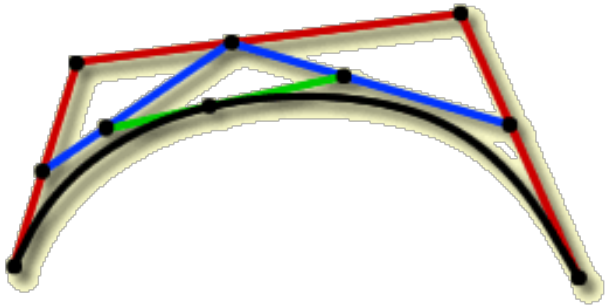
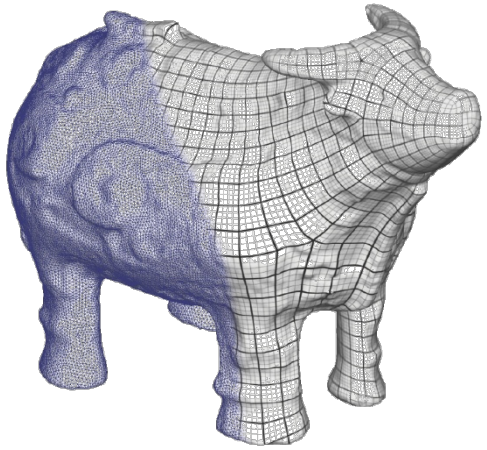
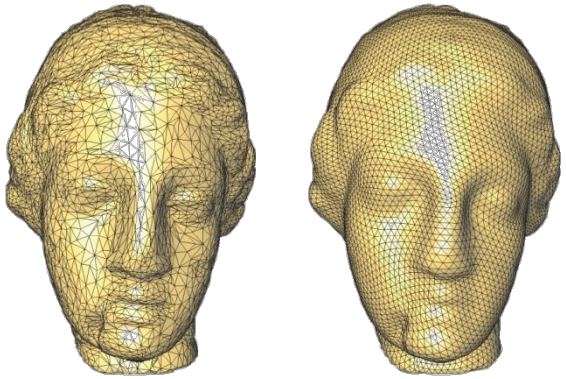
CS348a: Geometric Modeling and Processing







Leonidas Guibas
Computer Science Department
Stanford University

Last Time:
Neural Generative Models for Shapes

Generative Model (Unconditional)

Given training data, generate new samples from the same distribution:

Training data ~ $p_{data}(x)$

Generated samples~ $p_{model}(x)$

Objective: learn a $p_{model}(x)$ that matches $p_{data}(x)$.

Conditional Generative Model

- Data: (x, y) where x is a **condition** and y is the corresponding **content.**

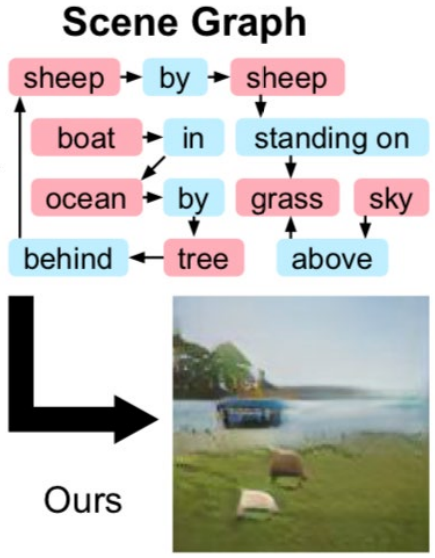
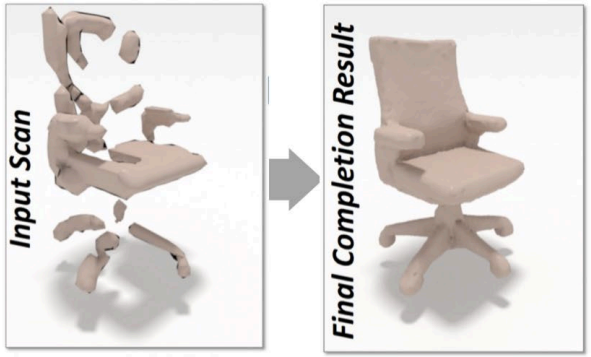


Image generation based on scene-graph

Single-view 3D reconstruction

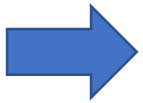


Shape completion

Objective: learn a $p_{model}(y|x)$ that matches $p_{data}(y|x)$.

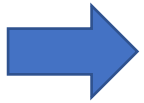
How to Learn Generative Models

- Explicitly modeling data probabilistic density, learn a network $p_\theta(x)$ that maximize data probability



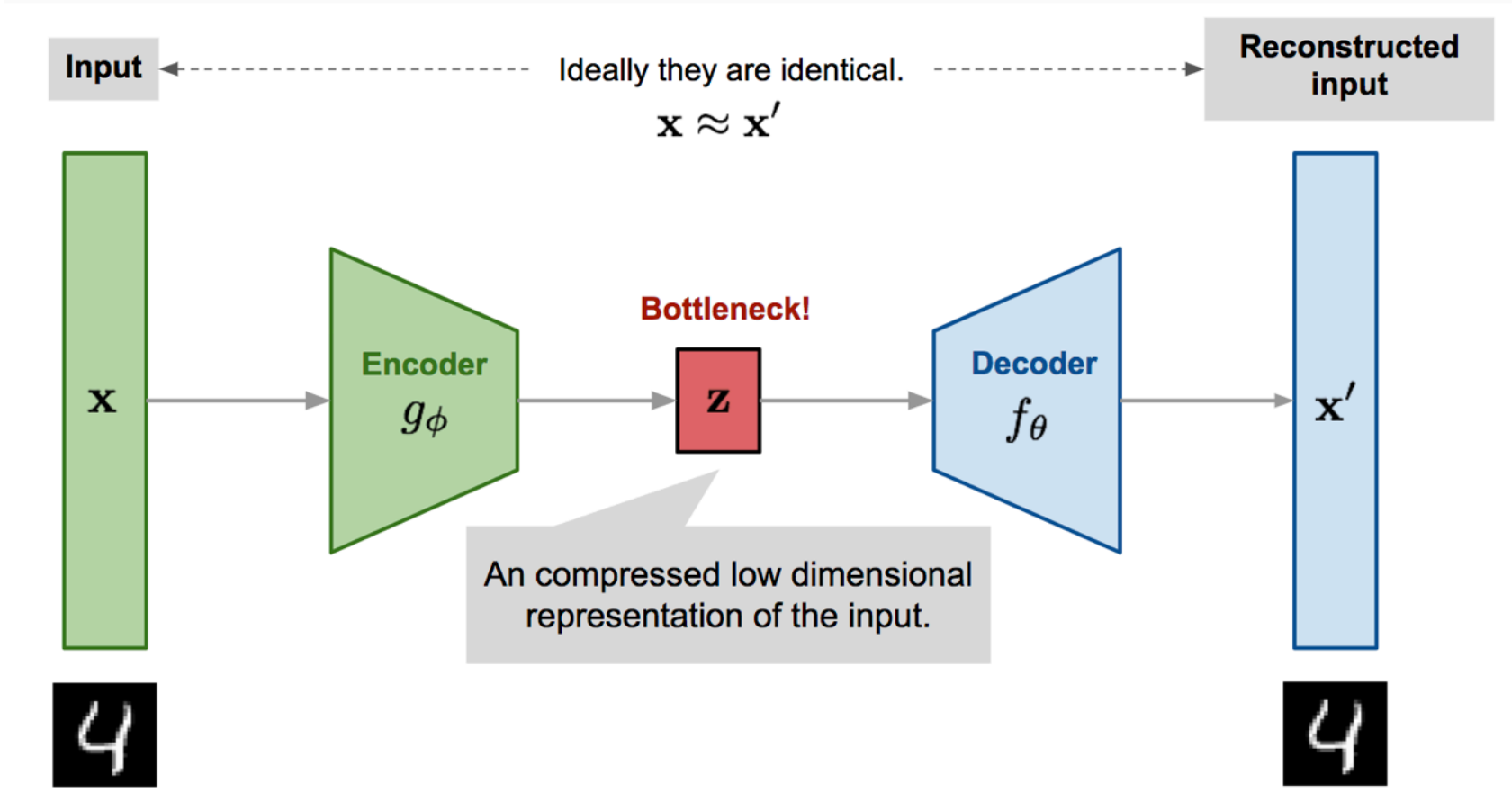
- Markov chain
- Autoregressive models
- **Variational autoencoder (VAE)**
- **Flow-based models**
- Energy based models
- …

- Implicitly modeling probabilistic density, e.g. learn a network that scores the "realness" of the data, $f_\theta(x)$



- **Generative adversarial network (GAN)**
- Score-based generative
- …

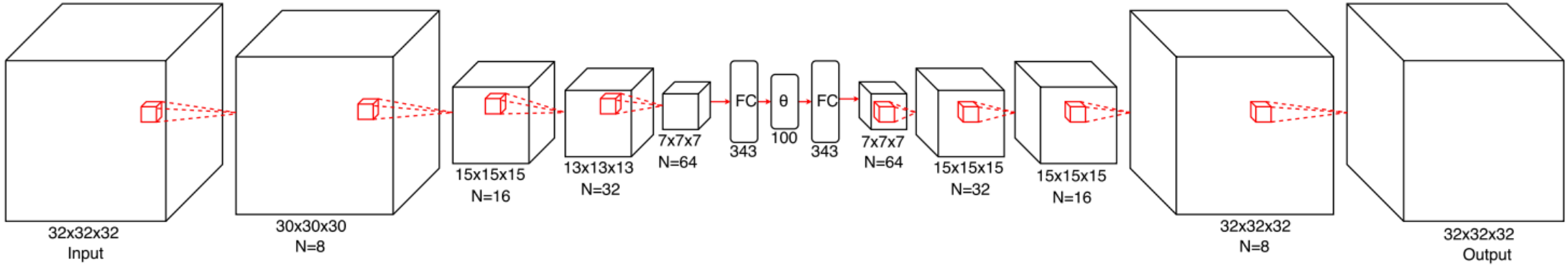
Auto-Encoder



Task: Learn to encode the input and decode itself

Reconstruction loss: measuring the distance between the input/output

Volumetric AE

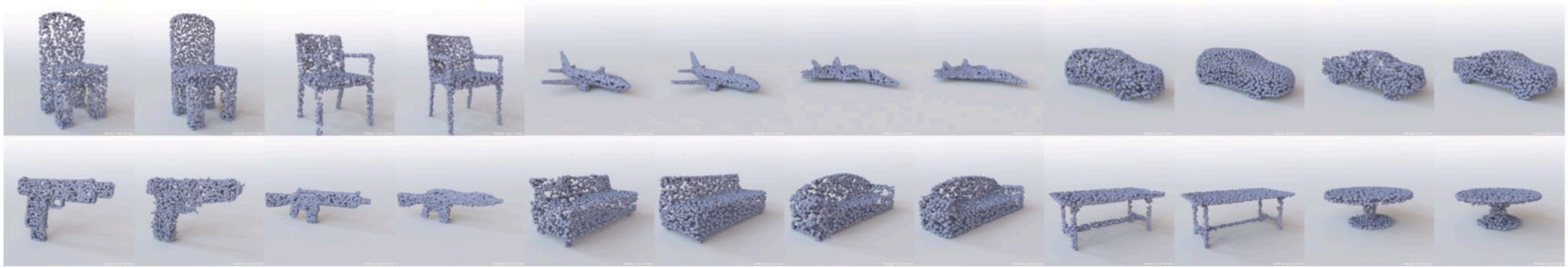


Binary Cross-Entropy Loss: $\mathcal{L} = -t\,log(o)\ -\ (1-t)\,log(1-o)$

CoRR 2016
Generative and Discriminative Voxel Modeling with Convolutional Neural Networks

Point Cloud AE

Encoder: PointNet (N*3 → L)
Decoder: MLP (L → 3N → N*3)



ICML, 2018, Learning Representations and Generative Models for 3D Point Clouds, Panos Achlioptas, et. al.

Decoding/Generation

Latent vectors *z*

Generated Shapes

Generator/Decoder: generating shapes from latent vectors

Making the Network Generative

- Variational Auto-Encoder (VAE): Learn a distribution that approximates the data distribution of true 3D structures

$$P(X) \approx P_{gt}(X)$$

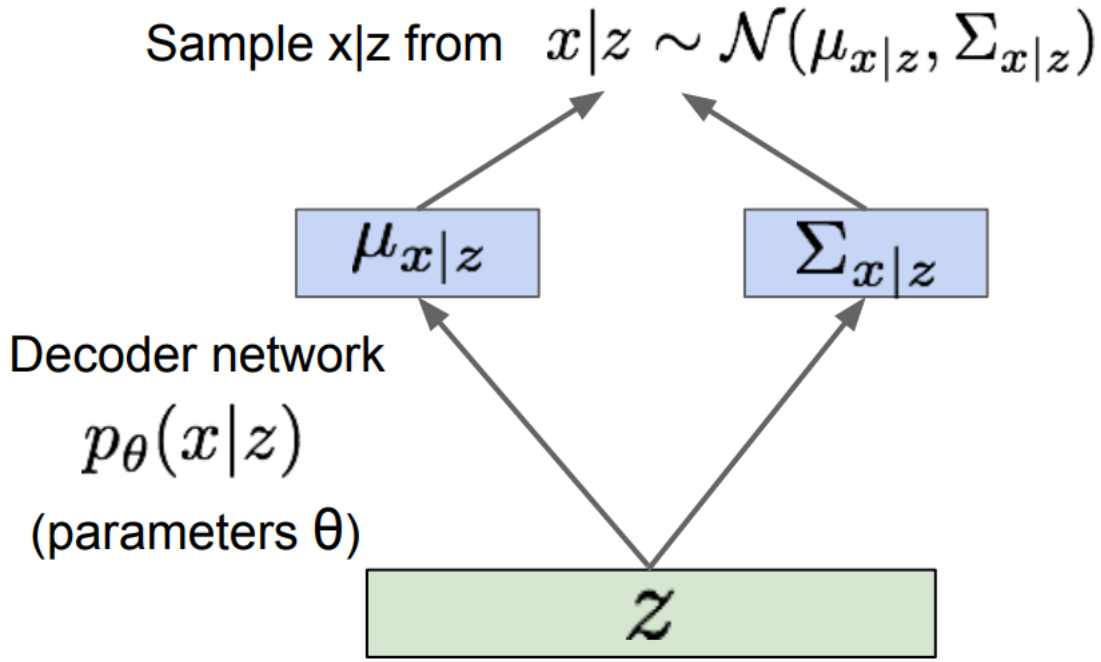
$$\text{maximize } P(X) = \int P(X|z;\theta)P(z)dz$$

$$\text{maximize } E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]$$

z should reconstruct X, given that it was drawn from $Q(z|X)$

Assuming z's follow a normal distribution

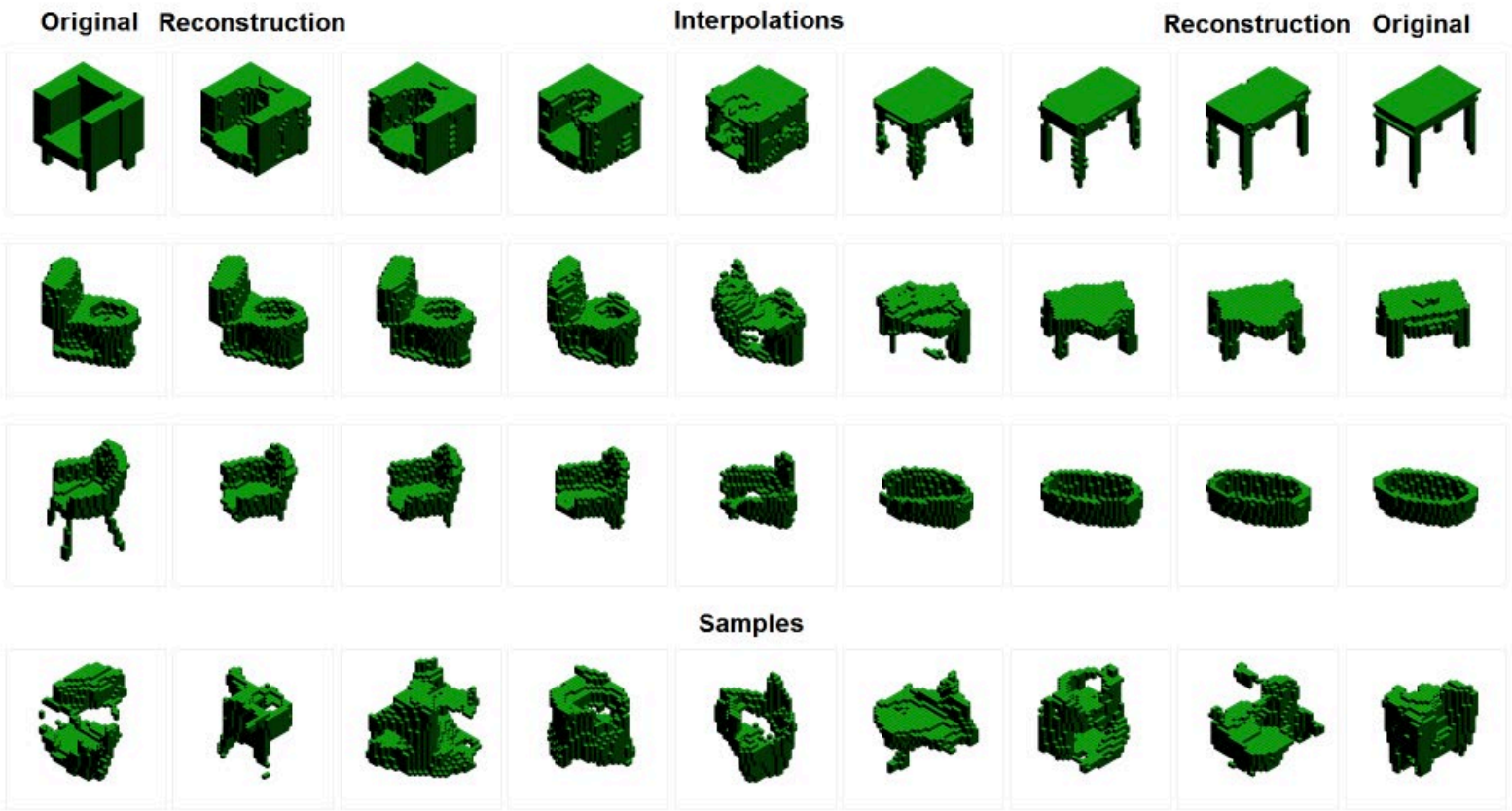
Variational Auto-Encoder



Decoder

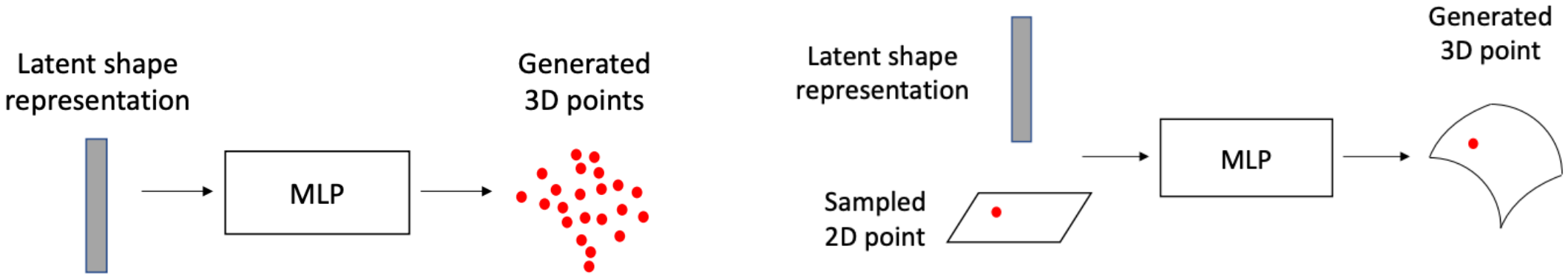
Image Credit: Stanford CS231N

Generating New Samples & Interpolation



CoRR 2016
Generative and Discriminative Voxel Modeling with Convolutional Neural Networks

Parametric Decoder: AtlasNet



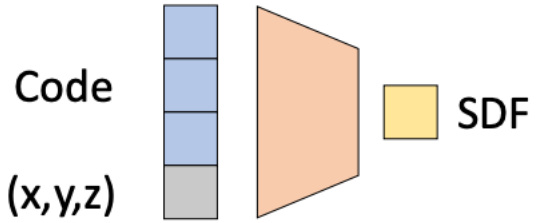
Given that the output points form a smooth surface, enforce such a parametrization in input. For each point (u, v) on the parameterization,

MLP([z, uv]) -> point

Also, you can get **Mesh!**

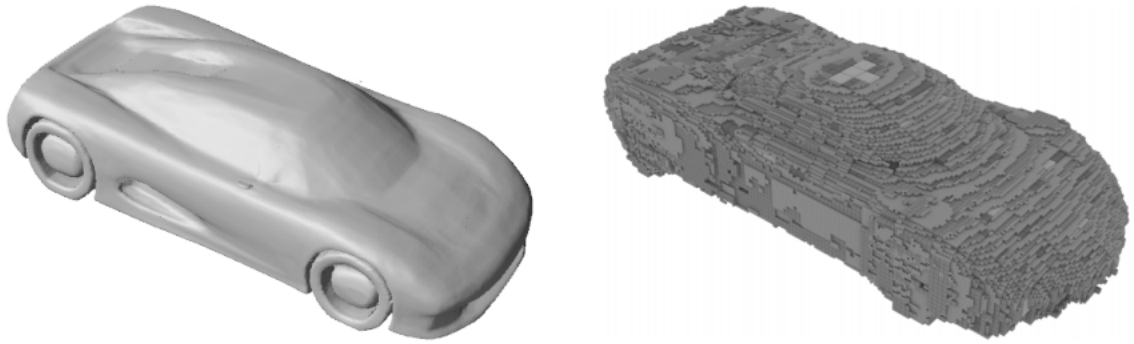
AtlasNet: A Papier-Machˆe Approach to Learning 3D Surface Generation, CVPR 2018

AutoEncoding SDF: Deep SDF



Decoder

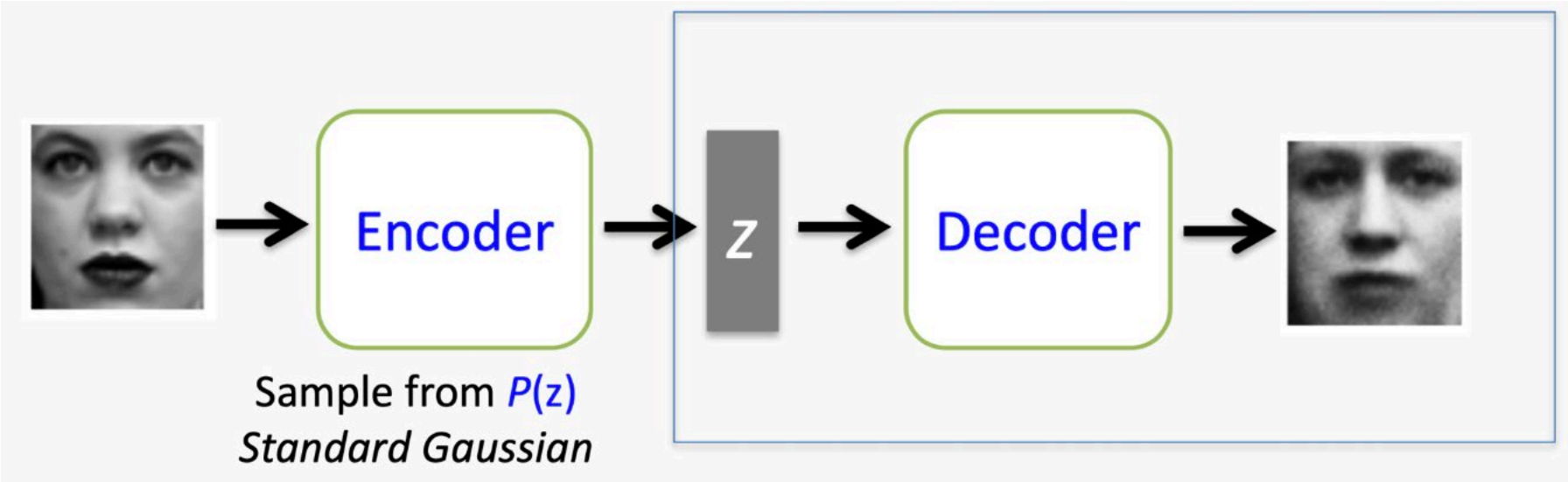
Comparison with Octree



(a) Ground-truth (b) Our Result (c) [22]-25 patch (d) [22]-sphere

DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, CVPR 2019

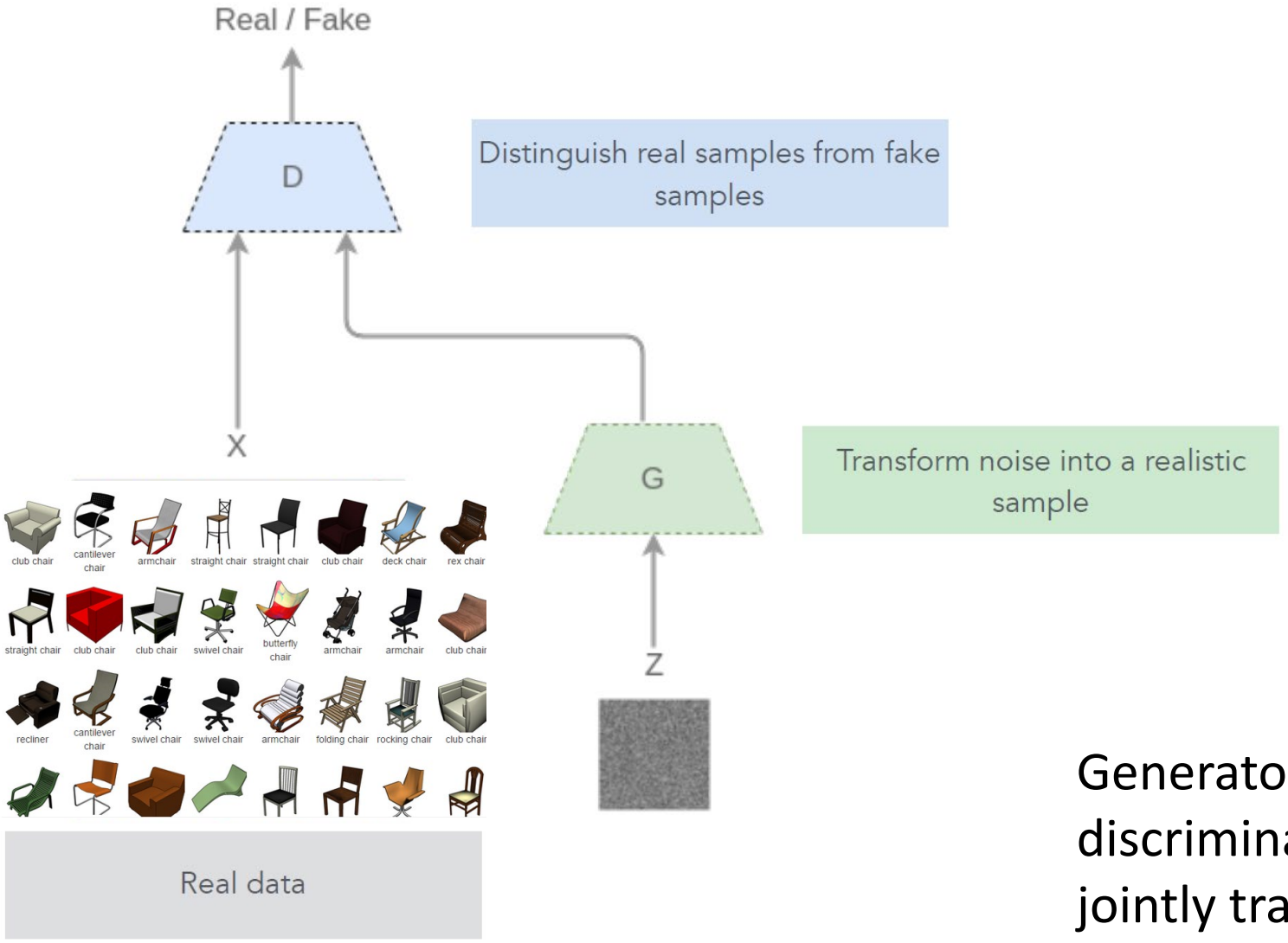
Issues for Autoencoders



Suffered from blurry issues.

