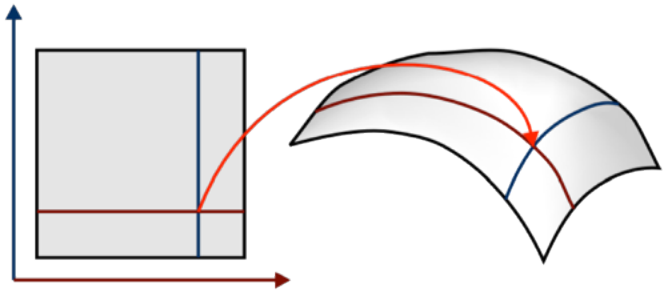
INF 574

Geometric Modeling: Digital Representation and Analysis of Shapes

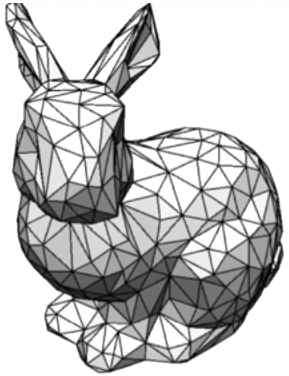
Today

- Surface scanning basics
- Point Cloud shape representation
- Shape registration: Iterative Closest Point
- Normal and outlier estimation
- Implicit Surface definition
- Reconstruction:
  - Marching Cubes
  - Extensions

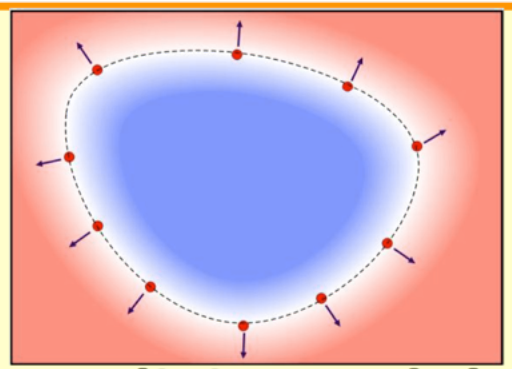
Modeling zoo



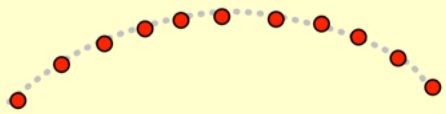
Parametric Models



Primitive Meshes



Implicit Models

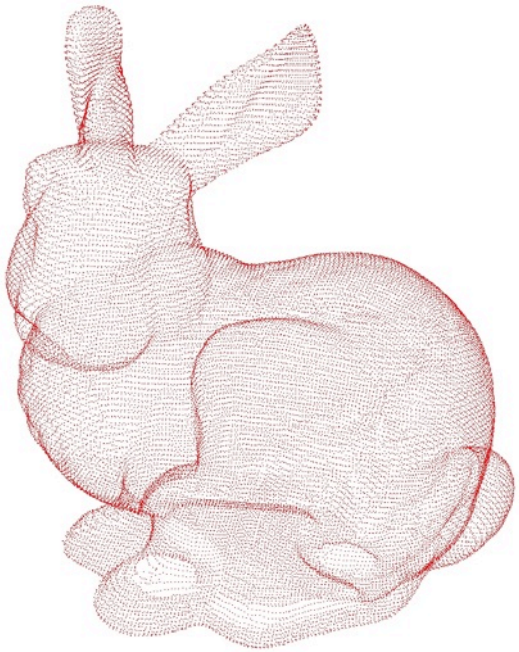


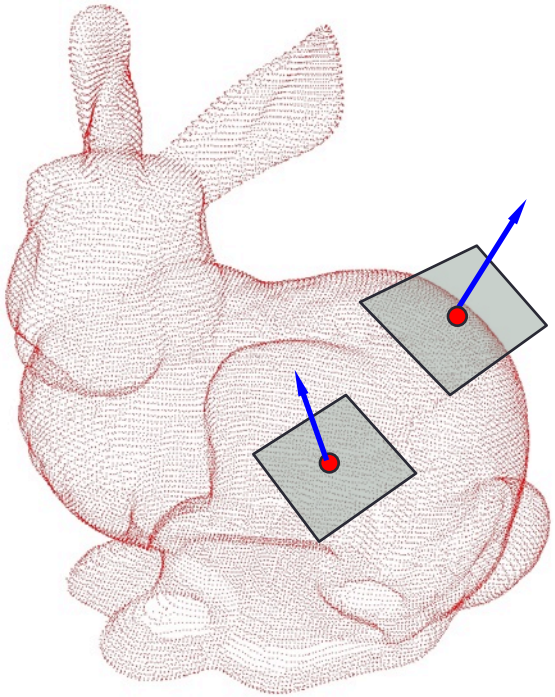
Particle Models

source: Michael Wand

Point Clouds

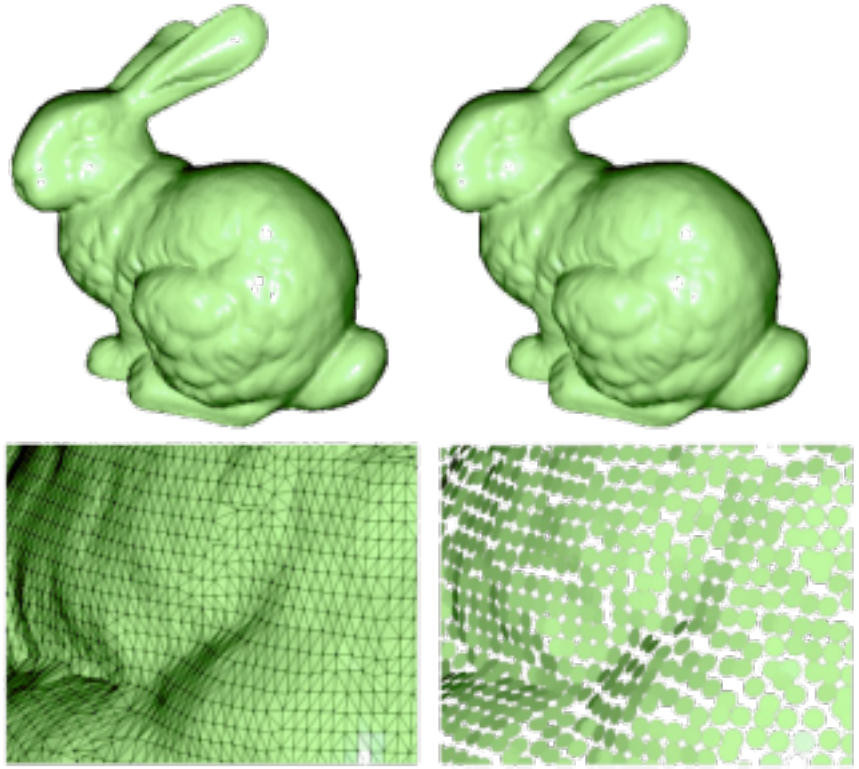
- Simplest representation: **only points,** no connectivity.
- Collection of (x,y,z) coordinates, possibly with normals





Point Clouds

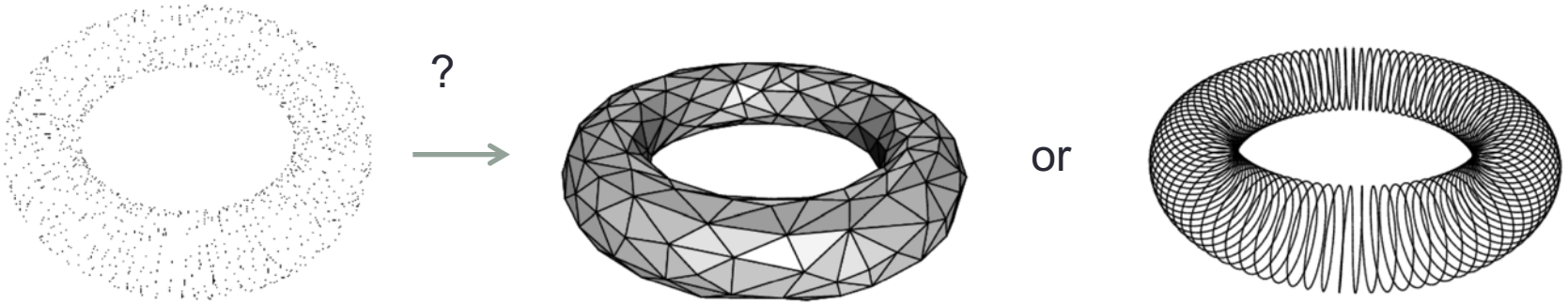
- Simplest representation: **only points,** no connectivity.
- Collection of (x,y,z) coordinates, possibly with normals.
- Points with orientation are called **surfels.**



Filip Van Bouwel

Point Clouds

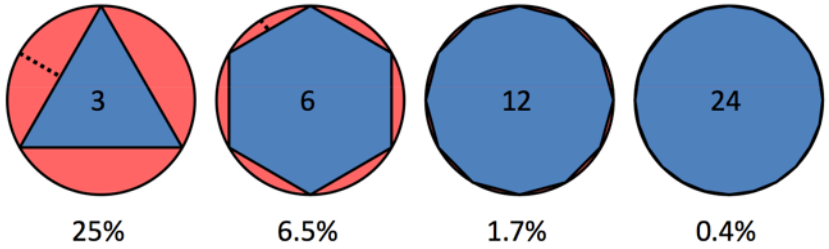
- Simplest representation: **only points,** no connectivity.
- Collection of (x,y,z) coordinates, possibly with normals.
- Points with orientation are called **surfels.**
- Severe limitations:
  - **no** Simplification or subdivision
  - **no** direct smooth rendering
  - **no** topological information



Point Clouds

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

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

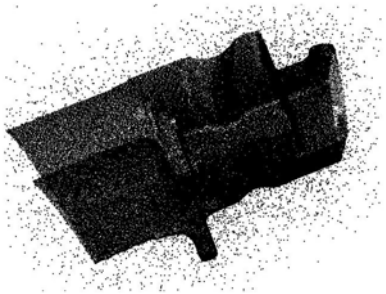


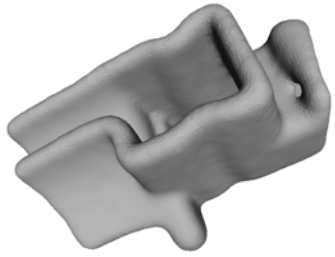
Point Clouds

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

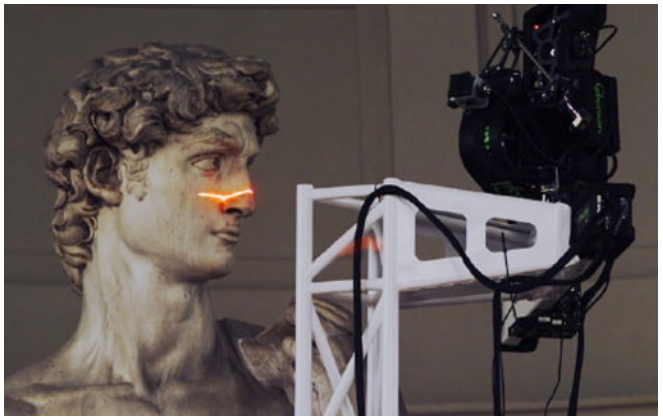




Why Point Clouds?

1) Typically, that's the only thing that's available

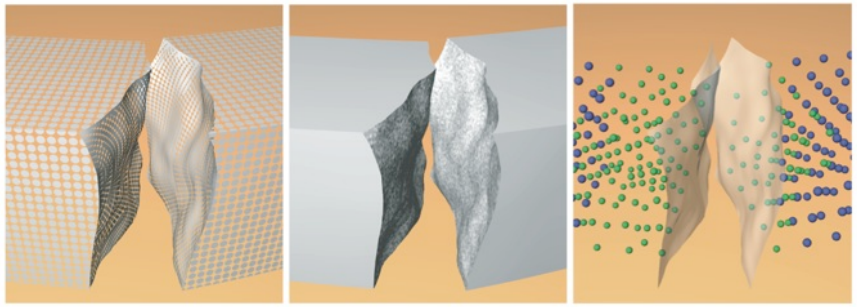
   Nearly all 3d scanning devices produce point clouds



Why Point Clouds?

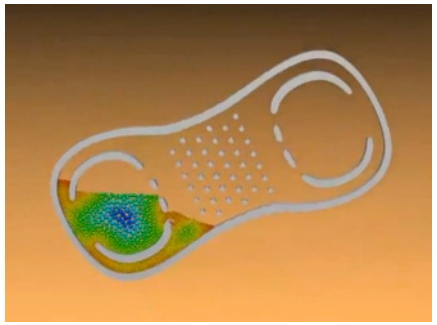
1) Typically, that's the only thing that's available
2) Locality: sometimes, easier to handle (esp. in hardware).

Fracturing Solids



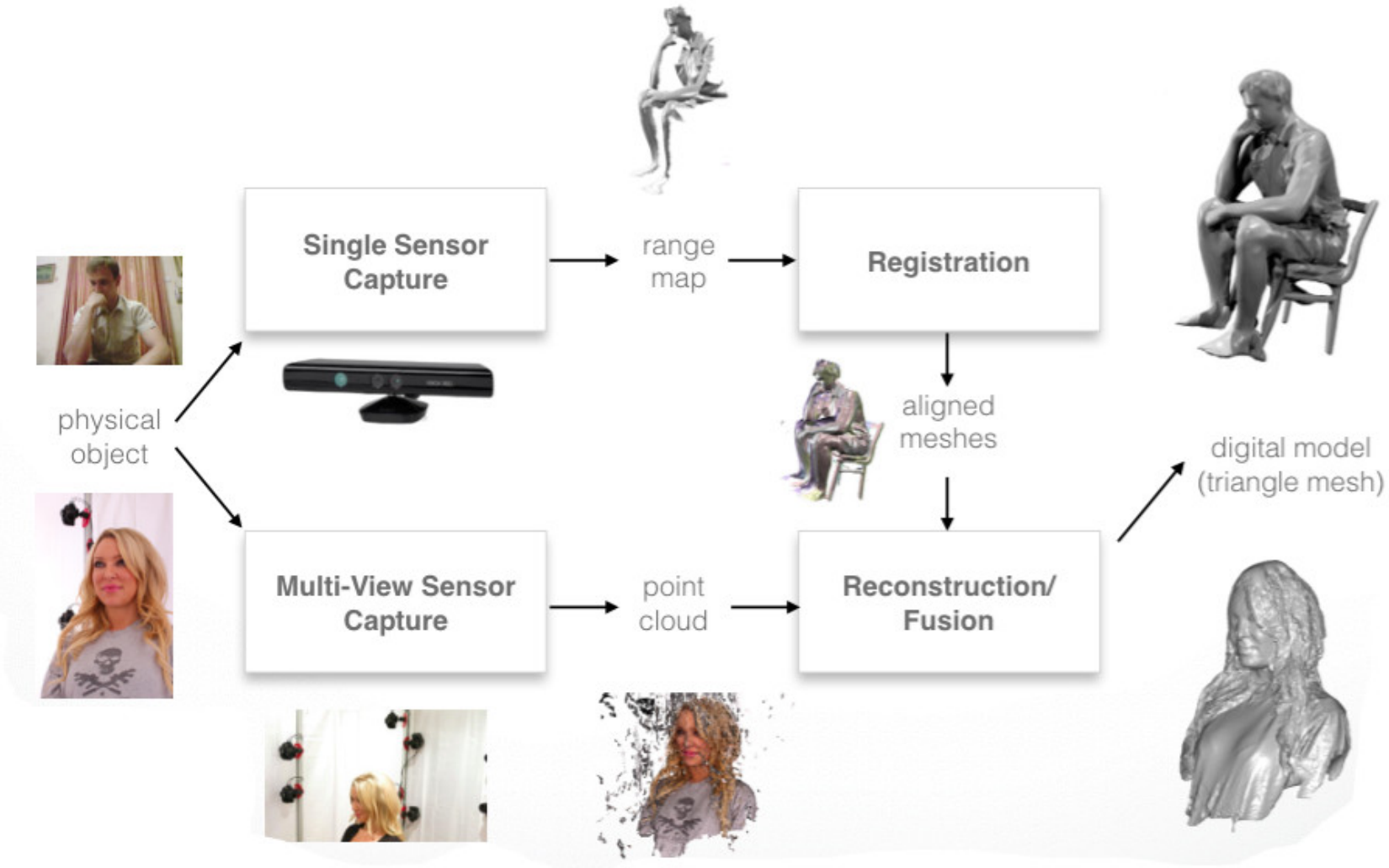
Meshless Animation of Fracturing Solids
Pauly et al., SIGGRAPH '05

