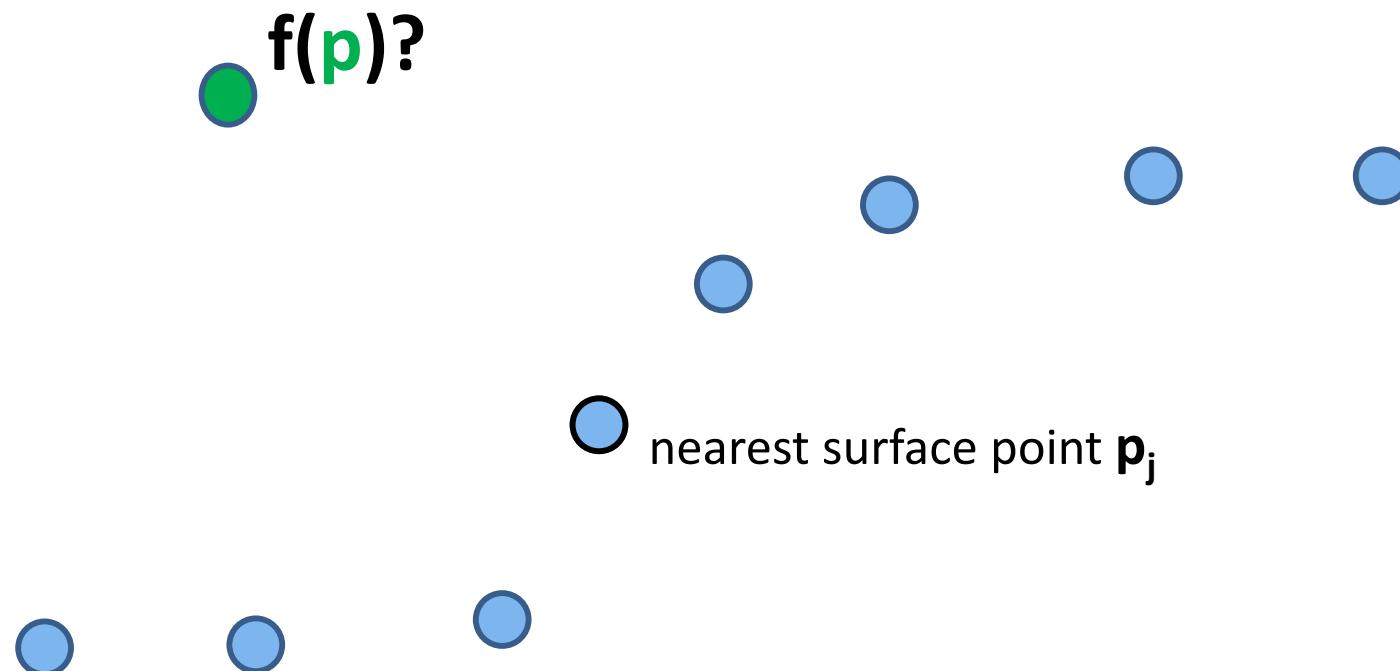
Let's focus on computing the **signed** distance  $f(x,y,z)$  for a single grid point  $\mathbf{p}=\{x,y,z\}$ :



# How to define a shape implicit from a point cloud?

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

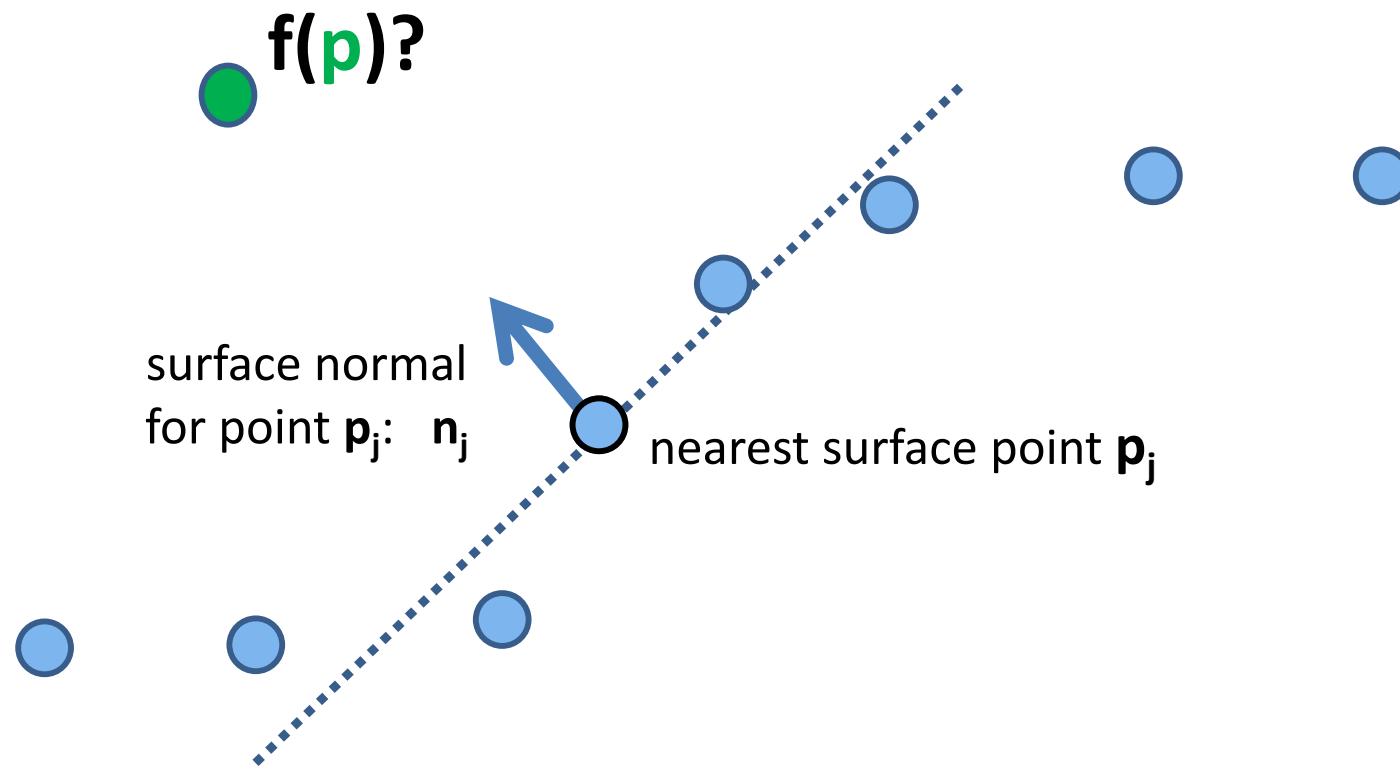
Then compute distance of the **grid point** to the tangent plane of that surface point



# How to define a shape implicit from a point cloud?

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

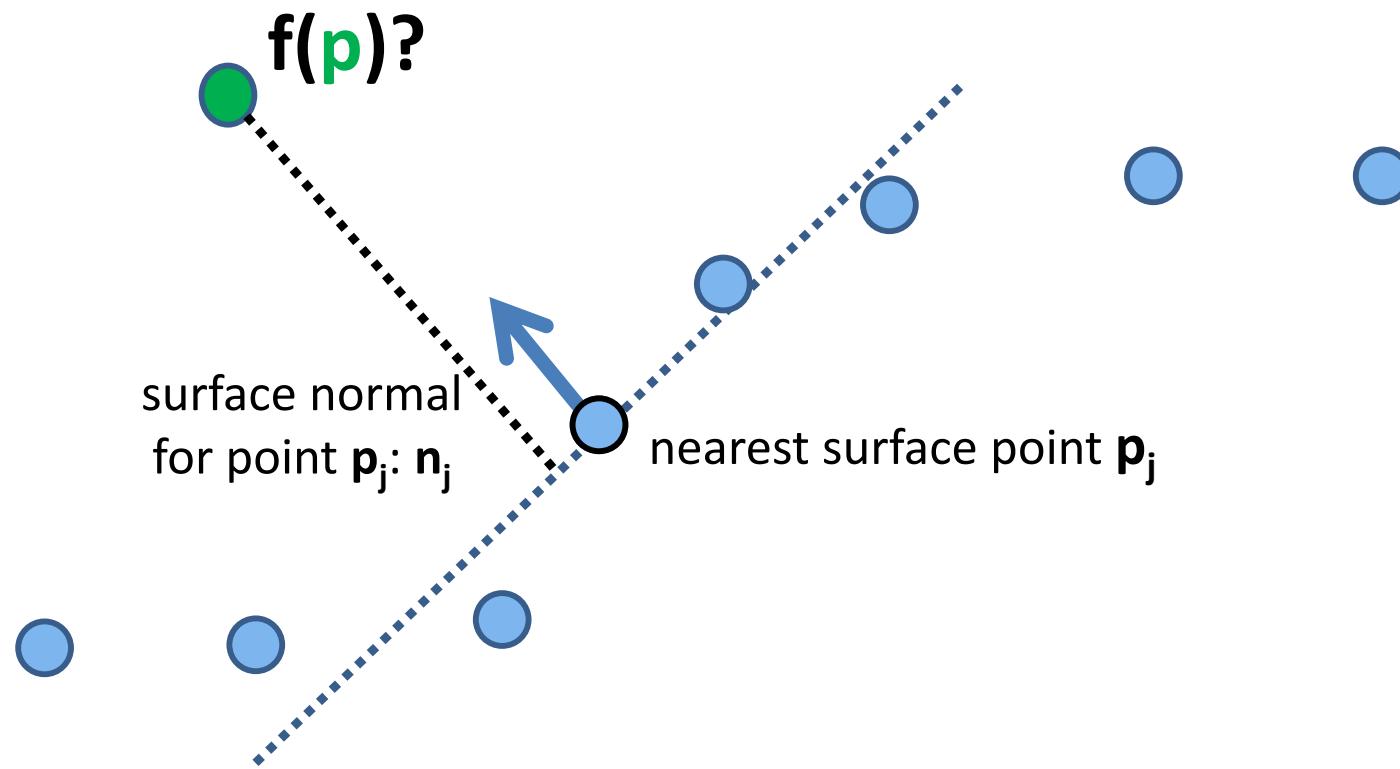
Then compute distance of the **grid point** to the tangent plane of that surface point



# How to define a shape implicit from a point cloud?

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

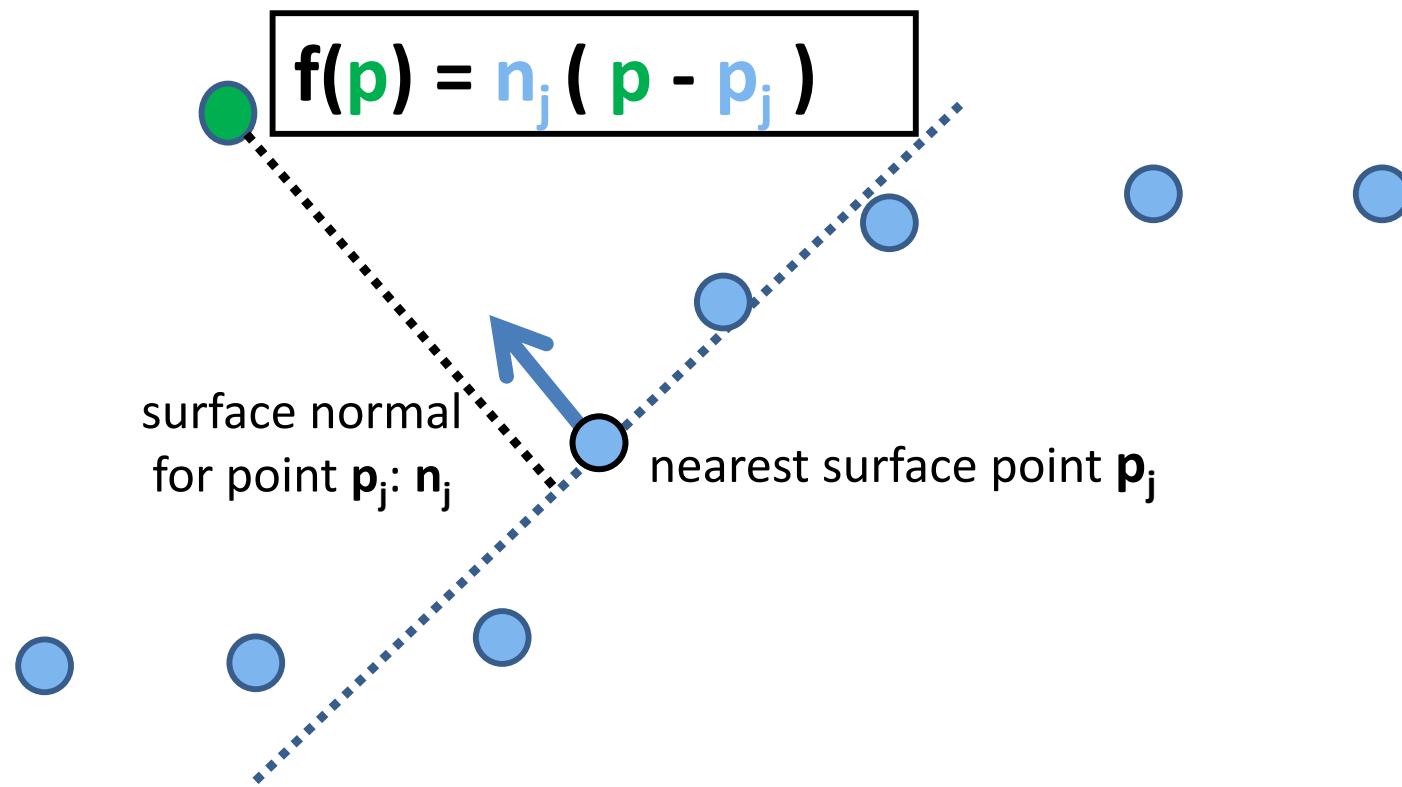
Then compute distance of the **grid point** to the tangent plane of that surface point



# How to define a shape implicit from a point cloud?

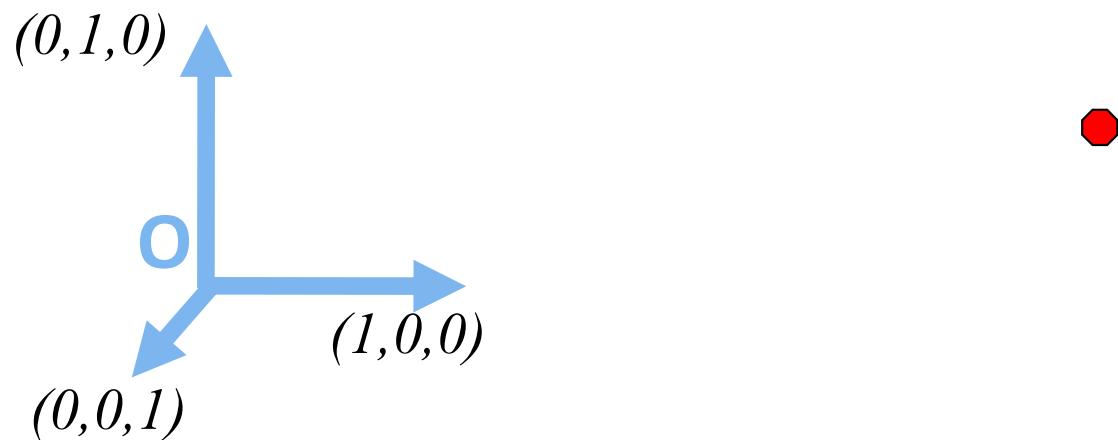
One way is to find the nearest surface point to the **grid point**.