Why? Loss function (L2).

Generative Adversarial Networks (GANs)



Generator G and discriminator D are jointly trained

Voxel GAN

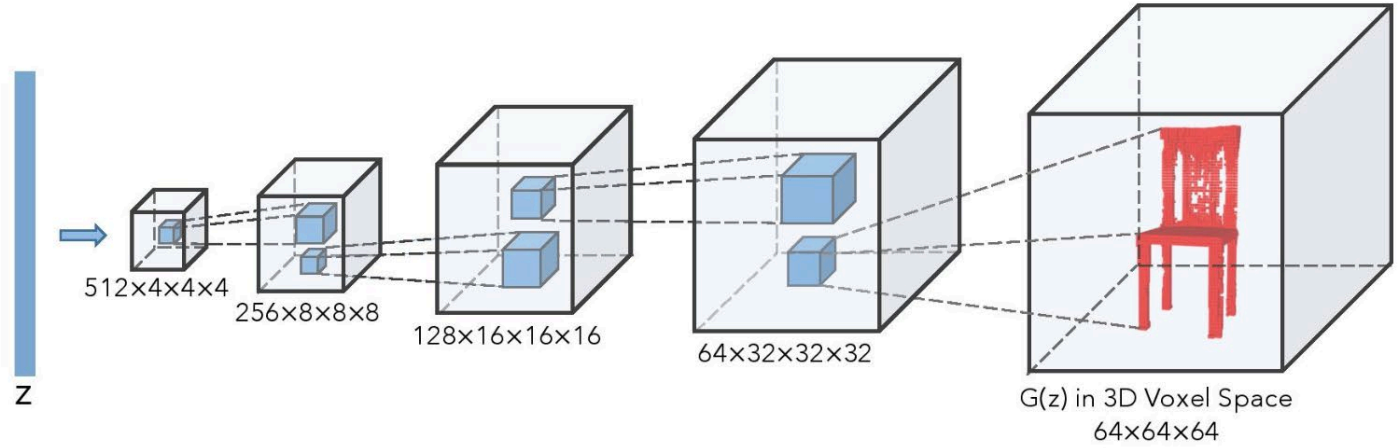
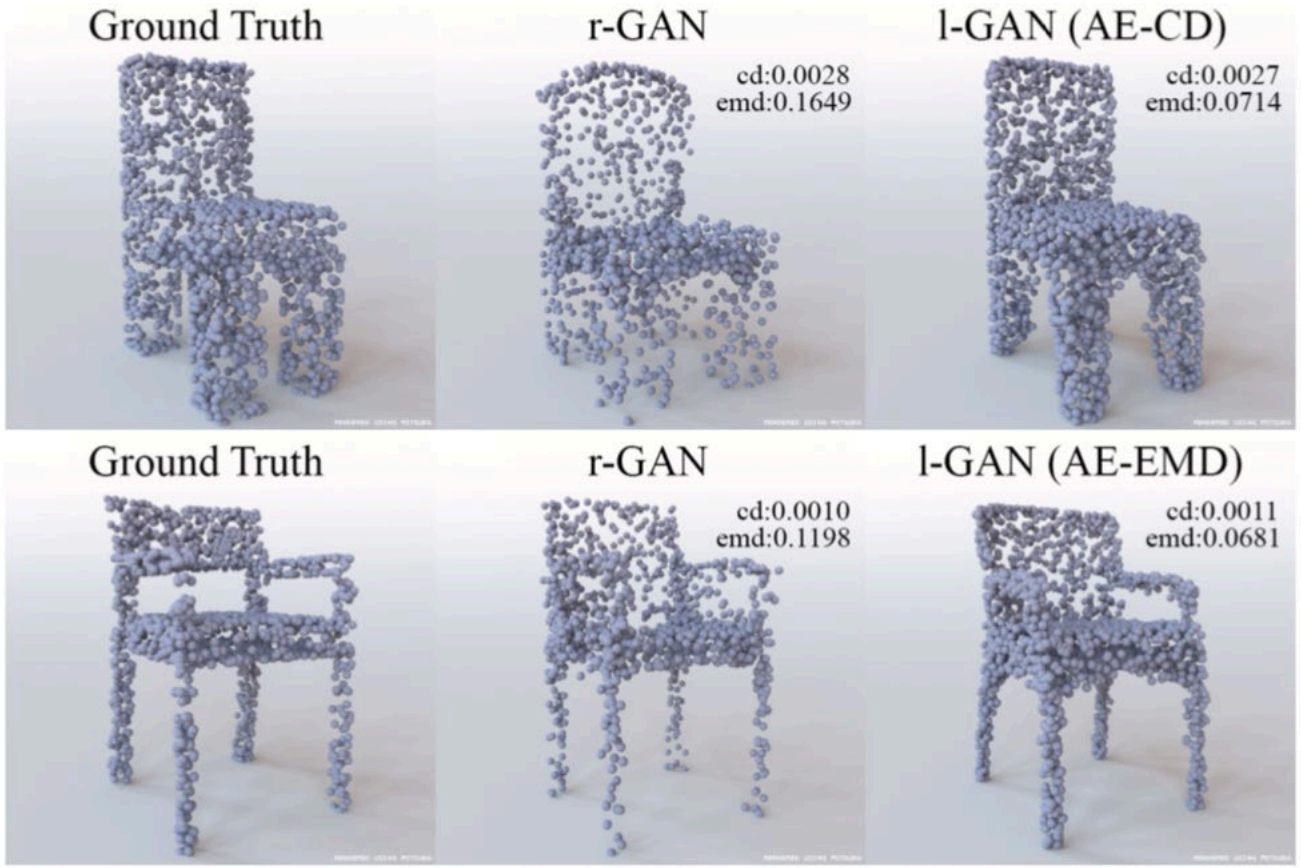


Figure 1: The generator of 3D Generative Adversarial Networks (3D-GAN)

Figure 2: Shapes synthesized by 3D-GAN

Wu et. al., **Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling**, NeurIPS 2016

Point Cloud GANs



ICML, 2018, Learning Representations and Generative Models for 3D Point Clouds, Panos Achlioptas, et. al.

Flow-based 3D Generative Model

Discrete Point Flow Networks
From Univ. Grenoble Alpes

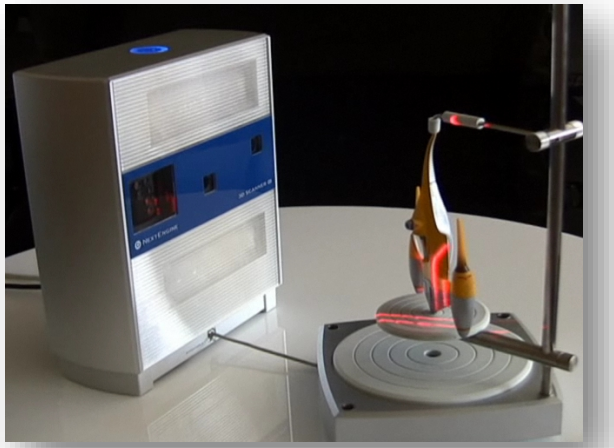
PointFlow (continuous normalizing flow)
From Cornell

Note that bijectivity requires same dimenisionality.
From left to right: latent points to generated points

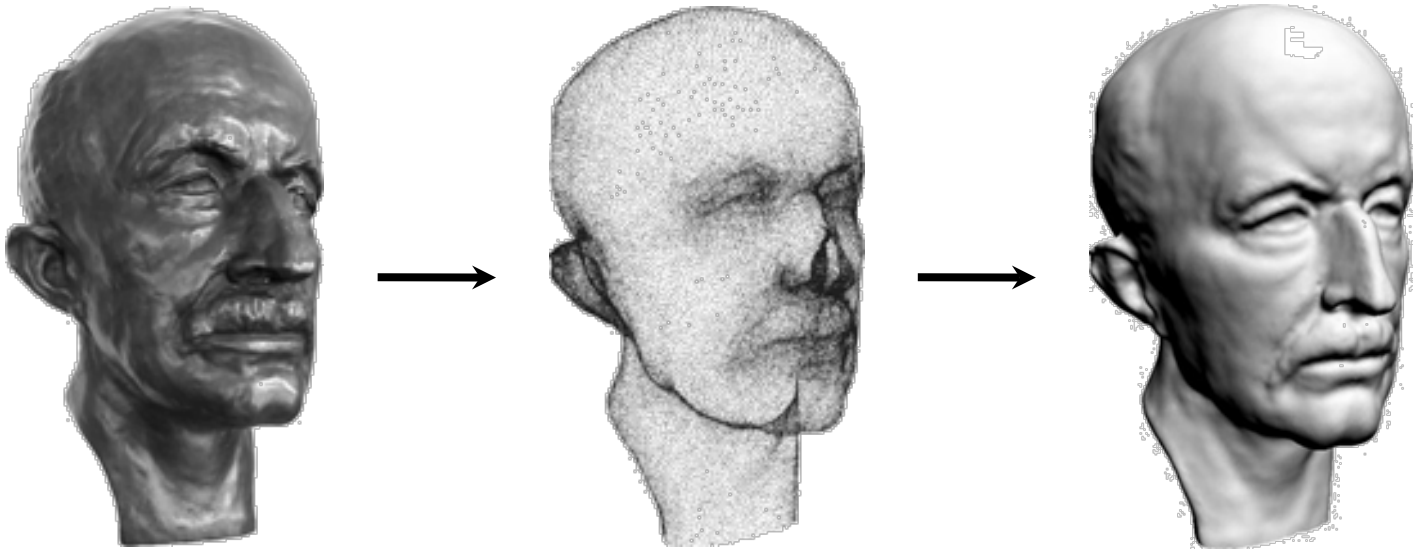
Today:
Acquired Shapes &
Geometry Processing

How Shape Models Arise: Acquisition

- Acquired shapes:



From Point Clouds to Surfaces



physical model

acquired point cloud

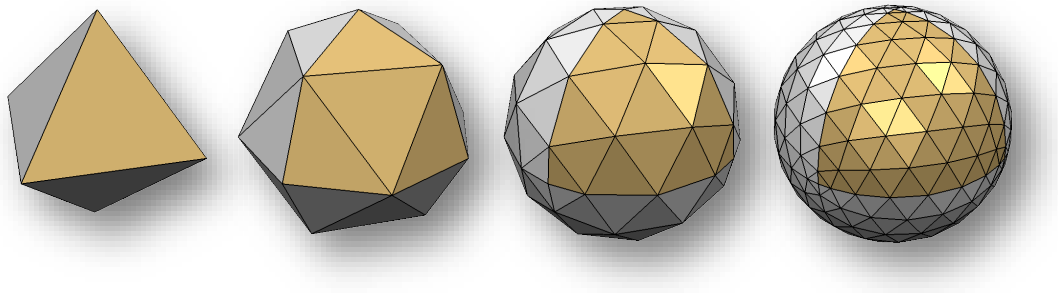
Reconstructed mesh or CAD 3D model

Acquired Shapes Overview I

- Shape acquisition
  - geometric 3D models derived from 3D scanners or other sensors (e.g., cameras)
- Geometry processing
  - techniques and algorithms for manipulating such raw geometric data to transform them into useful representations
- Intermediate geometry representation: triangle meshes
  - main questions:
    - why are triangle meshes a suitable representation for geometry processing?
    - what are the central processing algorithms?
    - how can they be implemented efficiently?

Acquired Shapes Overview II

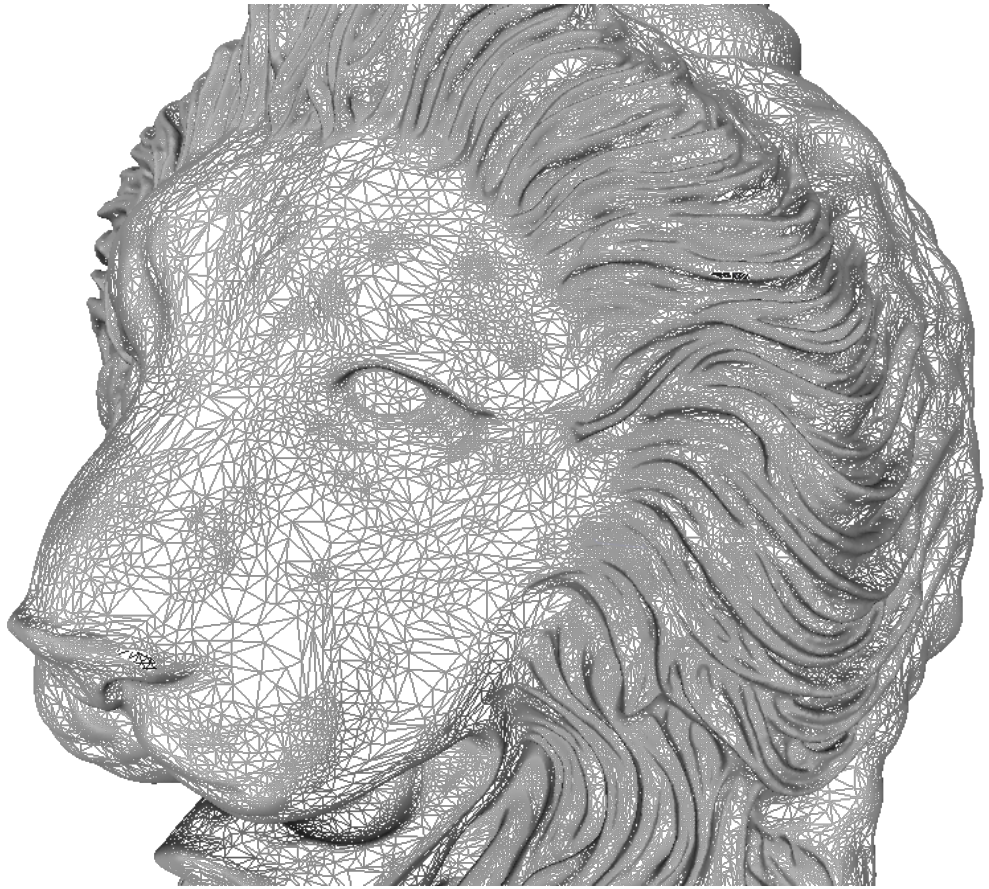
- Triangle meshes are splined surfaces
  - triangular patch surfaces of degree 1
- Triangle meshes can be big (1 billion vertices)
  - need efficient techniques and algorithms for manipulating such acquired geometric shapes
- Graphics hardware (GPUs) are able to consume meshes efficiently



Geometry Processing

What is Geometry Processing About?

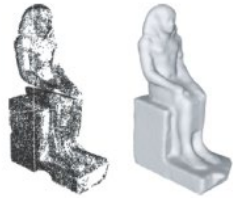
- Acquiring
- Analyzing/Repairing/Improving
- Manipulating



3D Models

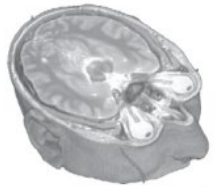
A Geometry Processing Pipeline: Low Level Algorithms

Input Data



Range-Scan

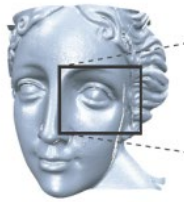
CAD

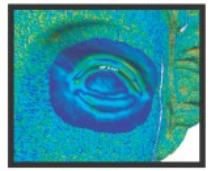


Tomography

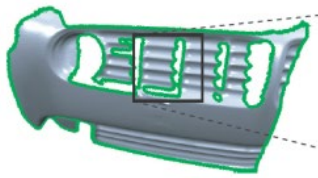
Analysis of surface quality

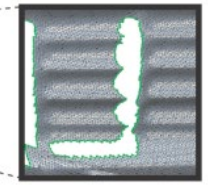
Surface smoothing for noise removal

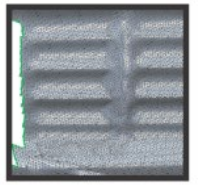




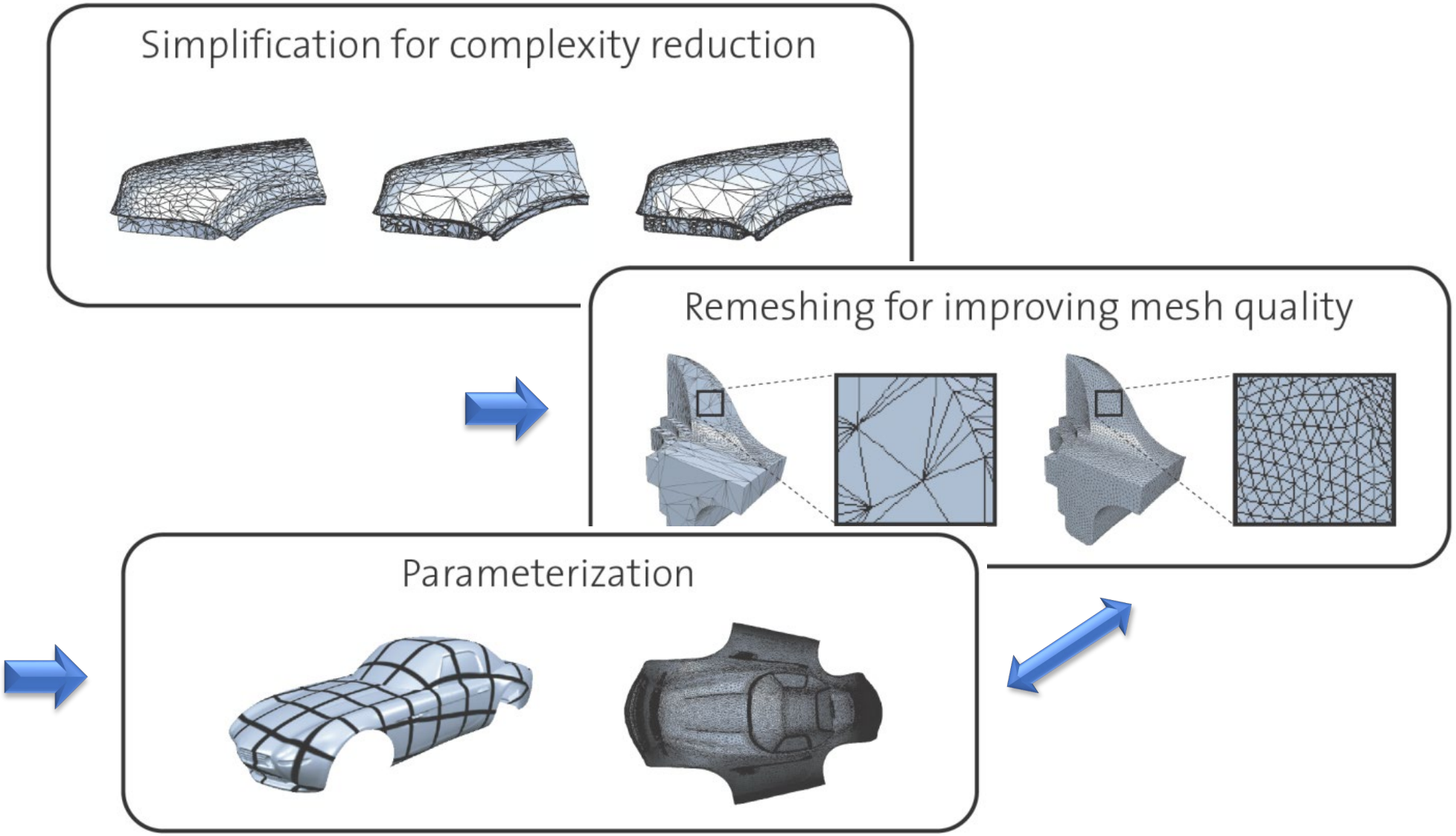
Removal of topological and geometrical errors



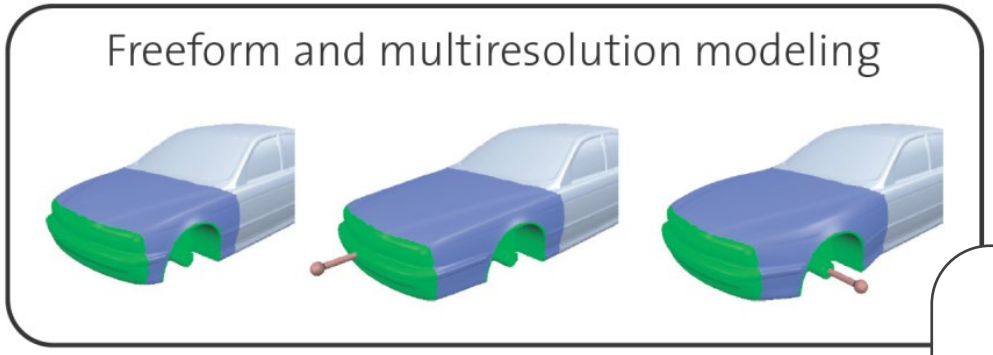




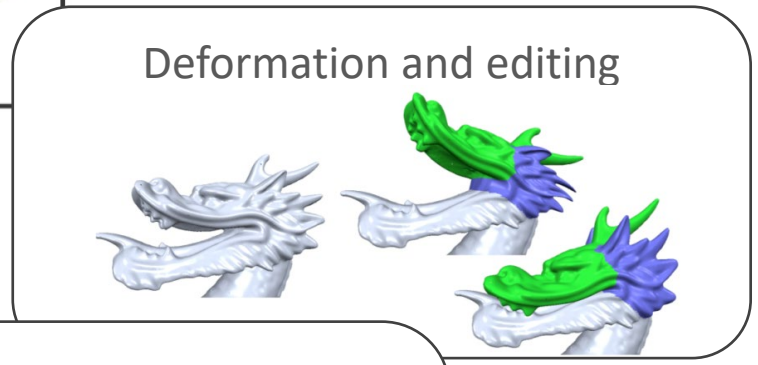
A Geometry Processing Pipeline: Intermediate Algorithms