Fluid Simulation

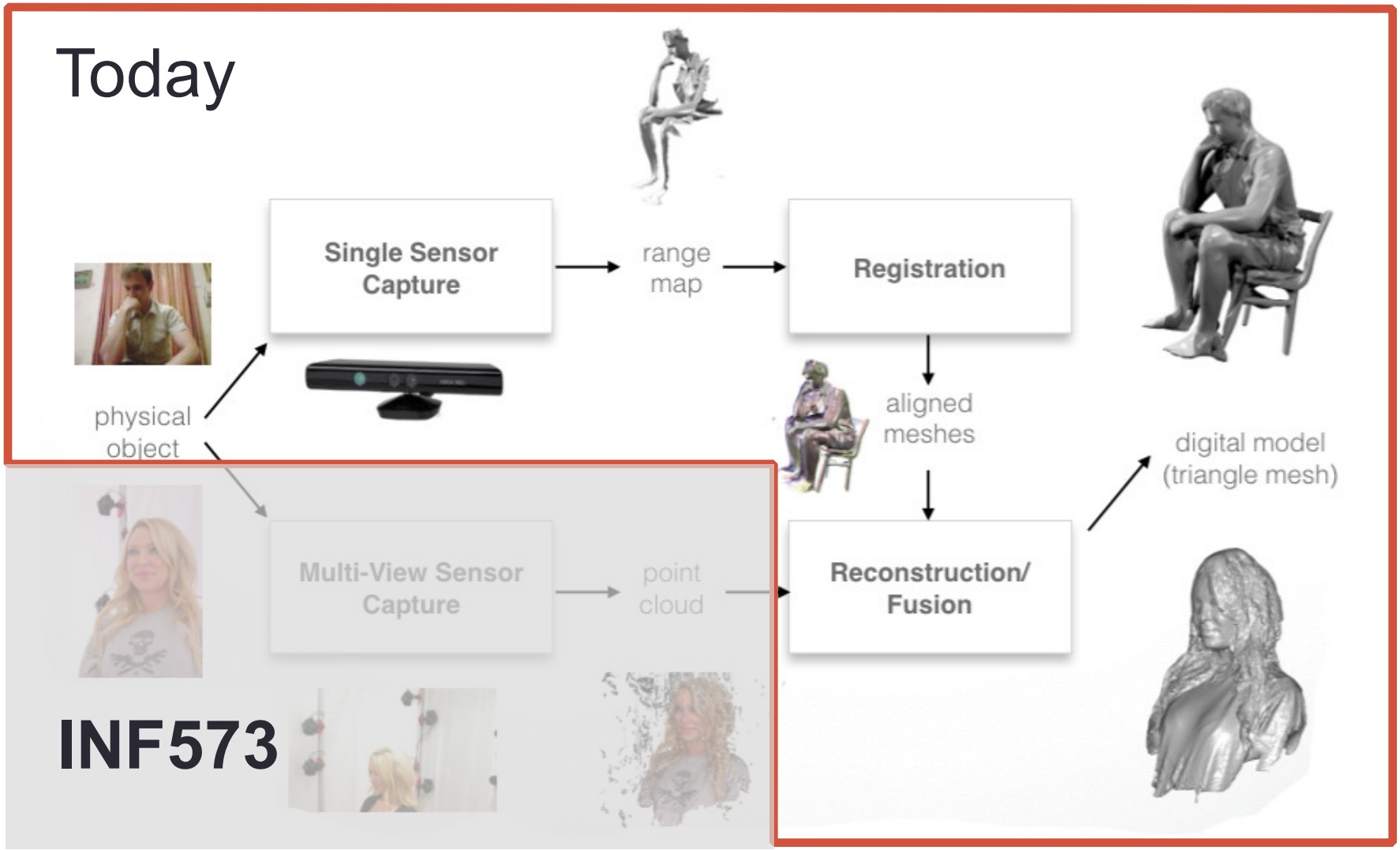


Adaptively sampled particle fluids,
Adams et al. SIGGRAPH '07

Two main scanning approaches



Two main scanning approaches



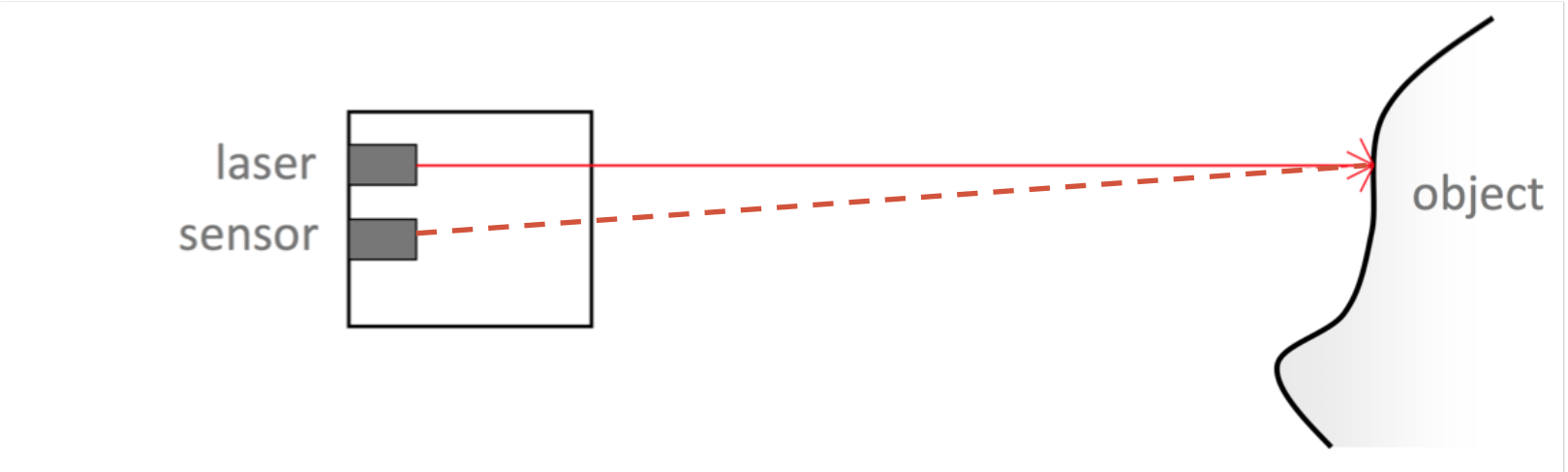
Typical Scanning and Reconstruction Pipeline

Single View Scanners

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



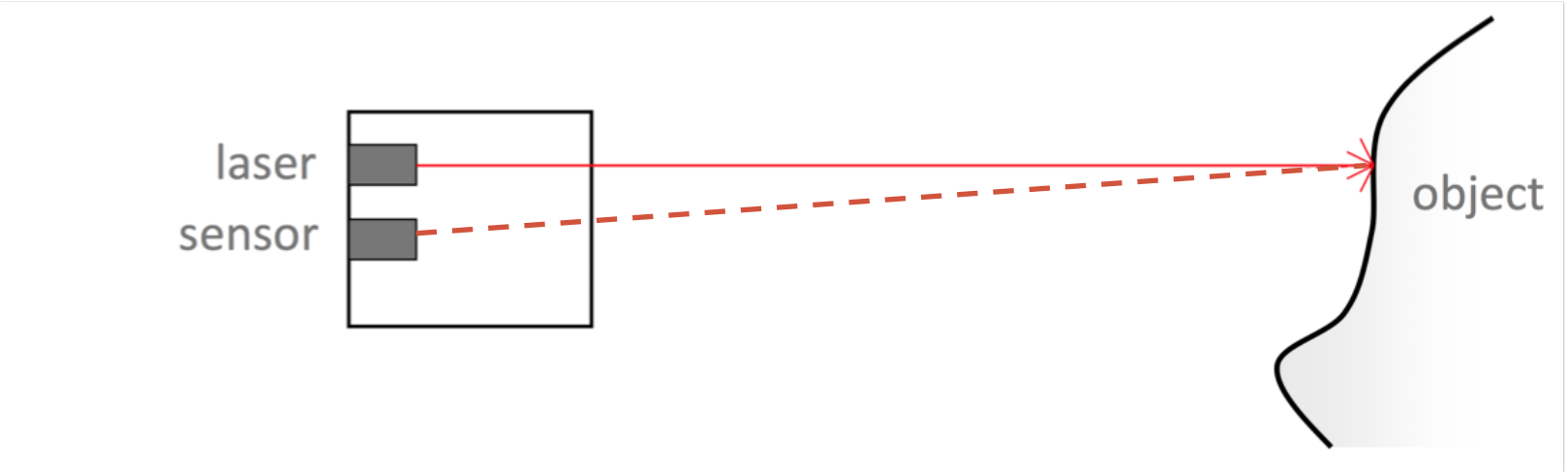
1. Emit a short short pulse of laser
2. Capture the reflection.
3. Measure the time it took to come back.

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

c: speed of light (≈ 299 792 458 m/s)

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

Time of Flight scanners

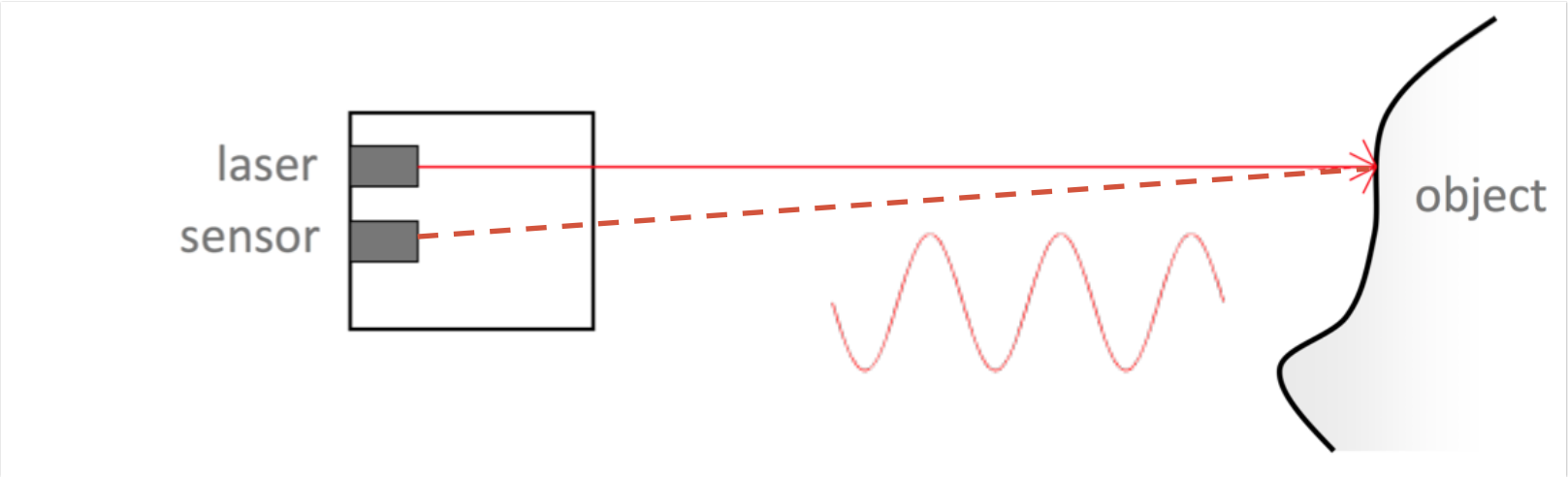


1. Emit a short short pulse of laser
2. Capture the reflection.
3. Measure the time it took to come back.
4. Need a very fast clock.
5. Main advantage: can be done over long distances.
6. Used in terrain scanning.

Time of Flight scanners

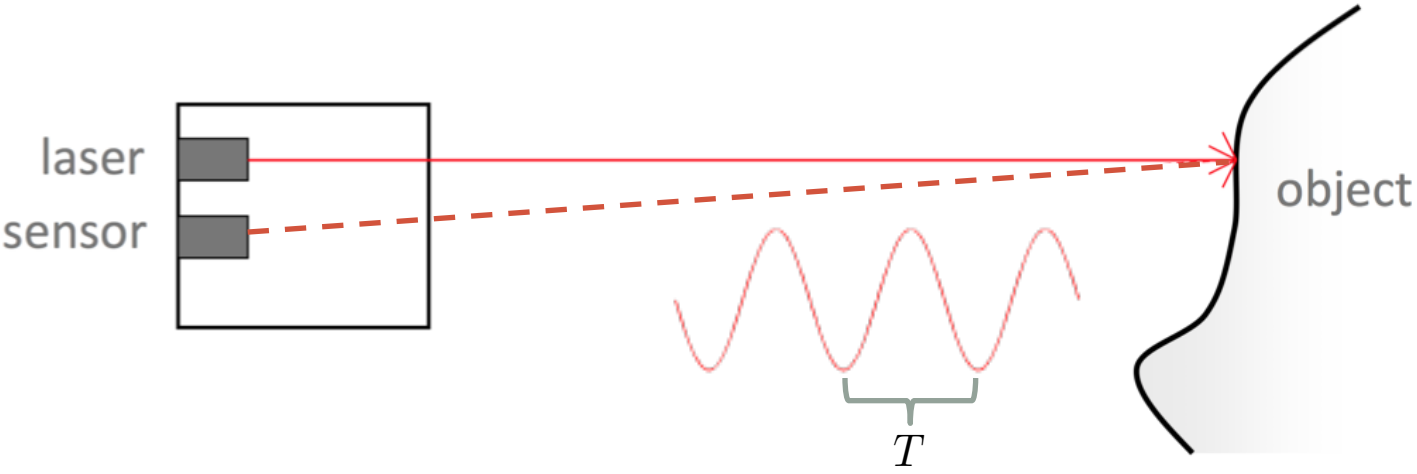
source: Michael Wand

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.