Then compute distance of the **grid point** to the tangent plane of that surface point... a dot product of two vectors



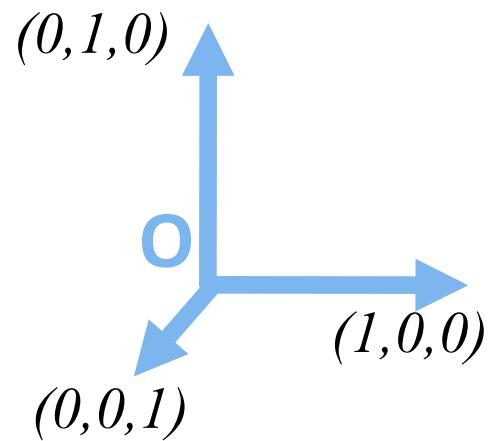
# Eh? Vector? Dot Product?

A **point** in 2D or 3D space is represented by a **vector**:



# Math review - vectors

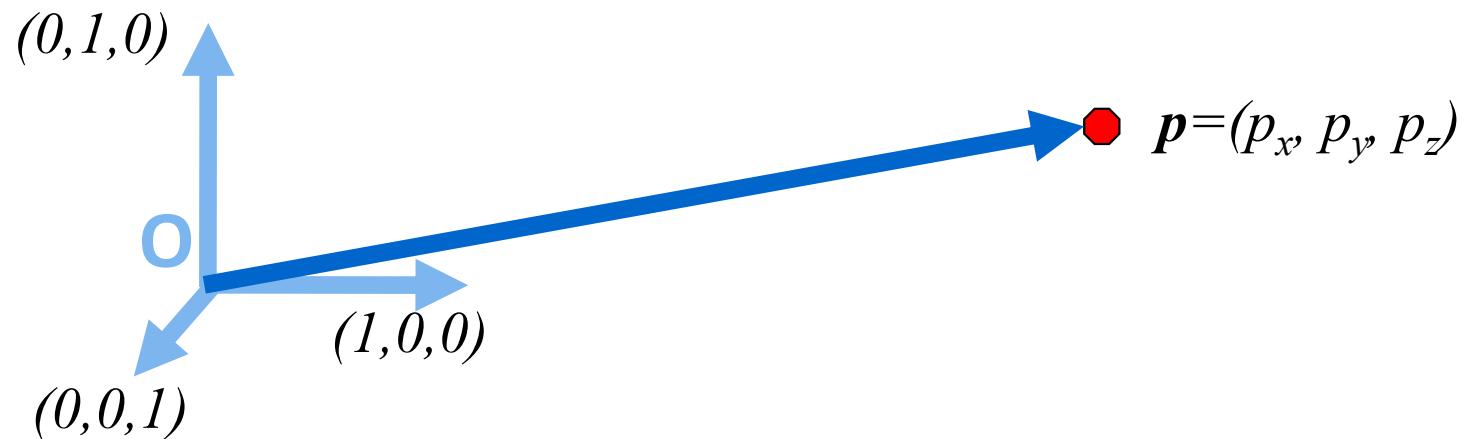
A **point** in 2D or 3D space is represented by a **vector**:



origin of world  
coordinate system  
at  $(0, 0, 0)$

# Math review - vectors

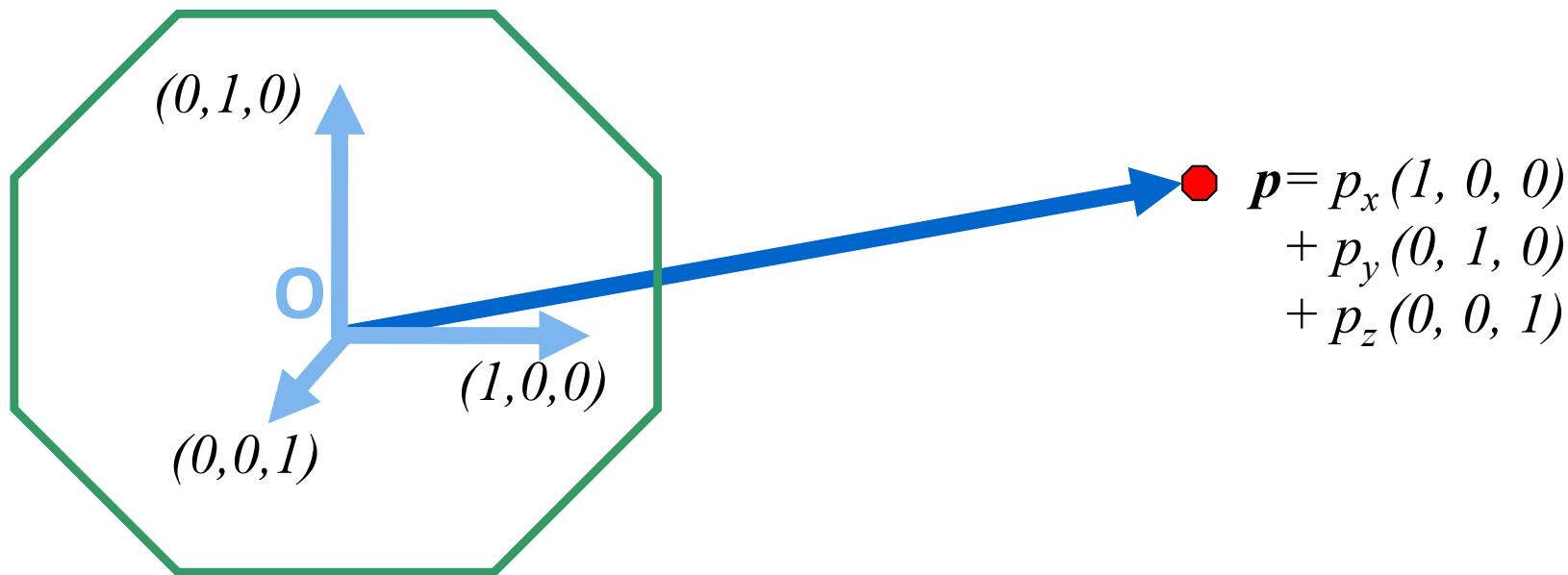
A **point** in 2D or 3D space is represented by a **vector**:



origin of world  
coordinate system  
at  $(0, 0, 0)$

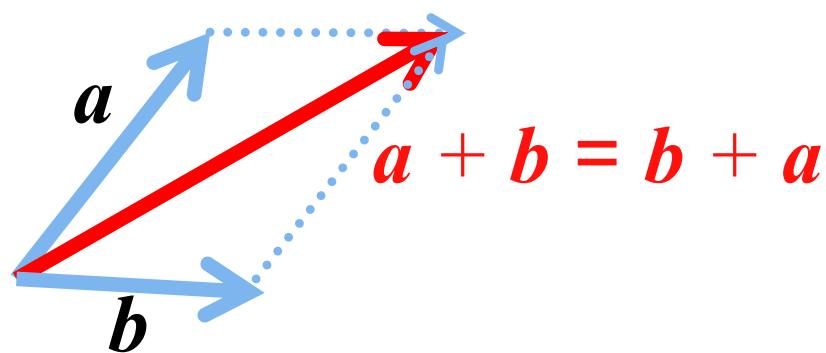
# Cartesian coordinate system

The **point** is essentially expressed as a linear combination of **three orthogonal unit vectors**



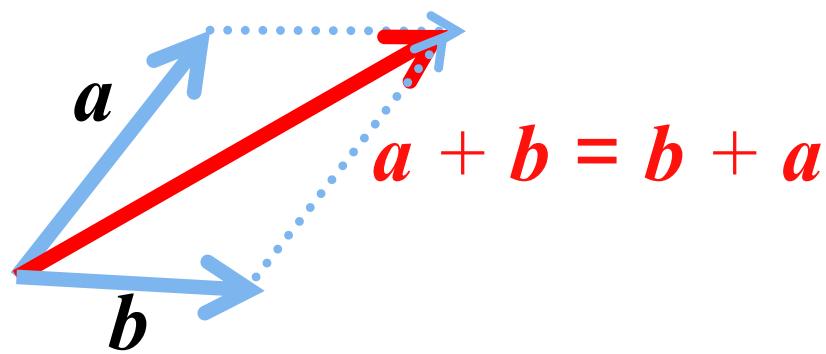
# What can we do with vectors?

Add two vectors:

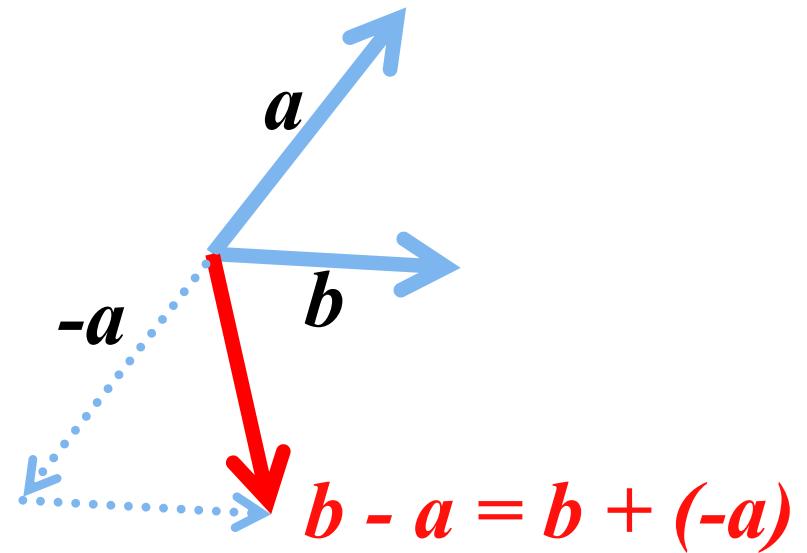


# What can we do with vectors?

Add two vectors:

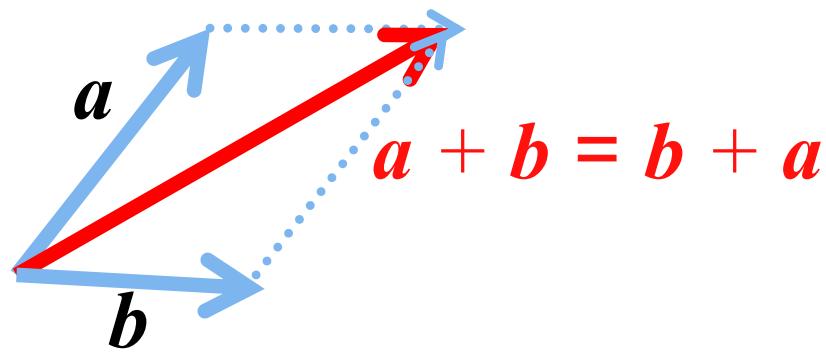


Subtract two vectors:

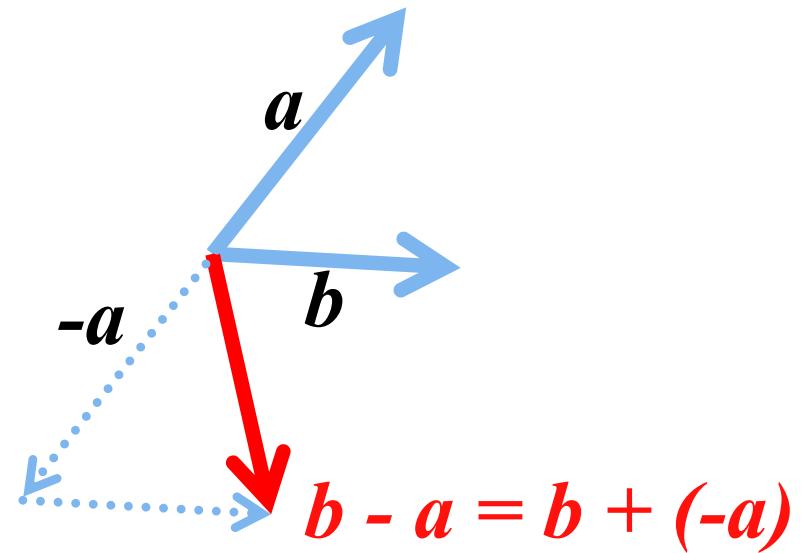


# What can we do with vectors?

Add two vectors:



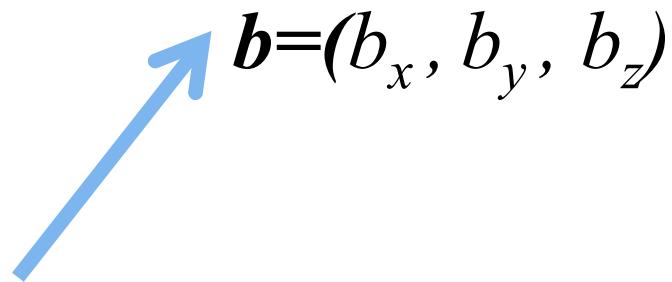
Subtract two vectors:



$$\mathbf{b} + \mathbf{a} = (b_x + a_x, b_y + a_y, b_z + a_z)$$

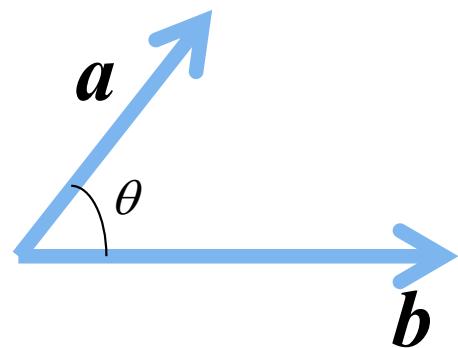
$$\mathbf{b} - \mathbf{a} = (b_x - a_x, b_y - a_y, b_z - a_z)$$

# Length of the vector



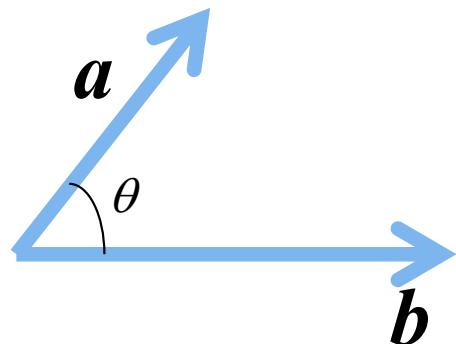
$$\| \mathbf{b} \| = \sqrt{{b_x}^2 + {b_y}^2 + {b_z}^2}$$

# Dot product



$$\mathbf{a} \cdot \mathbf{b} = \| \mathbf{a} \| \| \mathbf{b} \| \cos \theta$$

# Dot product

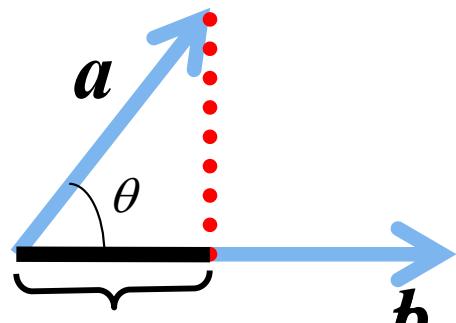


$$\mathbf{a} \cdot \mathbf{b} = \| \mathbf{a} \| \| \mathbf{b} \| \cos \theta$$

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z$$

# Dot product - Geometric interpretation

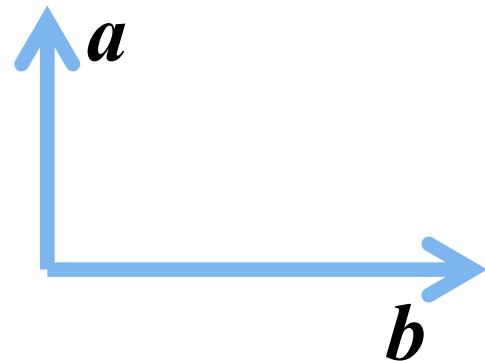
If the vector  $b$  is a unit vector ( $\|b\| = 1$ ):



$$\mathbf{a} \cdot \mathbf{b} = \| \mathbf{a} \| \cos \theta$$

# Orthogonal vectors

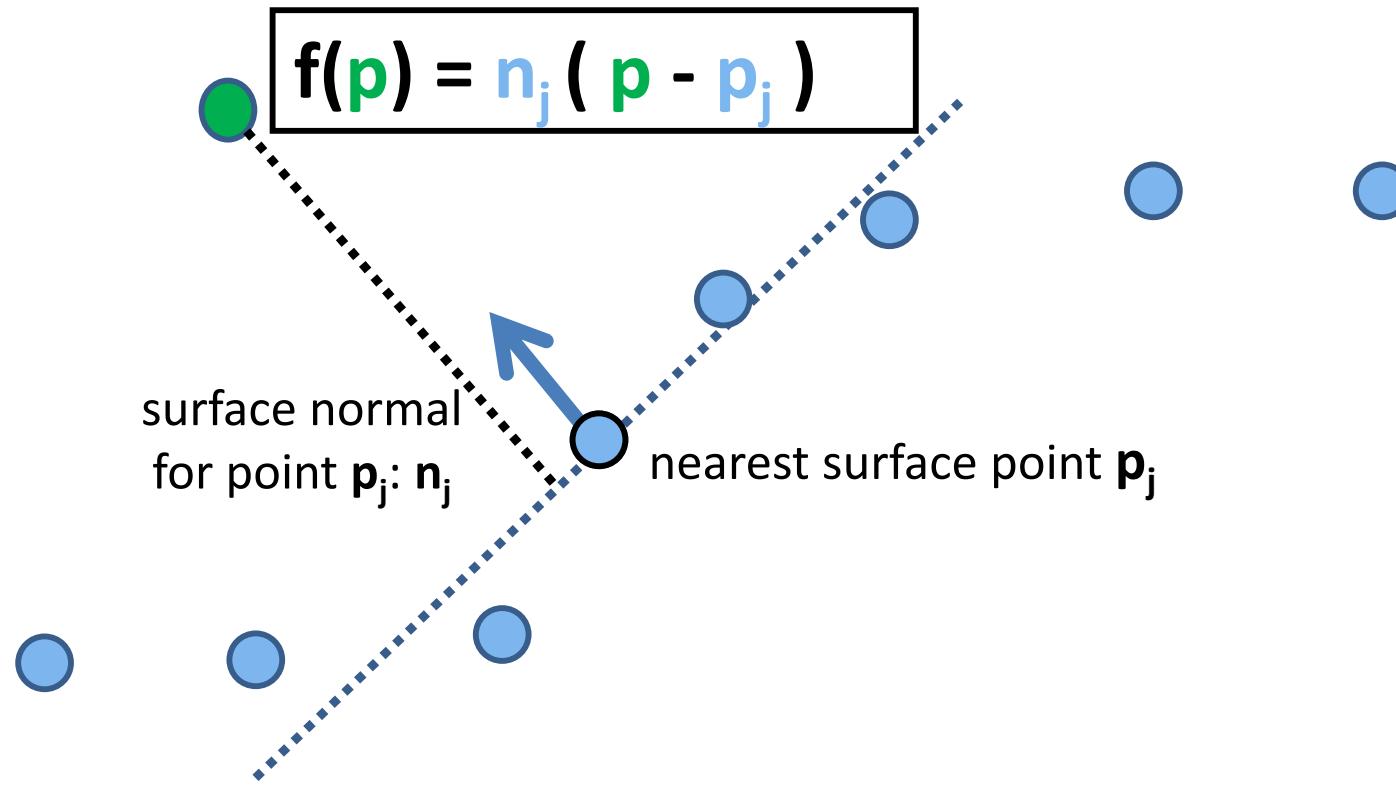
Two (non-zero) vectors are orthogonal if and only if their dot product is **zero**.



$$\mathbf{a} \cdot \mathbf{b} = 0$$

# How to define implicit surface for reconstruction?

Do the dot products for the whole grid!



# How to define a shape implicit from a point cloud?

Do the dot products for the whole grid!

2.3

1.7

0.9

0.2

1.2

0.4

0.1

-0.8

0.3

-0.5

-0.7

-1.4

0.2

-0.9

-1.7

-2.5

# How to define a shape implicit from a point cloud?

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

2.3 • 1.7 • 0.9 • 0.2 •

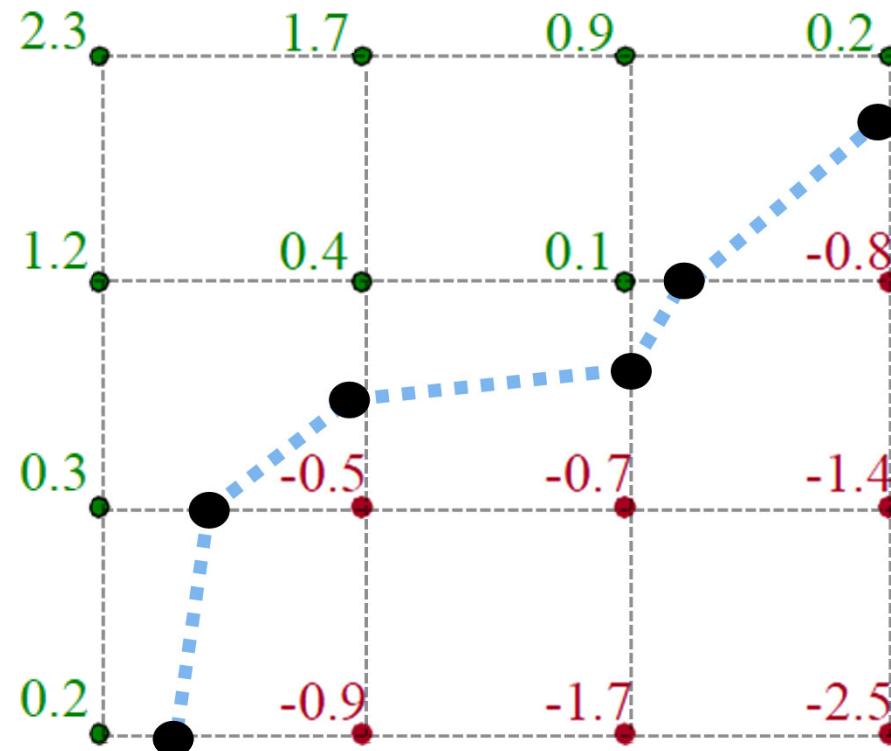
1.2 • 0.4 • 0.1 • -0.8 •

0.3 • -0.5 • -0.7 • -1.4 •

0.2 • -0.9 • -1.7 • -2.5 •

# Isosurface extraction

Linear interpolation to get a piecewise linear approximation of the surface



# Simplest interpolation: linear

Introduce a free parameter  $u \in [0,1]$

**Think of  $u$  as time** -- imagine we are walking along the line start at  $p_0$  at  $u=0$  and arriving at  $p_1$  at  $u=1$ .

$$\mathbf{f}(u) = (1 - u) \cdot \mathbf{f}(u_0) + u \cdot \mathbf{f}(u_1)$$

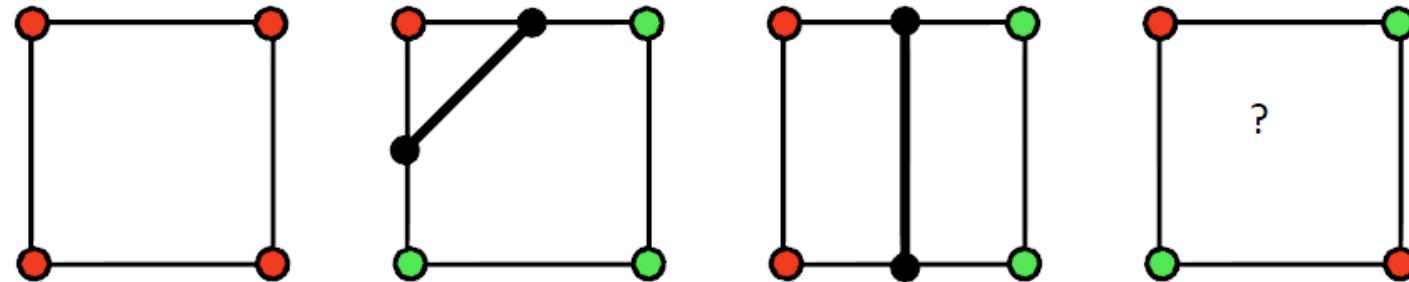


At time  $u=0.7$ , we are about here  
Evaluate  $(1 - 0.7) \cdot \mathbf{f}(u_0) + 0.7 \cdot \mathbf{f}(u_1)$

# Marching cubes – 2D

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

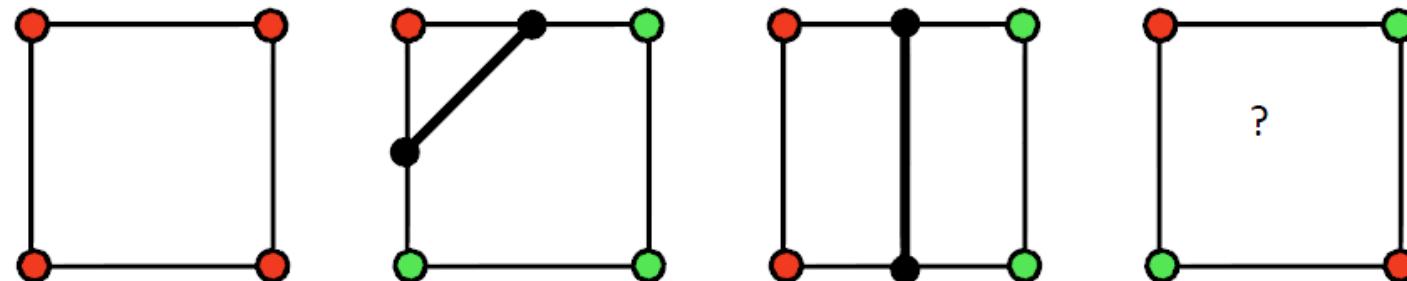
Due to symmetry, 4 unique configurations:



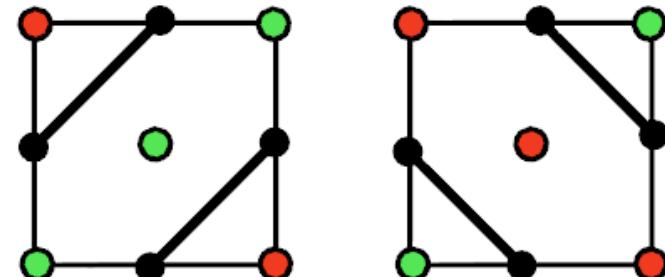
# Marching cubes – 2D

$2^4 = 16$  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.