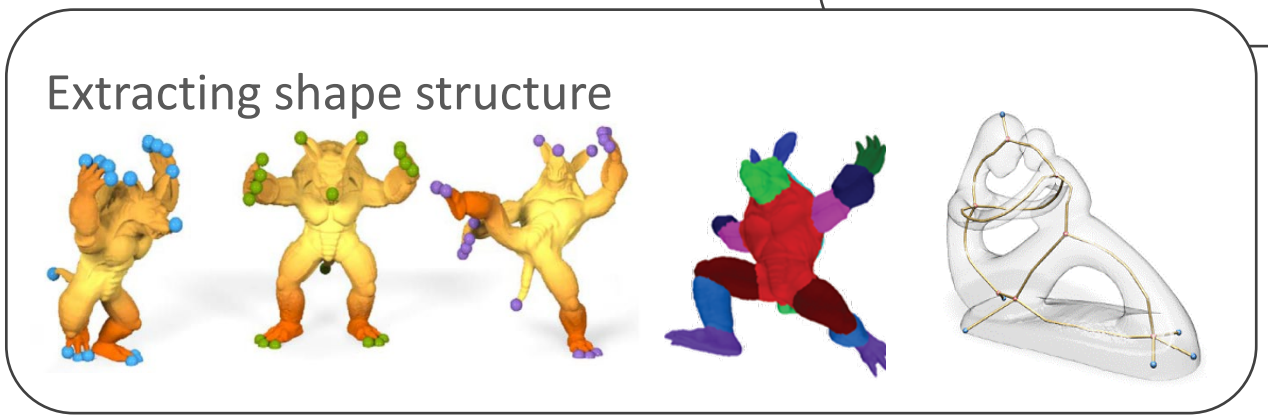


A Geometry Processing Pipeline: High Level Algorithms



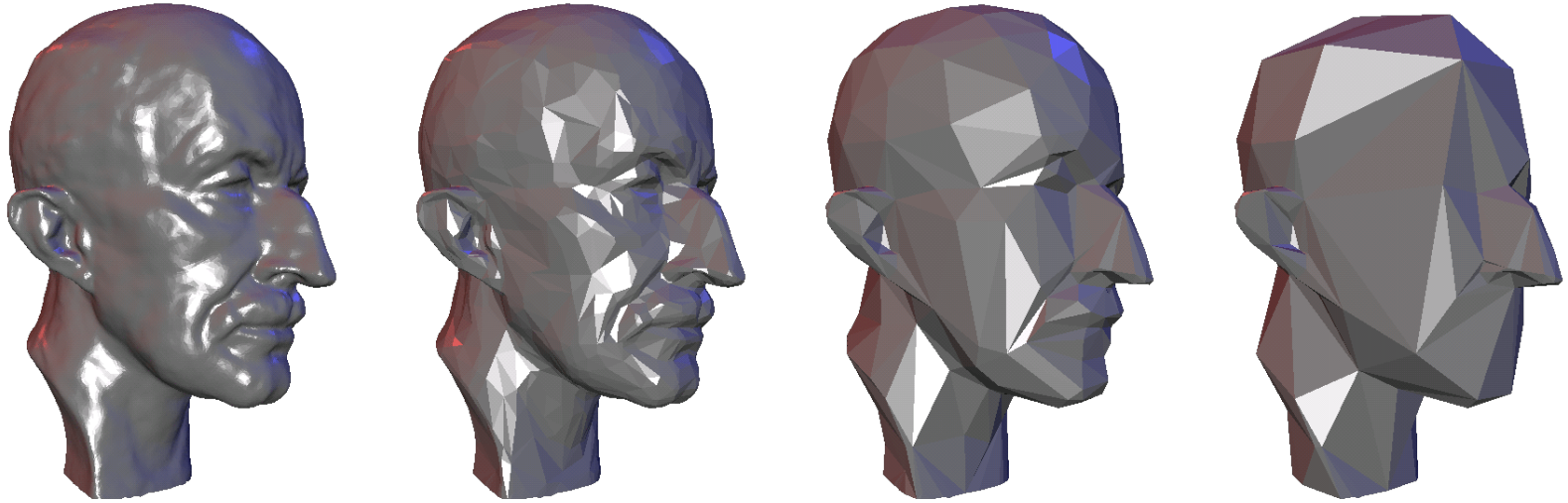
CAD model generation

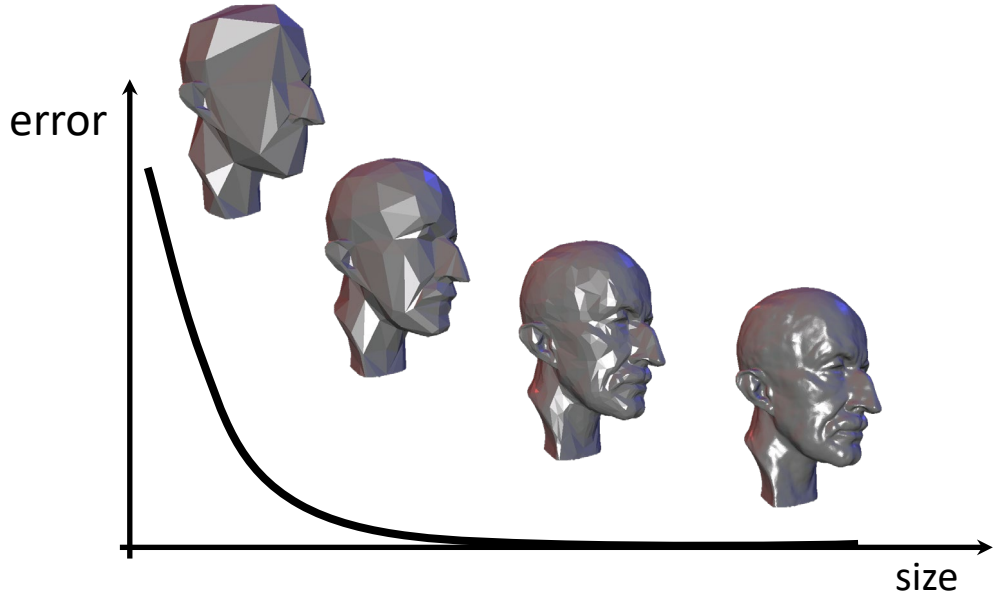




- keypoint extraction
- segmentation
- skeleton inference

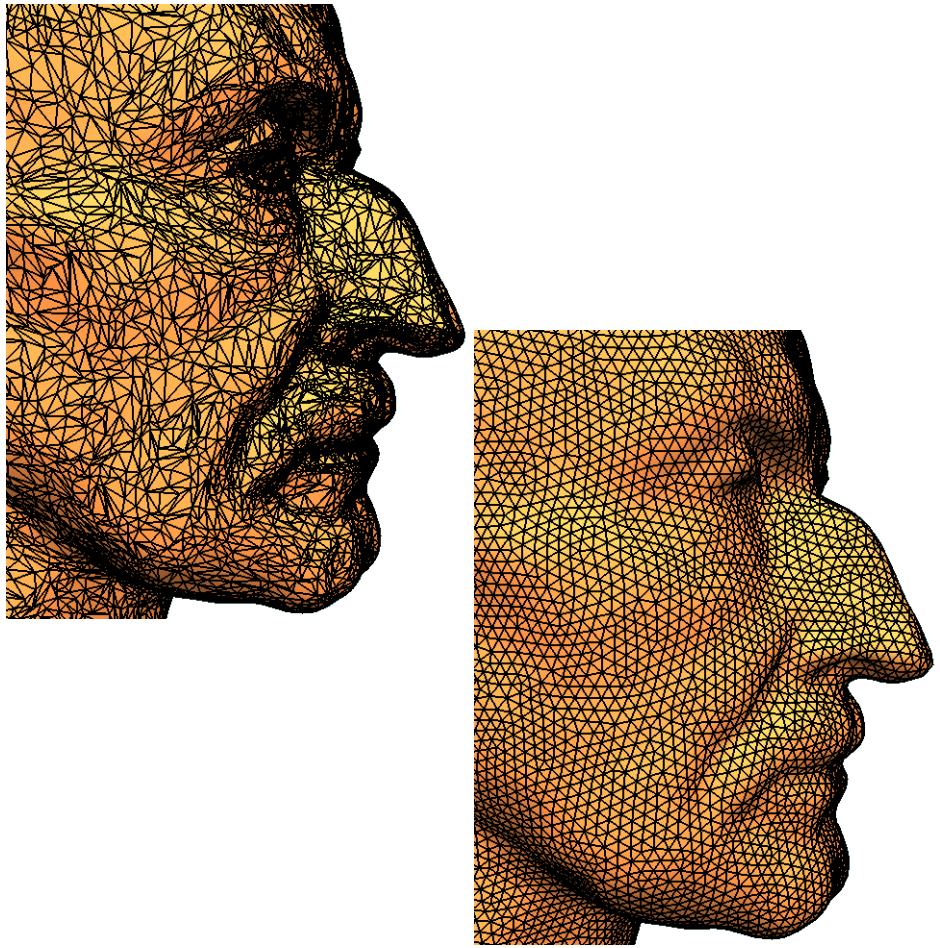
Always Trade-Offs for a Representation



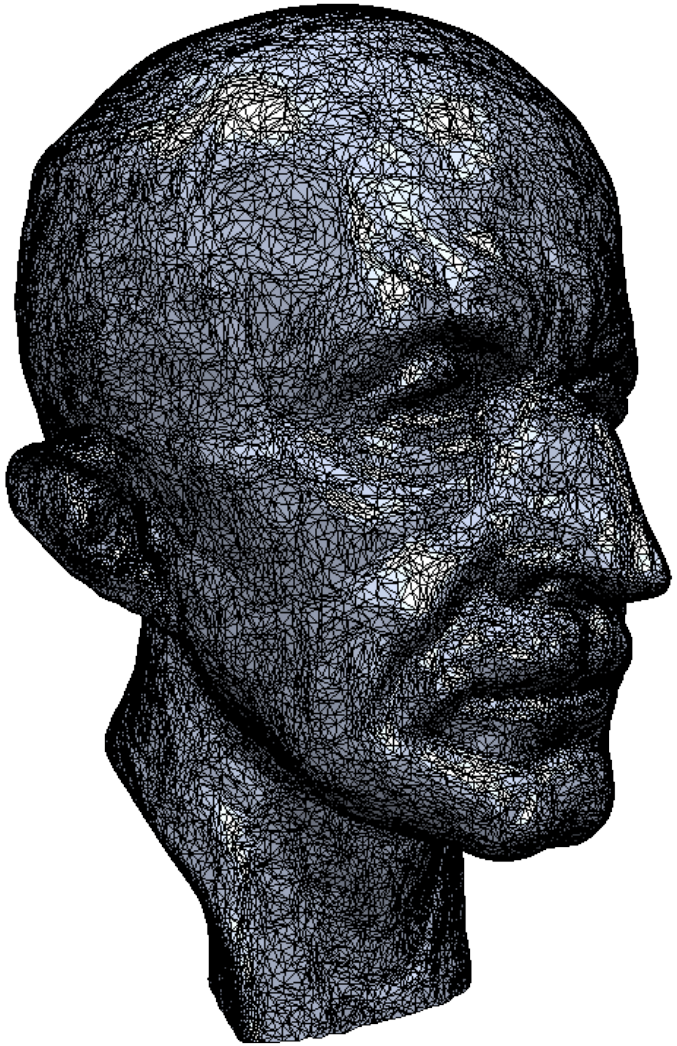


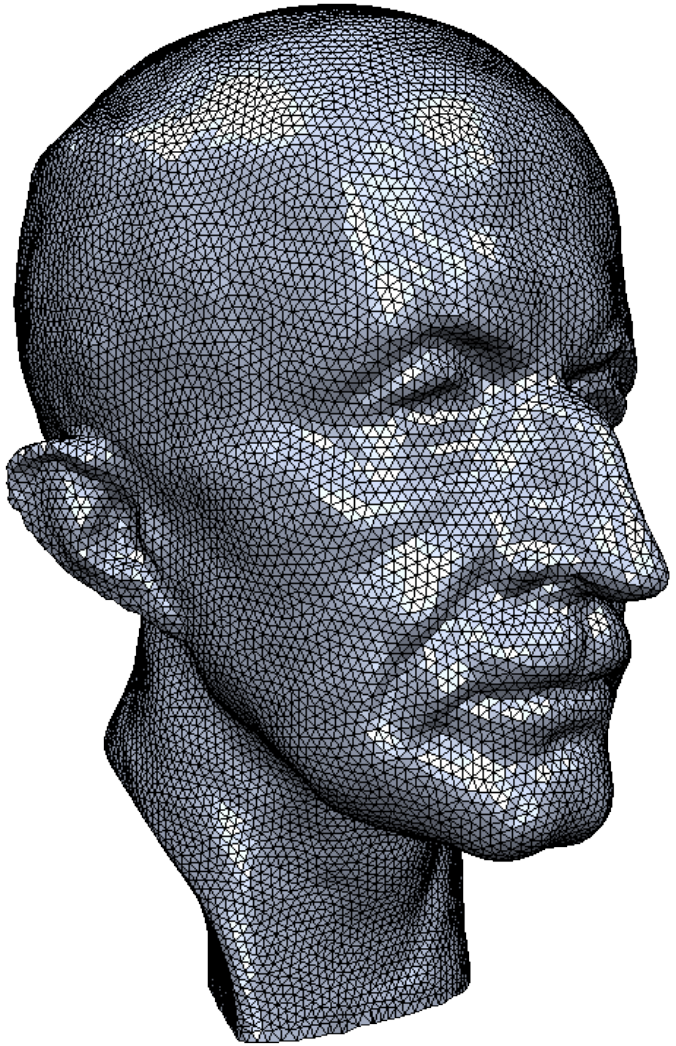
Mesh Quality Criteria

- Smoothness
  - Low geometric noise
- Adaptive tessellation
  - Low complexity
- Triangle shape
  - Numerical robustness



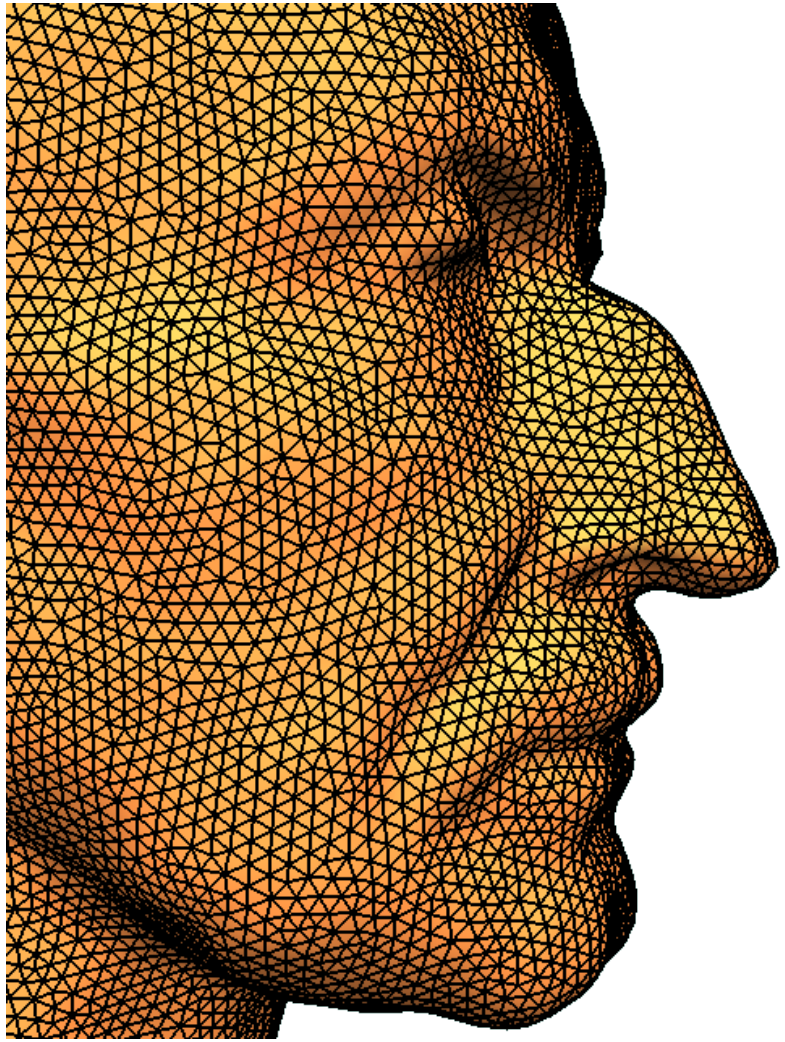
What is a Good Mesh?





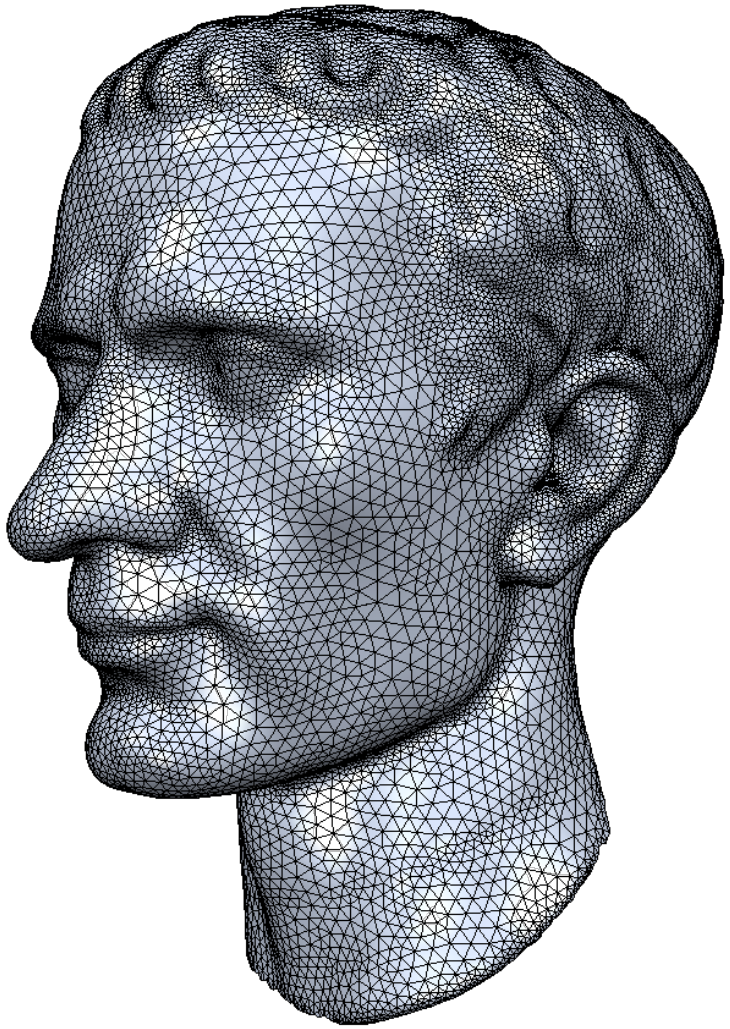
What is a Good Mesh?

- Equal edge lengths
- Equilateral triangles
- Valence close to 6



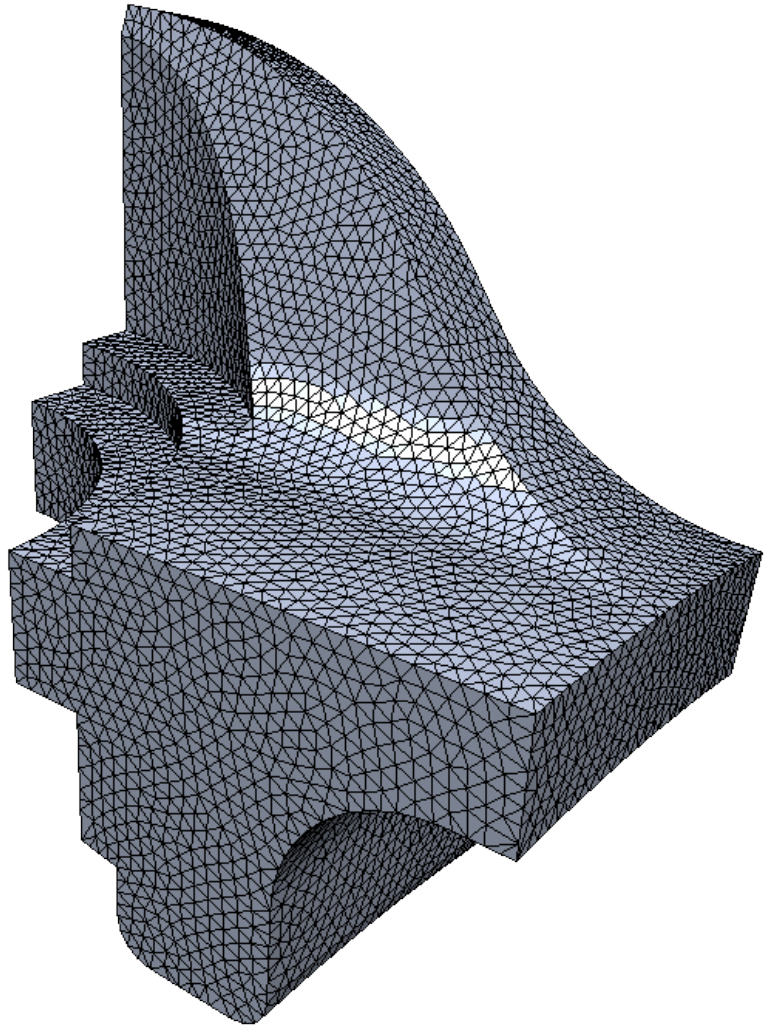
What is a Good Mesh?

- Equal edge lengths
- Equilateral triangles
- Valence close to 6
- Uniform vs. adaptive sampling



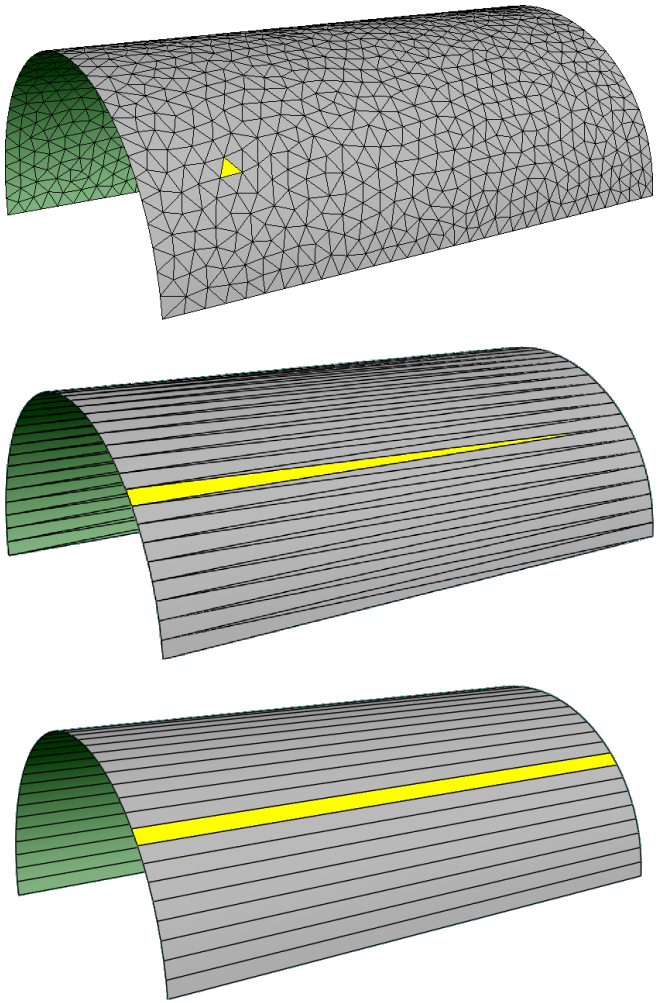
What is a Good Mesh?

- Equal edge lengths
- Equilateral triangles
- Valence close to 6
- Uniform vs. adaptive sampling
- Feature preservation



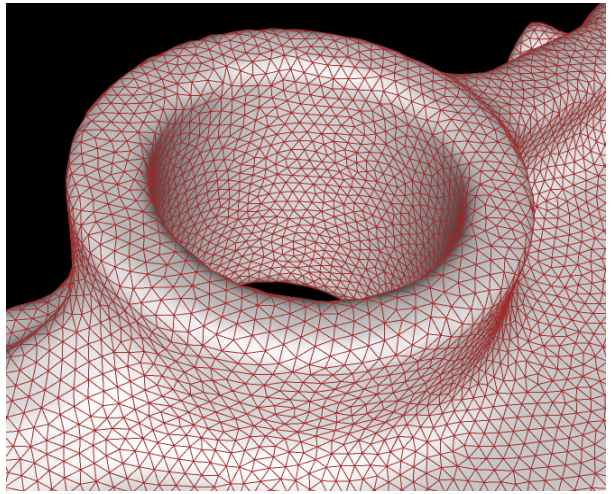
What is a Good Mesh?