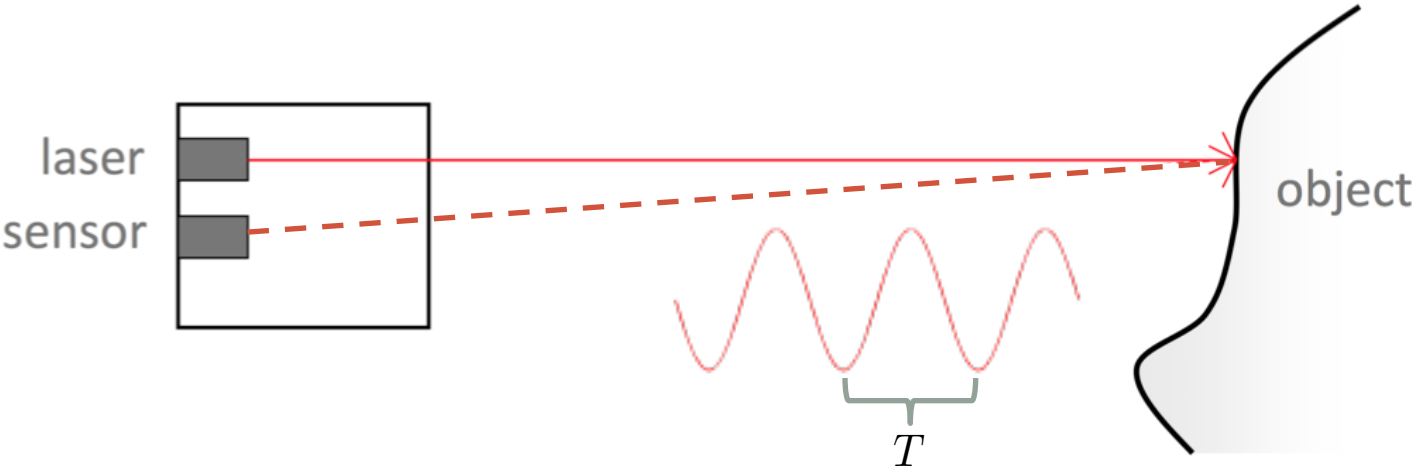
Phase-Based range-scanners



- Emitted signal: $s(t) = a\cos(2\pi f t)$
- Received signal: $r(t) = A\cos(2\pi f(t-\tau)) + B$
- Cross-correlation between emitted and received signals:

$$C(x) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} r(t)s(t+x)dt$$

Phase-Based range-scanners

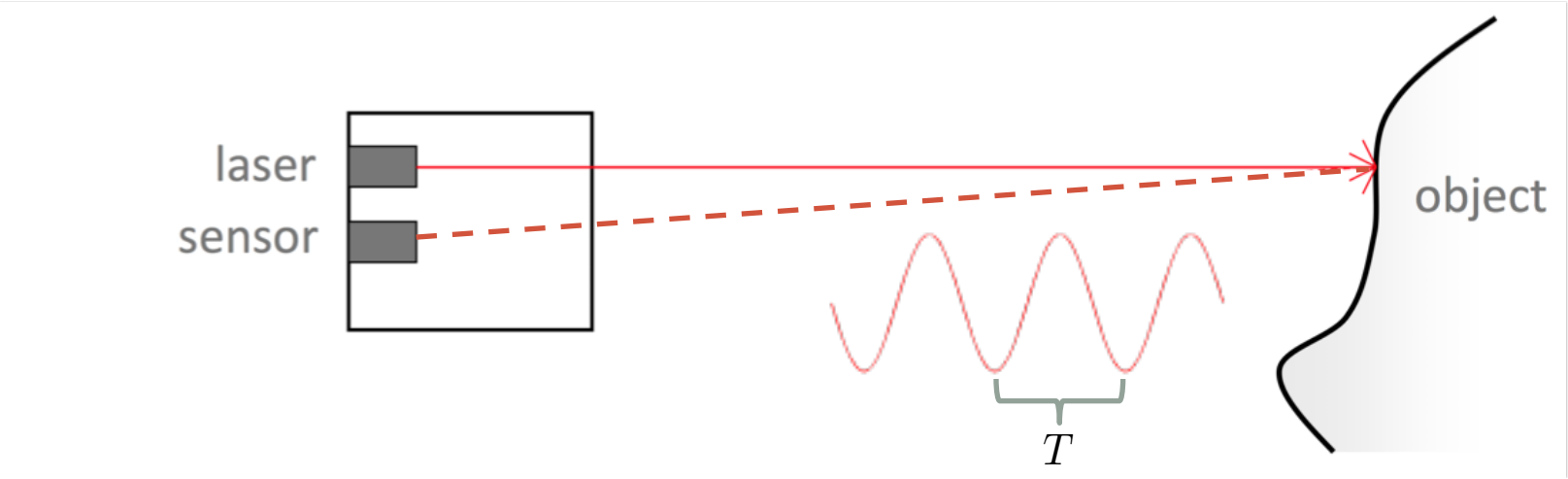


- Emitted signal: $s(t) = a\cos(2\pi f t)$
- Received signal: $r(t) = A\cos(2\pi f(t-\tau)) + B$
- Compute cross-correlation between emitted and received signals.

Unknowns: A, B, τ

- Four bucket trick: sample 4 different times in a period. Solve for unknowns.
- Compute depth (up to integer of wavelength): $d = \frac{1}{2}c\tau$

Phase-Based range-scanners

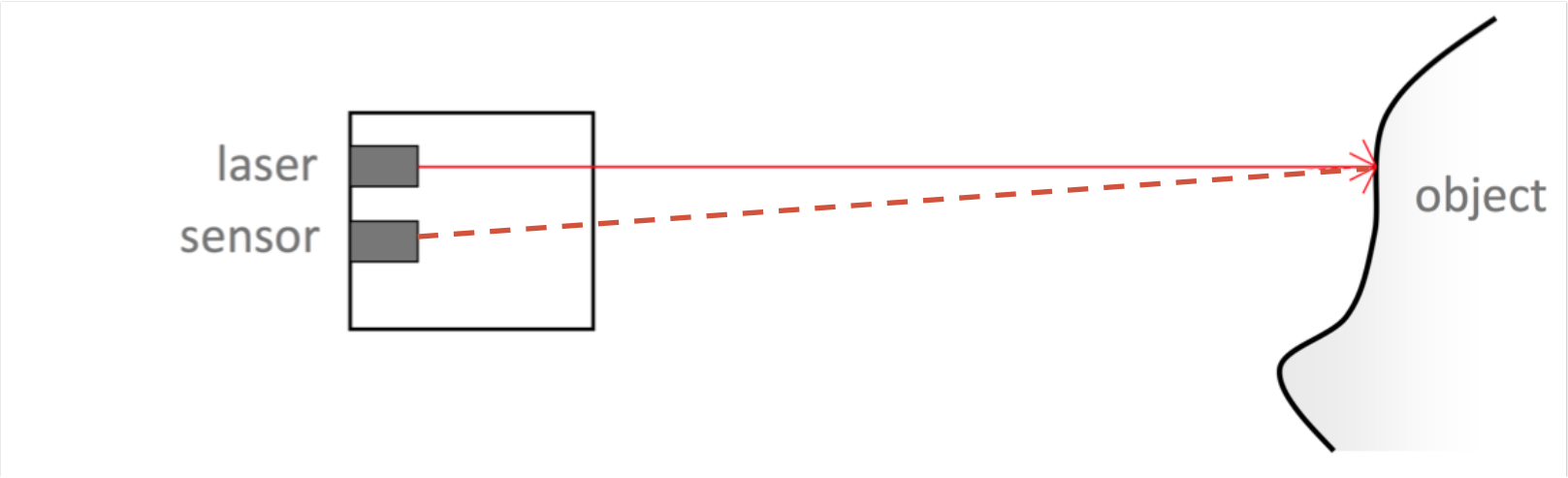


1. Instead of a pulse, emit a continuous **phase-modulated** beam.
2. Capture the reflection.
3. Measure the **phase-shift** between the output and input signals.
4. From the phase-shift, the distance can be computed **up to integer wavelength.**
5. **Greater frequency and accuracy** but **shorter range.**

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

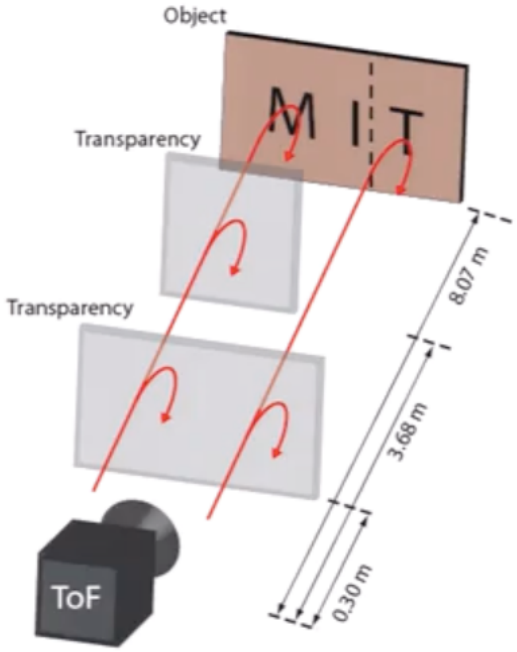
up to 79 meters vs. hundreds of meters

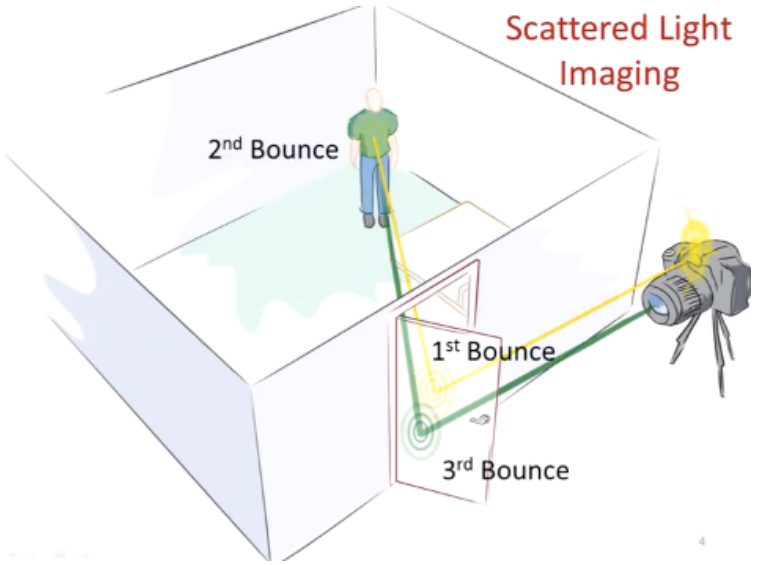
Range-scanners



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

Advanced Time-of-Flight.



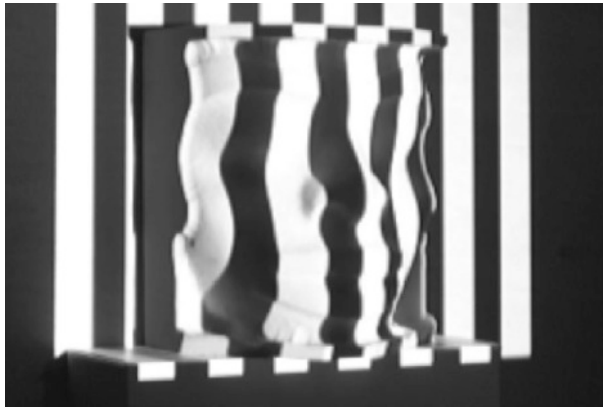


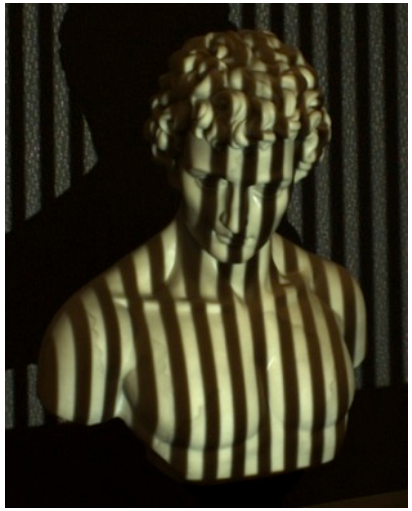
Izadi et al., SIGGRAPH 2014 Course on:
3D Imaging with Time-of-Flight cameras

Triangulation-based approaches

1. Add a **photometric sensor** (e.g. camera)
2. Record the position for a reference plane.
3. Change in recording position can be used to recover the depth.

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



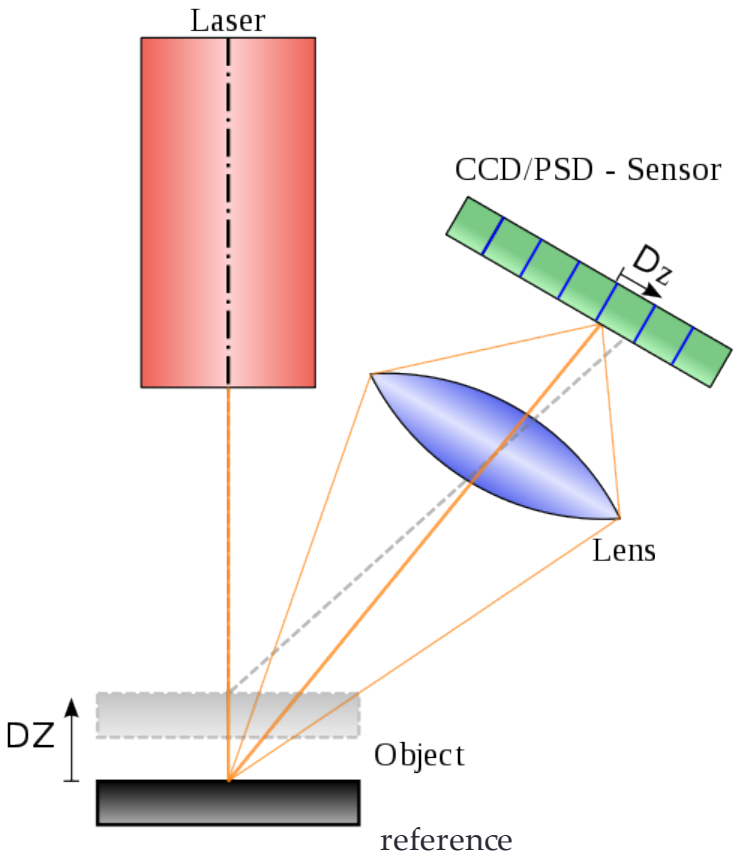


Triangulation-based approaches

1. Add a **photometric sensor** (e.g. camera)
2. Record the position for a reference plane.
3. Change in recording position can be used to recover the depth.

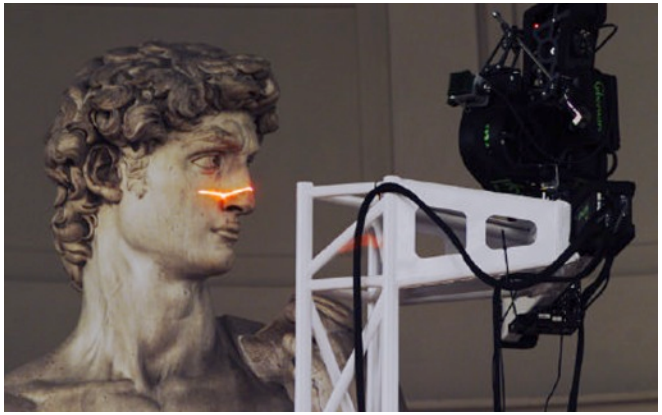
Intuition: the **depth** is related to the **shift** in the camera plane.