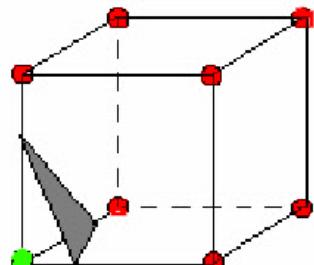
# Marching cubes - 3D

Classify grid vertices as inside/outside

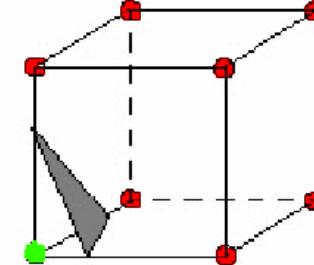
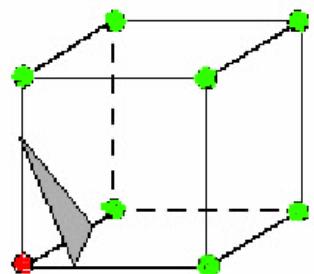
$2^8 = 256$  cases in 3D

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

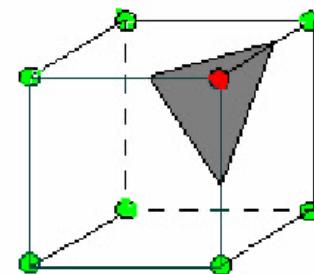
Due to symmetries (see below), 15 unique configurations



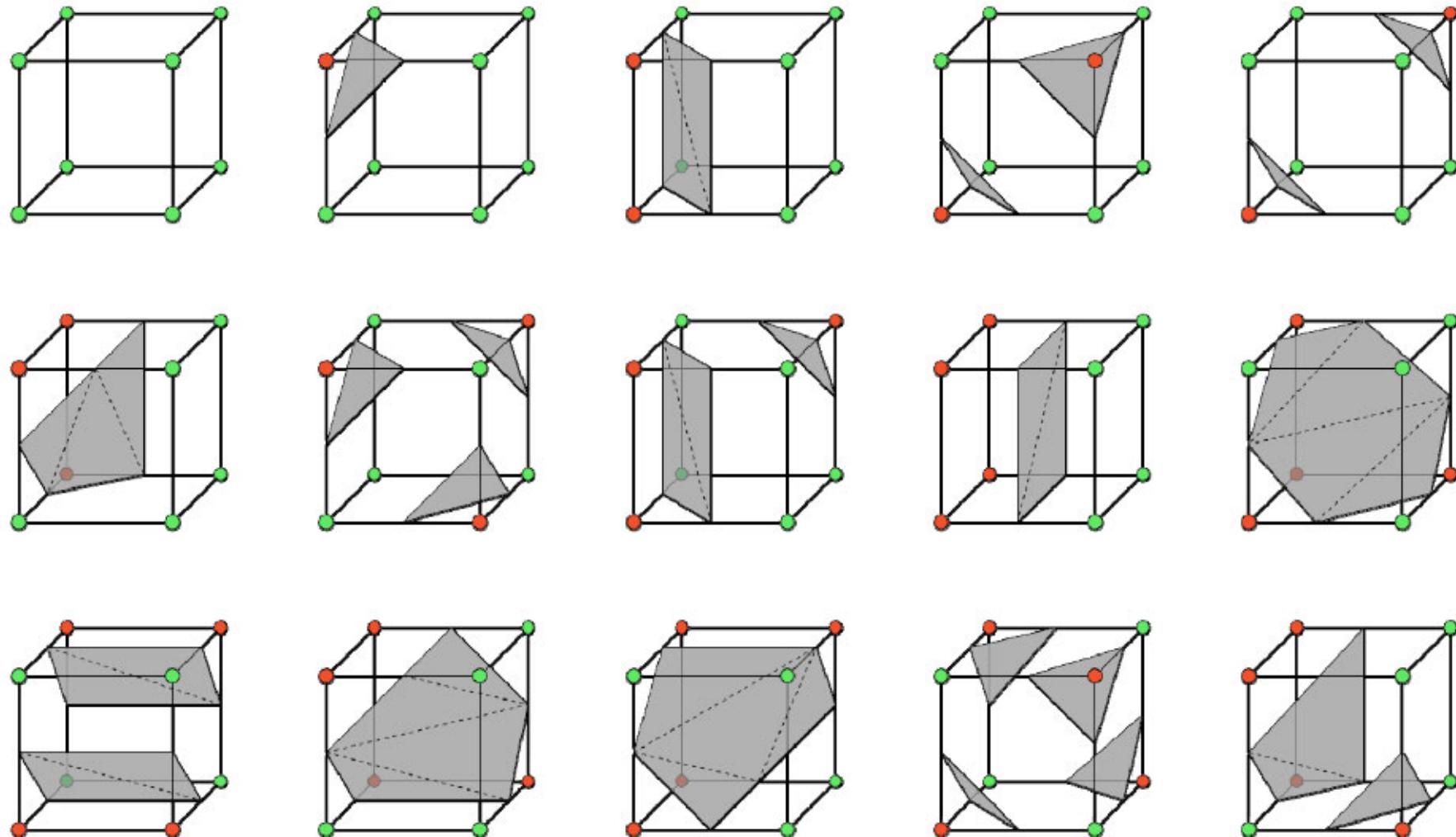
reverse case:



symmetric case:

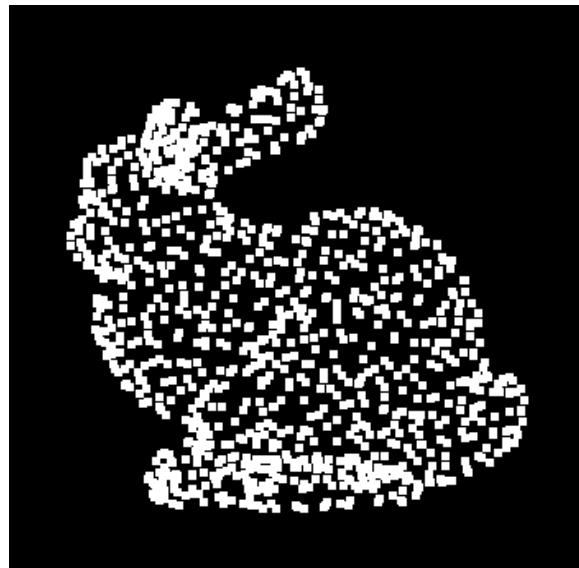


# Marching cubes - 3D

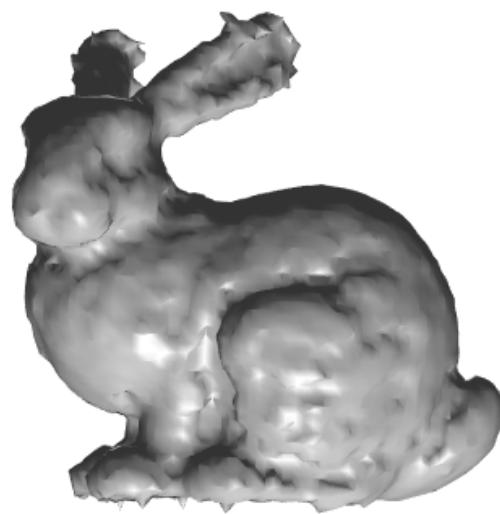


# Implicits

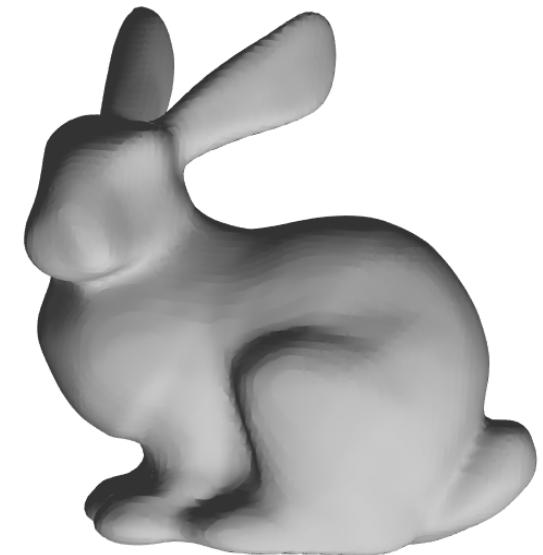
- **Continuous representation (infinite resolution)**
- **Compact to store esp if a neural net predicts  $f(x,y,z)$**
- **Errors in signed distance produce many artifacts**



Point Cloud



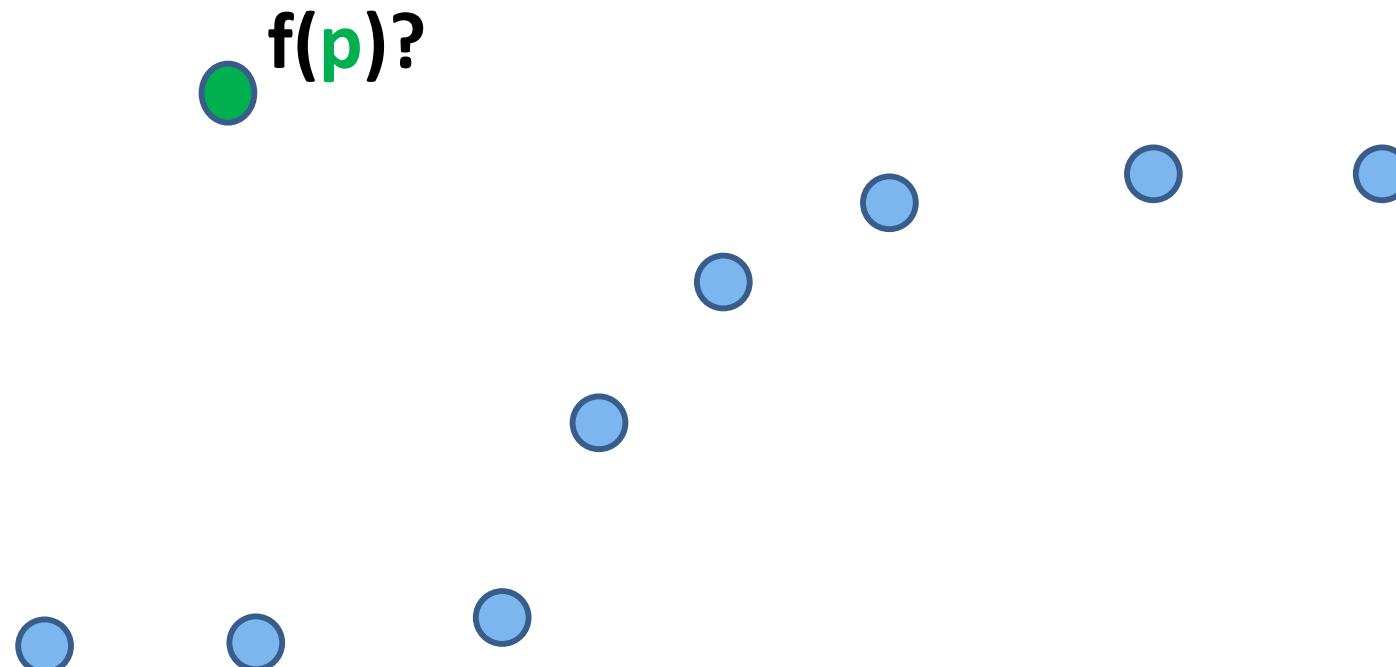
Simple implicit  
approximation  
based on dot products