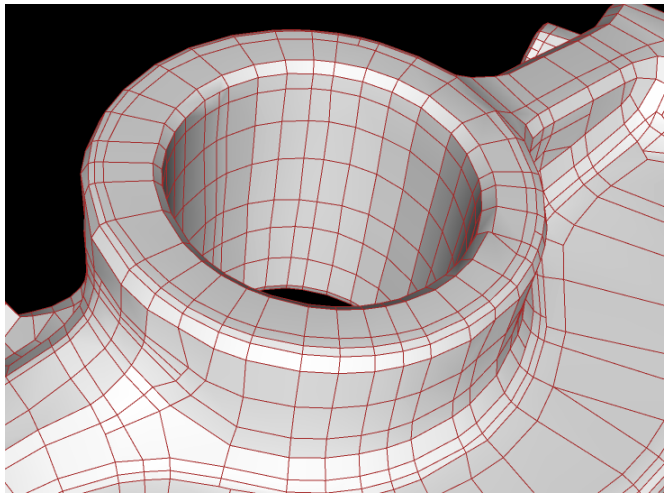
- Equal edge lengths
- Equilateral triangles
- Valence close to 6
- Uniform vs. adaptive sampling
- Feature preservation
- Alignment to curvature lines
- Isotropic vs. anisotropic



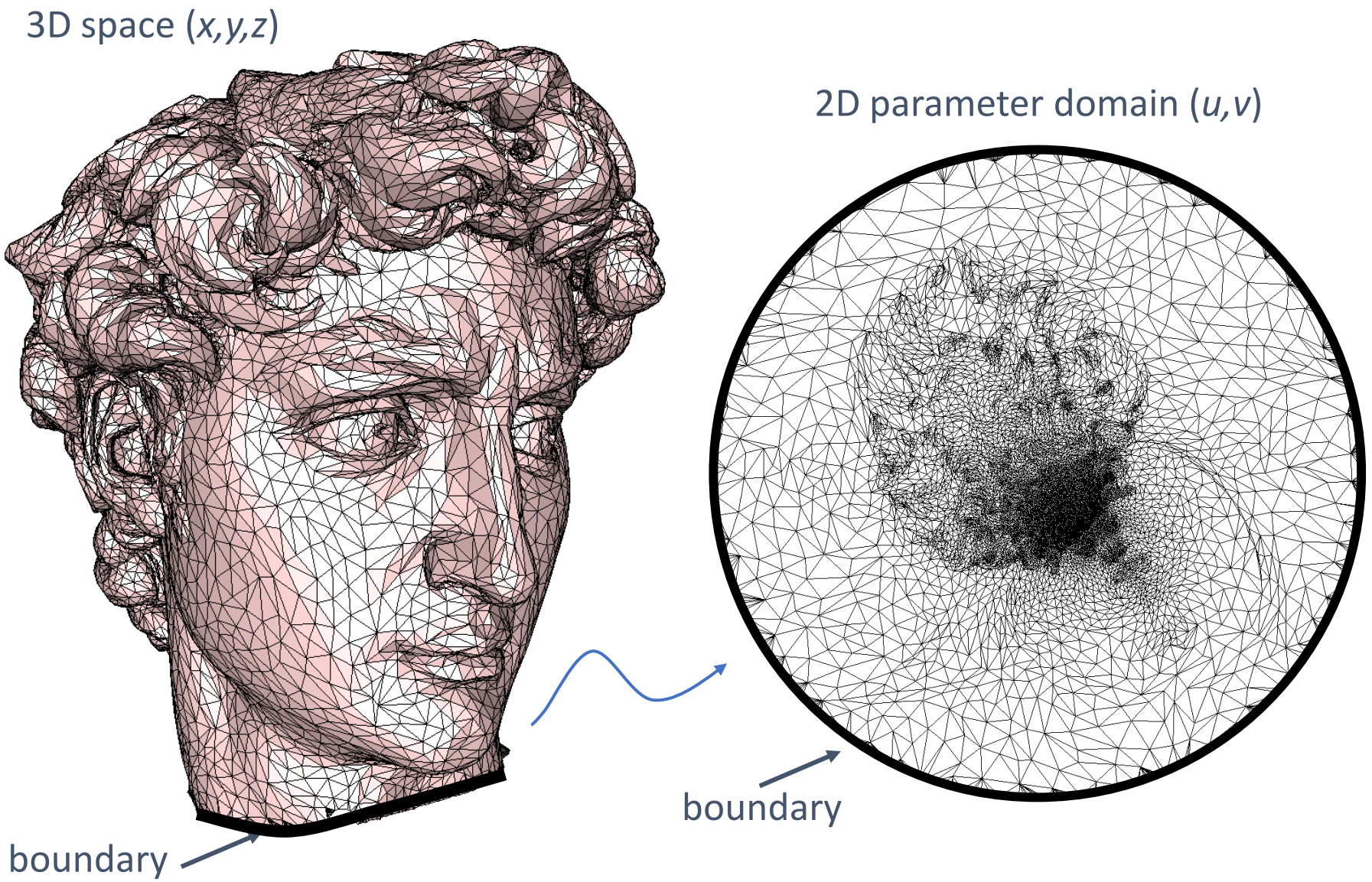
What is a Good Mesh?

- Equal edge lengths
- Equilateral triangles
- Valence close to 6
- Uniform vs. adaptive sampling
- Feature preservation
- Alignment to curvature lines
- Isotropic vs. anisotropic
- Triangles vs. quadrangles





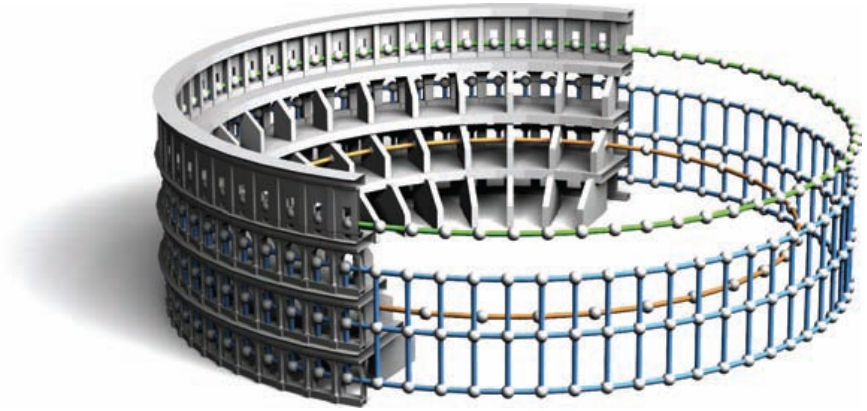
Key Topics: Parametrization

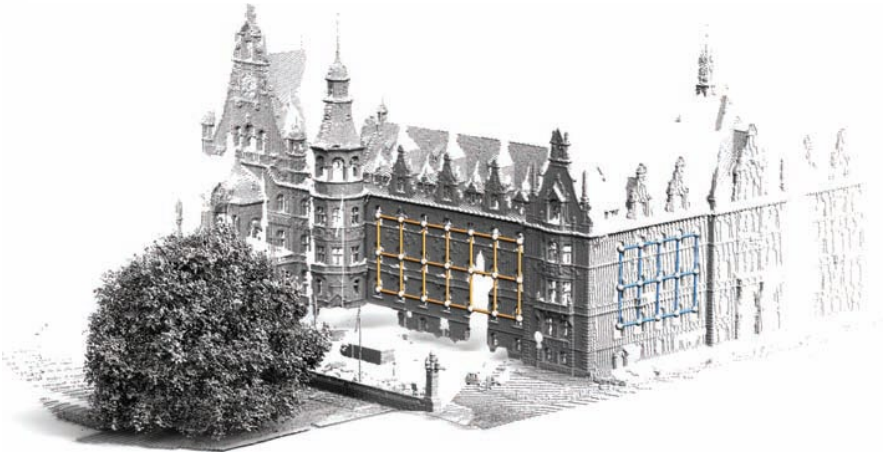


Application -- Texture Mapping

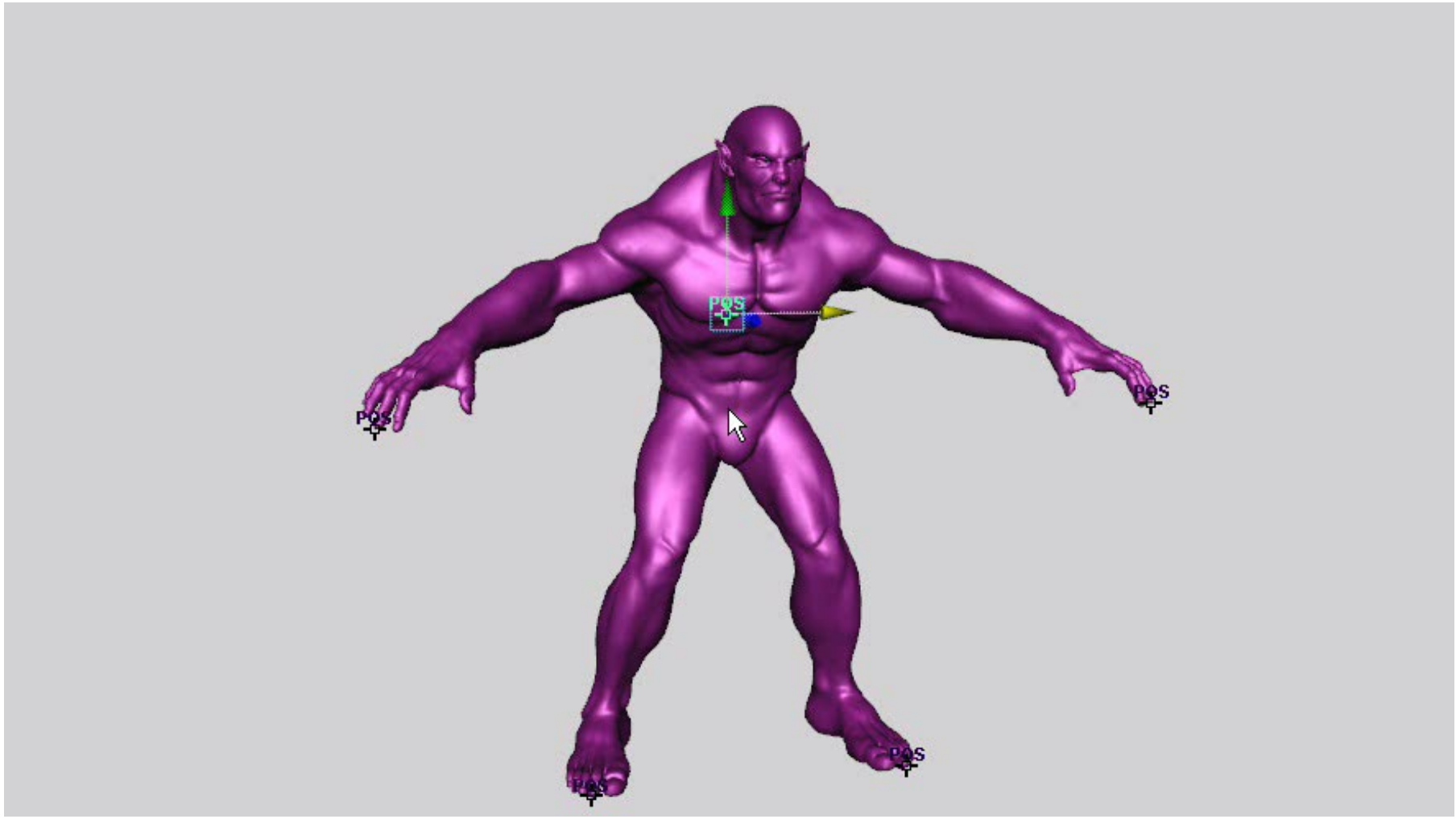
Key Topics: Segmentation

Key Topics: Symmetry Detection





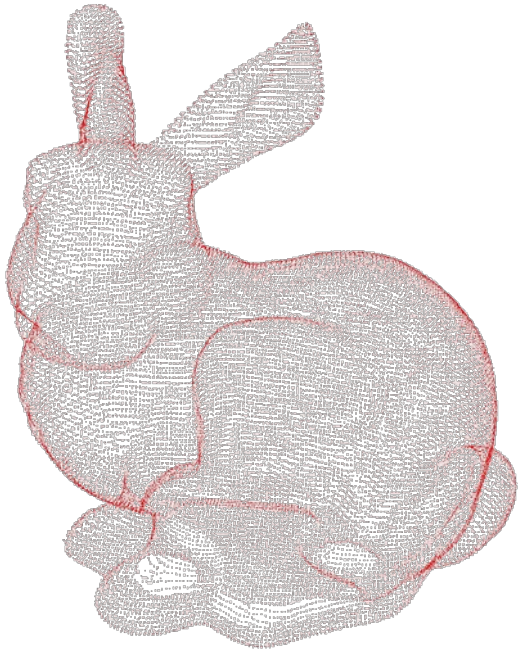
Key Topics: Deformation / Manipulation



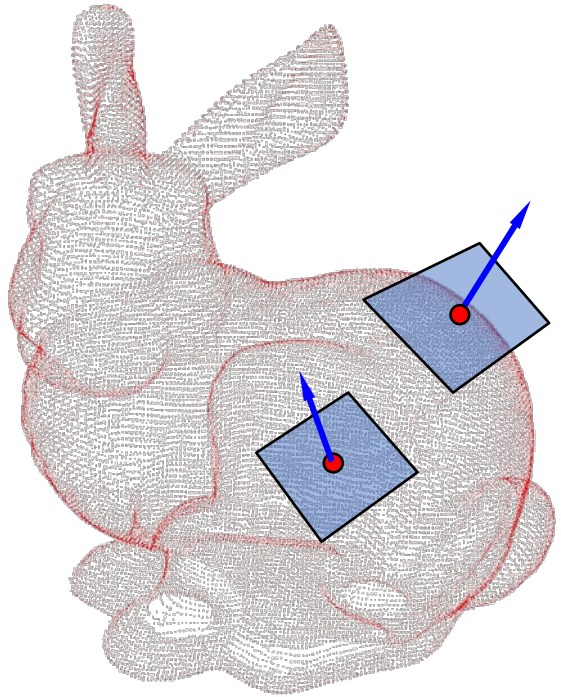
Point Clouds

Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals



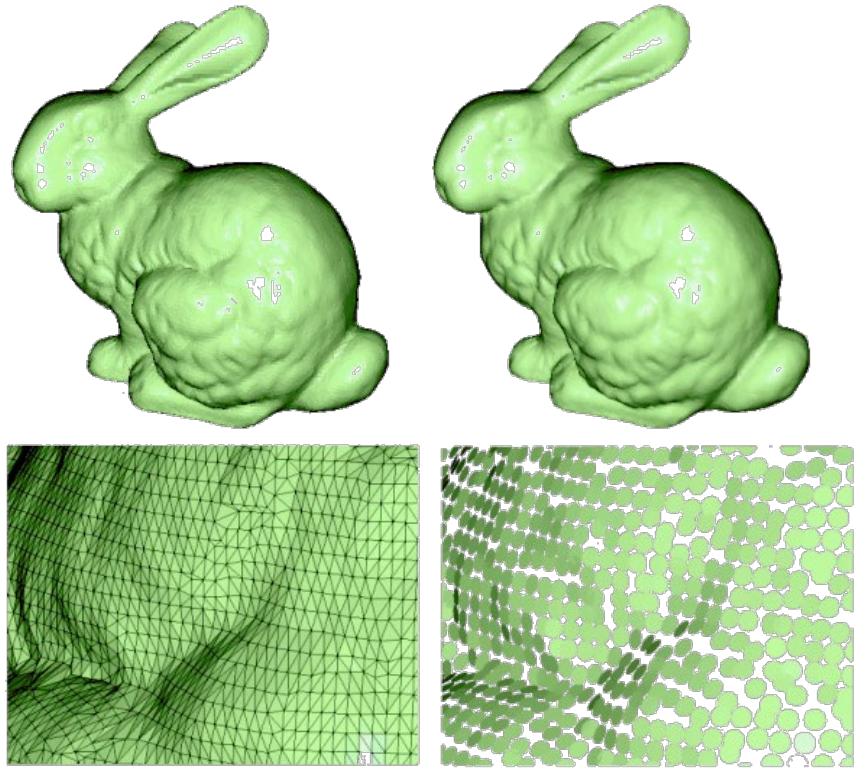
Stanford bunny



Stanford Bunny

Point Clouds

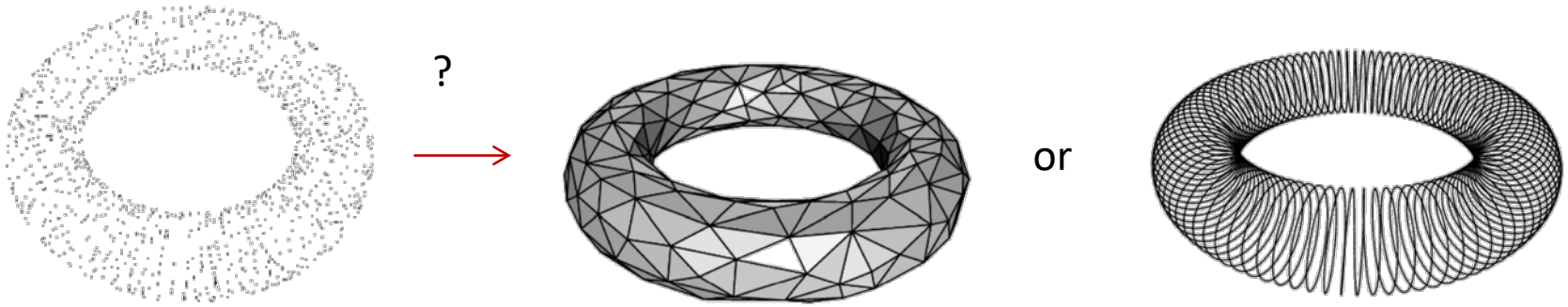
- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**



Filip van Bouwel

Point Clouds

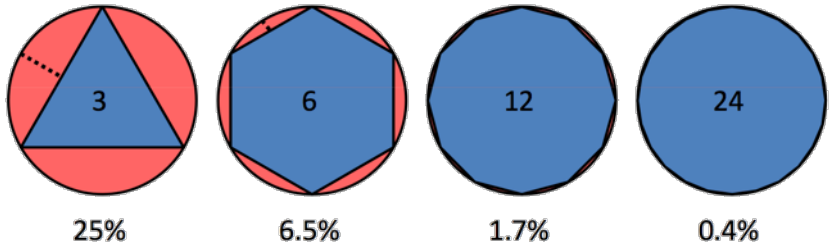
- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information



Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information
  - weak approximation power:

- Piecewise linear approximation
  - Error is $O(h^2)$

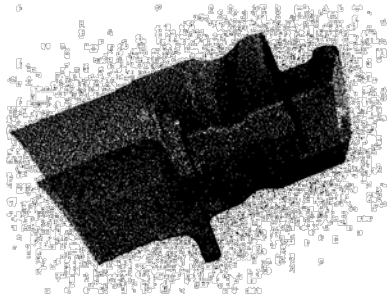


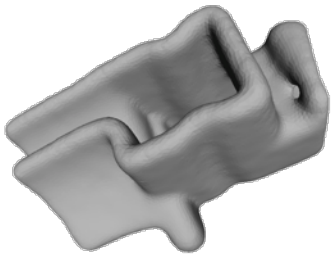
Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information
  - weak approximation power: $O(h)$ for point clouds
    - need *square* number of points for the same approximation power as meshes

Point Clouds

- Simplest representation: **only points,** no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**
- Several limitations:
  - **no** Simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information
  - weak approximation power
  - noise and outliers

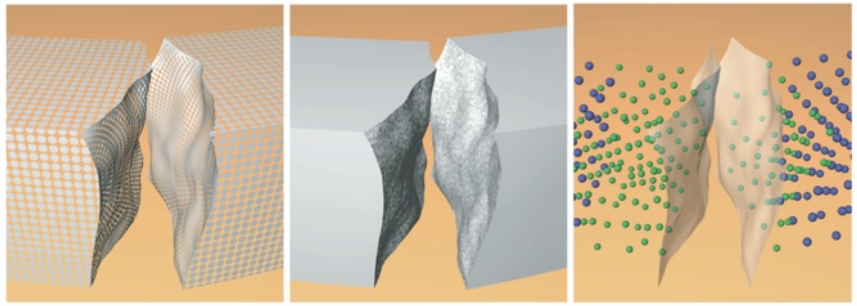




Why Point Clouds?

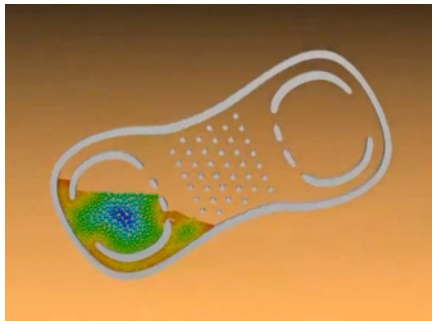
1) Typically, that's the only thing that's available from a large class of sensors
2) Isolation: sometimes, easier to handle (esp. in hardware).

Fracturing Solids



Meshless Animation of Fracturing Solids
Pauly et al., SIGGRAPH '05

Fluids

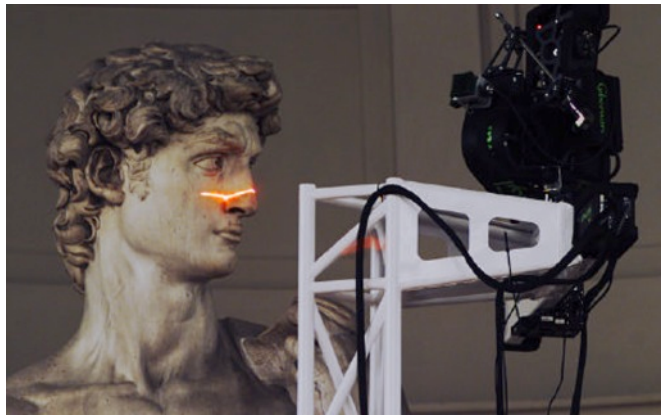


Adaptively sampled particle fluids,
Adams et al. SIGGRAPH '07

Why Point Clouds?

- Typically, that's the only thing that's available

  Nearly all 3D scanning devices produce point clouds



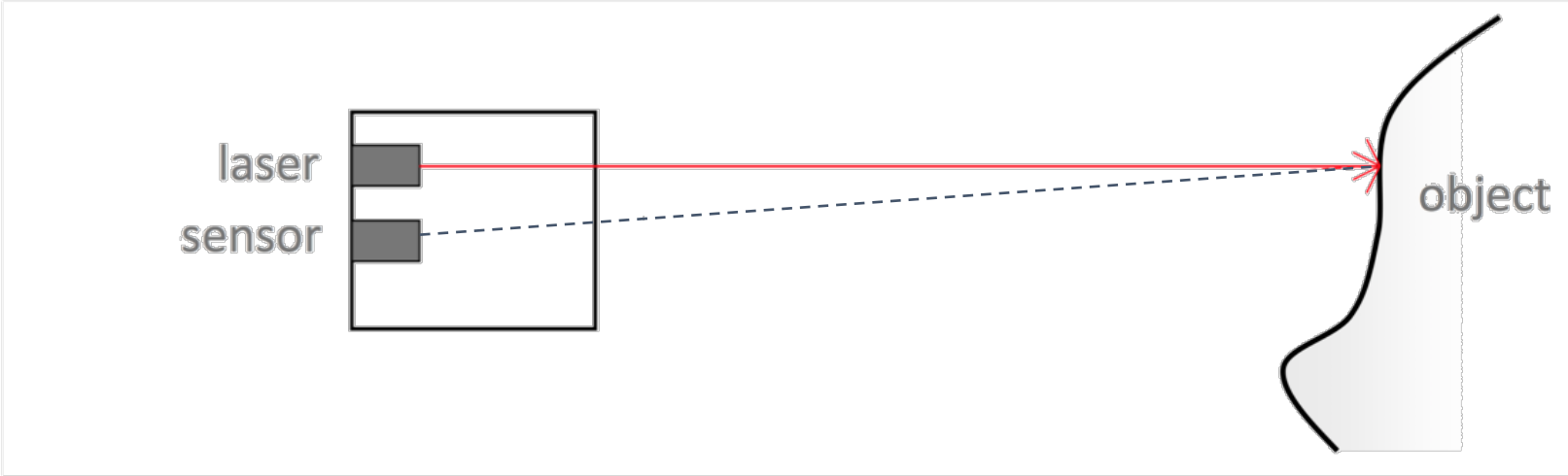
Surface Scanning

Surface Scanning Basics

Major types of 3D scanners

- **Range (emission-based) scanners**
  - Time-of-flight laser scanner
  - Phase-based laser scanner
- **Triangulation**
  - Laser line sweep
  - Structured light
- • **Stereo / computer vision**
  - Passive stereo
  - Active stereo / space-time stereo

Time of Flight Scanners



1. Emit a short short pulse of laser
2. Capture the reflection.
3. Measure the time it took to come back.

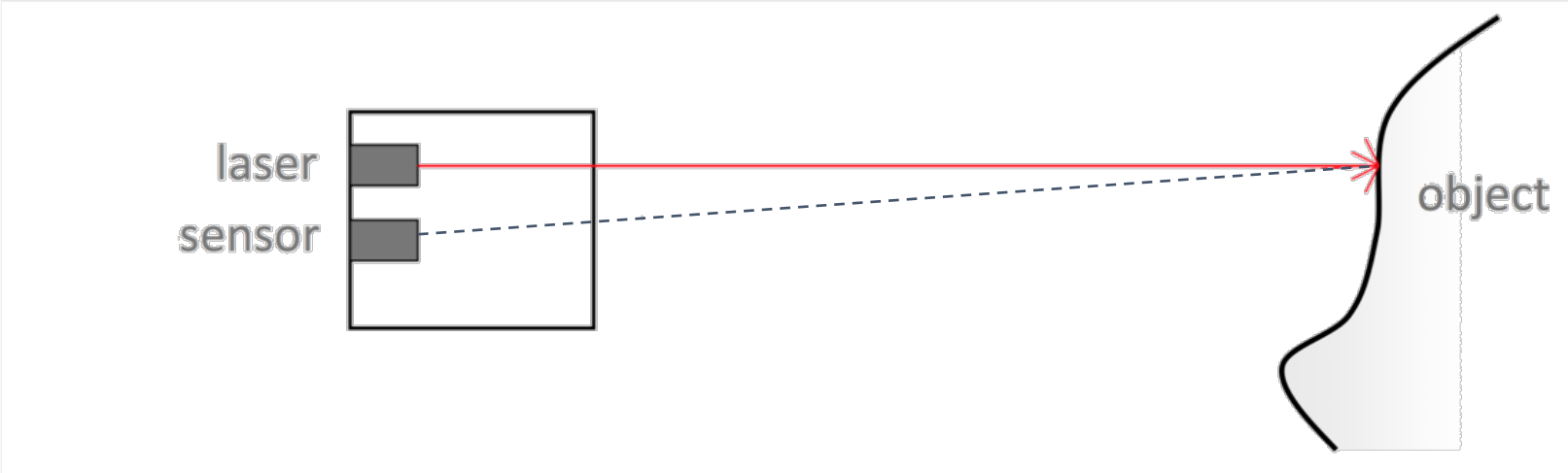
$$D = \frac{cT}{2}$$

c: speed of light (≈ 299 792 458 m/s)

Need a very fast clock: e.g. 1GHz achieves 0.15m (15cm) accuracy.