Need to compute DZ using D_Z.

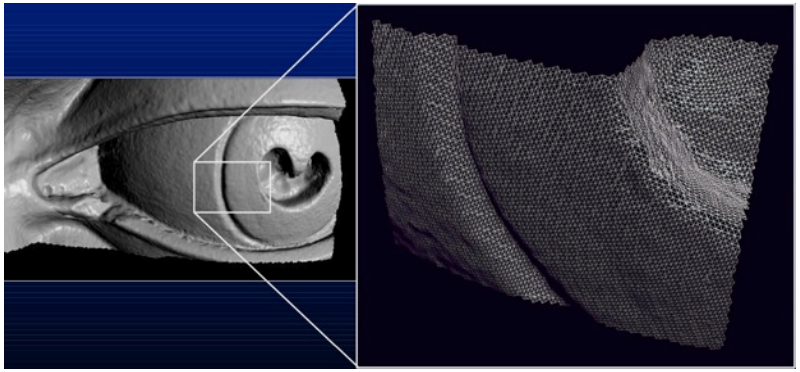


Triangulation-based approaches

1. Add a **photometric sensor** (e.g. camera)
2. Record the position for a reference plane.
3. If well-calibrated, can lead to extreme accurate depth measurements.
4. Main problem: slow and expensive.



David statue scan: over 1 billion points.



David's left eye:
source Levoy et al.

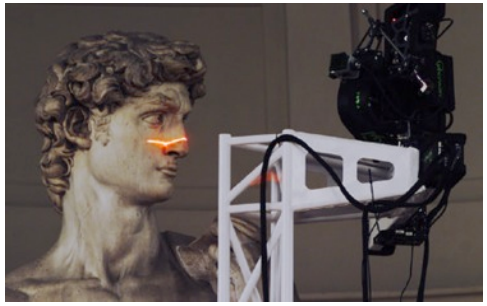
"The digital Michelangelo project: 3D scanning of large statues", Levoy et al., 2000

Structured-Light scanners

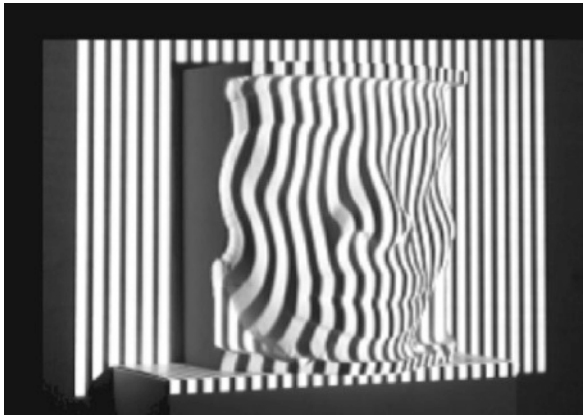
Same general idea as Triangulation based scanner.

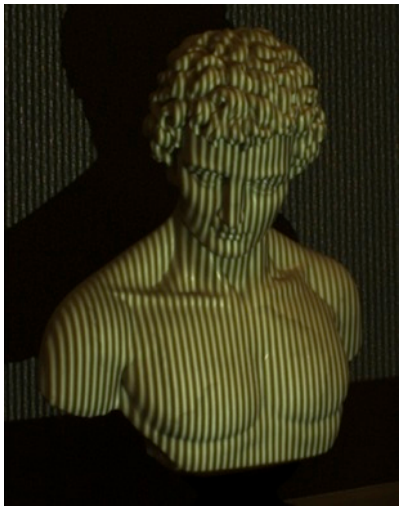
Main Idea: Replace laser with projector. Project stripes instead of sheets.

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



vs.

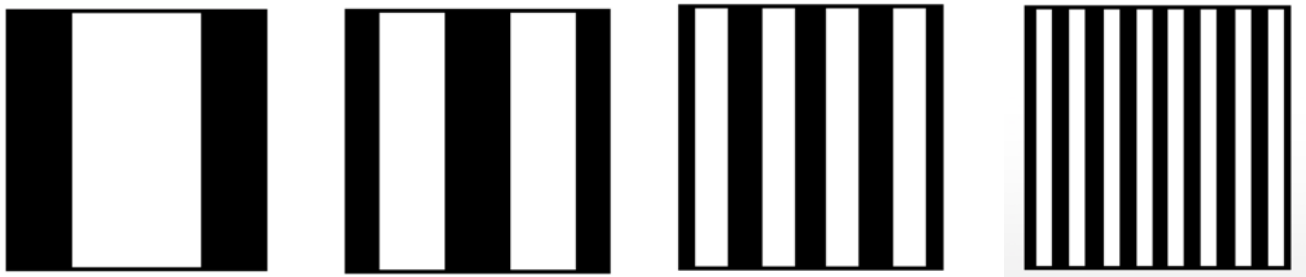




Structured-Light scanners

Same idea as Triangulation based scanner.

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

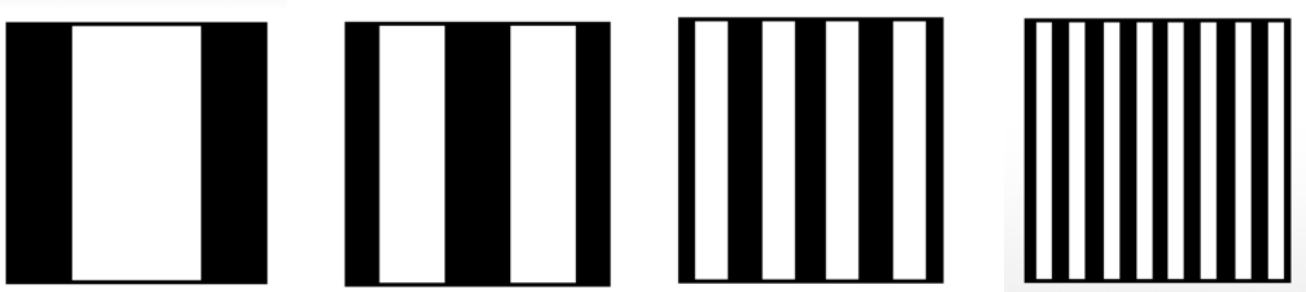


log(N) projections are sufficient to identify N stripes.

Structured-Light scanners

Same idea as Triangulation based scanner.

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



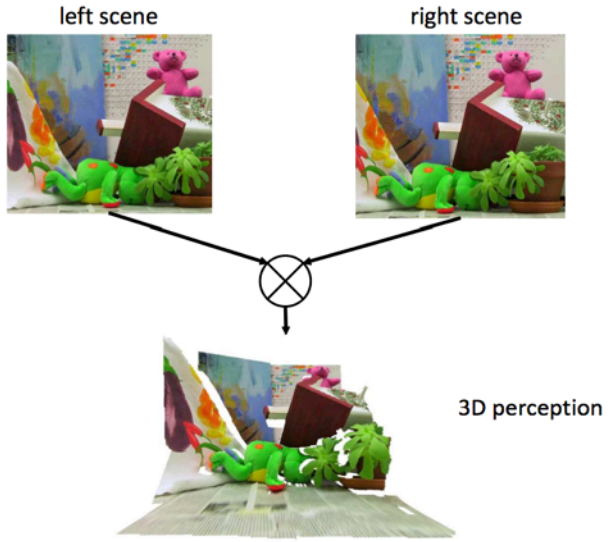
log(N) projections are sufficient to identify N stripes.

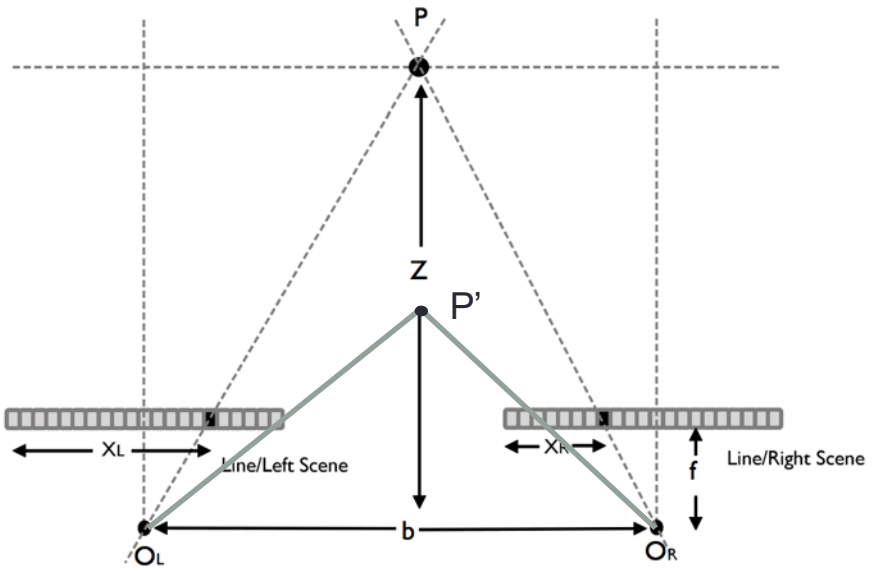
Advantage: cost and speed

Disadvantage: need controlled conditions & projector calibration.

Computer Vision based Techniques

Depth from stereo:





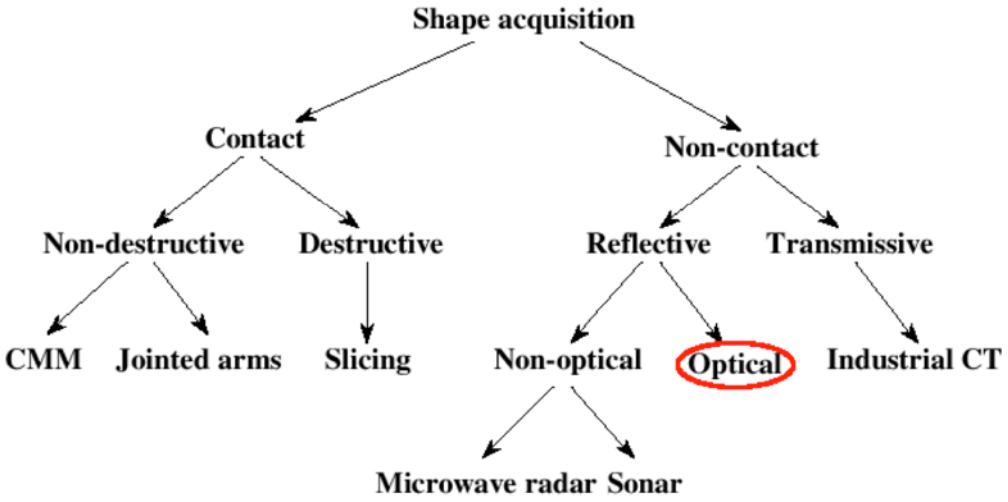
Given 2 images, shift in the x-axis is related to the depth.
Main challenge: establishing corresponding points across images: **very difficult.**

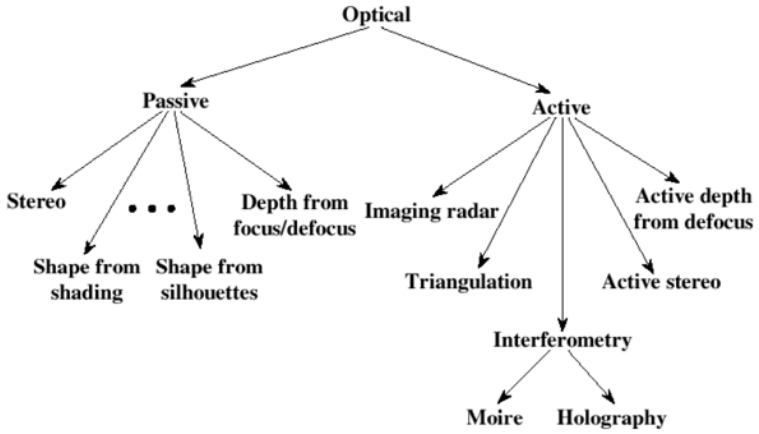
Computer Vision based Techniques

Depth from blur:

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

Multitude of other methods



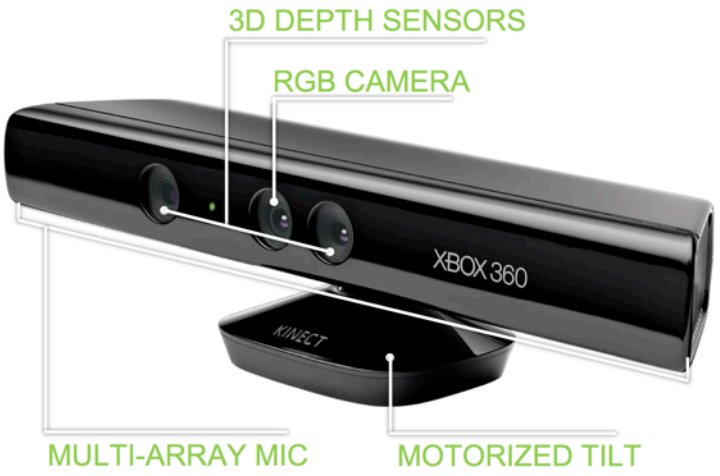


Rocchini et al. ‘01

Non-exhaustive taxonomy of 3d acquisition methods.

Microsoft Kinect scanner

Low-cost (100$) 3d scanner – gadget for Xbox.

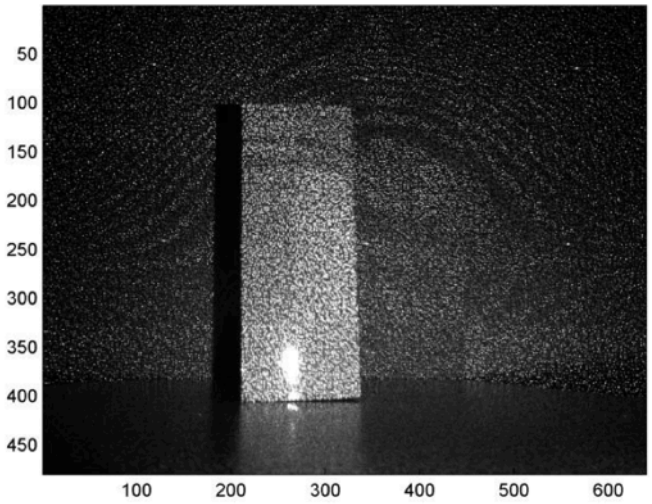


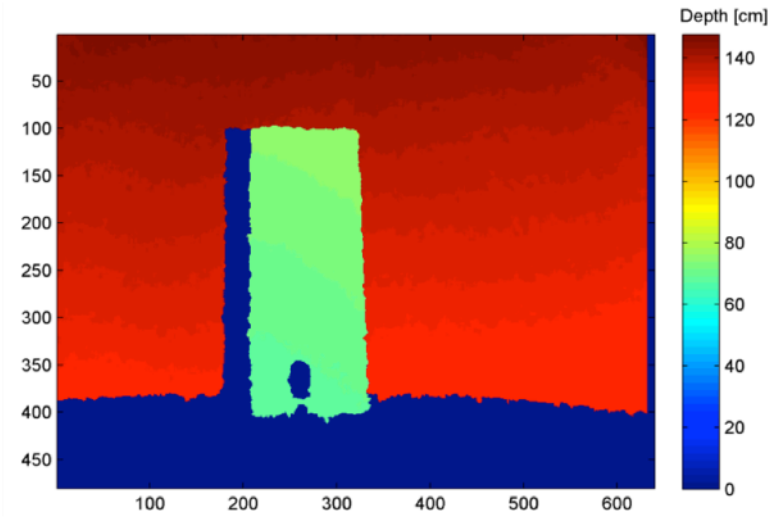
Allows to acquire Image (640 x 480) and 3d geometry (300k points) at 30 FPS.