Neural implicit  
approximation

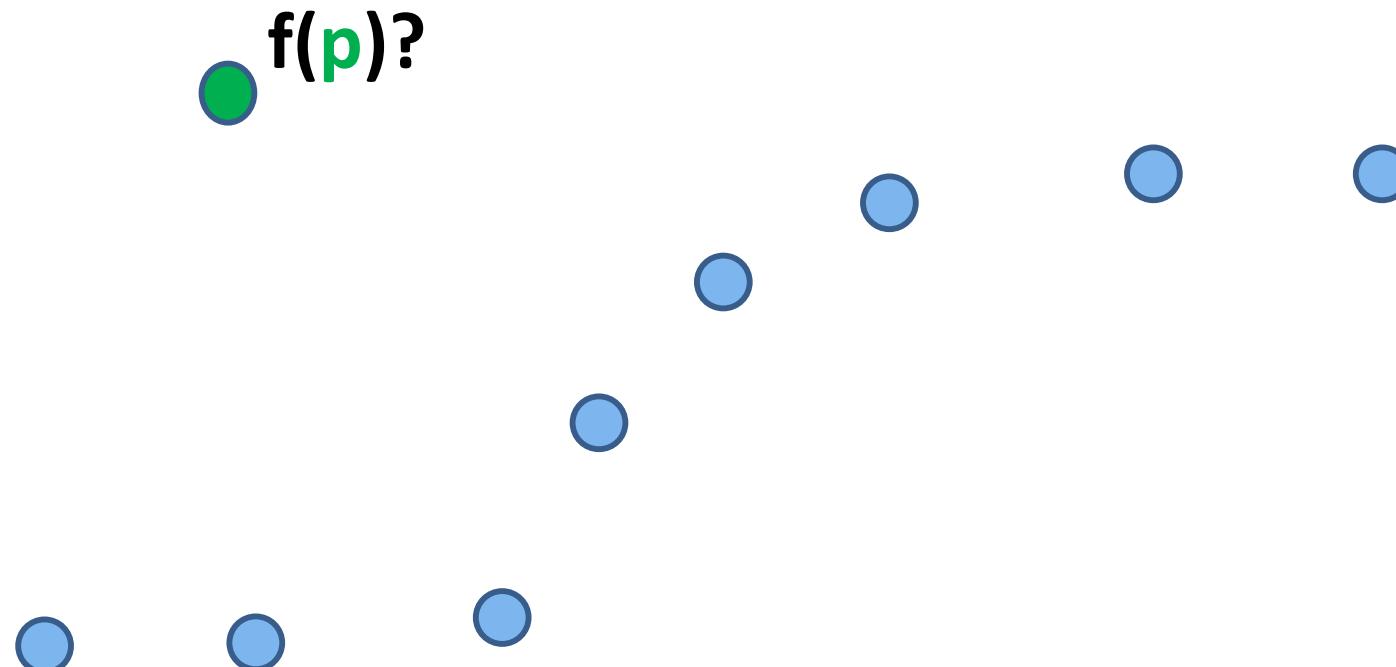
# Moving Least Squares

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



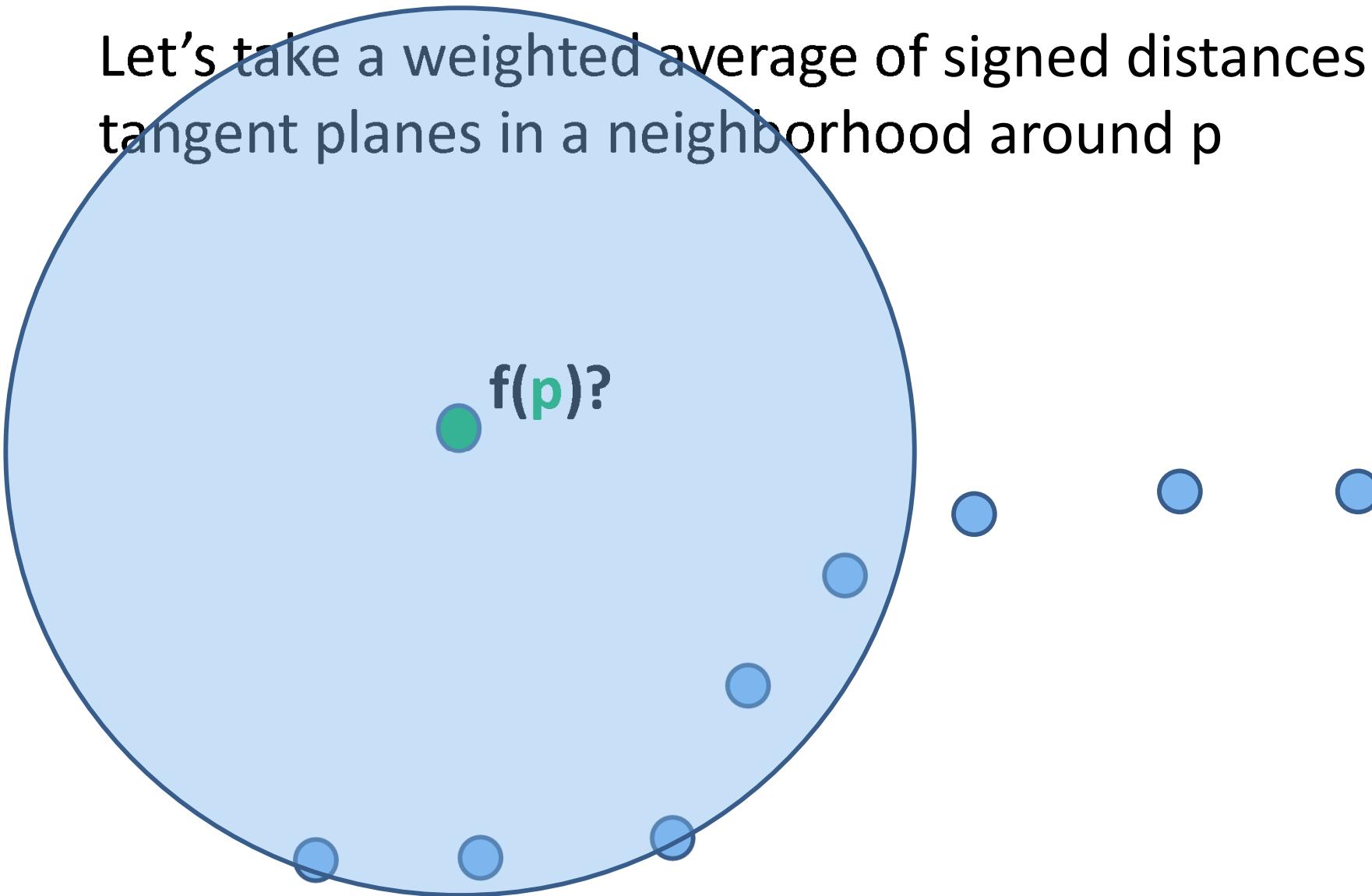
# Moving Least Squares

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



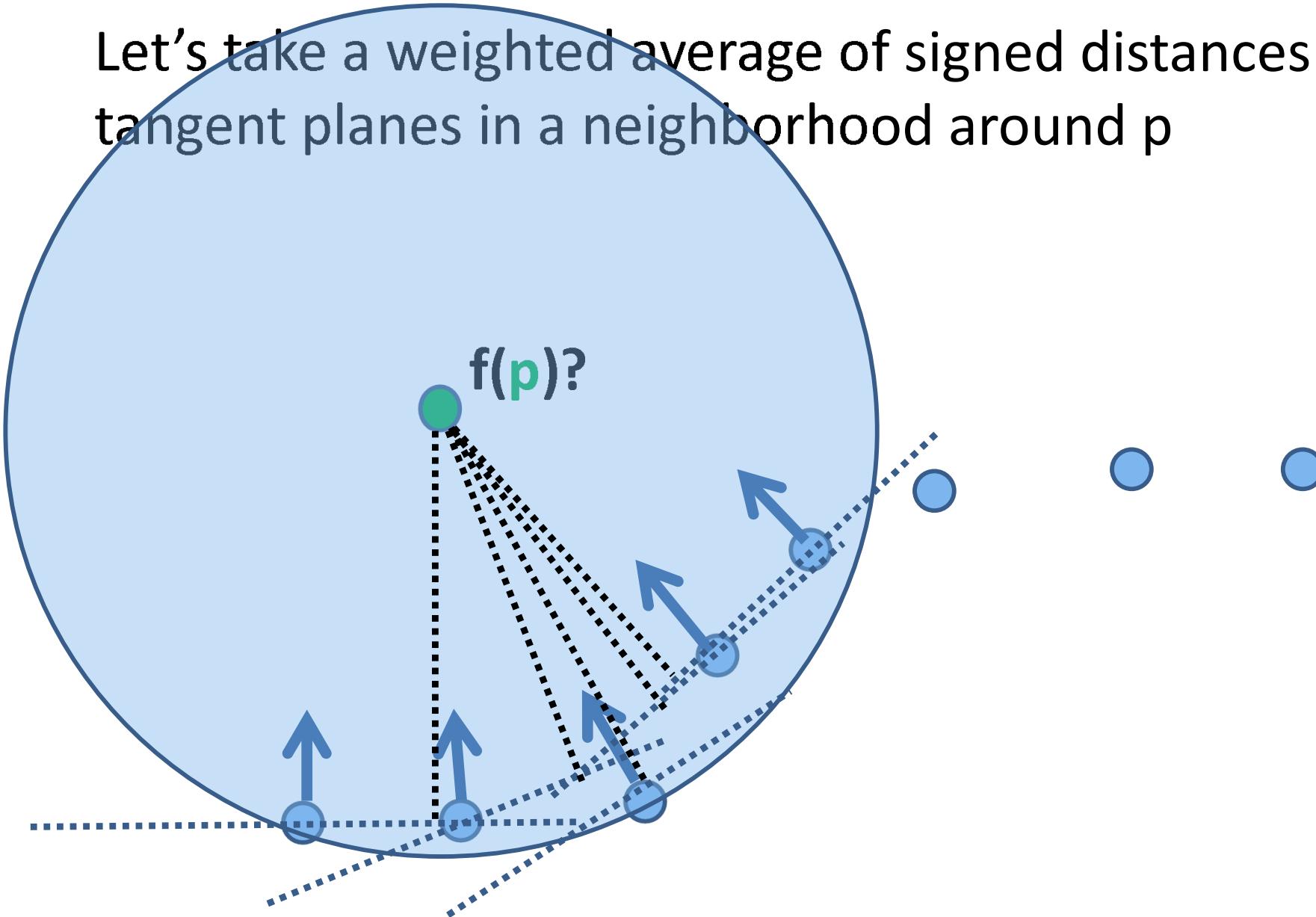
# Moving Least Squares

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



# Moving Least Squares

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

$$f(\mathbf{p}) = \frac{\sum_i \mathbf{n}_i (\mathbf{p} - \mathbf{p}_i) \varphi(\|\mathbf{p} - \mathbf{p}_i\|)}{\sum_i \varphi(\|\mathbf{p} - \mathbf{p}_i\|)}$$

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

# Moving Least Squares

Take a weighted average of all signed distances to tangent planes

$$f(\mathbf{p}) = \frac{\sum_i \mathbf{n}_i (\mathbf{p} - \mathbf{p}_i) \varphi(\|\mathbf{p} - \mathbf{p}_i\|)}{\sum_i \varphi(\|\mathbf{p} - \mathbf{p}_i\|)}$$

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



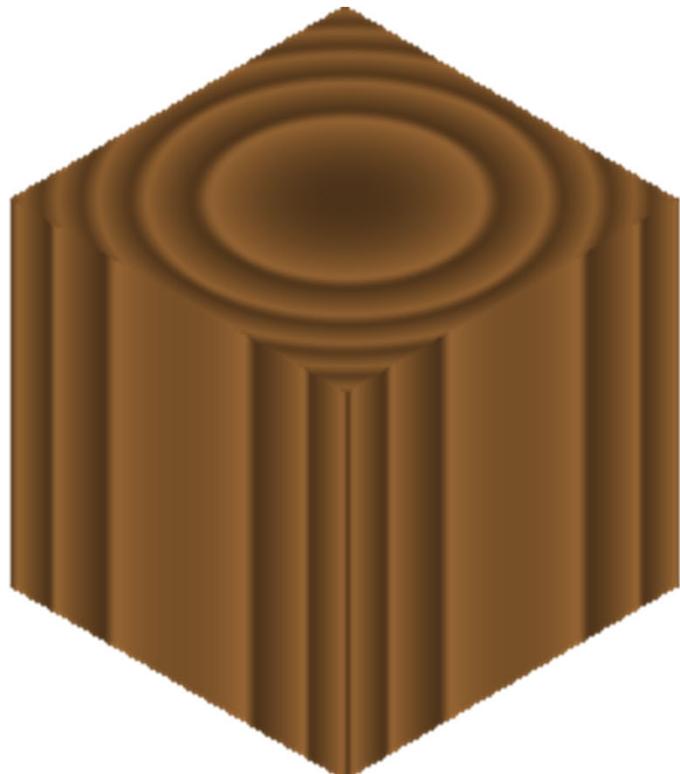
controls weight decay:  
Lots of hand-tuning for each shape...  
... that's why we prefer neural nets to  
approximate the implicit...

# 3D Textures

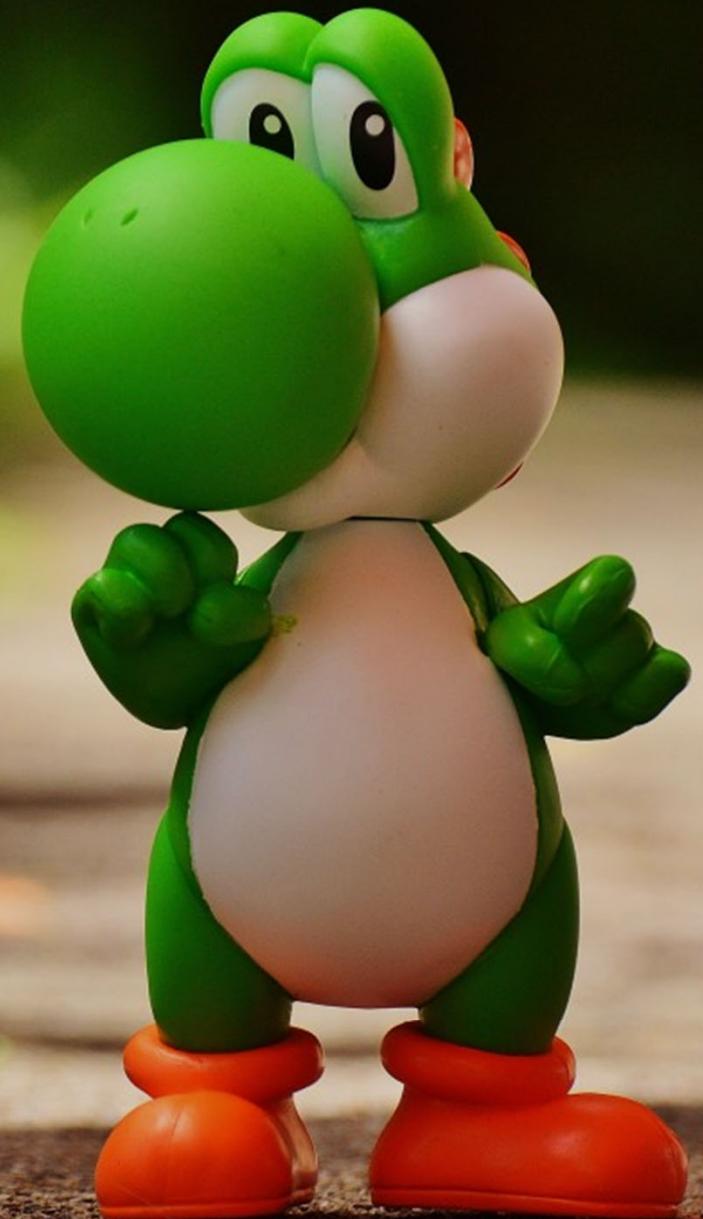
Instead of using a 2D image, we can instead use a 3D volumetric image:  $f(x, y, z) = [\text{unsigned distance}, \text{red}, \text{green}, \text{blue}]$

No need to flatten the mesh. **Specify 3D texture coordinates per vertex.**

Lots of memory might be required. Hard to define it by hand...