Guinness record: AMD Bulldozer, 8.429 GHz

Time of Flight Scanners

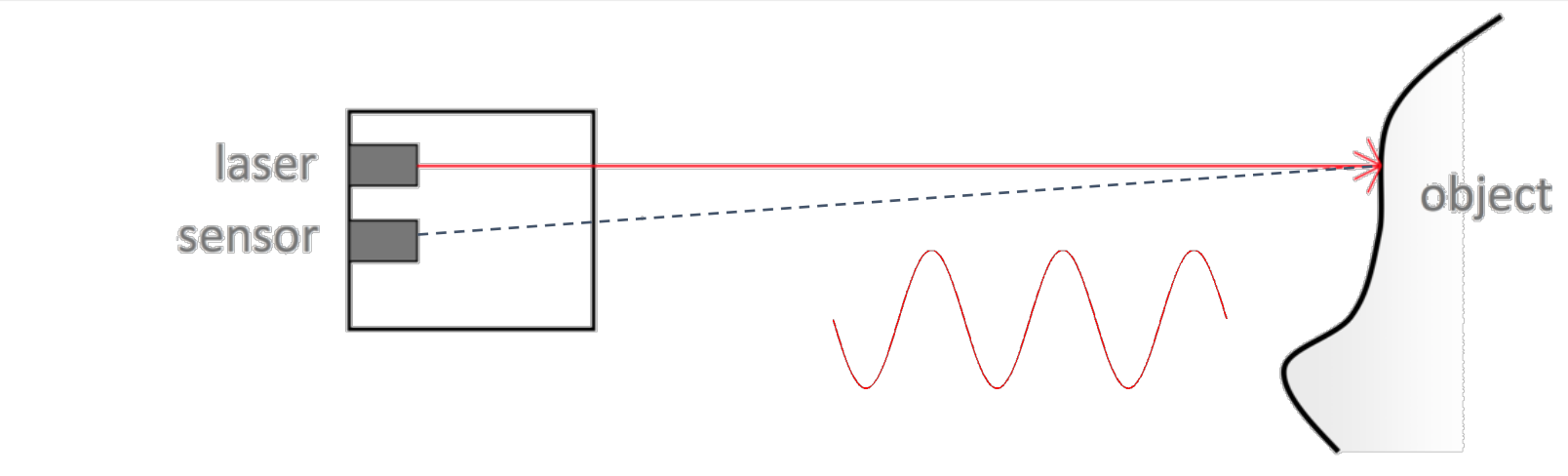


1. Emit a short short pulse of laser
2. Capture the reflection.
3. Measure the time it takes to come back.
4. Need a very fast clock.
5. Main advantage: can be done over long distances.
6. Used in terrain scanning.

Time of Flight Scanners

source: Michael Wand

Phase-Based Range-Scanners



1. Instead of a pulse, emit a continuous **phase-modulated** beam
2. Capture the reflection
3. Measure the **phase-shift** between the output and input signals

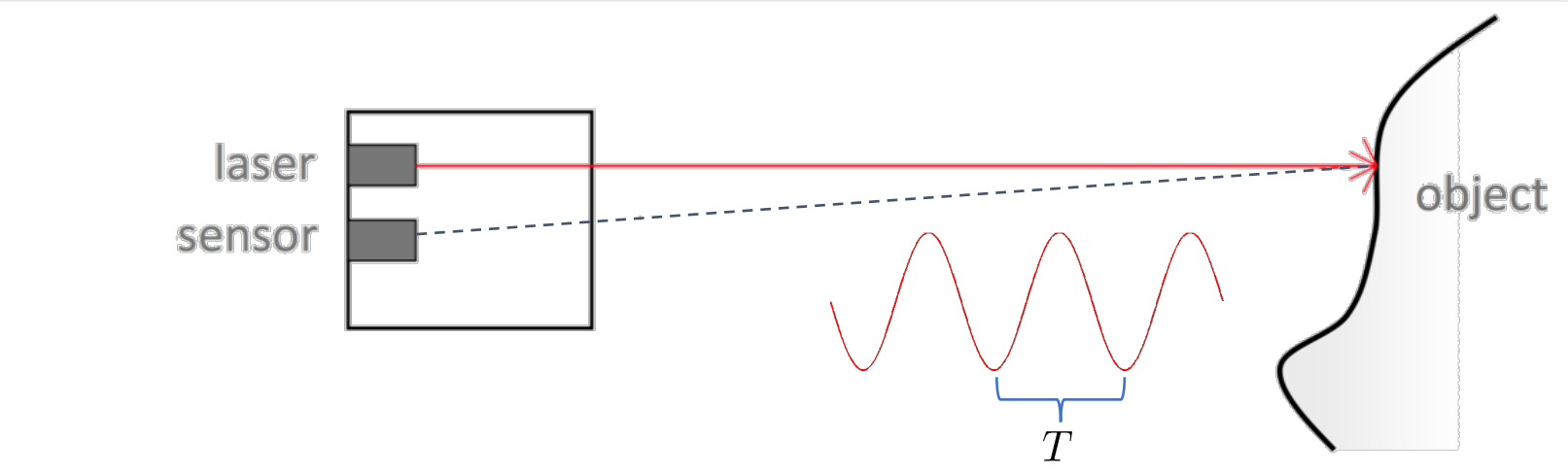
Output: $e(t) = e \cdot \left[1 + \sin\left(\frac{F}{2\pi} \cdot t\right)\right]$

F: Modulation frequency
e: emitted mean power

Input: $s(t) = e \cdot \left[1 + \sin\left(\frac{F}{2\pi} \cdot t - \phi\right)\right]$

φ: Phase delay arising from the object's distance

Phase-Based Range-Scanners

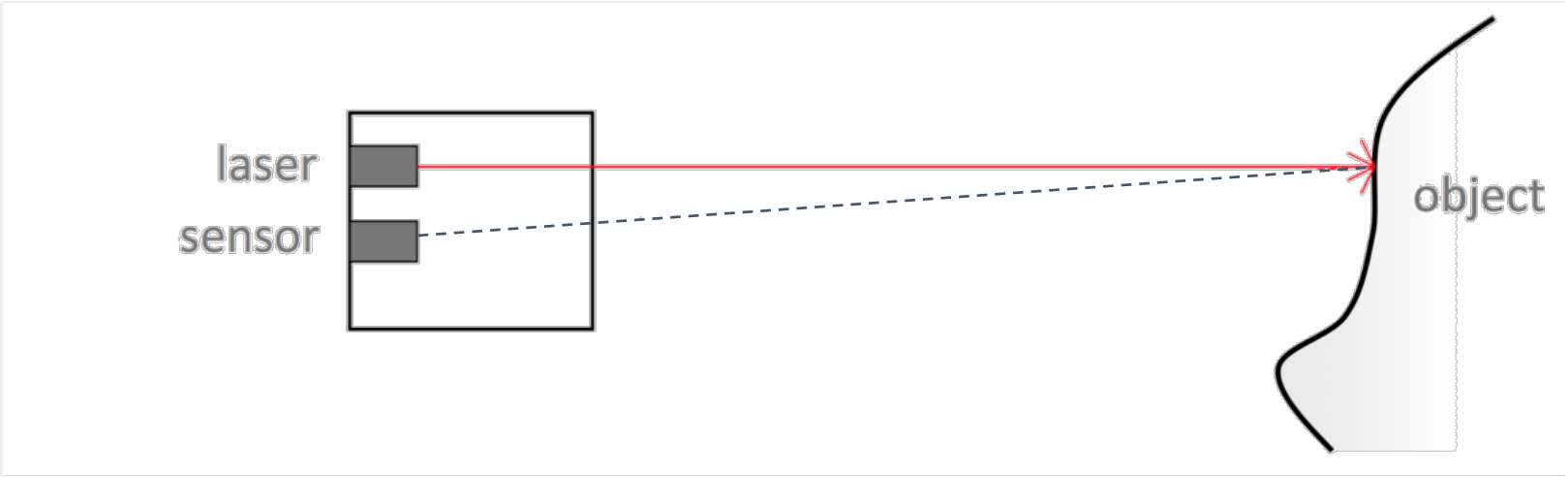


1. Instead of a pulse, emit a continuous **phase-modulated** beam
2. Capture the reflection
3. Measure the **phase-shift** between the output and input signals.
4. From the phase-shift, the distance can be computed **up to modulation period**
5. No fast clock required, **greater frequency and accuracy** but **shorter range**

e.g. 1,016,727 vs. 50,000 (ToF) points per second

up to 79 meters vs. hundreds of meters

Range-Scanners

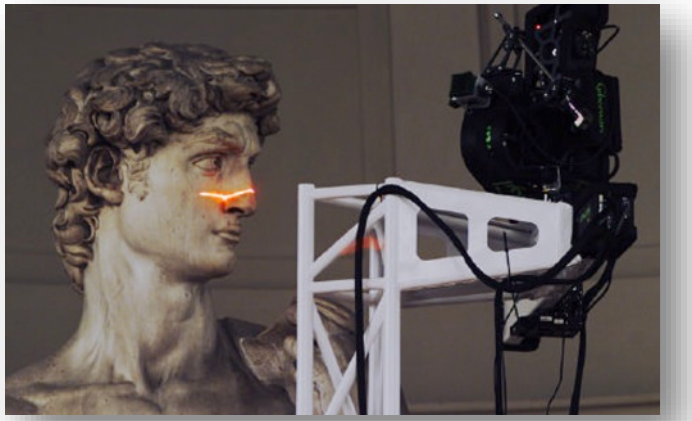


1. Typically, range scanners by themselves provide limited accuracy (noise, outliers, uneven sampling).
2. May require a lot of post-processing to get a good sampling.

Triangulation-Based Methods

1. Add a **controllable light source** (e.g., laser)
2. Add a **photometric sensor** (e.g., camera)
3. Record the projected feature position for a reference plane
4. Change in recording position can be used to recover the depth.

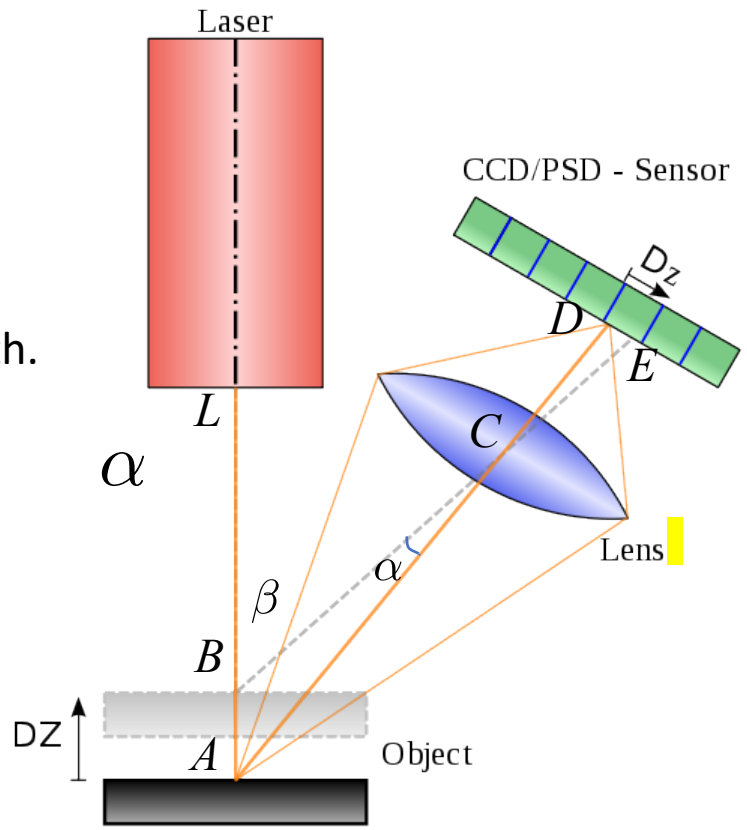
Intuition: the **depth** is related to the **shift** in the camera plane.



Triangulation-Based Methods

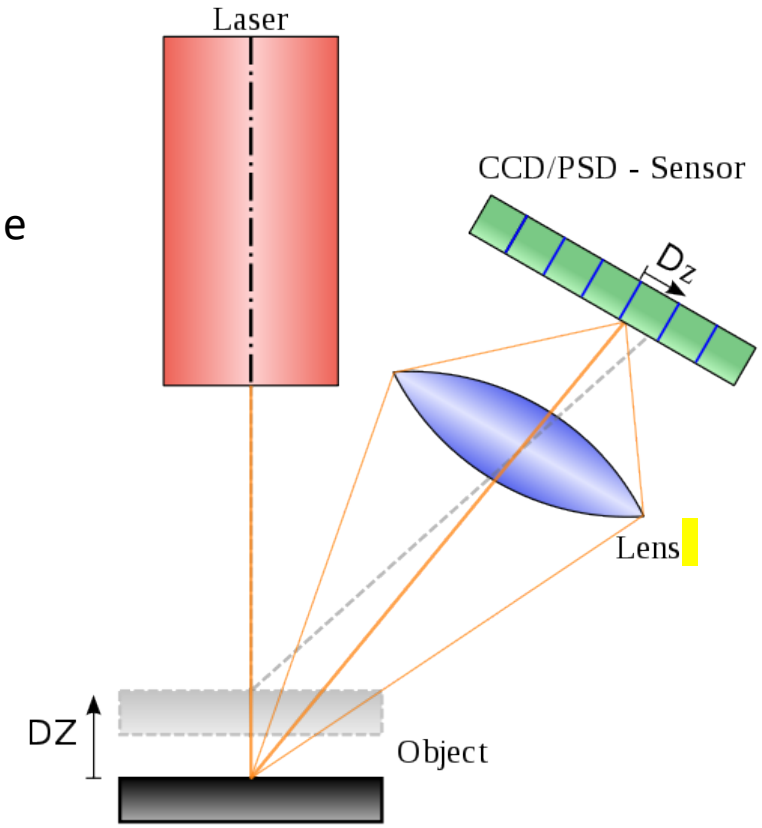
1. Add a **controllable light source** (e.g., laser)
2. Add a **photometric sensor** (e.g., camera)
3. Record the projected feature position for a reference plane
4. Change in recording position can be used to recover the depth.

    1. Using Dz, EC and $\angle CED$, compute $\angle DCE$ =
    2. Using α, β and $BC = BE - CE$, compute AB
    3. The depth $LE = LB + AB$



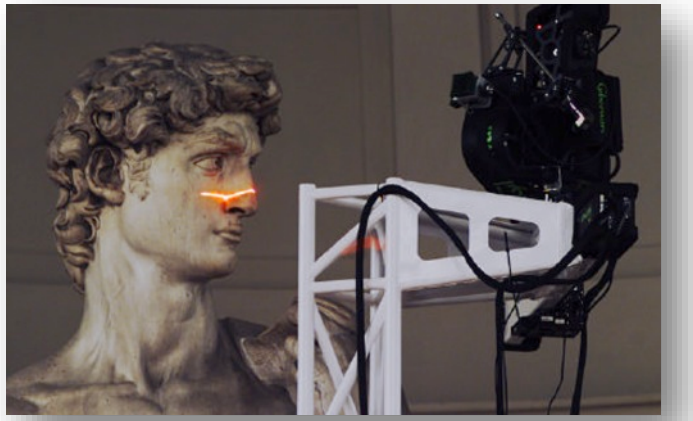
Triangulation-Based Methods

1. Add a **controllable light source** (e.g., laser)
2. Add a **photometric sensor** (e.g., camera)
3. Record the projected feature position for a reference plane
4. Change in recording position can be used to recover the depth.
5. If well-calibrated, can lead to extremely accurate depth measurements

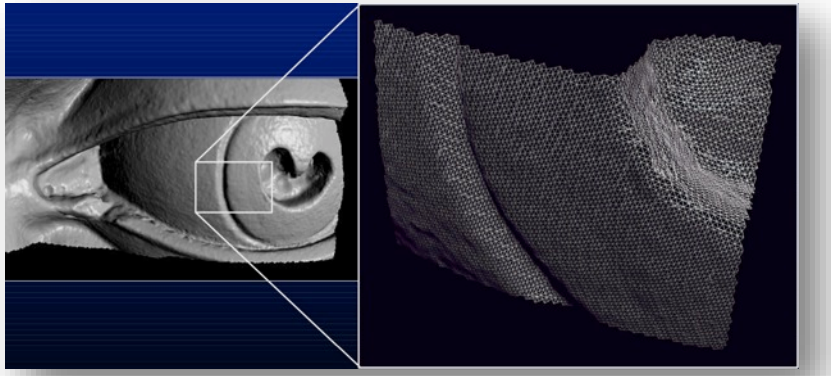


Triangulation-Based Methods

1. Add a **controllable light source** (e.g., laser)
2. Add a **photometric sensor** (e.g., camera)
3. Record the projected feature position for a reference plane
4. Change in recording position can be used to recover the depth.
5. If well-calibrated, can lead to extremely accurate depth measurements



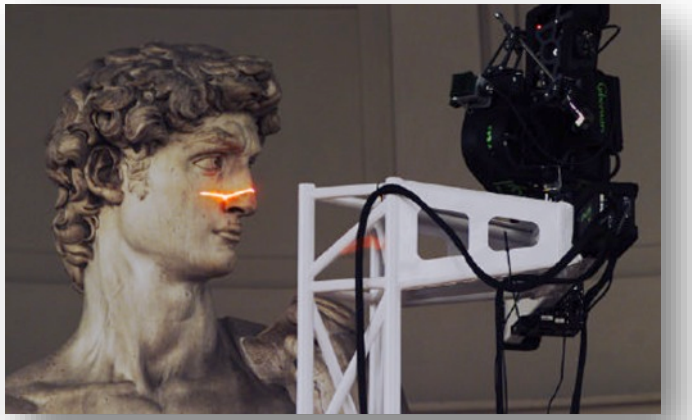
Similar technology used to scan Michelangelo's David 5m statue to 0.25mm accuracy.



David's left eye:
source Levoy et al.

Triangulation-Based Methods (Laser)

1. Add a **controllable light source** (e.g., laser)
2. Add a **photometric sensor** (e.g., camera)
3. Record the projected feature position for a reference plane
4. Change in recording position can be used to recover the depth.
5. If well-calibrated, can lead to extremely accurate depth measurements
6. Main problem: slow and expensive

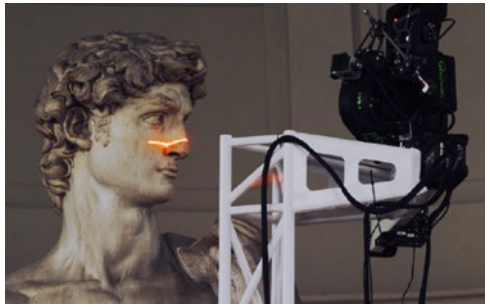


Structured-Light Scanners

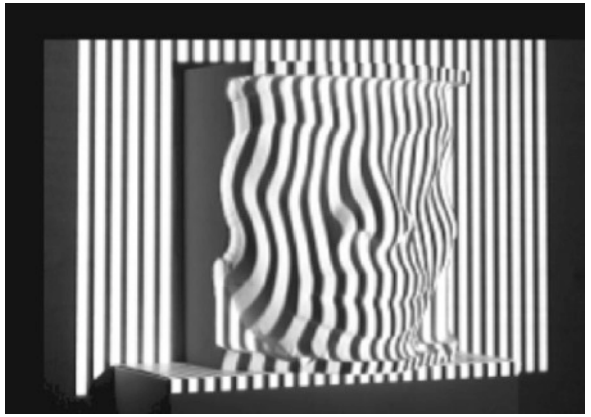
Same general idea as triangulation based scanner.

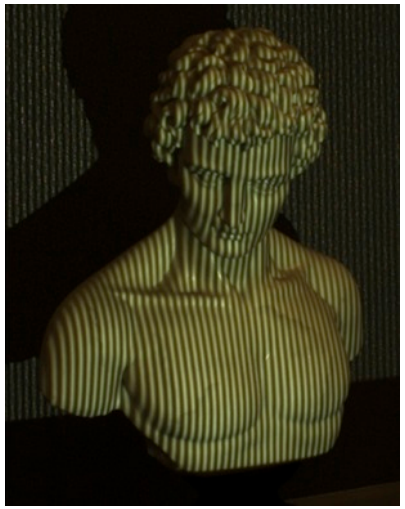
Main Idea: Replace laser with projector. Project stripes instead of sheets.

Challenge: Need to identify which (input/output) lines correspond.



vs.

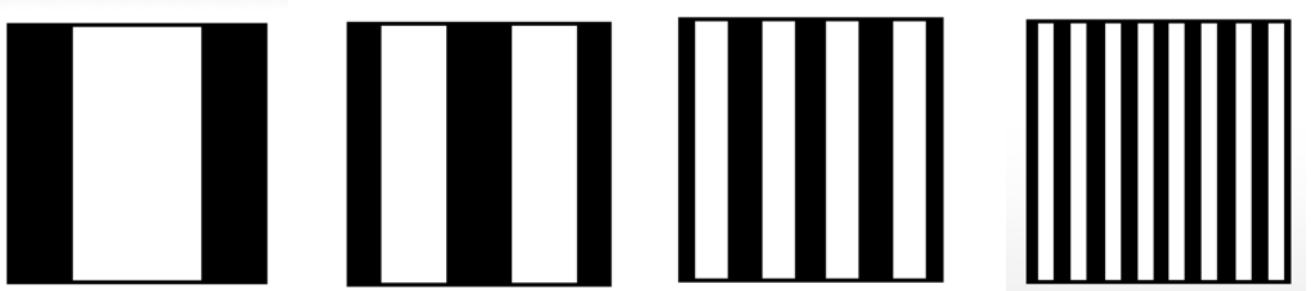




Structured-Light Scanners

Same basic idea as triangulation-based scanner. Use a projector.

Main Idea: Project multiple stripes to identify the position of a point.

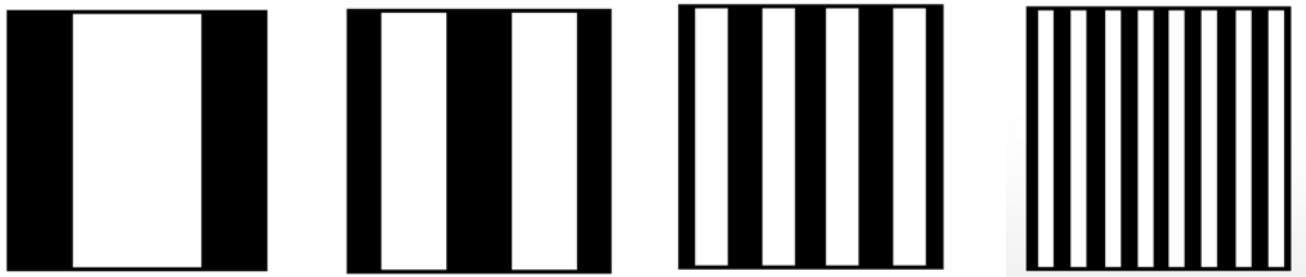


log(N) projections are sufficient to identify N stripes.

Structured-Light Scanners

Same basic idea as triangulation-based scanner. Use a projector.

Main Idea: Project multiple stripes to identify the position of a point.



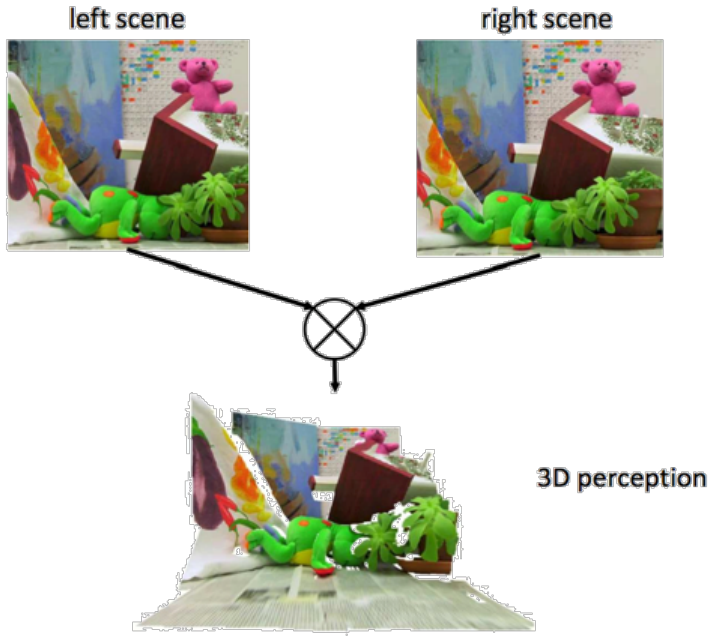
log(N) projections are sufficient to identify N stripes.