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 1 (2009)

Low-cost (100$) 3d scanner – gadget for Xbox.



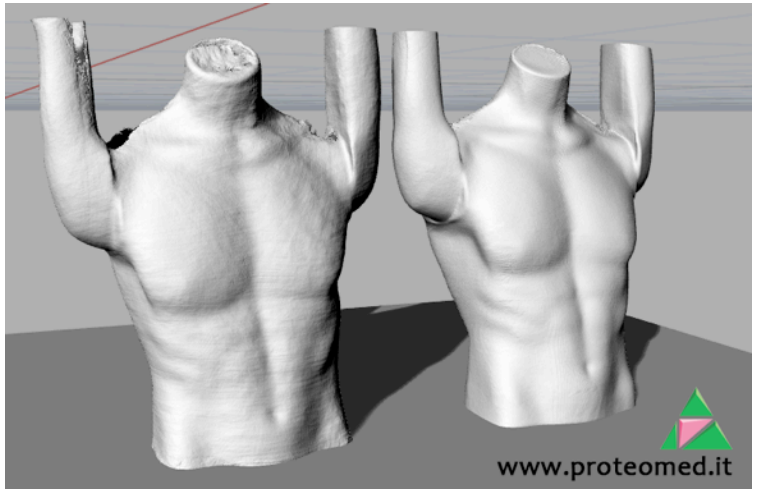


Allows to acquire Image (640 x 480) and 3d geometry (77k points) at 30 FPS.

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 v2 (2013)

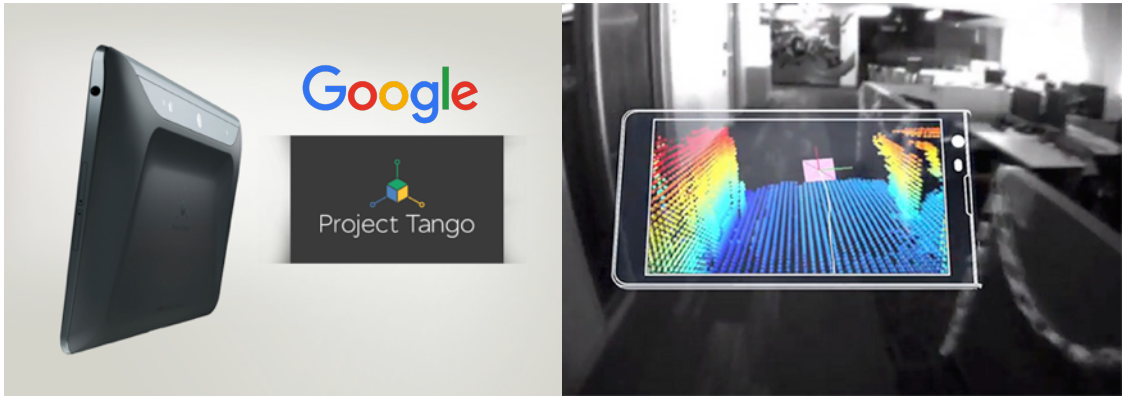
Low-cost (250$) 3d scanner – gadget for Xbox.

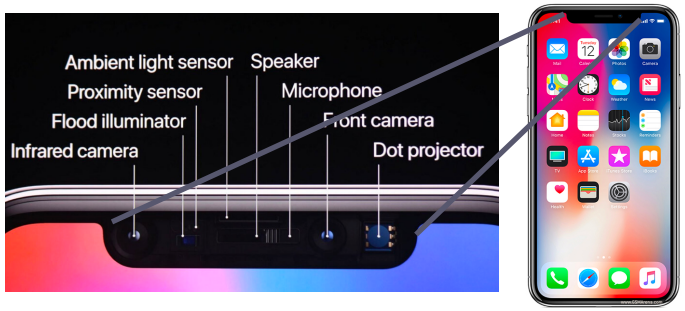


1080p video and 3d geometry (217k points) at 30 FPS.

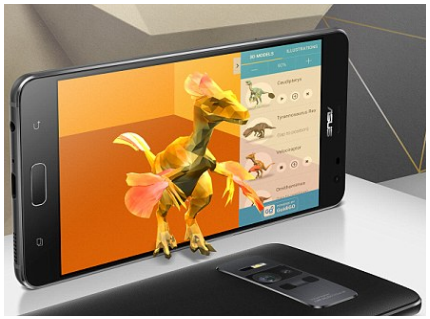
Uses Phase-based Time of Flight

Modern Mobile Devices (2017)





Apple iPhone X



Asus Zenfone AR

Sony Xperia XZ1

Typically use a combination of structured (infrared) light + stereo based depth.

Typical Scanning and Reconstruction Pipeline

Why Registration?

- Fundamental problem in geometric data processing.
- Appears in many shape *analysis* applications.
- One of the best-known algorithms in computer graphics and computational geometry.
- For you: very nice programming exercise.
- Quick introduction to an active research area.

Other Applications

- Manufacturing:

  One shape is a **model** and the other is a **scan** of a product. Finding defects.

- Medicine:

  Finding correspondences between 3D MRI scans of the same person or different people.

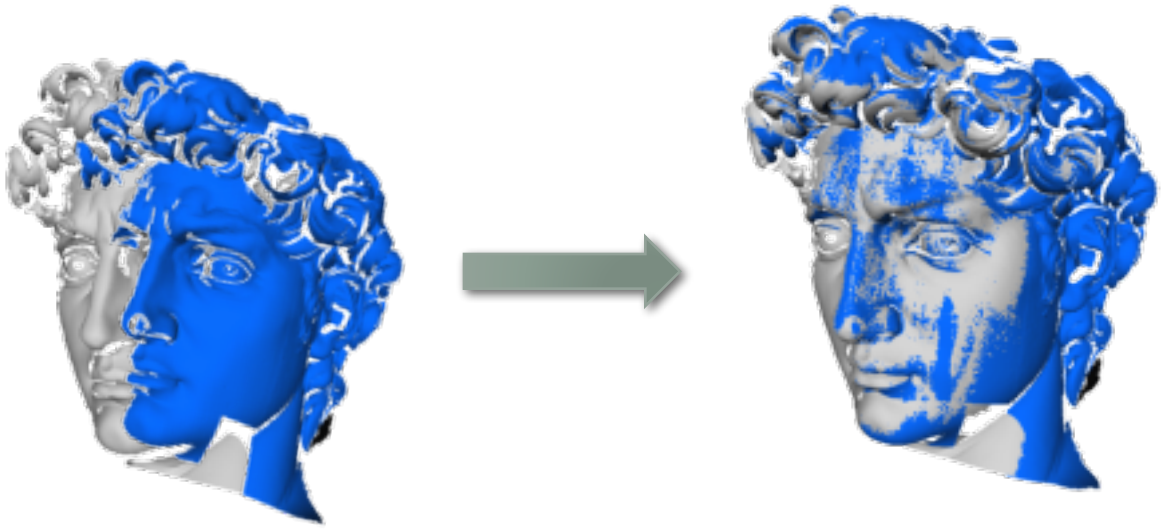
- Animation Reconstruction & 3D Video.

- Statistical Shape Analysis:

  Building models for a collection of shapes.

Local Alignment

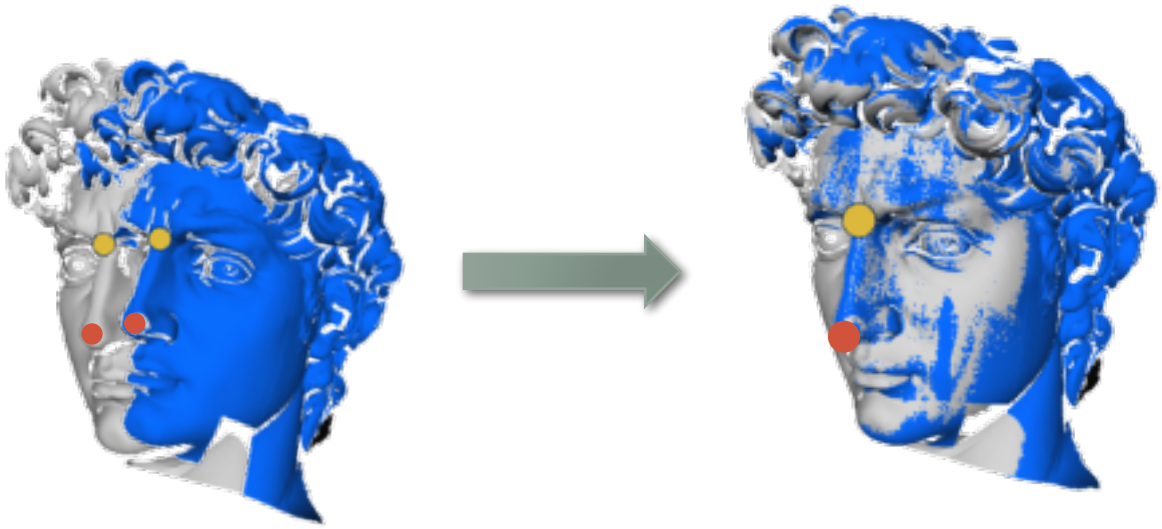
- Simplest instance of the registration problem



Given two shapes that are **approximately aligned** (e.g. by a human) we want to find the optimal tranformation.

Local Alignment

- What does it mean for an alignment to be **good**?



Intuition: want corresponding points to be close after transformation.

Problems

1. We don't know what points correspond.
2. We don't know the optimal alignment.

Iterative Closest Point (ICP)

- Approach: iterate between finding correspondences and finding the transformation:

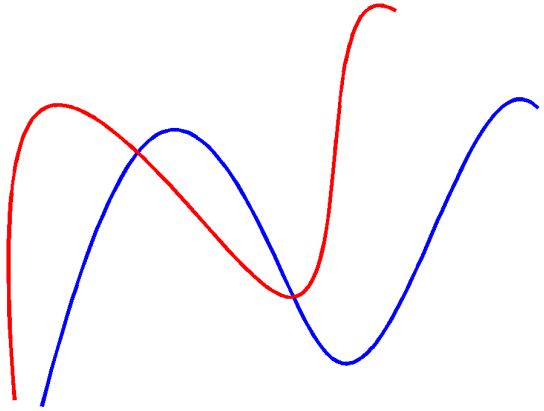
Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$$

Iterative Closest Point (ICP)

- Approach: iterate between finding correspondences and finding the transformation:

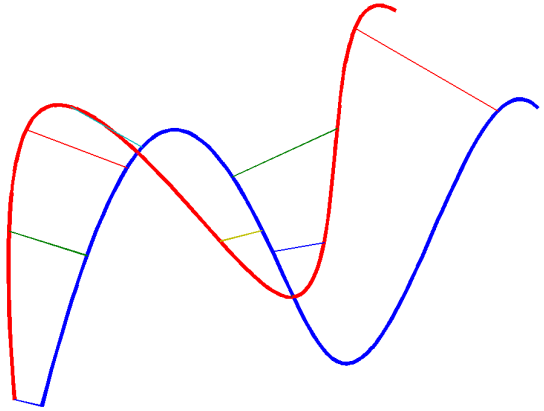


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

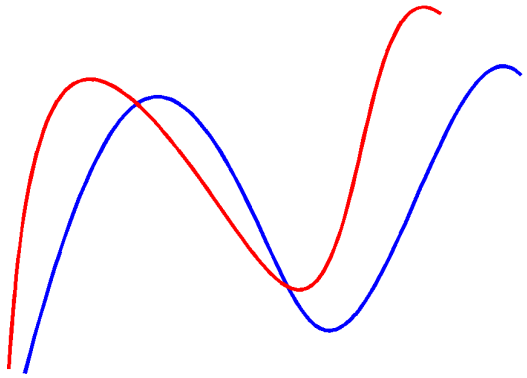


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

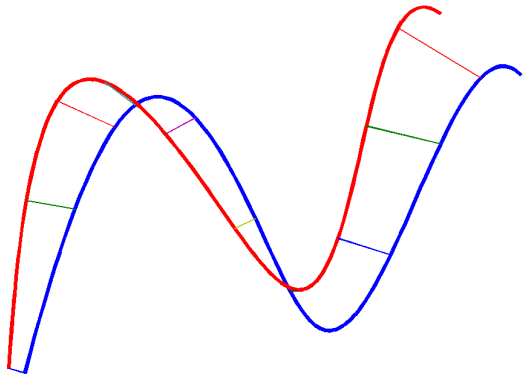


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

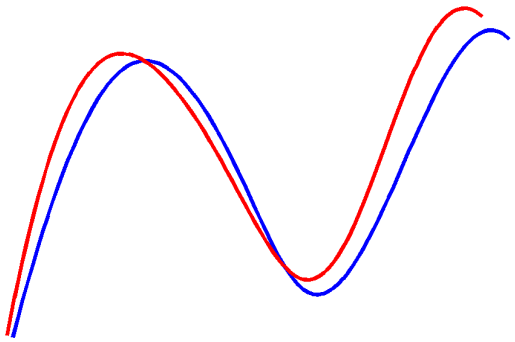


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

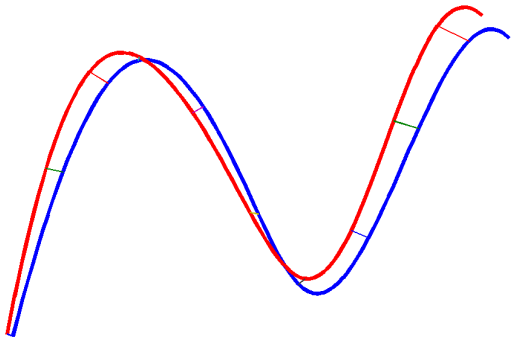


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

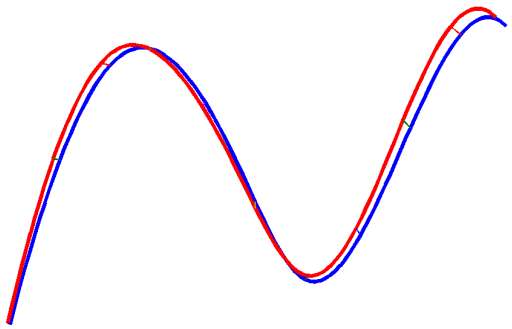


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:

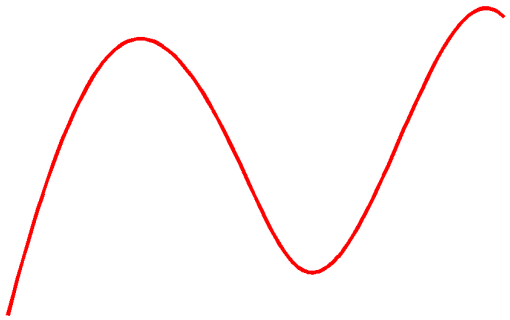


Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Approach: iterate between finding correspondences and finding the transformation:



Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

Iterative Closest Point

- Requires two main computations:

1. Computing nearest neighbors.
2. Computing the optimal transformation

ICP: Nearest Neighbor Computation

Closest points

$$y_i = \arg\min_{y \in Y} \|y - x_i\|$$

- How to find closest points **efficiently**?
- Straightforward complexity: $\mathcal{O}(MN)$