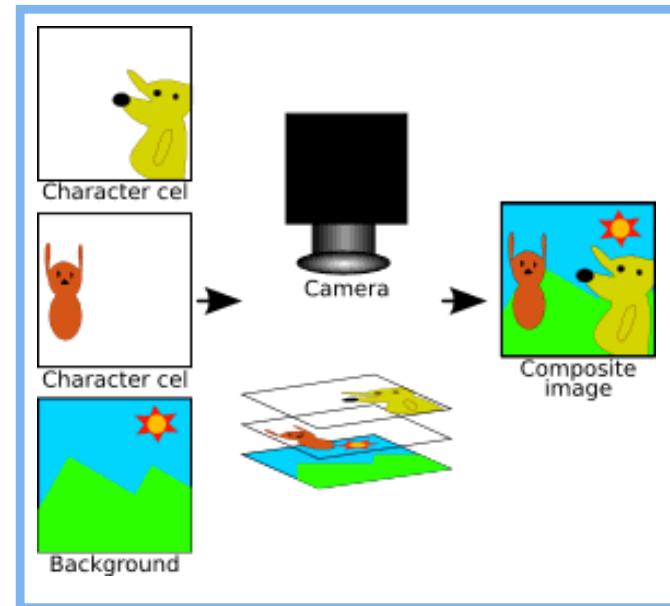


# Part III: Animation Representations



# History of Animation

Traditional cel animation:



First animated characters in old games were 2D sprites:



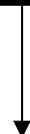
# History of Animation

**Stop Motion:** Physically manipulate real-world objects and photograph them one frame at a time to create the illusion of movement



# History of Animation

<https://www.youtube.com/watch?v=4kOQz2sVQ34>



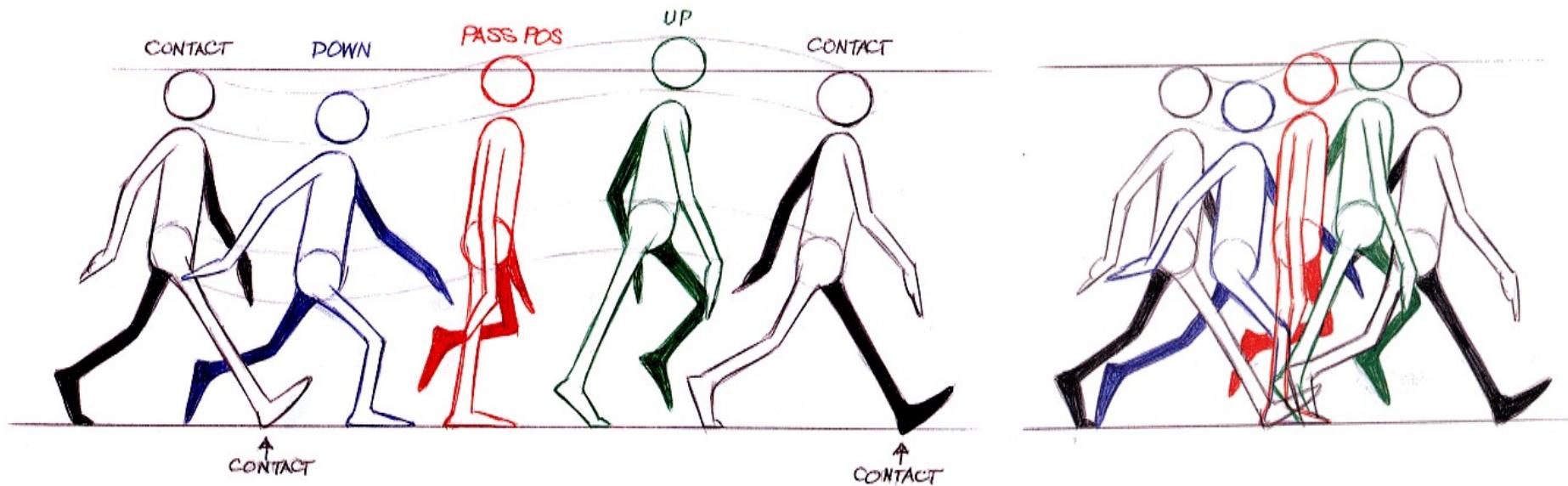
# Specifying each single frame is hard!



# Keyframing

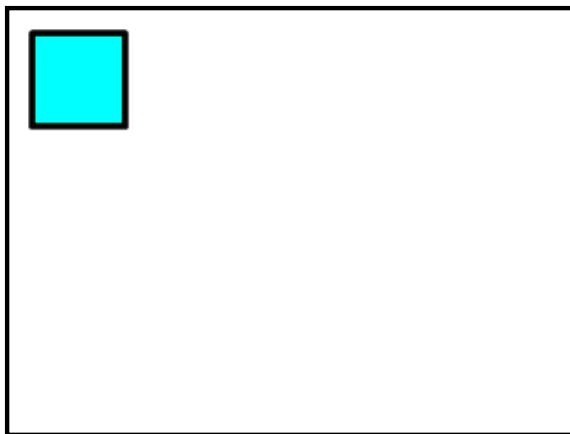
Need to stay cost efficient:

- Advanced artists model a set of key poses (keyframes) for characters
- Beginner artists filled in the motion in between these key poses

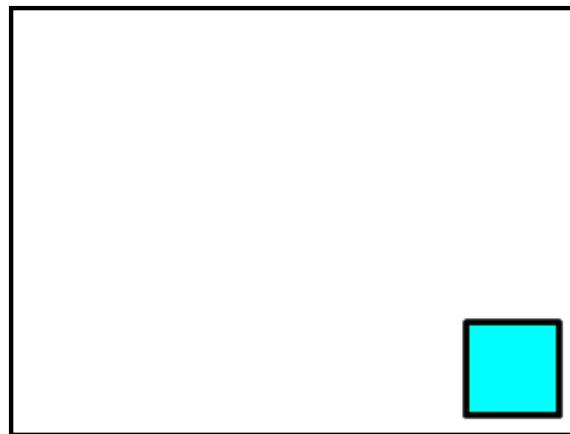


# Computer keyframing

Fill in motion automatically between sparse **keyframes** specified by the user.



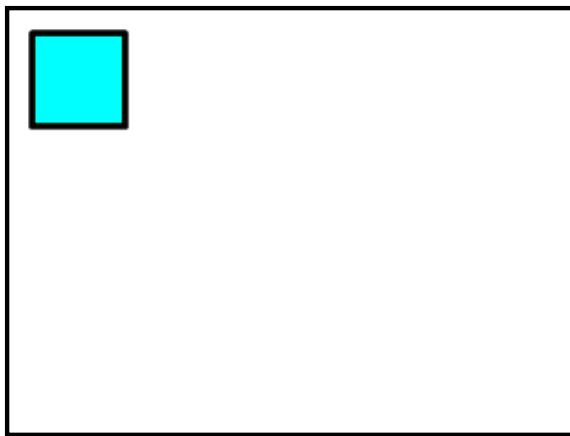
**Position of object  
at first keyframe**



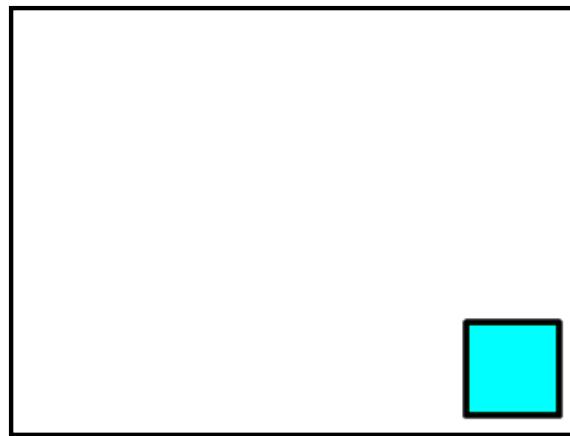
**Position of object at  
second keyframe**

# Computer keyframing

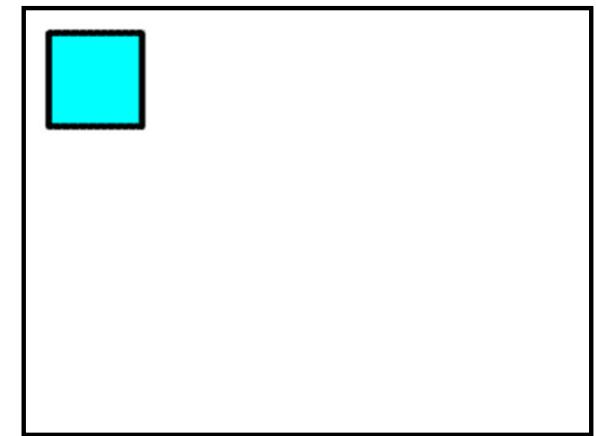
Fill in motion automatically between sparse **keyframes** specified by the user.



Position of object  
at first keyframe



Position of object at  
second keyframe

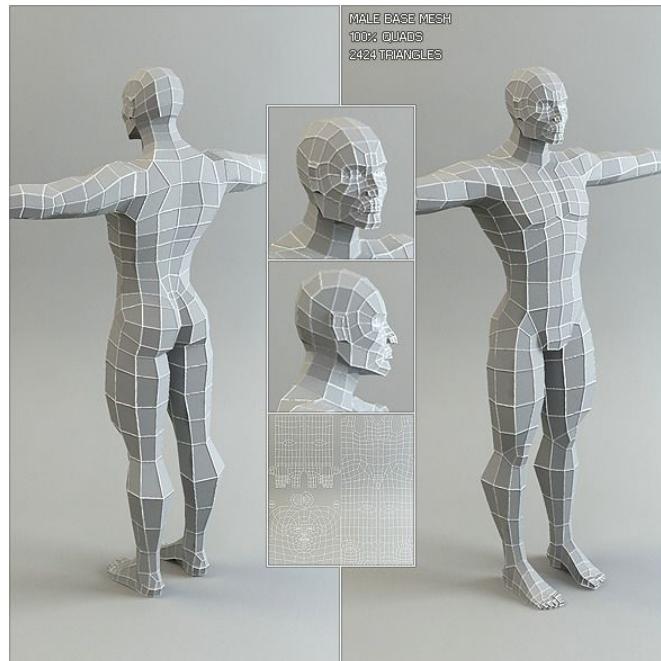


Interpolated  
position  
[animation]

If we have a complex shape, do we specify keyframes for each of its points?



If we have a complex shape, do we specify keyframes for each of its points?



Too many points to handle!



# Character rigging and skinning

Basic idea:

- *Create a jointed skeleton*
- *Attach the surface of a 3D model (skin) to the skeleton*
- *Animation: move skeleton around – the surface follows it*

Skeleton is invisible – only the skin is seen by the viewer!

# Character rigging and skinning

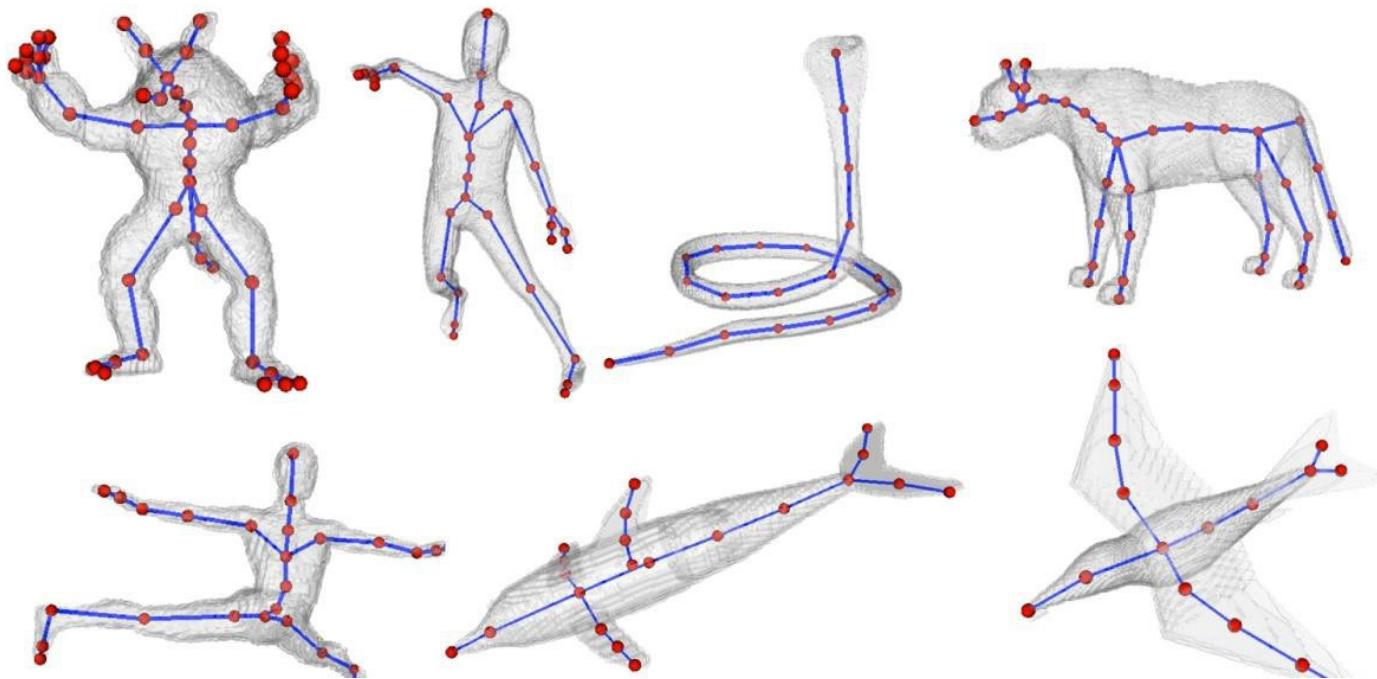
Basic idea:

- *Create a jointed skeleton*
- *Attach the surface of a 3D model (skin) to the skeleton*
- *Animation: move skeleton around – the surface follows it*

Skeleton is invisible – only the skin is seen by the viewer!