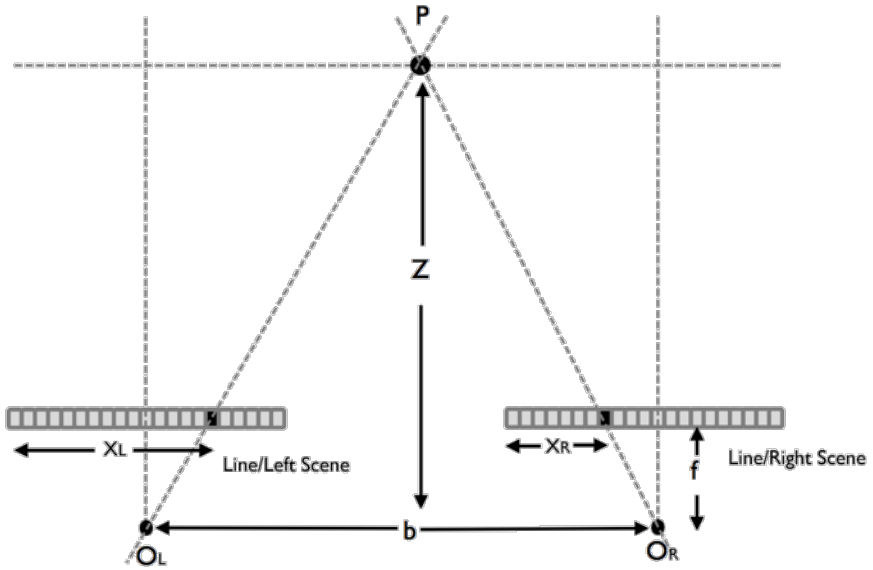
Advantage: cost and speed

Disadvantage: need controlled conditions & projector calibration.

Computer Vision Based Techniques

Depth from stereo:

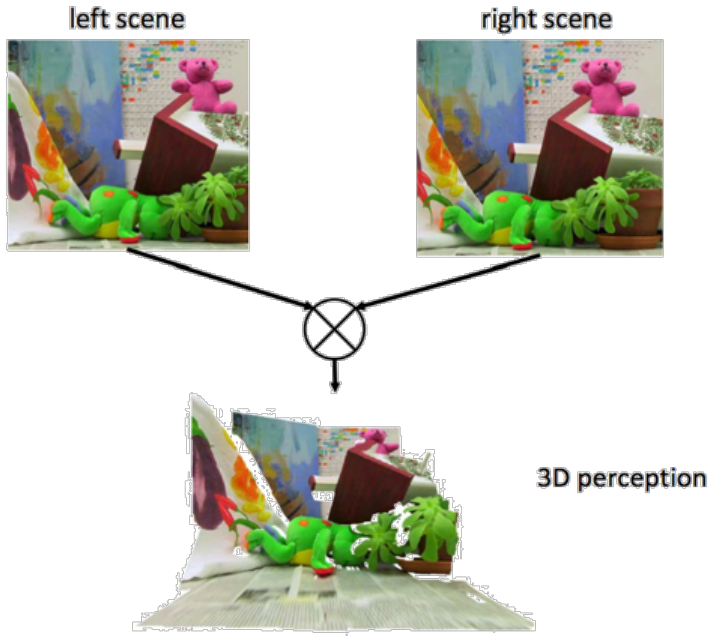


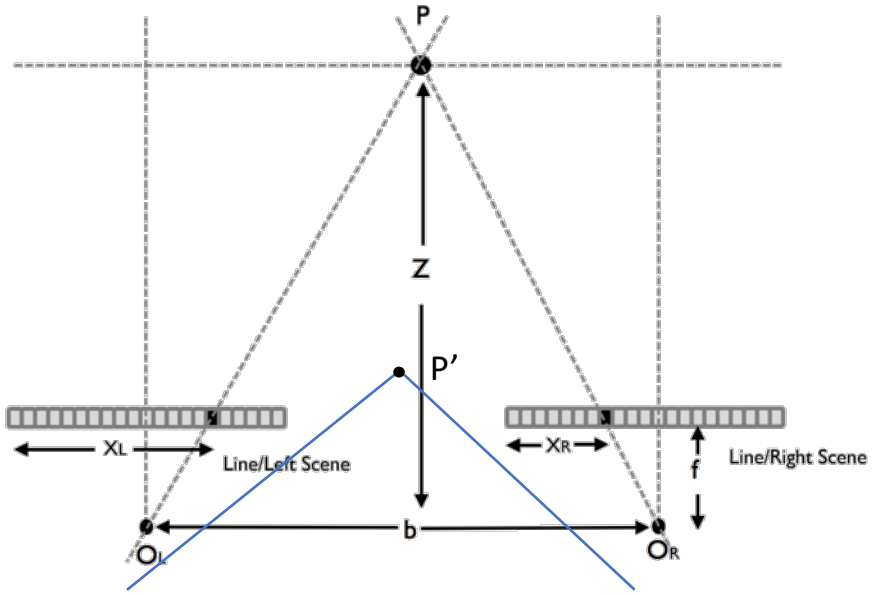


Given 2 images, shift in the x-axis is related to the depth.

Computer Vision Based Techniques

Depth from stereo:





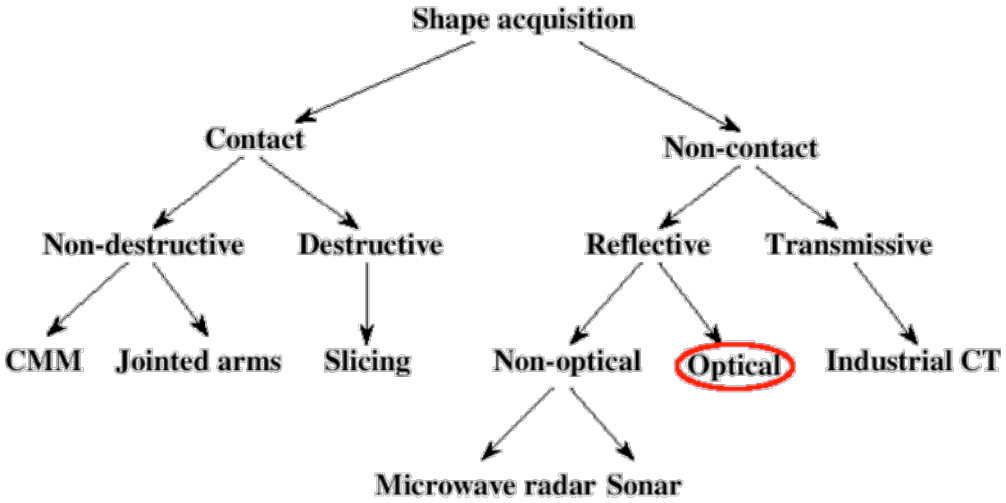
Given 2 images, shift in the x-axis is related to the depth.
Main challenge: establishing corresponding points across images: **very difficult.**

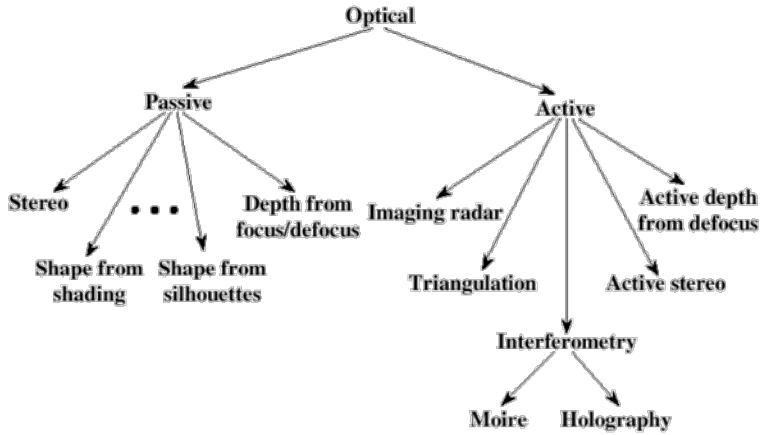
Computer Vision Based Techniques

Depth from blur:

Can approximate depth by detecting how blurry part of the image is for a **known focal length.**

Multitude of Other Methods





Rocchini et al. ‘01

Non-exhaustive taxonomy of 3D acquisition methods.

Scanning for Everyone: Microsoft Kinect Scanner

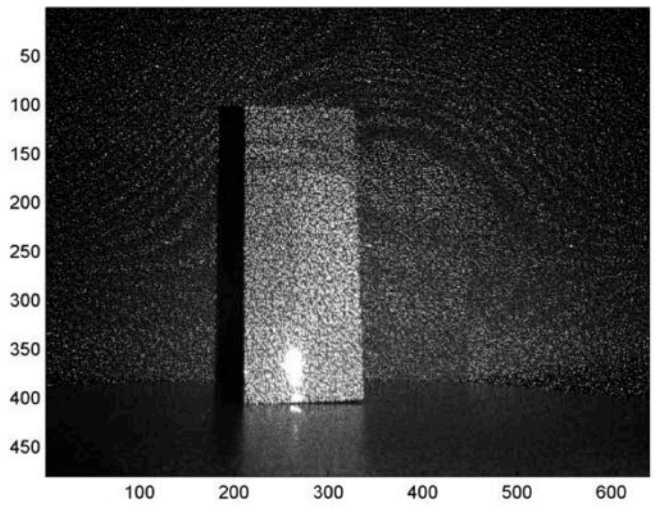
Low-cost ($200) 3D scanner – gadget for Xbox.

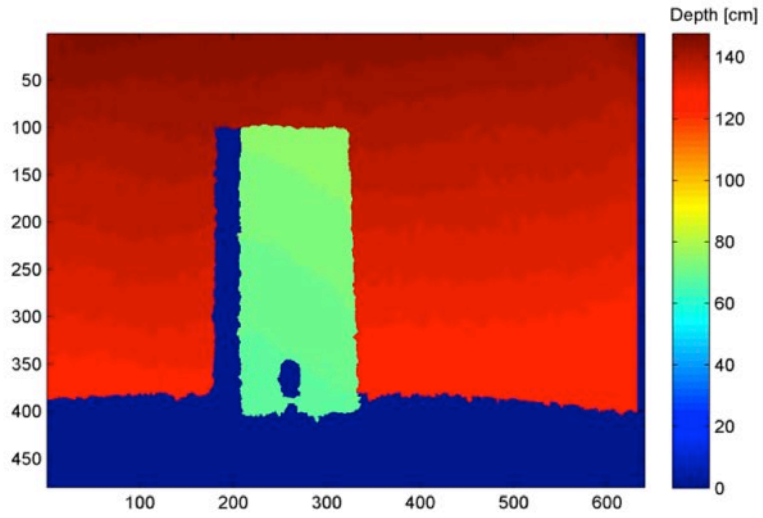
Allows to acquire Image (640 x 480) and 3D geometry (300k points) at 30 FPS.

Uses infrared active illumination with an infrared sensor **and** depth-from blur.
accuracy of ~1mm (at 0.5m distance) to 4cm (at 2m distance).

Microsoft Kinect Scanner

Low-cost ($200) 3D scanner – gadget for Xbox.





Allows us to acquire Image (640 x 480) and 3D geometry (300k points) at 30 FPS.

Uses infrared active illumination with an infrared sensor **and** depth-from blur.
accuracy of ~1mm (at 0.5m distance) to 4cm (at 2m distance).

Affordable 3D Scanners

Microsoft Kinect

Google Tango

Intel RealSense

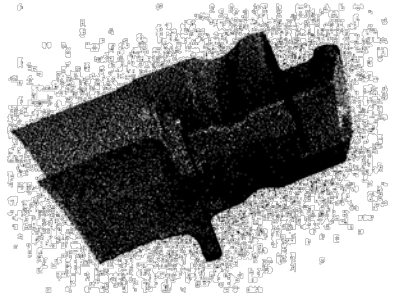
iSense 3D for iPad

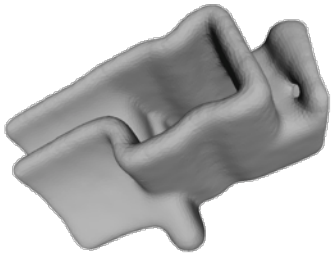
Point Cloud Processing

3D Point Cloud Processing

Typically point cloud sampling of a shape is insufficient for most applications. Main stages in processing:

1. Outlier removal – throw away samples from non-surface areas
2. If we have multiple scans, align them
3. Smoothing – remove local noise
4. Estimate surface normals
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh extraction

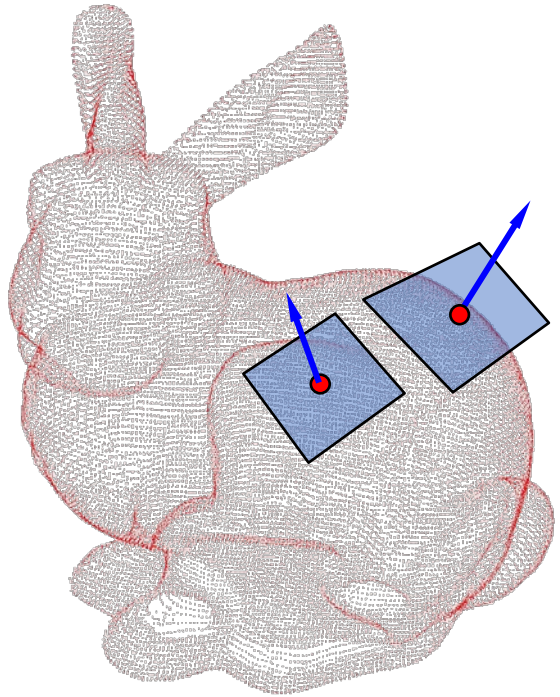




Normal Estimation and Outlier Removal

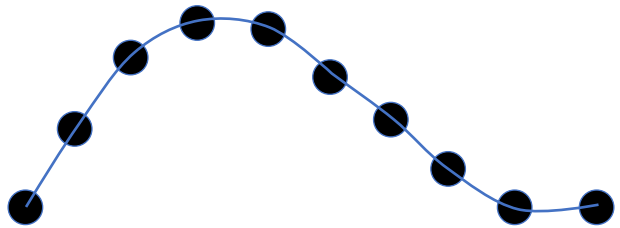
Fundamental problems in point cloud processing.

Although seemingly very different, can be solved with the same general approach – look at the “shape of neighborhoods” …



Normal Estimation

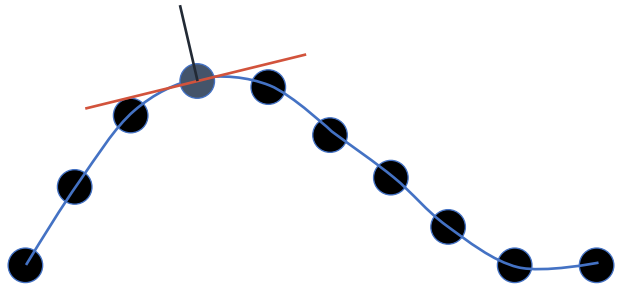
Assume we have a clean sampling of the surface. OK, start with a curve.



Our goal is to find the best approximation of the tangent direction, and thus of the normal to the curve.

Normal Estimation

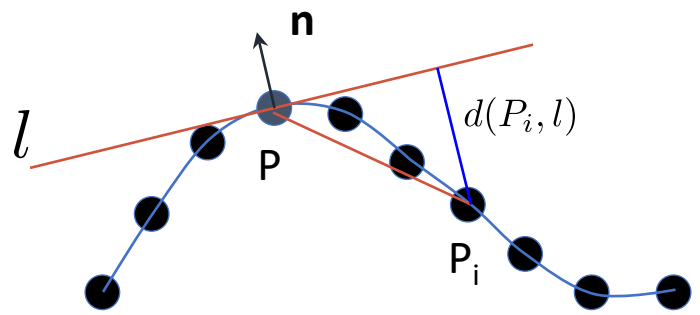
Assume we have a clean sampling of the surface. OK, start with a curve.



Our goal is to find the best approximation of the tangent direction, and thus of the normal to the line.

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



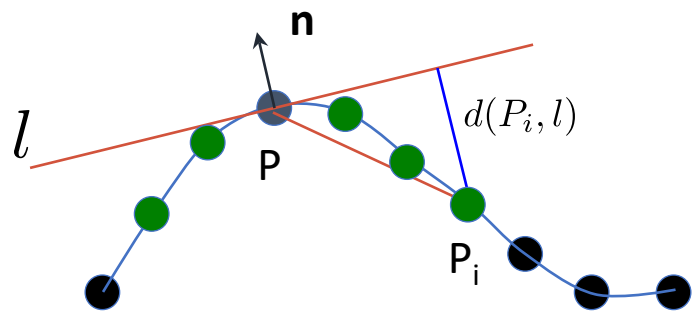
Goal: find best approximation of the normal at P.

Method: Given line l through P with normal **n**, for another point p_i:

$$d(p_i, l)^2 = \frac{((p_i - P)^T \mathbf{n})^2}{\mathbf{n}^T \mathbf{n}} = ((p_i - P)^T \mathbf{n})^2 \text{ if } \|\mathbf{n}\| = 1$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



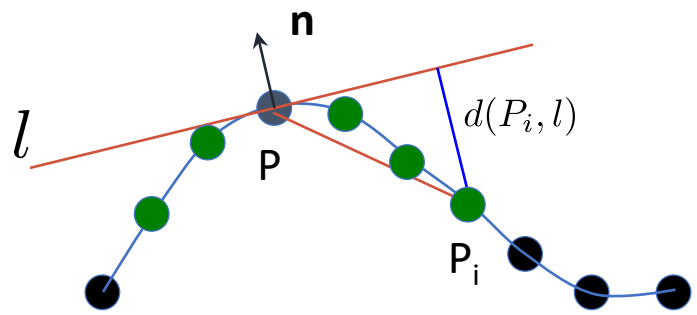
Goal: find best approximation of the normal at P.

Method: Find **n**, minimizing $\sum_{i=1}^{k} d(p_i, l)^2$ for a set of *k* points near P (e.g. *k* nearest neighbors of P).

$$\mathbf{n}_{\text{opt}} = \arg \min_{\|\mathbf{n}\|=1} \sum_{i=1}^{k} ((p_i - P)^T \mathbf{n})^2$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



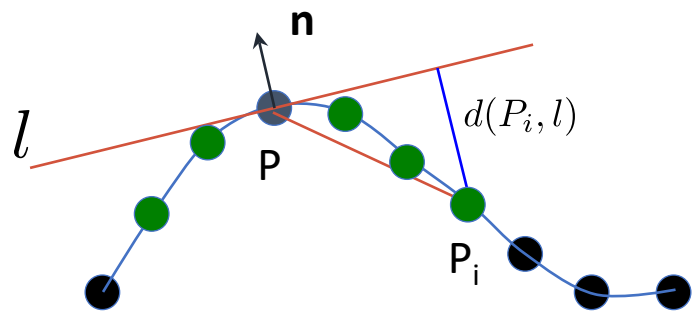
Using Lagrange multiplier:

$$\frac{\partial}{\partial \mathbf{n}} \left(\sum_{i=1}^{k} ((p_i - P)^T \mathbf{n})^2 \right) - \lambda \frac{\partial}{\partial \mathbf{n}} \left(\mathbf{n}^T \mathbf{n} \right) = 0$$

$$\sum_{i=1}^{k} 2(p_i - P)(p_i - P)^T \mathbf{n} = 2\lambda \mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



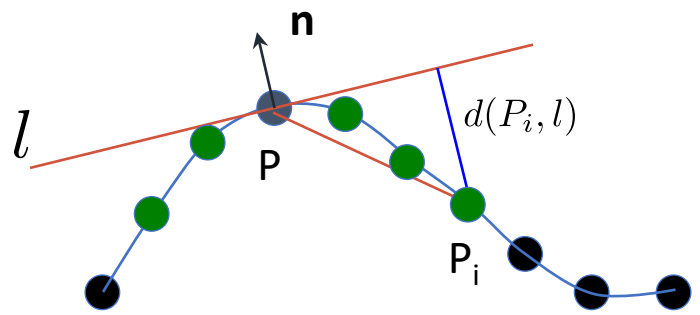
Using Lagrange multiplier:

$$\frac{\partial}{\partial \mathbf{n}} \left(\sum_{i=1}^{k} ((p_i - P)^T \mathbf{n})^2 \right) - \lambda \frac{\partial}{\partial \mathbf{n}} \left(\mathbf{n}^T \mathbf{n} \right) = 0$$

$$\left(\sum_{i=1}^{k} (p_i - P)(p_i - P)^T \right) \mathbf{n} = \lambda \mathbf{n} \quad \Rightarrow \quad C\mathbf{n} = \lambda \mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



The normal **n** must be an eigenvector of the matrix:

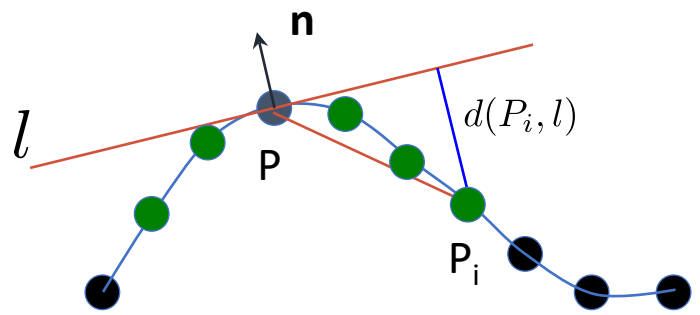
$$C\mathbf{n} = \lambda\mathbf{n} \qquad C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$$

Moreover, since:

$$\mathbf{n}_{\text{opt}} = \arg\min_{\|\mathbf{n}\|=1} \sum_{i=1}^{k}((p_i - P)^T\mathbf{n})^2 = \arg\min_{\|\mathbf{n}\|=1} \mathbf{n}^T C\mathbf{n}$$

Normal Estimation

Assume we have a clean sampling of the surface. OK, start with a curve.



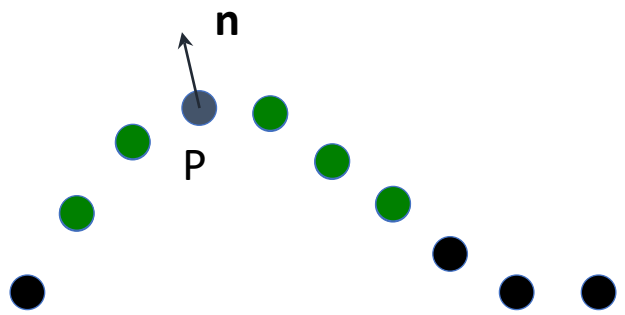
The normal **n** must be an eigenvector of the matrix:

$$C\mathbf{n} = \lambda\mathbf{n} \qquad C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$$

So, $\mathbf{n}_{opt}$ must be the eigenvector corresponding to the **smallest eigenvalue** of C.

Normal Estimation

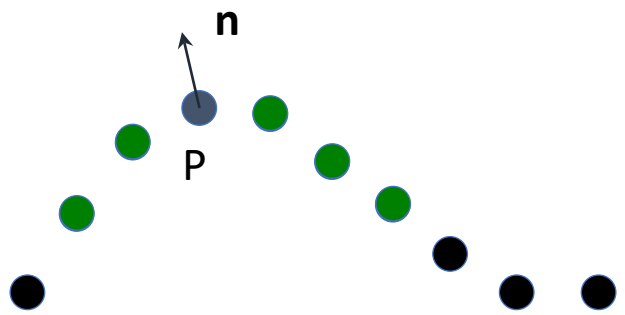
Method Outline (PCA):



1. Given a point P in the point cloud, find its *k* nearest neighbors.
2. Compute $C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$
3. **n:** eigenvector corresponding to the smallest eigenvalue of *C*.

Normal Estimation

Method Outline (PCA):



1. Given a point P in the point cloud, find its *k* nearest neighbors.
2. Compute $C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$
3. **n:** eigenvector corresponding to the smallest eigenvalue of *C*.

Variant on the theme: use $C = \sum_{i=1}^{k} (p_i - \overline{P})(p_i - \overline{P})^T, \quad \overline{P} = \frac{1}{k} \sum_{i=1}^{k} p_i$

Normal Estimation

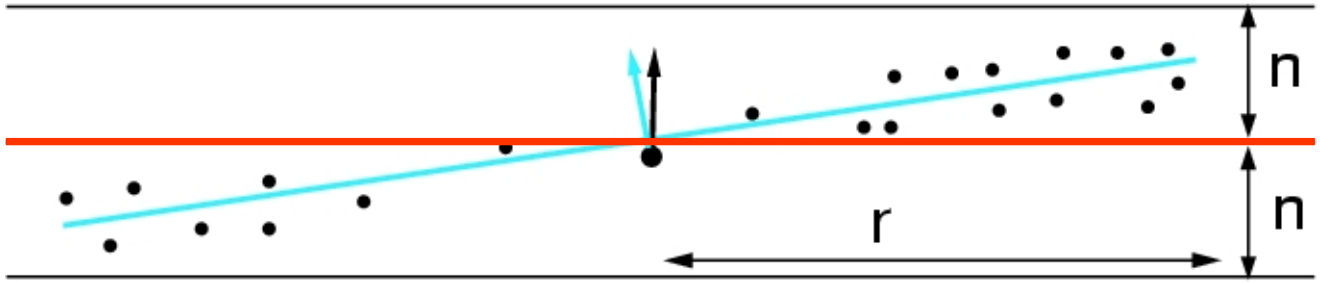
Critical parameter: *k*. Because of uneven sampling typically fix a radius *r*,

and use all points **inside a ball of radius *r*.**

How to pick an optimal r?