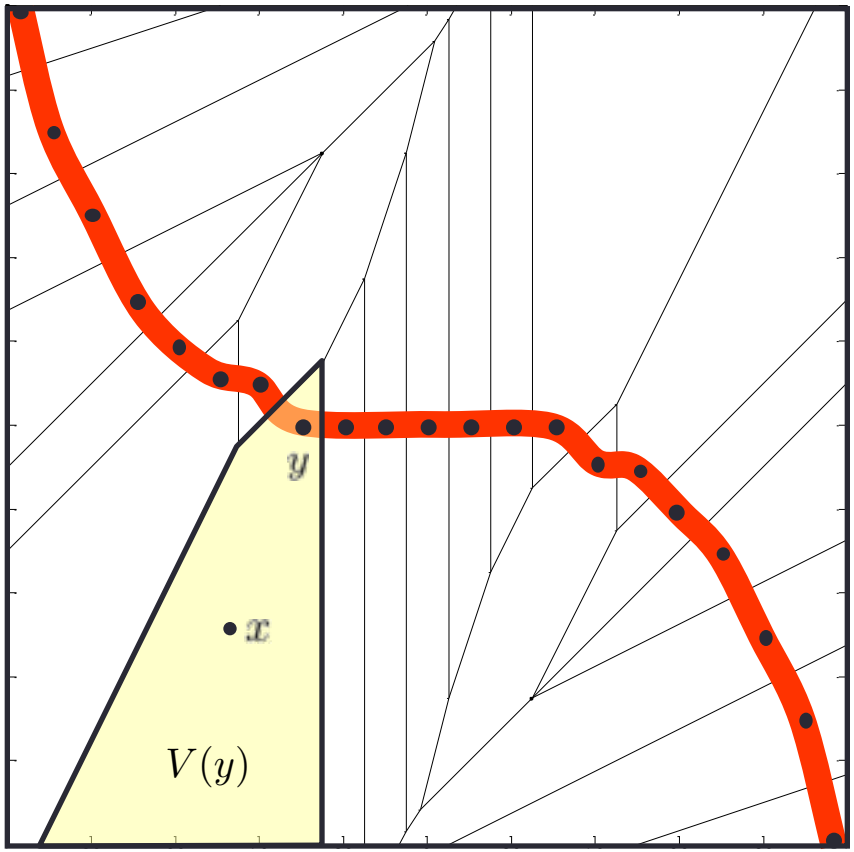
  M number of points on X, N number of points on Y.

- Y divides the space into **Voronoi cells**

$$V(y \in Y) = \{z \in \mathbb{R}^3 : \|y - z\| < \|y' - z\| \ \forall\ y' \in Y \neq y\}$$

- Given a query point y, **determine to which cell it belongs**.

Closest points: Voronoi Cells

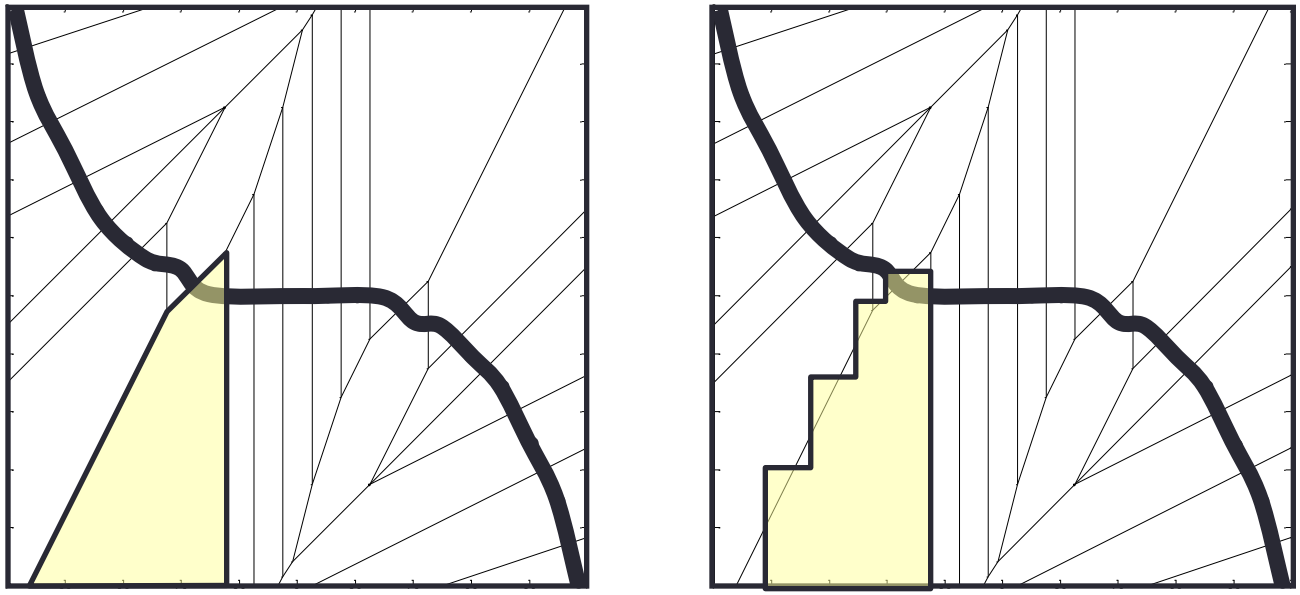


$$V(y \in Y) = \{z \in \mathbb{R}^3 : \|y - z\| < \|y' - z\| \ \forall\ y' \in Y \neq y\}$$

Image source:
M. Bronstein

Closest points: Voronoi Cells

Approximate nearest neighbors

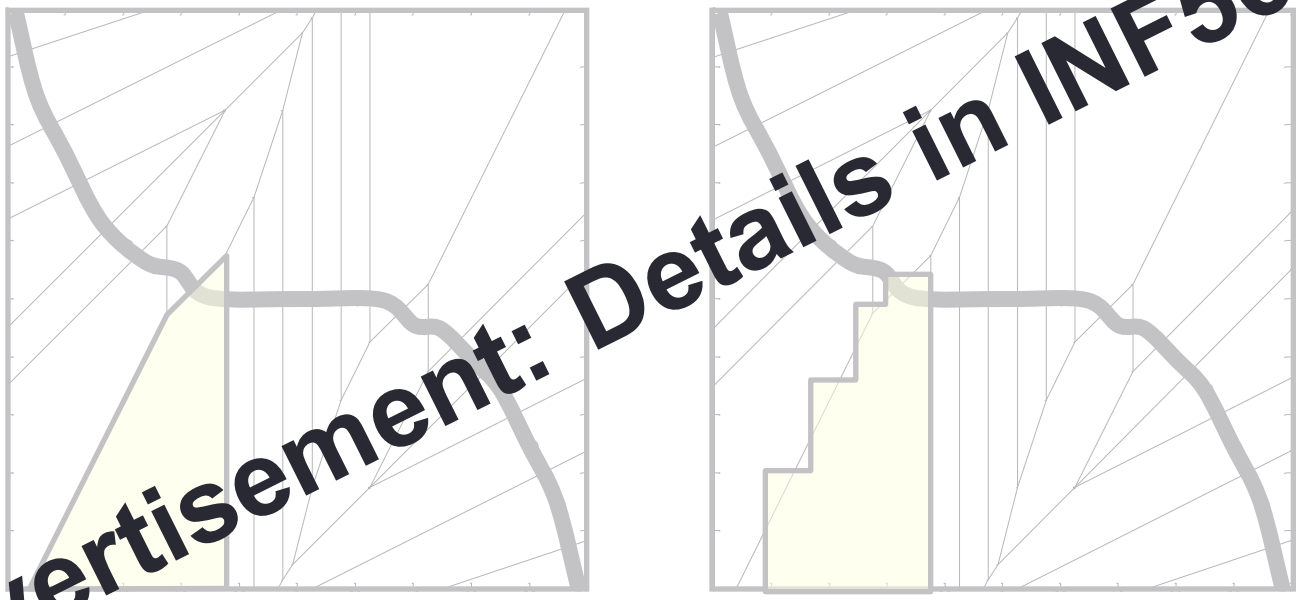


M. Bronstein

- To reduce search complexity, **approximate** Voronoi cells.
- Use **binary space partition trees** (e.g. **kd-trees** or **octrees**).
- Approximate nearest neighbor search complexity: $\mathcal{O}(N \log M)$.

Closest points: Voronoi Cells

Approximate nearest neighbors



M. Bronstein

Advertisement: Details in INF562

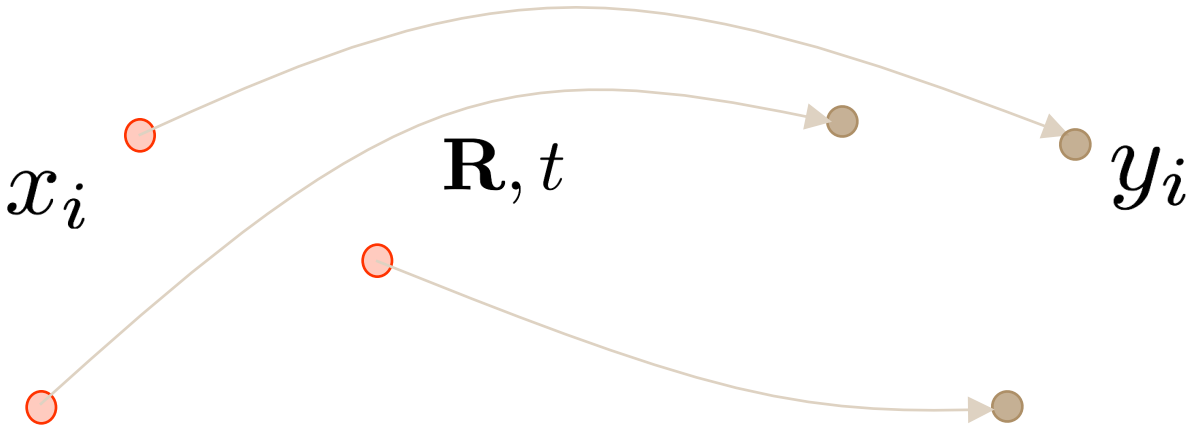
- To reduce search complexity, **approximate** Voronoi cells.
- Use **binary space partition trees** (e.g. **kd-trees** or **octrees**).
- Approximate nearest neighbor search complexity: $\mathcal{O}(N \log M)$.

ICP: Optimal Transformation

Problem Formulation:

1. Given two sets points: $\{x_i\}, \{y_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: $\mathbf{R}, t$ that minimizes:

$$\mathbf{R}_{\text{opt}}, t_{\text{opt}} = \underset{\mathbf{R}, t}{\arg\min} \sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$$



ICP: Optimal Transformation

Problem Formulation:

1. Given two sets points: $\{x_i\}, \{y_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: $\mathbf{R}, t$ that minimizes:

$$\mathbf{R}_{\text{opt}}, t_{\text{opt}} = \underset{\mathbf{R}^T\mathbf{R}=\text{Id}, t}{\arg\min} \sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$$

Closed form solution with quaternions.

Horn, B. K. P., Closed-form solution of absolute orientation using unit quaternions, 1987

Closed-form solution of absolute orientation using unit quaternions.

Horn, B. K. P., *Journal of the Optical Society of America*, Vol. 4 1987

Problem Formulation:

1. Given two sets points: $\{p_i\}, \{q_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: R and q_0 that minimizes:

$$\sum_{i=1}^{n} \|R(q_i) + q_0 - p_i\|^2$$

2. In practice, have many points. Perform this step many times. Want a **fast** way to get an **exact solution**. Representation is important.

Given two sets points: $\{p_i\}, \{q_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: R and q_0 that minimizes: $\sum_{i=1}^{n} \|R(q_i) + q_0 - p_i\|^2$

Finding the translation part q_0.

1. Let $\bar{q} = \frac{1}{n}\sum_1^n q_i$ and $\bar{p} = \frac{1}{n}\sum_1^n p_i$: the centroids of $\{q_i\}$ and $\{p_i\}$

   and let $q_i' = q_i - \bar{q}$ and $p_i' = p_i - \bar{p}$. Then $\sum q_i' = \sum p_i' = 0$.

2. $$\sum_{i=1}^{n} \|R(q_i) + q_0 - p_i\|^2 = \sum_{i=1}^{n} \|R(q_i') + q_0' - p_i'\|^2 =$$

   where: $q_0' = q_0 + R(\bar{q}) - \bar{p}$

   $$\sum_{i=1}^{n} \|R(q_i') - p_i'\|^2 - 2q_0' \sum_{i=1}^{n} (R(q_i') - p_i') + n\|q_0'\|^2 =$$

   $$\sum_{i=1}^{n} \|R(q_i') - p_i'\|^2 + n\|q_0'\|^2$$

   minimized if $q_0' = 0$ $\Longrightarrow$ $q_0 = \bar{p} - R(\bar{q})$

Given two sets points: $\{p_i\}, \{q_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: R and q_0 that minimizes: $\sum_{i=1}^{n} \|R(q_i) + q_0 - p_i\|^2$

Need to find the rotation R.

1. Several ways to represent rotation: Orthonormal matrix, Euler angle, Gibbs vector, quaternions, etc. Which one is most useful in this case?

2. Plan of action:

   Want to minimize $\sum_{i=1}^{n} \|R(q_i') - p_i'\|^2 = \sum_{i=1}^{n} \|R(q_i')\|^2 - 2R(q_i) \cdot p_i' + \|p_i'\|^2$

   Since rotation preserves lengths. Need to *maximize*: $\sum_{i=1}^{n} R(q_i') \cdot p_i'$

   Question: What **unit** vector maximizes $u^{\mathbf{T}} N u$ if N is symmetric?

   Answer: Eigenvector corresponding to the maximum eigenvalue of N.

Quaternions basics

4. Conjugation: $q^* = q_0 - q_x i - q_y j - q_z k$. Matrix corresponding to the conjugate is the transpose of the original.
5. Regular vectors can be represented with purely imaginary quaternions:

$$q = q_x i + q_y j + q_z k$$

6. Rotation can be represented with composite product by a **unit quaternion** r **:**

$$R(q) = r\, q\, r^*$$

where $r = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(w_x i + w_y j + w_z k)$**.**

Conversely, every unit quaternion corresponds to a rotation.

Want to optimize the rotation:

$$\min_R \sum_{i=1}^{n} \|R(q_i') - p_i'\|_2^2$$

Need to find the rotation R.

2. Plan of action:

Need to *maximize*: $\sum_{i=1}^{n} R(q_i) \cdot p_i'$

3. If represent R using orthonormal matrices, then get: $\sum_{i=1}^{n} p_i^{\mathbf{T}} \, R \, q_i$

Would prefer: $\sum_{i=1}^{n} u(R)^{\mathbf{T}} N_i(q_i, p_i) u(R)$, where u is a vector and N_i - matrices

Quaternions allow us to express rotation in such form.

Quaternions (two slides intro).

1. Vectors with 4 components, 3 of which are imaginary:

$$q = q_0 + q_x i + q_y j + q_z k$$

2. Multiplication is defined using the rules:

$$i^2 = j^2 = k^2 = ijk = -1$$

from which follows: $ij = k \quad jk = i \quad ki = j$

$ji = -k \quad kj = -i \quad ik = -j$

In matrix form: $qr = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} r = \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{bmatrix} q$

$$qr = Qr = \bar{R}q$$

Quaternions (two slides intro).