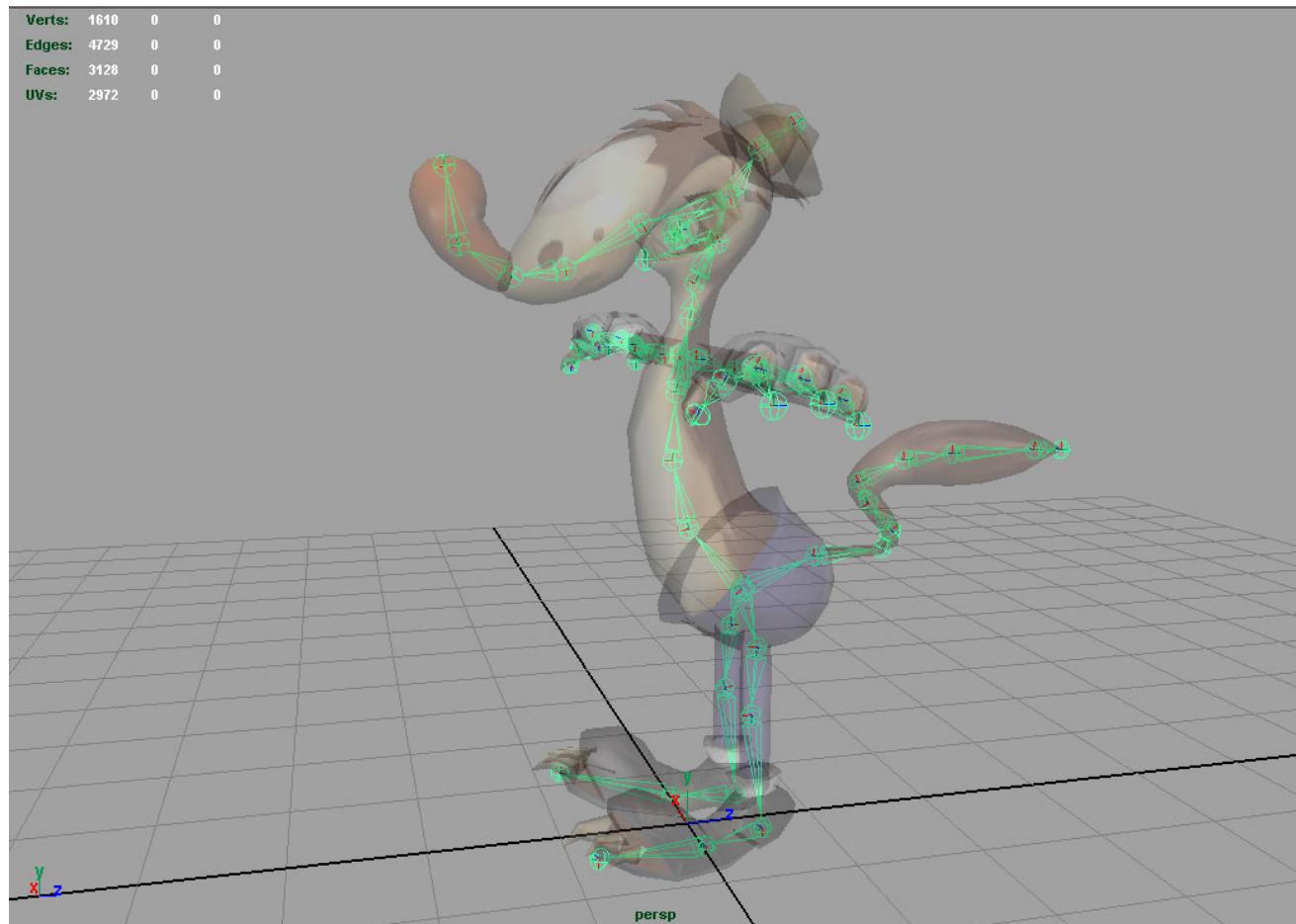
# Animation skeletons

An animation **skeleton** is a collection of “bones” (rigid segments) connected by articulated joints



# Animation skeletons

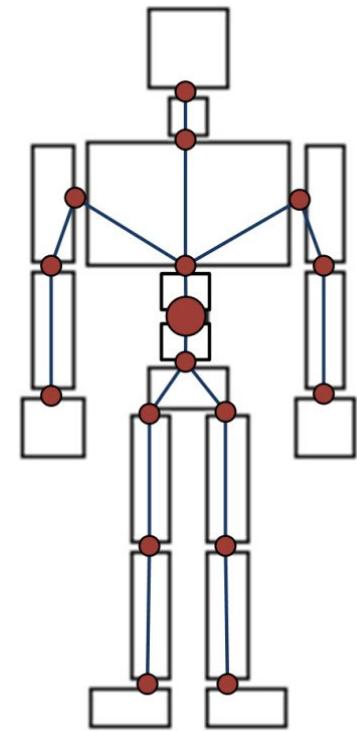
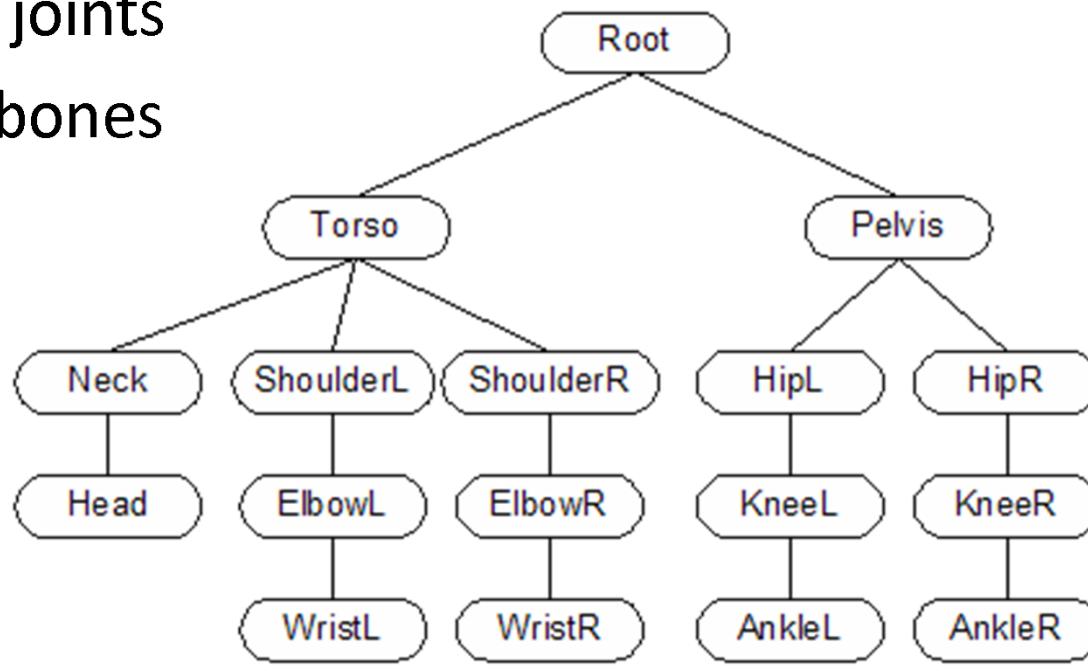
Artists create the skeleton suited for the character in a neutral pose (**bind pose**).



# Joint hierarchy

The articulated skeleton is described by a **tree hierarchy**

- Nodes: joints
- Edges: bones



- The transformation of a joint is defined **relative** to its parent joint/body in the hierarchy

# Joint hierarchy

Joints are organized in a hierarchy

The **root** is the position of the “base” of the skeleton

- typically the backbone or pelvis
- the root has all 6 DoF so it can be placed anywhere with any orientation

Typically, other joints have only rotational DoFs

- but in reality some have translational components as well



3 translational and  
48 rotational DoFs

# Character rigging and skinning

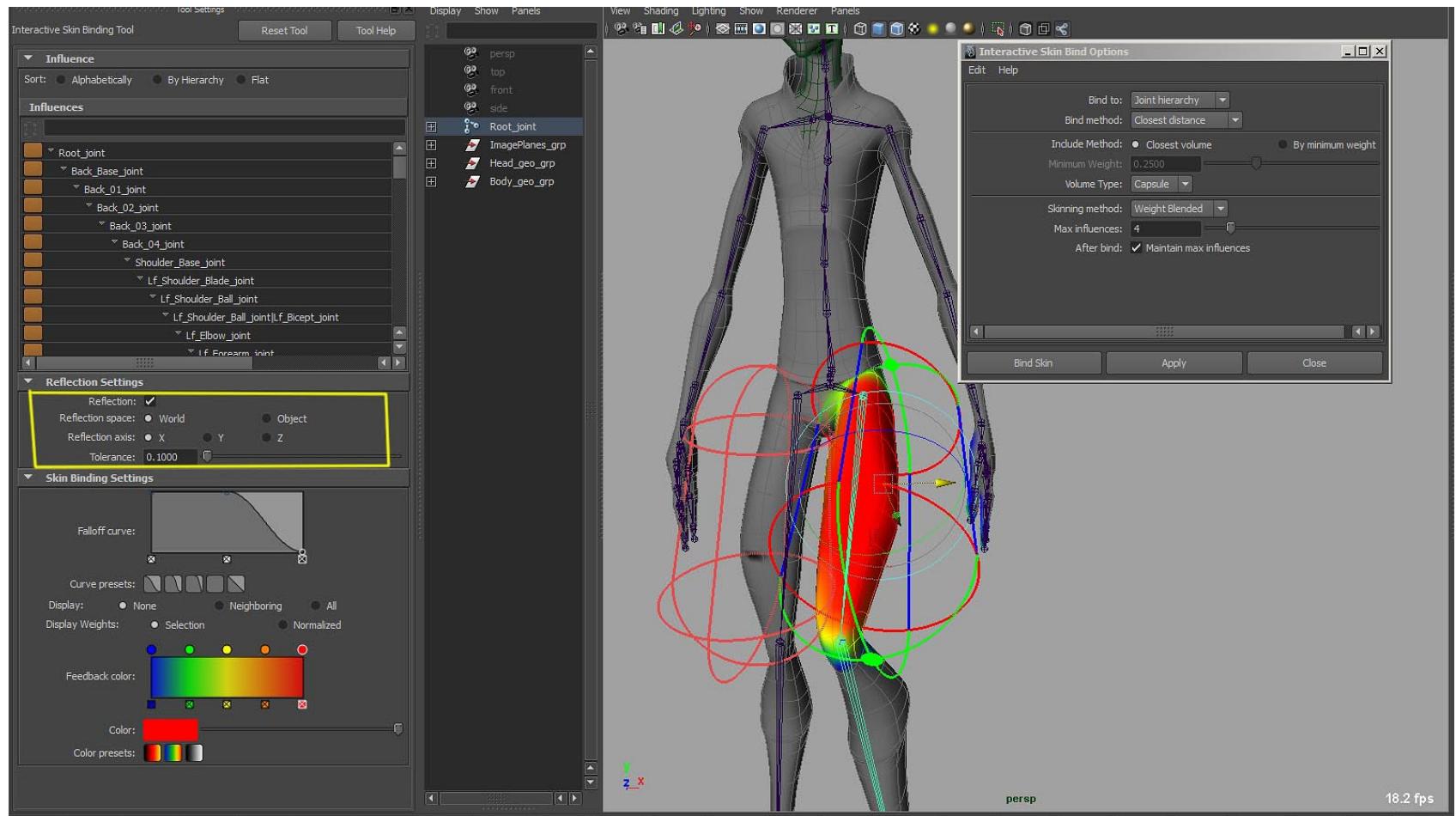
Basic idea:

- *Create a jointed skeleton*
- ***Attach the surface of a 3D model (skin) to the skeleton***
- *Animation: move skeleton around – the surface follows it*

Skeleton is invisible – only the skin is seen by the viewer!

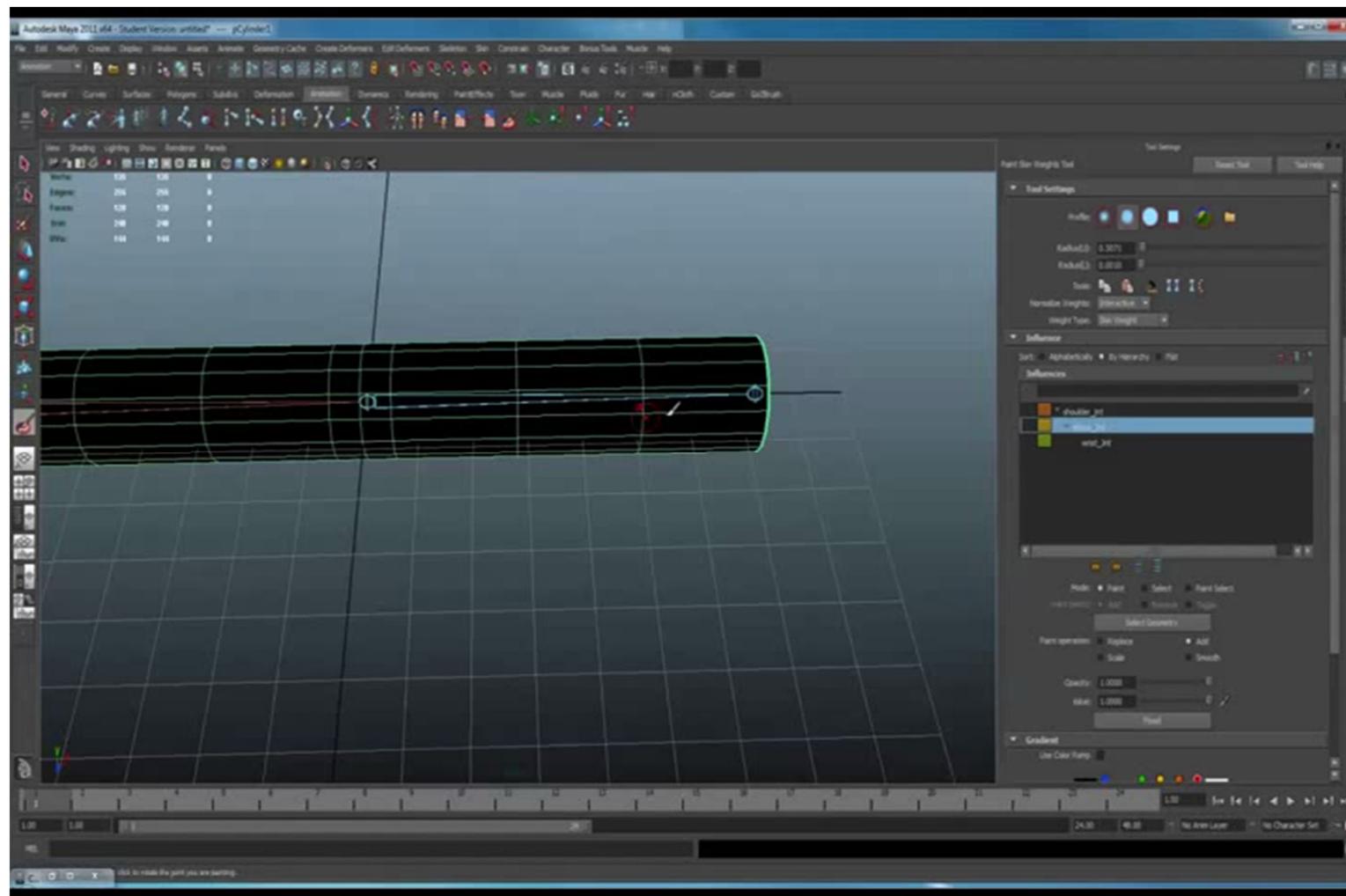
# Skinning

Each vertex in the polygon mesh is attached to **one or more** bones of the skeleton. We use **weights** to define bones' influences on each vertex (must sum to 1 per vertex).