Normal Estimation

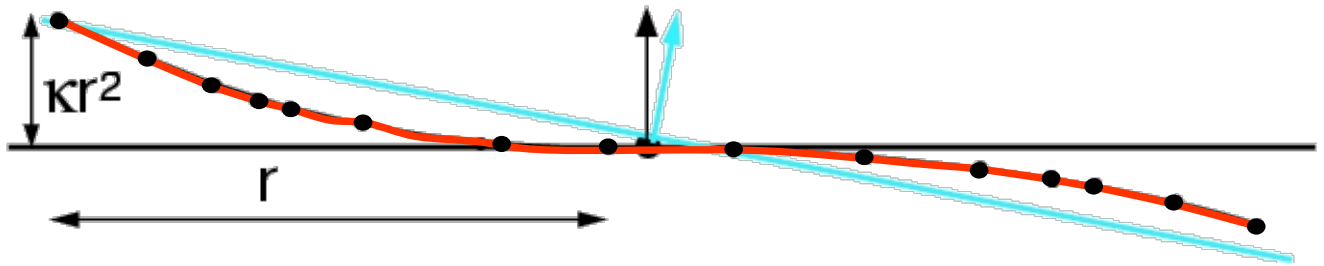
Collusive noise



Because of noise in the data, small *r* may lead to underfitting.

Normal Estimation

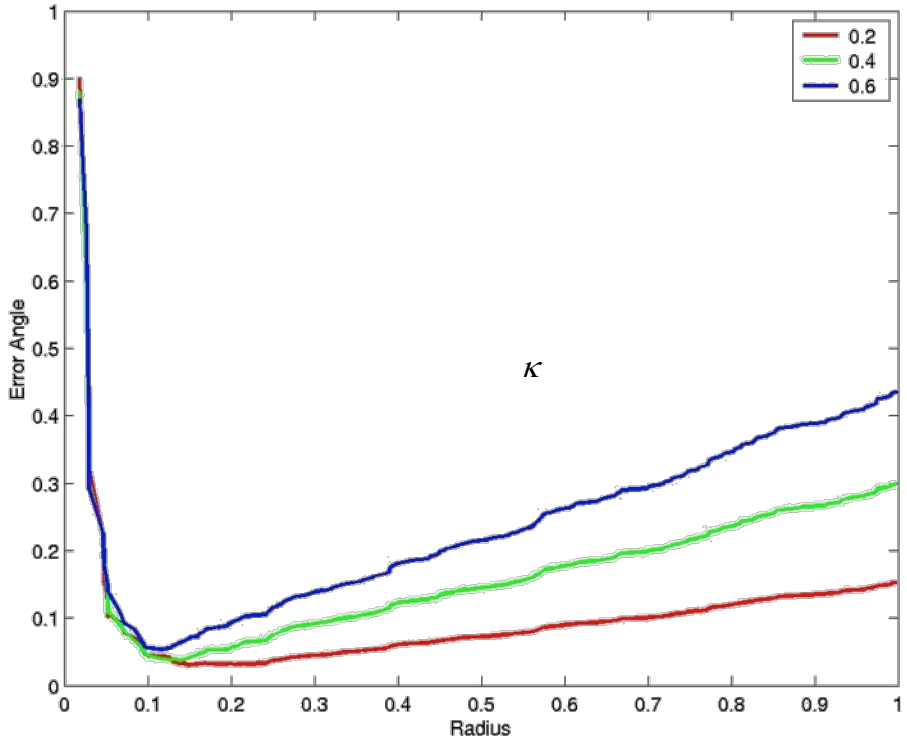
Curvature effect



Due to curvature, large *r* can lead to estimation bias.

Normal Estimation

κ



κ

κ = curvature

σ_n = noise

source: Mitra et al. '04

$$error \leq \Theta(1)\kappa r \quad + \quad \Theta(1)\frac{\sigma_n}{\sqrt{\varepsilon\rho r^3}} + \Theta(1)\frac{\sigma_n^2}{r^2}$$

Estimation error under Gaussian noise for different values of curvature (2D)

Normal Estimation

A similar but involved analysis results in 3D,

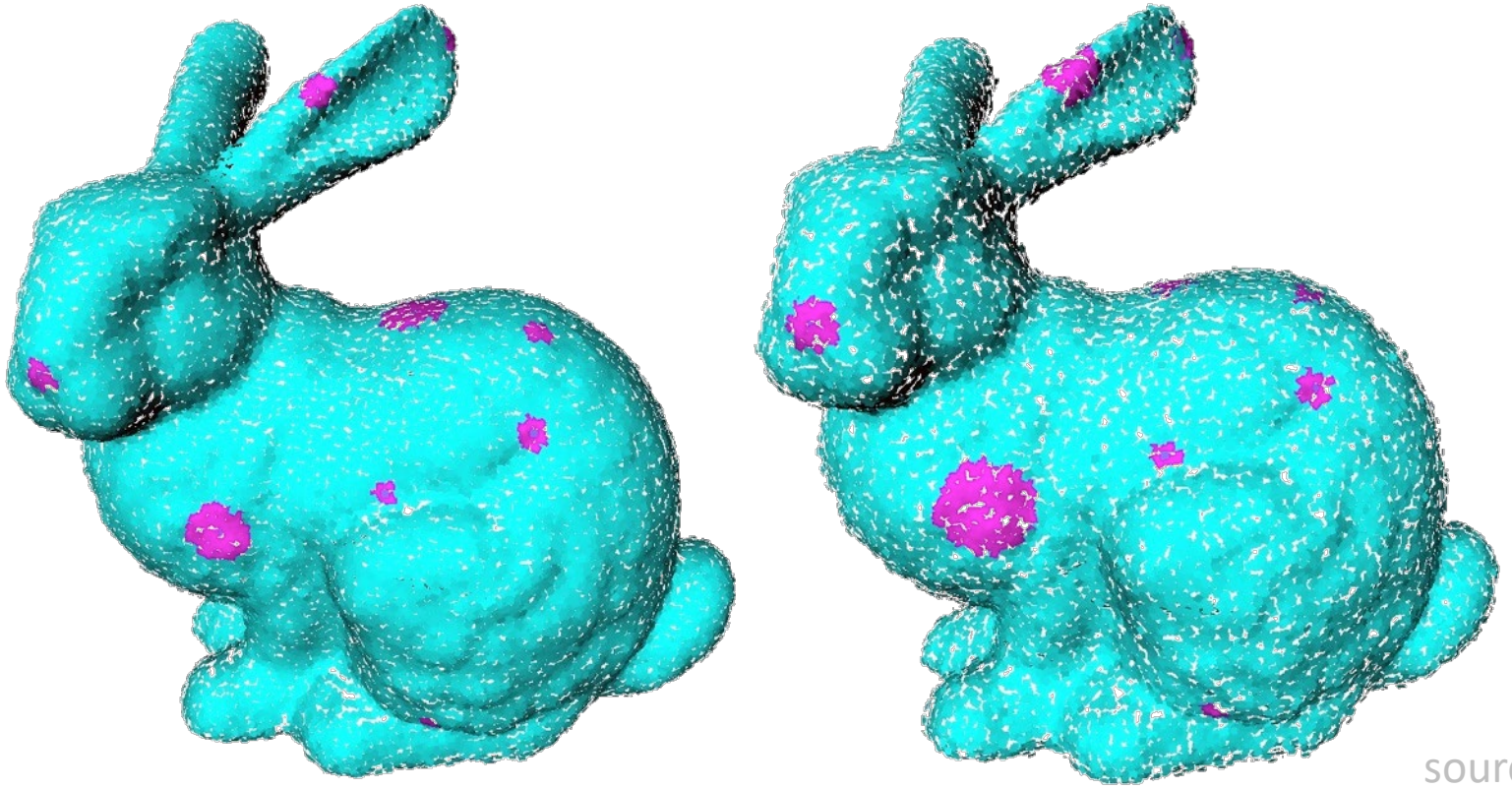
$$error \leq \Theta(1)\kappa r + \Theta(1)\sigma_n /(r^2\sqrt{\varepsilon\rho}) + \Theta(1)\sigma_n^2 / r^2$$

A good choice of r is,

$$r = \left(\frac{1}{\kappa} \left(c_1 \frac{\sigma_n}{\sqrt{\varepsilon\rho}} + c_2\sigma_n^2 \right) \right)^{1/3}$$

source: Mitra et al. ‘04

Normal Estimation – Neighborhood Size

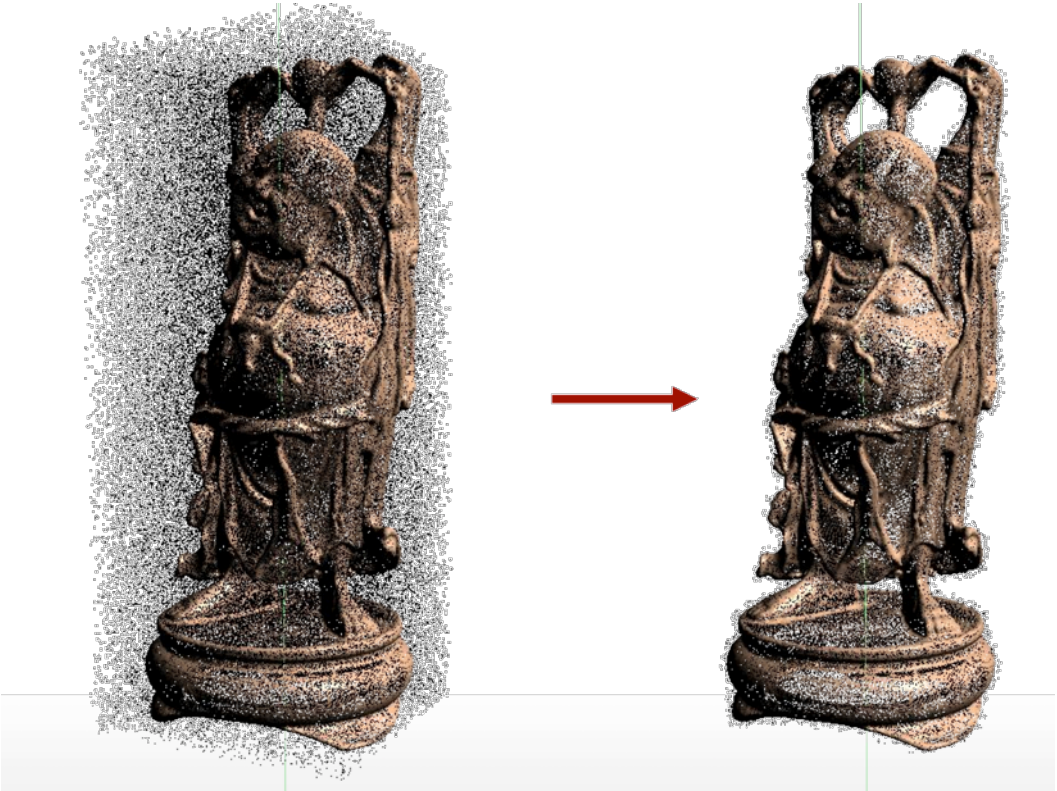


1x noise

2x noise

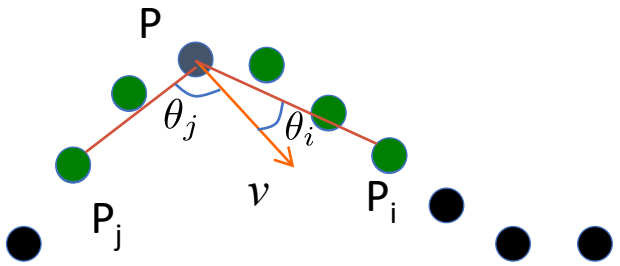
source: Mitra et al. ‘04

Outlier Removal



Goal: remove points that do not lie close to a surface.

Outlier Estimation



From the covariance matrix: $C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$ we have:

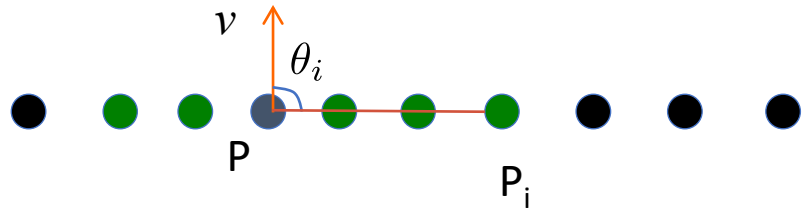
for any vector v, the Rayleigh quotient:

$$\frac{v^T C v}{v^T v} = \sum_{i=1}^{k} \left((p_i - P)^T v\right)^2 \text{ if } \|v\| = 1$$

$$= \sum_{i=1}^{k} \left(\|p_i - P\| \cos(\theta_i)\right)^2$$

Intuitively, v_{min}, maximizes the sum of angles to each vector $(p_i - P)$.

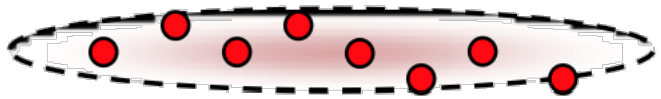
Outlier Estimation



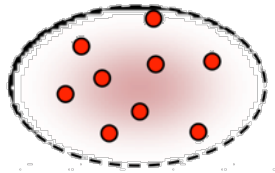
If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{\min}(C) = 0$ and $\lambda_{\max}(C)$ is large.

There exists a direction along which the point cloud has no variability.

If points are scattered randomly, then: $\lambda_{\max}(C) \approx \lambda_{\min}(C)$

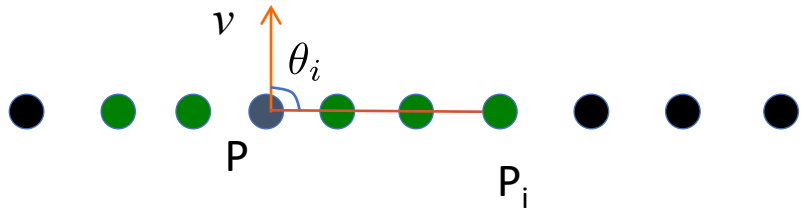


$\frac{\lambda_1}{\lambda_2}$ small



$\frac{\lambda_1}{\lambda_2} \approx 1$

Outlier Estimation



If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{\min}(C) = 0$ and $\lambda_{\max}(C)$ is large.

There exists a direction along which the point cloud has no variability.

If points are scattered randomly, then: $\lambda_{\max}(C) \approx \lambda_{\min}(C)$

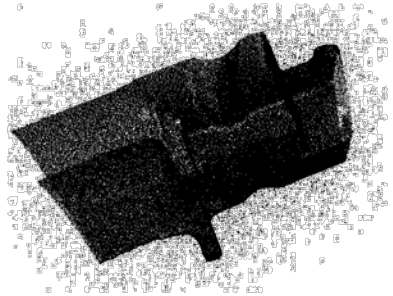
Thus, can remove points where $\frac{\lambda_1}{\lambda_2} > \epsilon$ for some threshold.

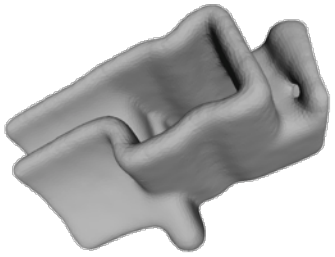
In 3D we expect two zero eigenvalues, so use $\frac{\lambda_2}{\lambda_3} > \epsilon$ for some threshold.

3D Point Cloud Processing

Typically point cloud sampling of a shape is insufficient for most applications. Main stages in processing:

1. Outlier removal – throw away samples from non-surface areas
2. If we have multiple scans, align them
3. Smoothing – remove local noise
4. Estimate surface normals
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh extraction



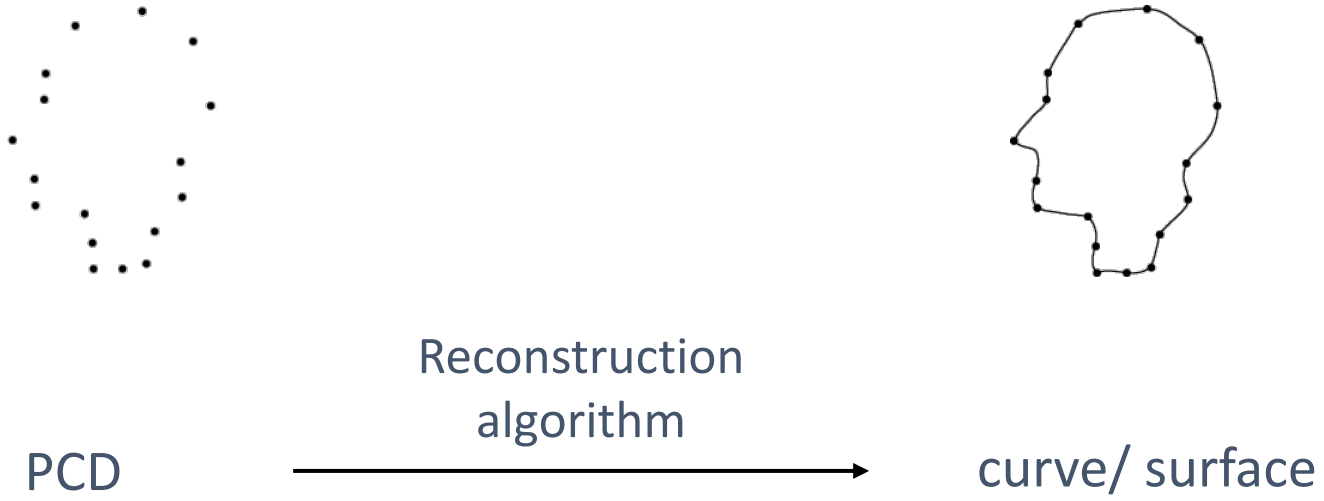


From Point Clouds to Surfaces

3D Point Cloud Reconstruction

Main Goal:

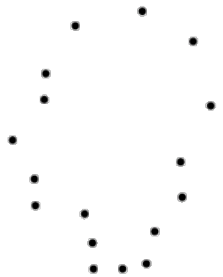
Construct a polygonal (e.g. triangle mesh) representation of the point cloud.

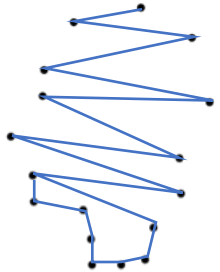


3D Point Cloud Reconstruction

Main Problem:

Data is **unstructured.** E.g. in 2D the points are not **ordered.**





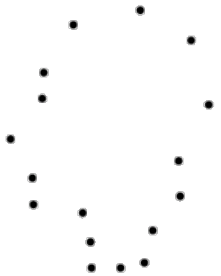
Reconstruction algorithm

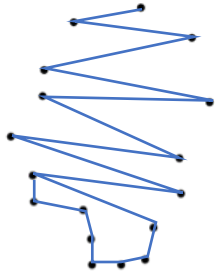
PCD → curve/ surface

3D Point Cloud Reconstruction

Main Problem:

Data is **unstructured.** E.g. in 2D the points are not **ordered.**
Inherently **ill-posed** (aka difficult) problem.



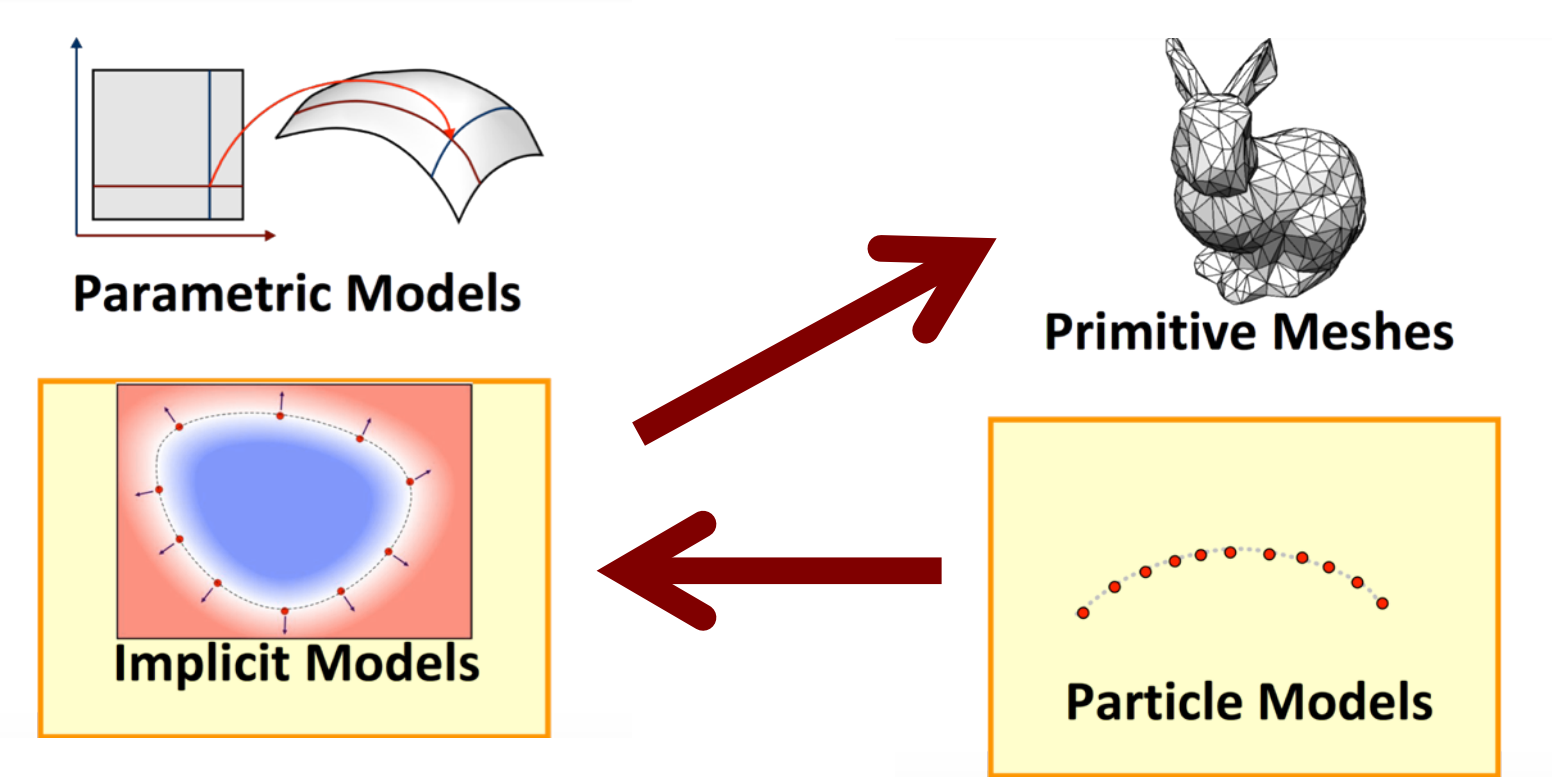


Reconstruction algorithm

PCD → curve/ surface

3D Point Cloud Reconstruction

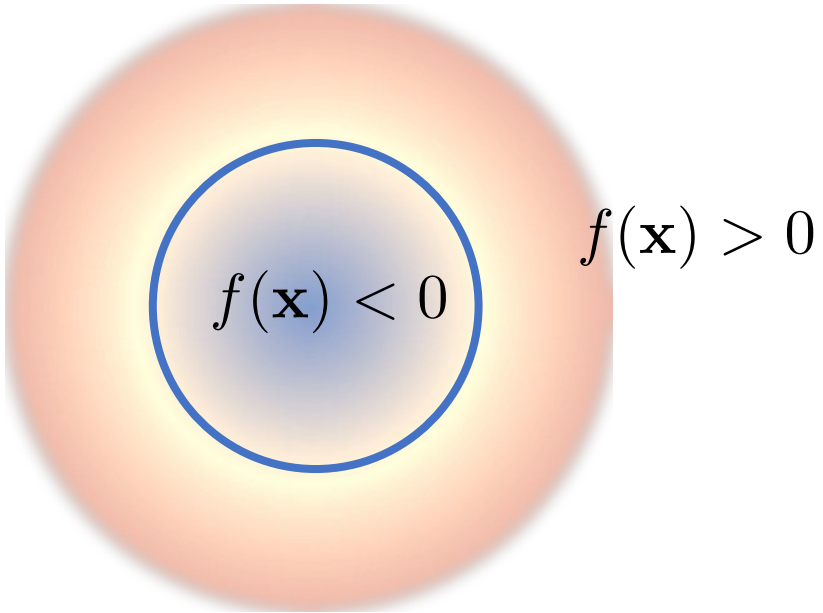
Reconstruction through Implicit models.