1. Vectors with 4 components, 3 of which are imaginary:

$$q = q_0 + q_x i + q_y j + q_z k$$

2. Multiplication is defined using the rules:

$$i^2 = j^2 = k^2 = ijk = -1$$

from which follows: $ij = k \quad jk = i \quad ki = j$

$ji = -k \quad kj = -i \quad ik = -j$

In matrix form: $qr = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} r = \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{bmatrix} q$

3. Dot product:

$$p \cdot q = \frac{1}{2}\left(pq^* + qp^*\right) \implies (pr) \cdot q = p \cdot (qr^*)$$

Matrices associated with unit quaternions are orthonormal.

Back to Alignment.

1. Want to maximize: $\sum_{i=1}^{n} R(q_i') \cdot p_i'$

2. In quaternion form: $\sum_{i=1}^{n} (r q_i r^*) \cdot p_i = \sum_{i=1}^{n} (r q_i) \cdot (p_i r)$

$$= \sum_{i=1}^{n} (\bar{Q} r) \cdot (P r) = \sum_{i=1}^{n} r^T \bar{Q}^T P r$$

$$= r^T \left(\sum_{i=1}^{n} \bar{Q}^T P \right) r$$

3. The optimal rotation quaternion is the eigenvector corresponding to the maximum eigenvalue of: $\sum_{i=1}^{n} \bar{Q}^T P$

ICP: Optimal Transformation

Problem Formulation:

1. Given two sets points: $\{x_i\}, \{y_i\}, i = 1..n$ in $\mathbb{R}^3$. Find the rigid transform: $\mathbf{R}, t$ that minimizes: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

2. Closed form solution with rotation matrices:
    1. Construct: $C = \sum_{i=1}^{N} (y_i - \mu^Y)(x_i - \mu^X)^T$, where $\mu^X = \frac{1}{N}\sum_i x_i$,
    2. Compute the SVD of C: $C = U\Sigma V^T$ $\mu^Y = \frac{1}{N}\sum_i y_i$
        1. If $\det(UV^T) = 1, R_{\text{opt}} = UV^T$
        2. Else $R_{\text{opt}} = U\tilde{\Sigma}V^T, \tilde{\Sigma} = \text{diag}(1, 1, \ldots, -1)$
    3. Set $t_{\text{opt}} = \mu^Y - R_{\text{opt}}\mu^X$

Note that C is a 3x3 matrix. SVD is very fast.

Arun et al., Least-Squares Fitting of Two 3-D Point Sets

Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing: $\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$

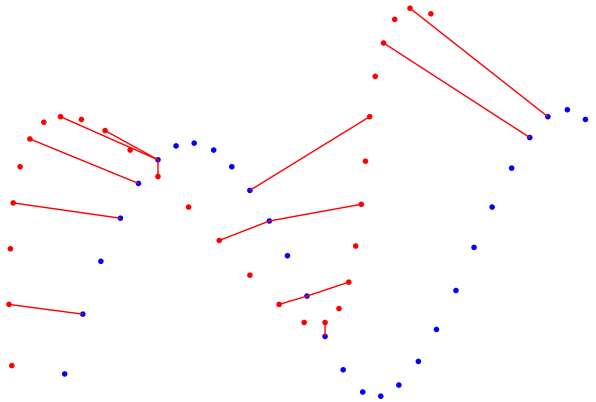
Convergence:

- at each iteration $\sum_{i=1}^{N} d^2(x_i, Y)$ decreases.
- Converges to local minimum
- Good initial guess: global minimum.

[Besl&McKay92]

Variations of ICP

1. Selecting source points (from one or both scans): sampling
2. Matching to points in the other mesh
3. Weighting the correspondences
4. Rejecting certain (outlier) point pairs
5. Assigning an error metric to the current transform
6. Minimizing the error metric w.r.t. transformation

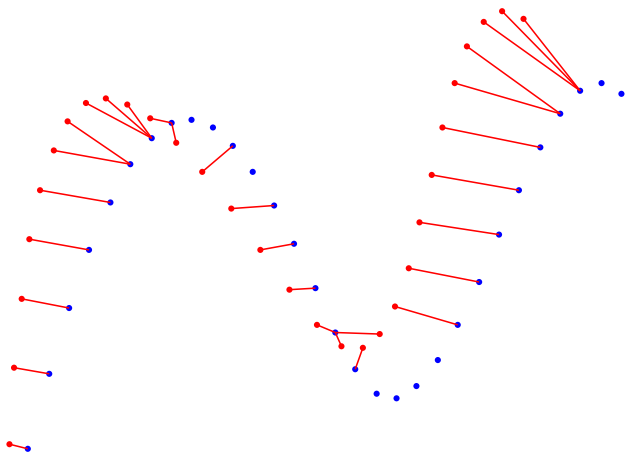


Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$$



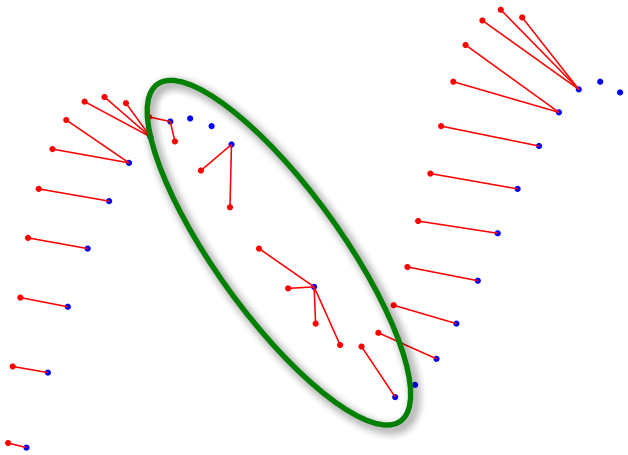
Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\sum_{i=1}^{N} \|\mathbf{R}x_i + t - y_i\|_2^2$$

Problem:
uneven sampling



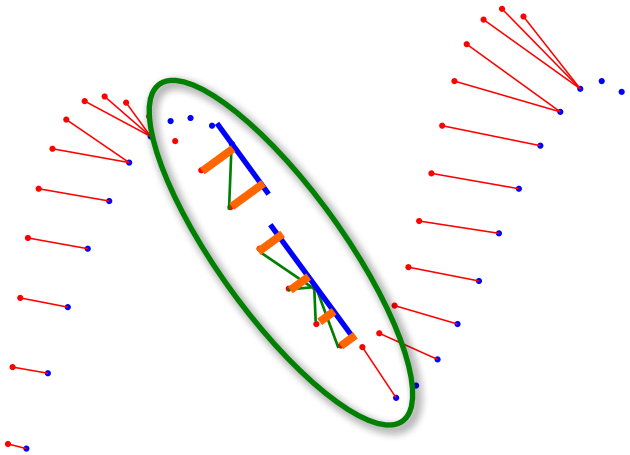
Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\sum_{i=1}^{N} d(\mathbf{R}x_i + t, P(y_i))^2 = \sum_{i=1}^{N} \left((\mathbf{R}x_i + t - y_i)^T \mathbf{n}_{y_i}\right)^2$$

Solution:
Minimize distance to the tangent plane



Chen, Medioni, '91

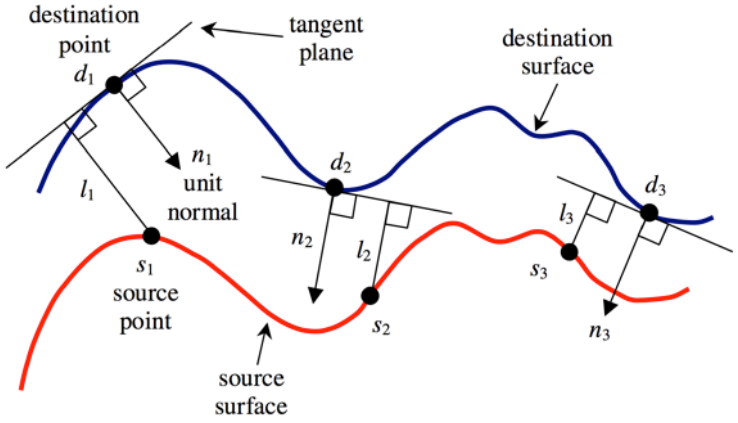
Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\sum_{i=1}^{N} d(\mathbf{R}x_i + t, P(y_i))^2 = \sum_{i=1}^{N} \left((\mathbf{R}x_i + t - y_i)^T \mathbf{n}_{y_i}\right)^2$$

Solution:
Minimize distance to the tangent plane



Kok-Lim Low, '04

Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\mathbf{R}_{\text{opt}}, t_{\text{opt}} = \underset{\mathbf{R}^T\mathbf{R}=\text{Id},\, t}{\arg\min} \sum_{i=1}^{N} \left((\mathbf{R}x_i + t - y_i)^T \mathbf{n}_{y_i} \right)^2$$

Question:

How to minimize the error?

Challenge:

Although the error is **quadratic** (linear derivative), the space of rotation matrices is **not linear.**

Problem:

No closed form solution!

Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation $\mathbf{R}, t$ minimizing:

$$\mathbf{R}_{\text{opt}}, t_{\text{opt}} = \underset{\mathbf{R}^T\mathbf{R}=\text{Id},\, t}{\arg\min} \sum_{i=1}^{N} \left((\mathbf{R}x_i + t - y_i)^T \mathbf{n}_{y_i} \right)^2$$

Common Solution:

Linearize rotation. Assume rotation angle is small.

$$\mathbf{R}x_i \approx x_i + r \times x_i$$

r : axis,
$\|r\|_2$: angle of rotation.

Note: follows from Rodrigues's formula

$$R(r, \alpha)x_i = x_i \cos(\alpha) + (r \times x_i)\sin(\alpha) + r(r^T x_i)(1 - \cos(\alpha))$$

And first order approximations: $\sin(\alpha) \approx \alpha, \;\; \cos(\alpha) \approx 1$

Iterative Closest Point.

Given a pair of shapes, X and Y, iterate:

1. For each $x_i \in X$ find **nearest** neighbor $y_i \in Y$.
2. Find deformation r, t minimizing:

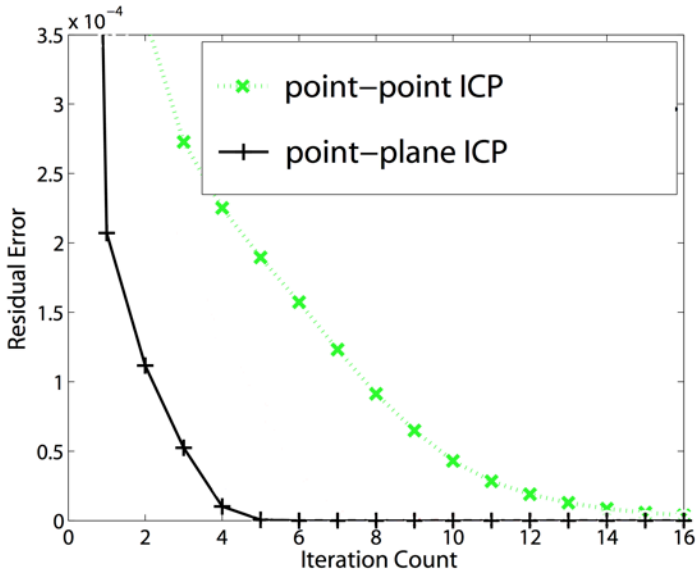
$$E(r,t) = \sum_{i=1}^{N} \left((x_i + r \times x_i + t - y_i)^T \mathbf{n}_{y_i} \right)^2$$

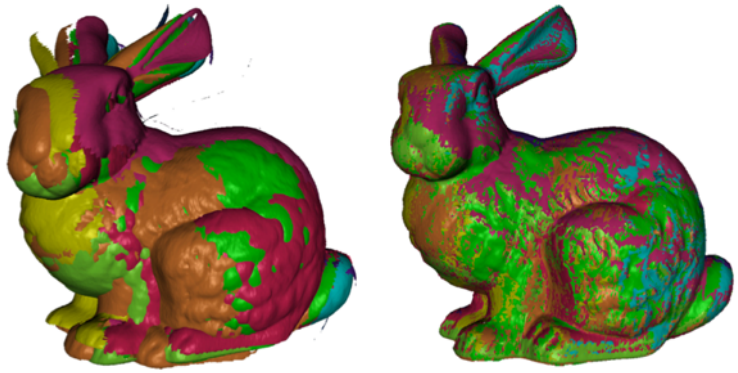
Setting: $\frac{\partial}{\partial r} E(r,t) = 0$ and $\frac{\partial}{\partial t} E(r,t) = 0$ leads to a 6x6 linear system

$$Ax = b$$

$$x = \begin{pmatrix} r \\ t \end{pmatrix} \quad A = \sum \begin{pmatrix} x_i \times \mathbf{n}_{y_i} \\ \mathbf{n}_{y_i} \end{pmatrix} \begin{pmatrix} x_i \times \mathbf{n}_{y_i} \\ \mathbf{n}_{y_i} \end{pmatrix}^T \quad b = \sum (y_i - x_i)^T \mathbf{n}_{y_i} \begin{pmatrix} x_i \times \mathbf{n}_{y_i} \\ \mathbf{n}_{y_i} \end{pmatrix}$$

Iterative Closest Point.





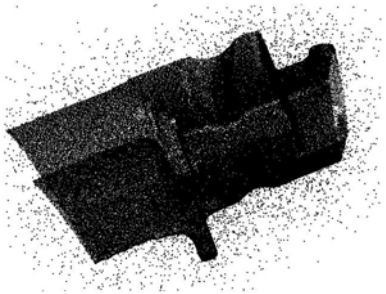
Aligning the bunny to itself.

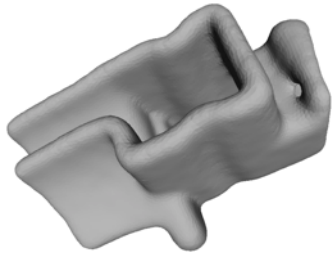
N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas. Registration of Point Cloud Data from a Geometric Optimization Perspective. Proc. SGP, 2004.

3d Point Cloud Processing

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

1. Shape scanning (acquisition)
2. If have multiple scans, align them.
3. Smoothing – remove local noise and outliers.
4. Estimate surface normals.
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh

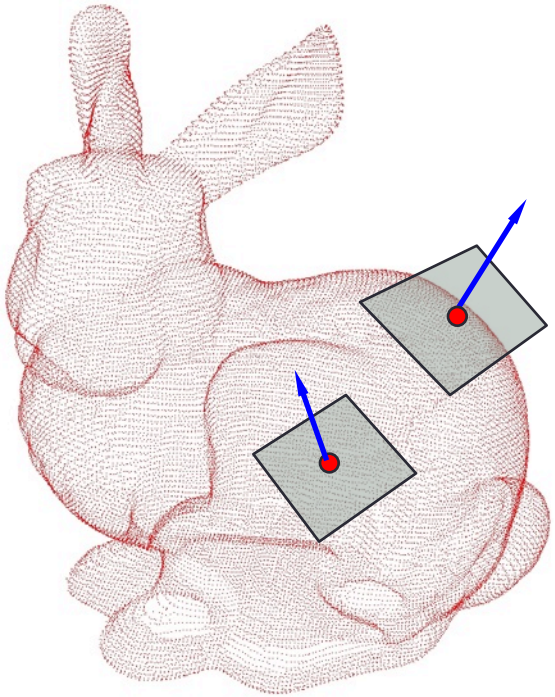




Normal Estimation and Outlier Removal

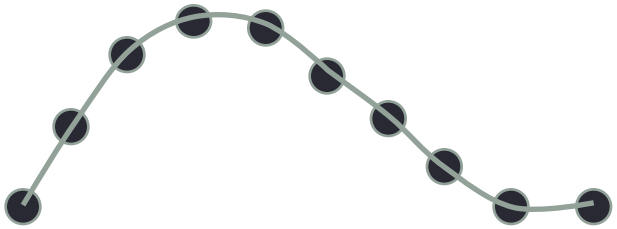
Fundamental problems in Point cloud processing.