# How are skinning weights specified?

An artist “paints” the influence of each bone to vertices:



... discussion

# Character rigging and skinning

Basic idea:

- *Create a jointed skeleton*
- *Attach the surface of a 3D model (skin) to the skeleton*
- ***Animation: move skeleton around – the surface follows it***

Skeleton is invisible – only the skin is seen by the viewer!

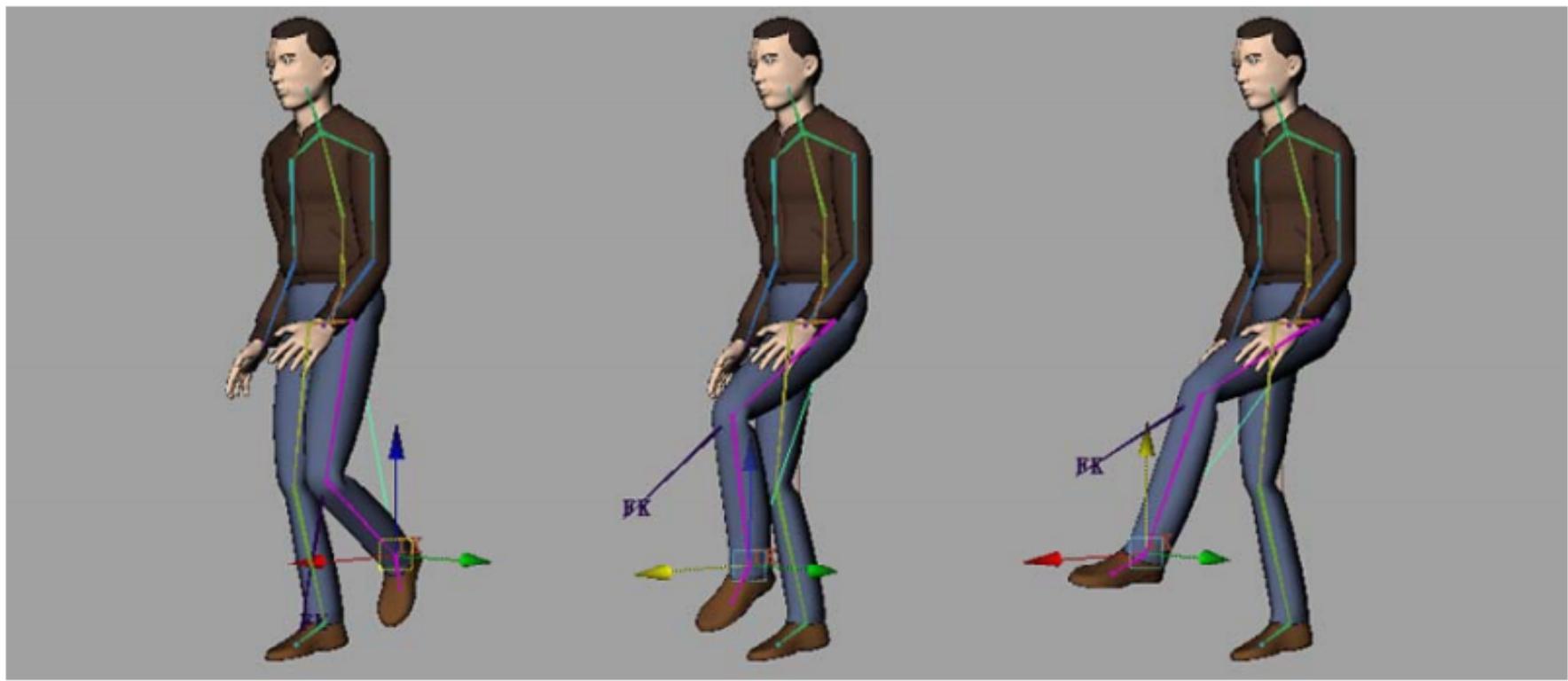
# Forward Kinematics (FK)

Animator manually specifies joint movements (rotations, translations) for each keyframe.

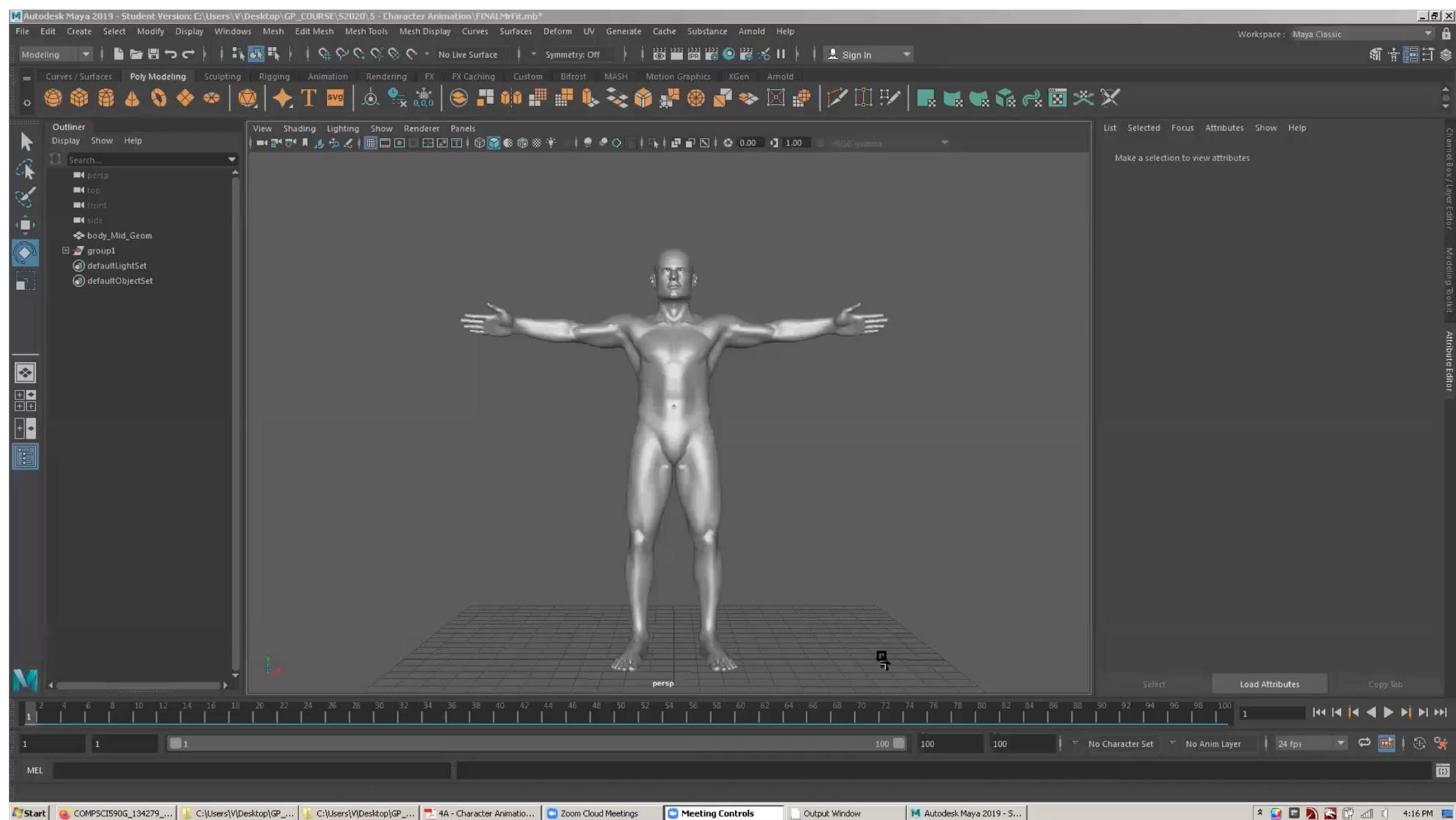


# Inverse Kinematics (IK)

Animator specifies desired location and sometimes orientation for a particular joint (called **end-effector**):



# Animation in Maya



# Blend-shape animation

Another popular animation technique esp for faces!

$B$ : base face [ $3 \times V$  matrix,  $V$  is #vertices]

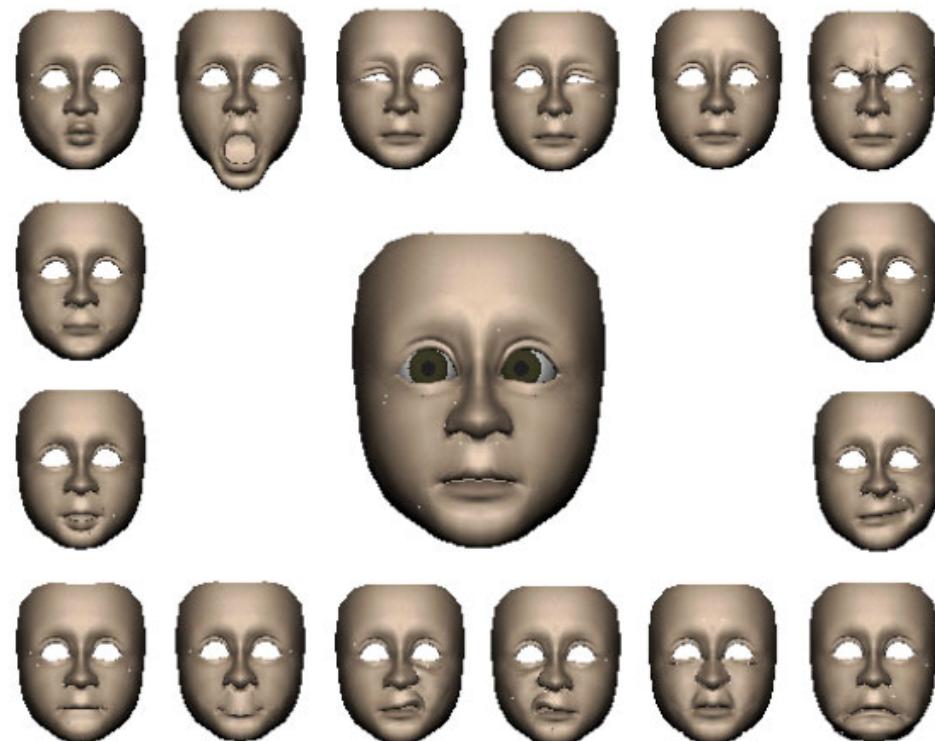
$U_i$ : displacement (translation per vertex)

from base shape for each expression  $i$

[ $3 \times V$  matrix]

$a_i$ : desired weight (blending) of expression  $i$

set at each keyframe



# Blend-shape animation

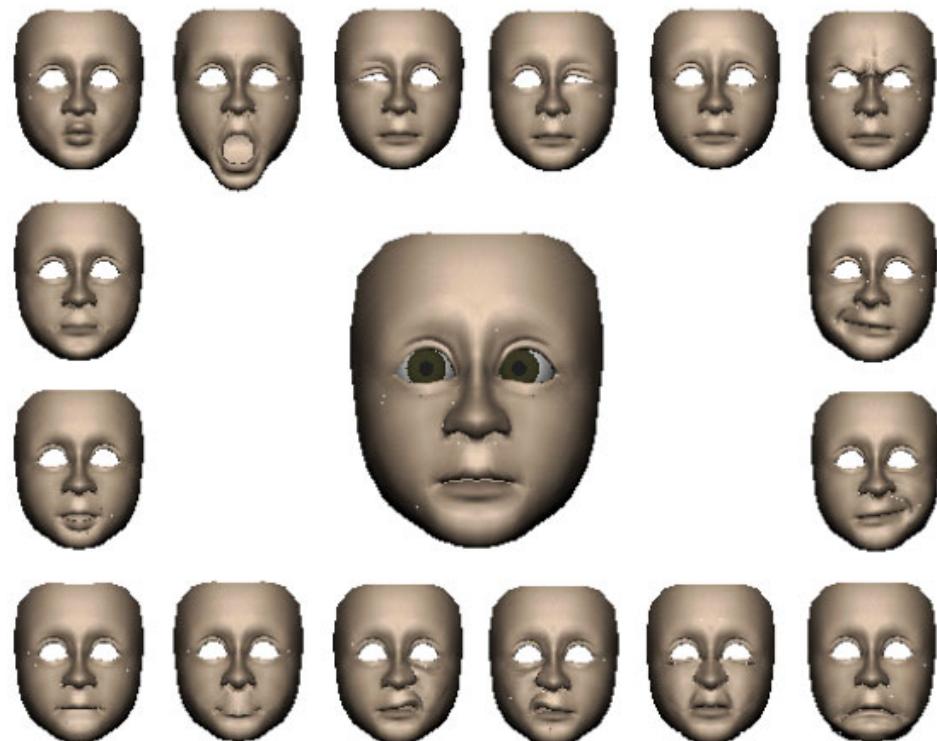
$B$ : base face [ $3 \times V$  matrix,  $V$  is #vertices]

$U_i$ : displacement (translation per vertex)  
from base shape for each expression  $i$   
[ $3 \times V$  matrix]

$a_i$ : desired weight (blending) of expression  $i$   
set at each keyframe

**Animated face:**

$$F = B + \sum_i a_i U_i$$



# Blend-shape animation