Implicit Surfaces

Given a function f(**x**), the surface is defined as:

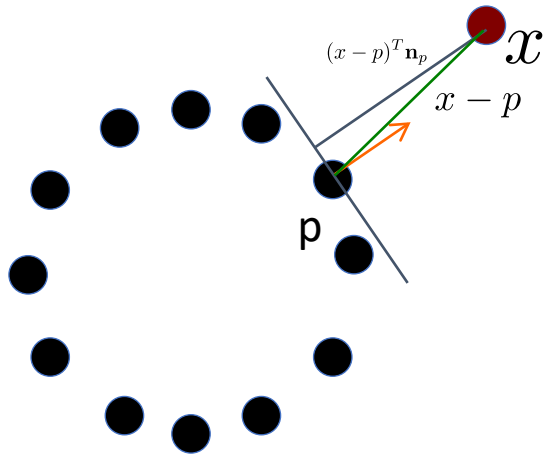
$$\{\mathbf{x}, \mathrm{s.t.} f(\mathbf{x}) = 0\}$$



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

Implicit Surfaces

Converting from a point cloud to an implicit surface:



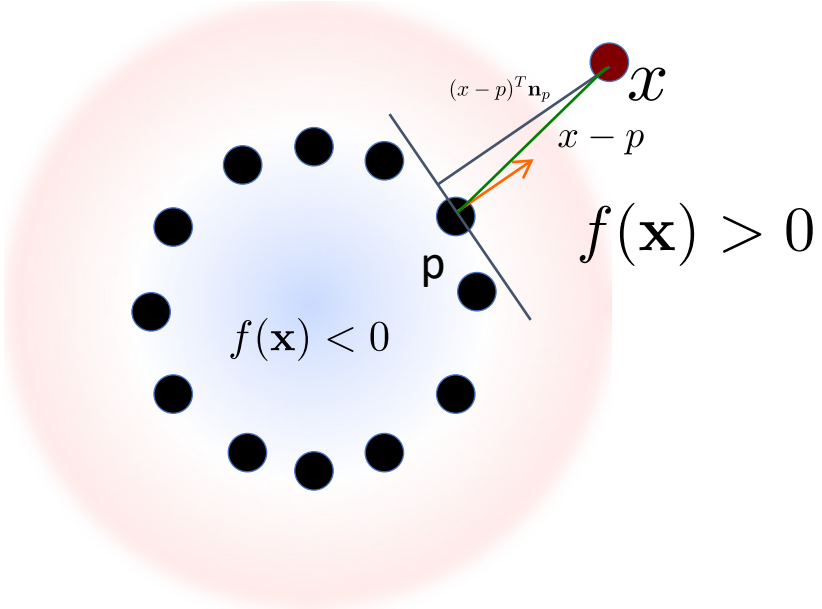
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



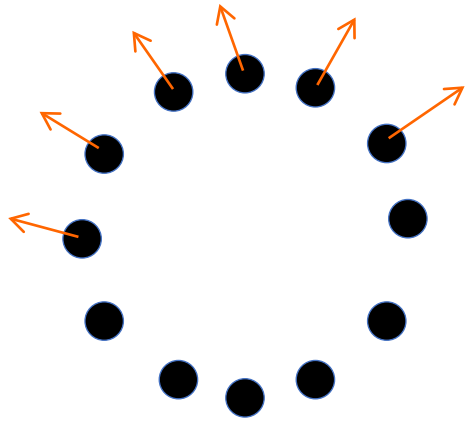
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



Simplest method:

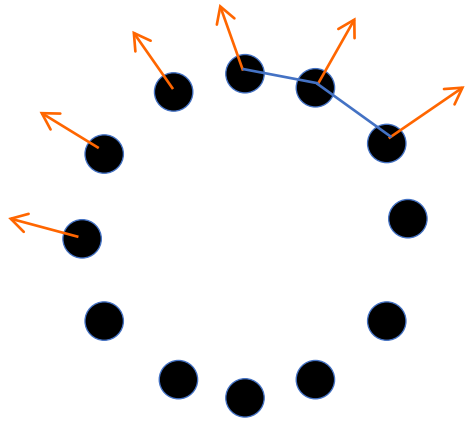
1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals.

   PCA only gives normals **up to orientation**

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



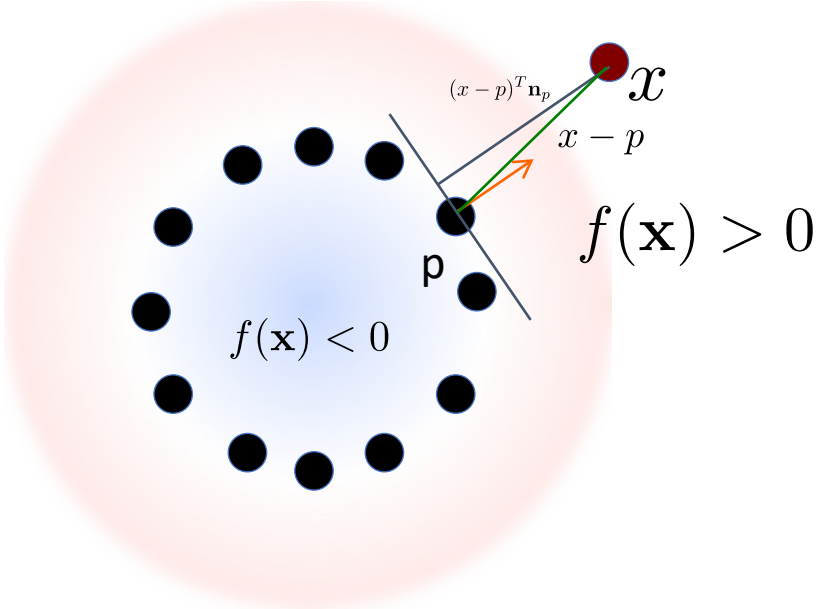
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals. In general, difficult problem, but can try to locally connect points and fix orientations.

Hugues Hoppe: Surface reconstruction from unorganized points

Implicit Surfaces

Converting from a point cloud to an implicit surface:



Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

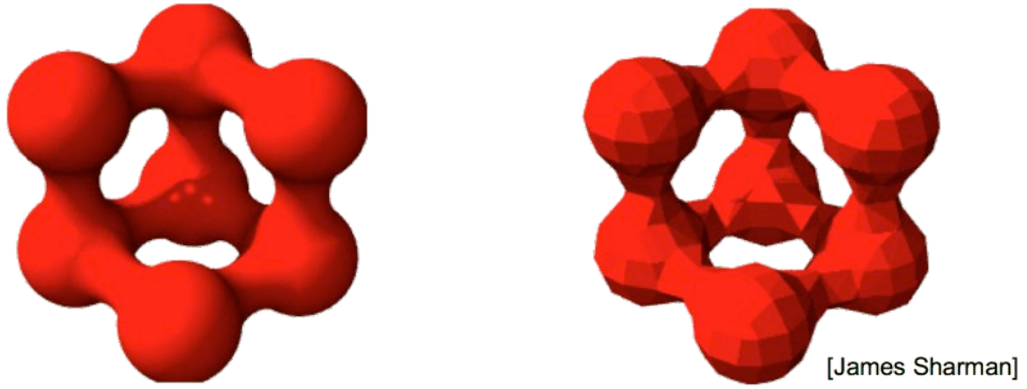
Note: many more advanced methods exist:
e.g., Moving Least Squares (MLS)

Marching Cubes

Converting from implicit to explicit representations.

Goal: Given an implicit representation: $\{\mathbf{x}, \text{s.t.} f(\mathbf{x}) = 0\}$

Create a triangle mesh that approximates the surface.



Lorensen and Cline, SIGGRAPH ‘87

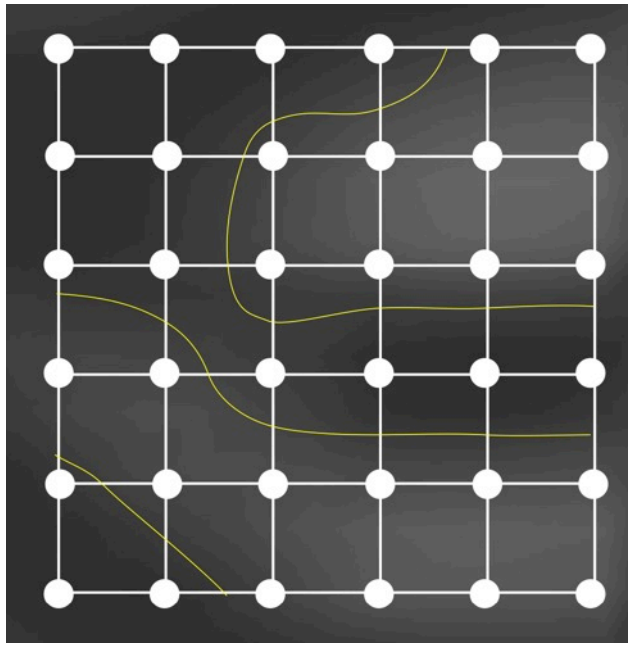
One of the most cited computer graphics papers of all time.

Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside

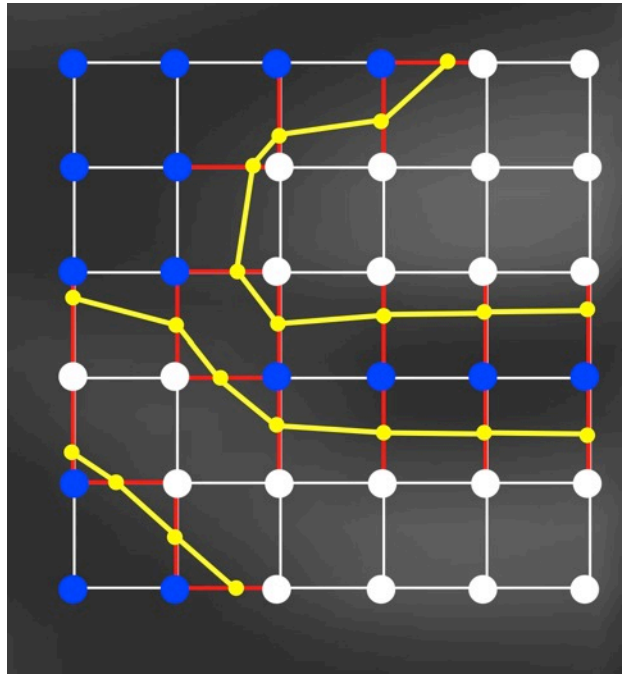
1. Discretize space.
2. Evaluate $f(x)$ on a grid.



Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside



1. Discretize space.
2. Evaluate $f(x)$ on a grid.
3. Classify grid points (+/-)
4. Classify grid edges
5. Compute intersections
6. Connect intersections

Marching Squares (2D)

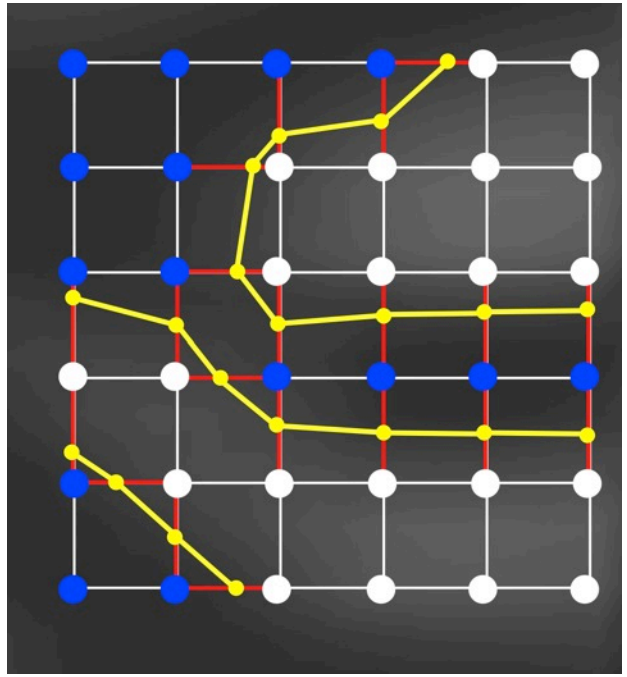
Computing the intersections:

- Edges with a sign switch contain intersections.

$$f(x_1) < 0, f(x_2) > 0 \Rightarrow$$
$$f(x_1 + t(x_2 - x_1)) = 0$$
$$\text{for some } 0 \leq t \leq 1$$

- Simplest way to compute t: assume f is linear between x1 and x2:

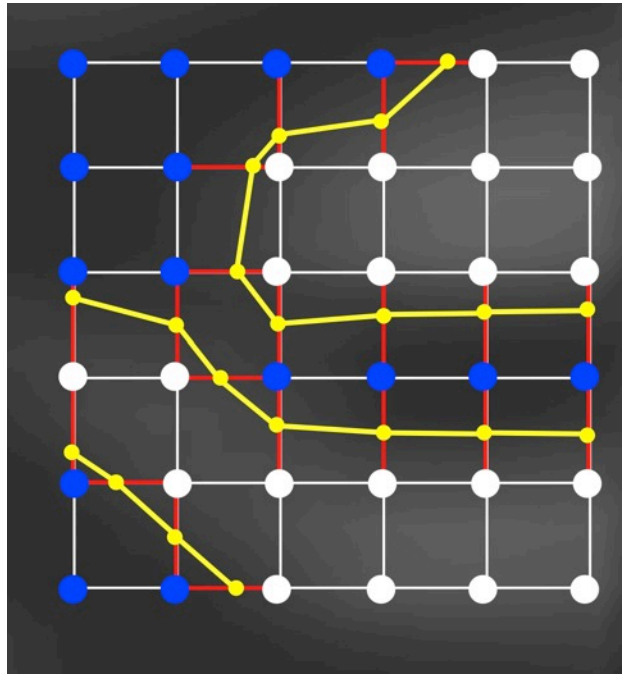
$$t = \frac{f(x_1)}{f(x_2) - f(x_1)}$$



Marching Squares (2D)

Connecting the intersections:

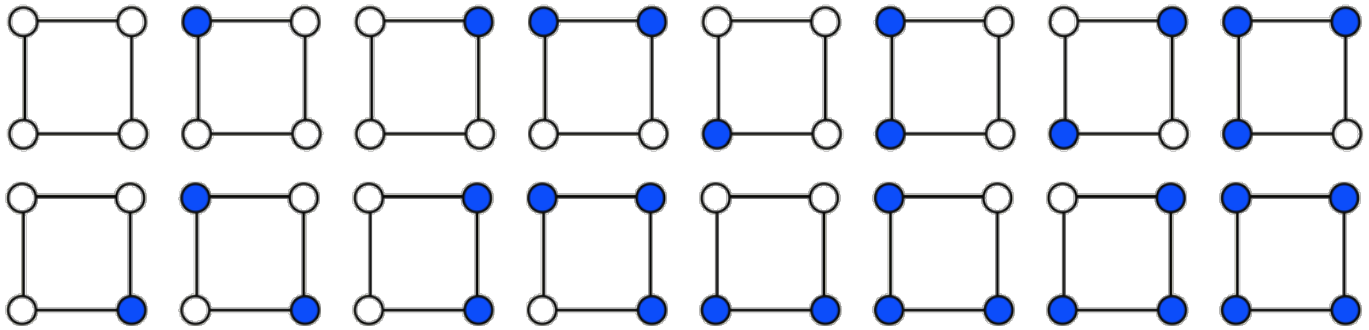
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



Marching Squares (2D)

Connecting the intersections:

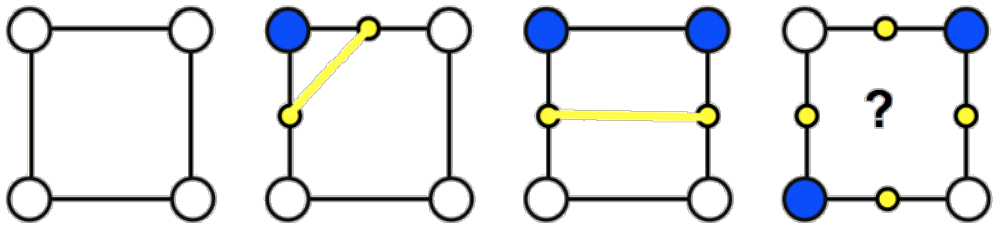
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections



Marching Squares (2D)

Connecting the intersections:

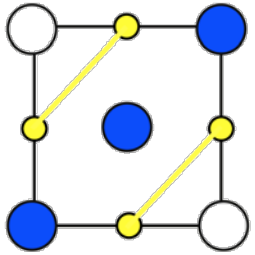
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections

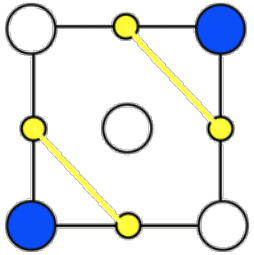


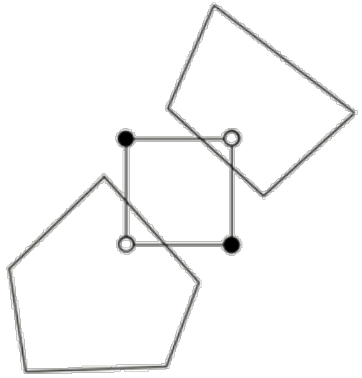
Marching Squares (2D)

Connecting the intersections:

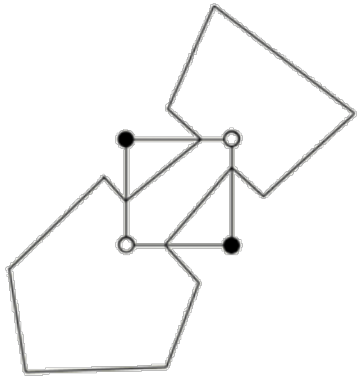
Ambiguous cases:







Break contour



Join contour

2 options:

1) Can resolve ambiguity by subsampling inside the cell.
2) If subsampling is impossible, pick one of the two possibilities.

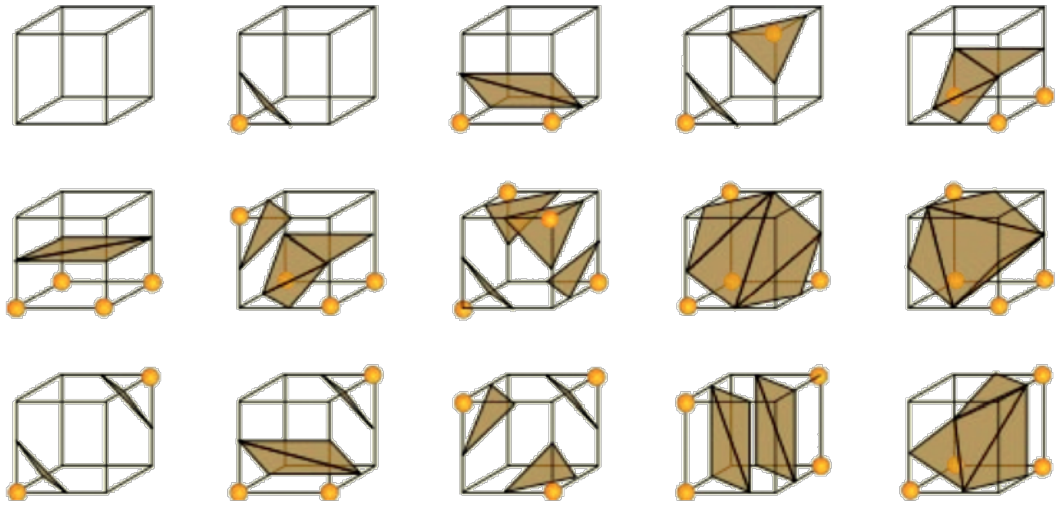
Marching Cubes (3D)

Same basic machinery applies to 3D.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 15 after symmetries



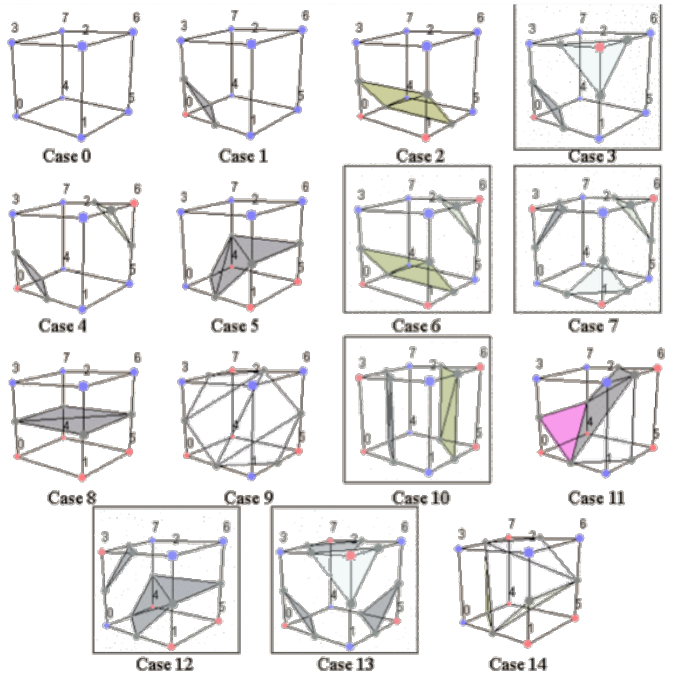
Marching Cubes (3D)

Same basic machinery applies to 3D.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 15 after symmetries
- 6 ambiguous cases (in boxes)



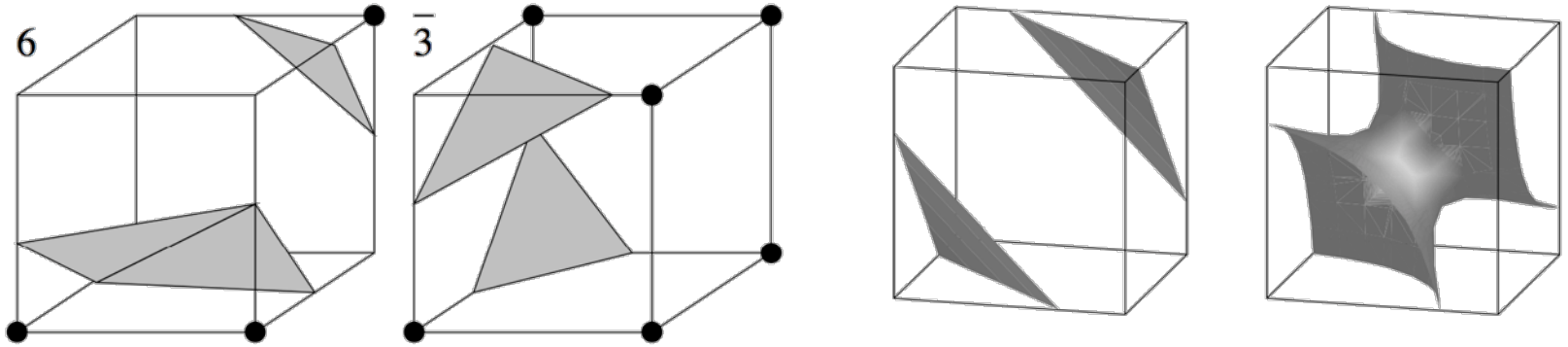
Marching Cubes (3D)

Same basic machinery applies to 3D.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 15 after symmetries
- 6 ambiguous cases (in boxes)
- Inconsistent triangulations can lead to holes and wrong topology.



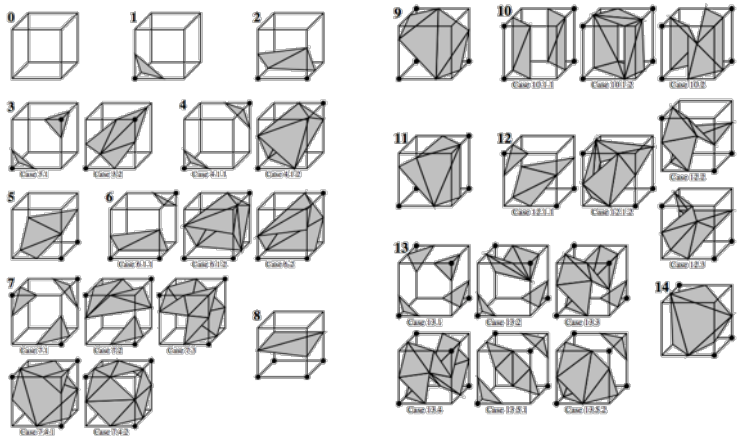
Marching Cubes (3D)

Same basic machinery applies to 3D.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 15 after symmetries
- 6 ambiguous cases (in boxes)
- Inconsistent triangulations can lead to holes and wrong topology.
- More subsampling rules – leads to 33 unique cases.



Chernyaev, Marching Cubes 33,'95

Marching Cubes (3D)

Main Strengths:

- Very multi-purpose.
- Extremely fast and parallelizable.
- Relatively simple to implement.

Main Weaknesses:

- Can create badly shaped (skinny) triangles.
- Basic versions do not provide topological guarantees.
- Many special cases (implemented as big lookup tables).

Marching Cubes (3D)

Main Strengths:

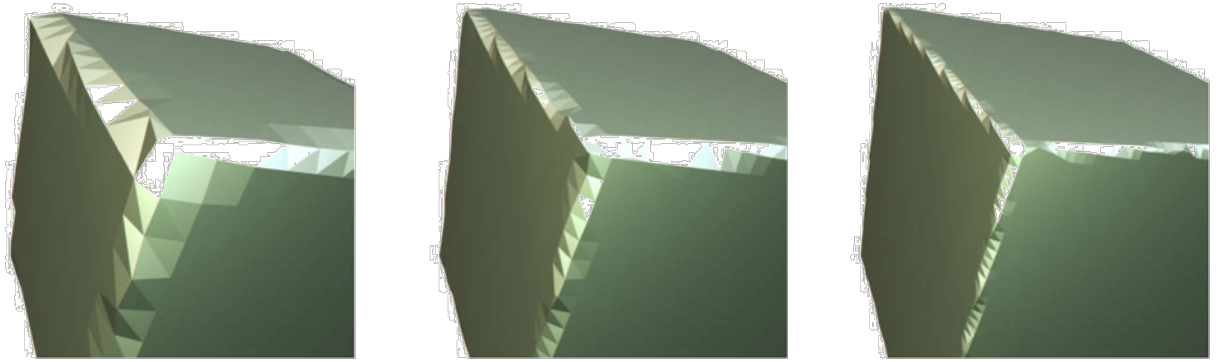
- Very multi-purpose.
- Extremely fast and parallelizable.
- Relatively simple to implement.
- Virtually parameter-free

Main Weaknesses:

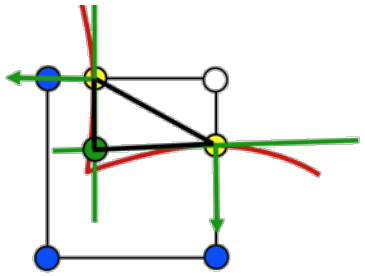
- Can create badly shaped (skinny) triangles.
- Basic versions do not provide topological guarantees.
- Many special cases (implemented as big lookup tables).
- No sharp features.

Marching Cubes (3D)

No sharp features.



1. Increasing grid resolution does not help
2. Normals do not converge.
3. Use normal information to find corners.
   Special treatment for corners



Extended Marching Cubes

That’s All