Although seemingly very different, can be solved with the same general approach.



Normal Estimation

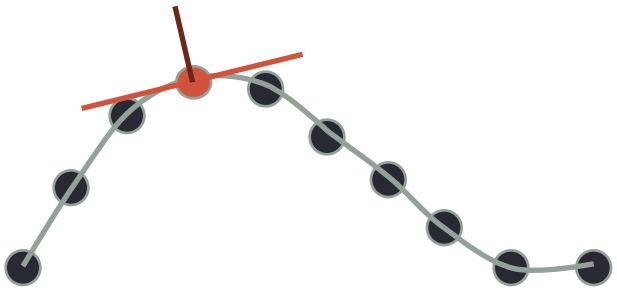
Assume we have a clean sampling of the surface.



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

Normal Estimation

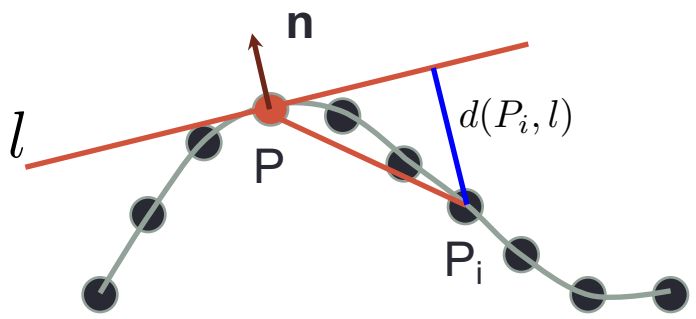
Assume we have a clean sampling of the surface.



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

Normal Estimation

Assume we have a clean sampling of the surface.



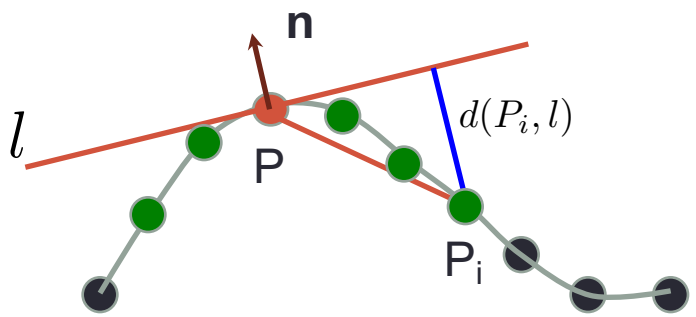
Goal: find best approximation of the normal at P.

Method: Given line l through P with normal $\mathbf{n}$, 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.



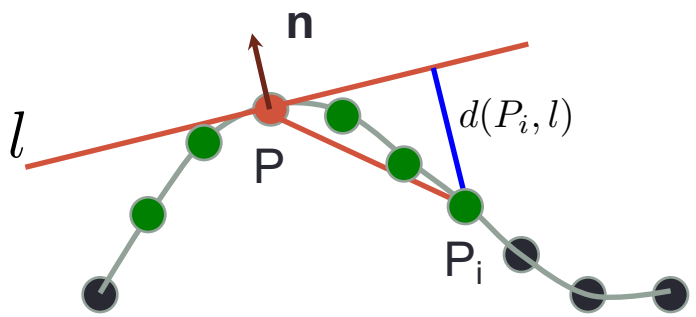
Goal: find best approximation of the normal at P.

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



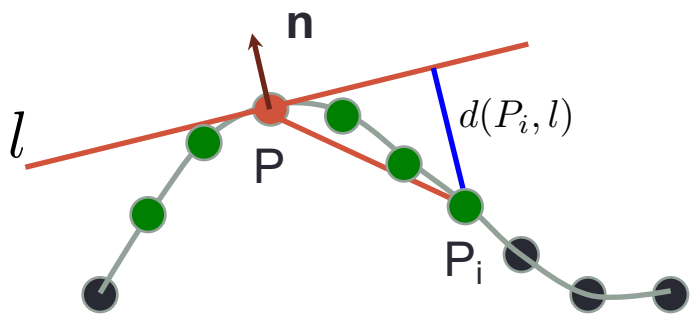
Using Lagrange multiplier:

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

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

Normal Estimation

Assume we have a clean sampling of the surface.



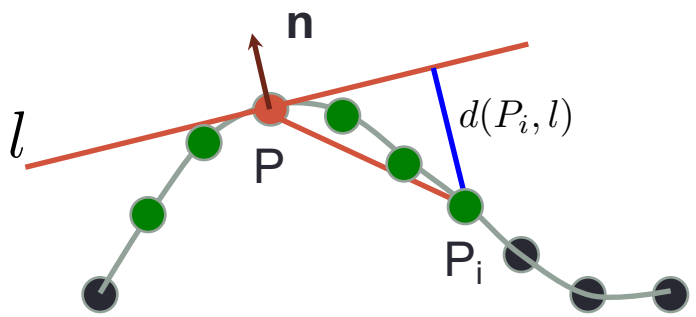
Using Lagrange multiplier:

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

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

Normal Estimation

Assume we have a clean sampling of the surface.



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

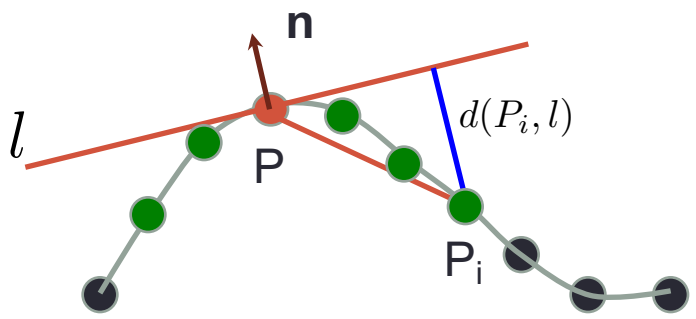
$$C\mathbf{n} = \lambda\mathbf{n} \qquad C = \sum_{i=1}^{k}(p_i - P)(p_i - P)^T$$

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.



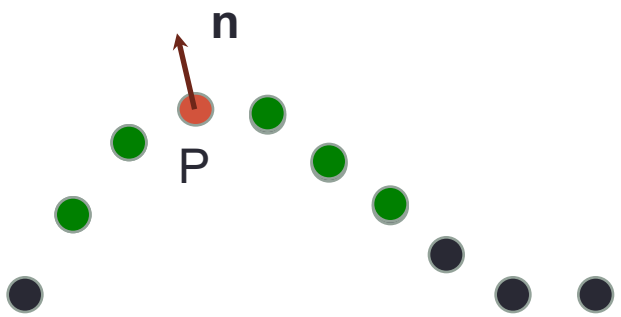
The normal **n** must be an eigenvector of the matrix:

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

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

Normal Estimation

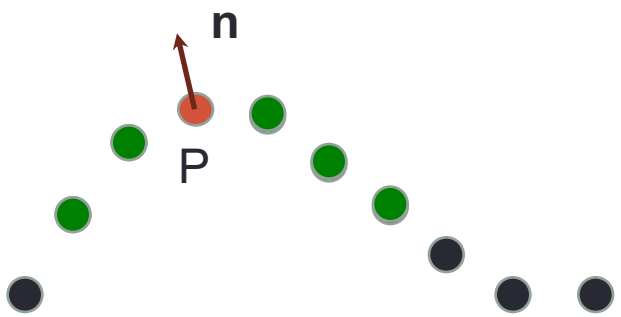
Method Outline (PCA):



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):



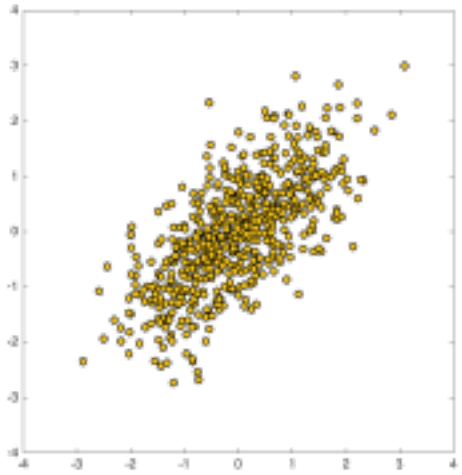
1. Given a point P in the point cloud, find its *k* nearest neighbors.
2. Compute $C = \sum_{i=1}^{k} (p_i - P)(p_i - P)^T$
3. **n:** eigenvector corresponding to the smallest eigenvalue of *C*.

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

Principal Component Analysis

PCA more generally (works in any dimension):

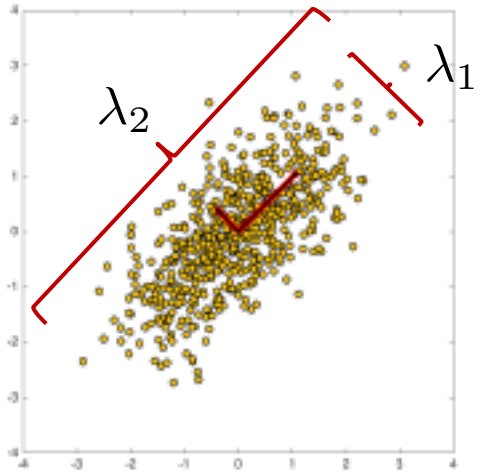
1. Given a point cloud with n points in $\mathbb{R}^d$
2. Covariance matrix $C \in \mathbb{R}^{d \times d}$, $C = \sum_{i=1}^{n} (p_i - \mu)(p_i - \mu)^T$
3. Eigenvectors of C encode *principal directions*
4. Eigenvalues (all non-negative!) encode the *variance.*



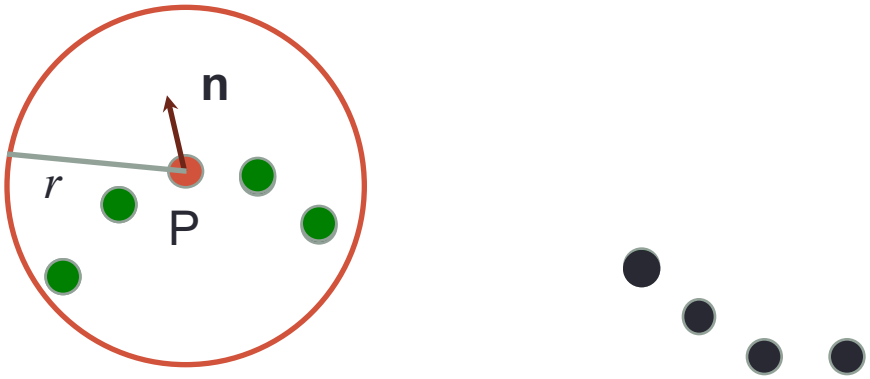
Principal Component Analysis

PCA more generally (works in any dimension):

1. Given a point cloud with n points in $\mathbb{R}^d$
2. Covariance matrix $C \in \mathbb{R}^{d \times d}$, $C = \sum_{i=1}^{n} (p_i - \mu)(p_i - \mu)^T$
3. Eigenvectors of C encode *principal directions*
4. Eigenvalues (all non-negative!) encode the *variance.*



Normal Estimation

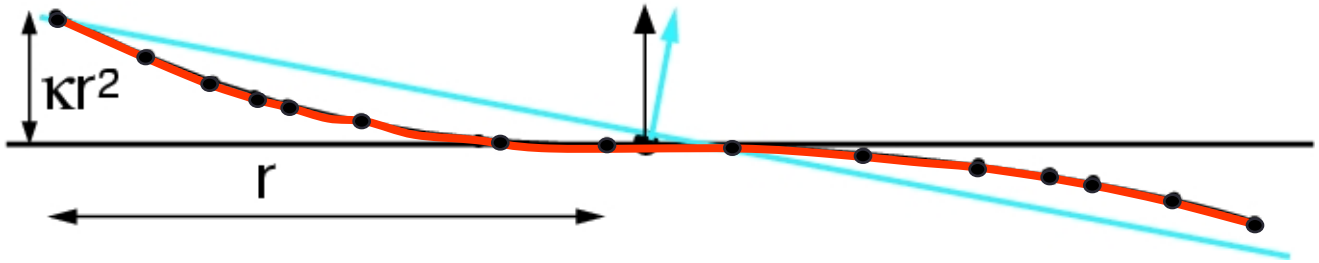


Critical parameter: *k*. Because of uneven sampling typically fix a radius *r*, and use all points **inside a ball of radius *r*.**

How to pick an optimal r?

Normal Estimation

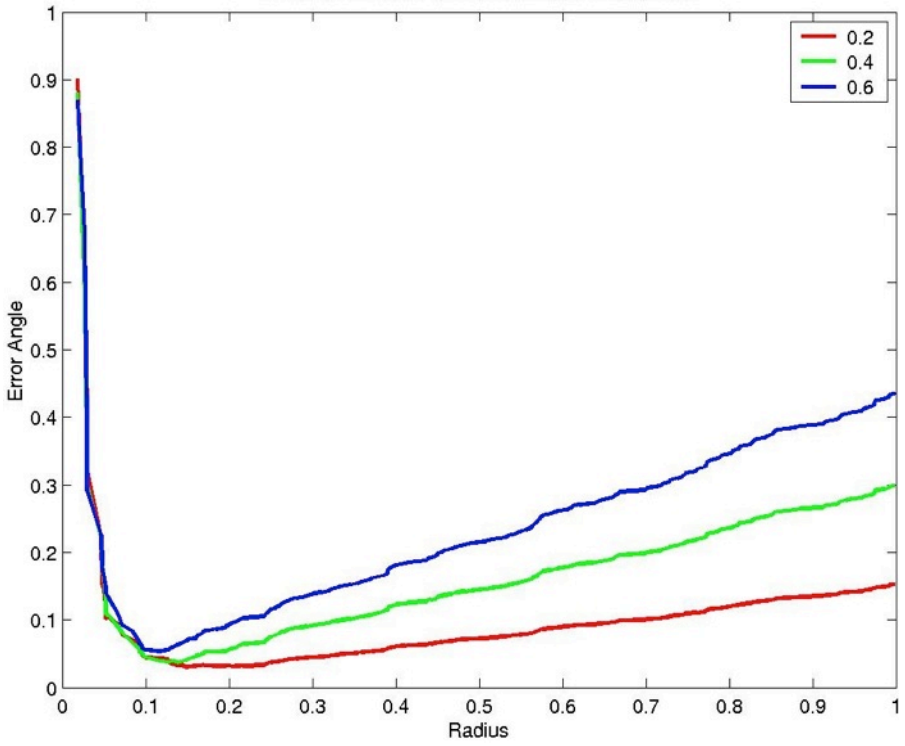
Curvature effect



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

Due to noise, small *r* can lead to errors

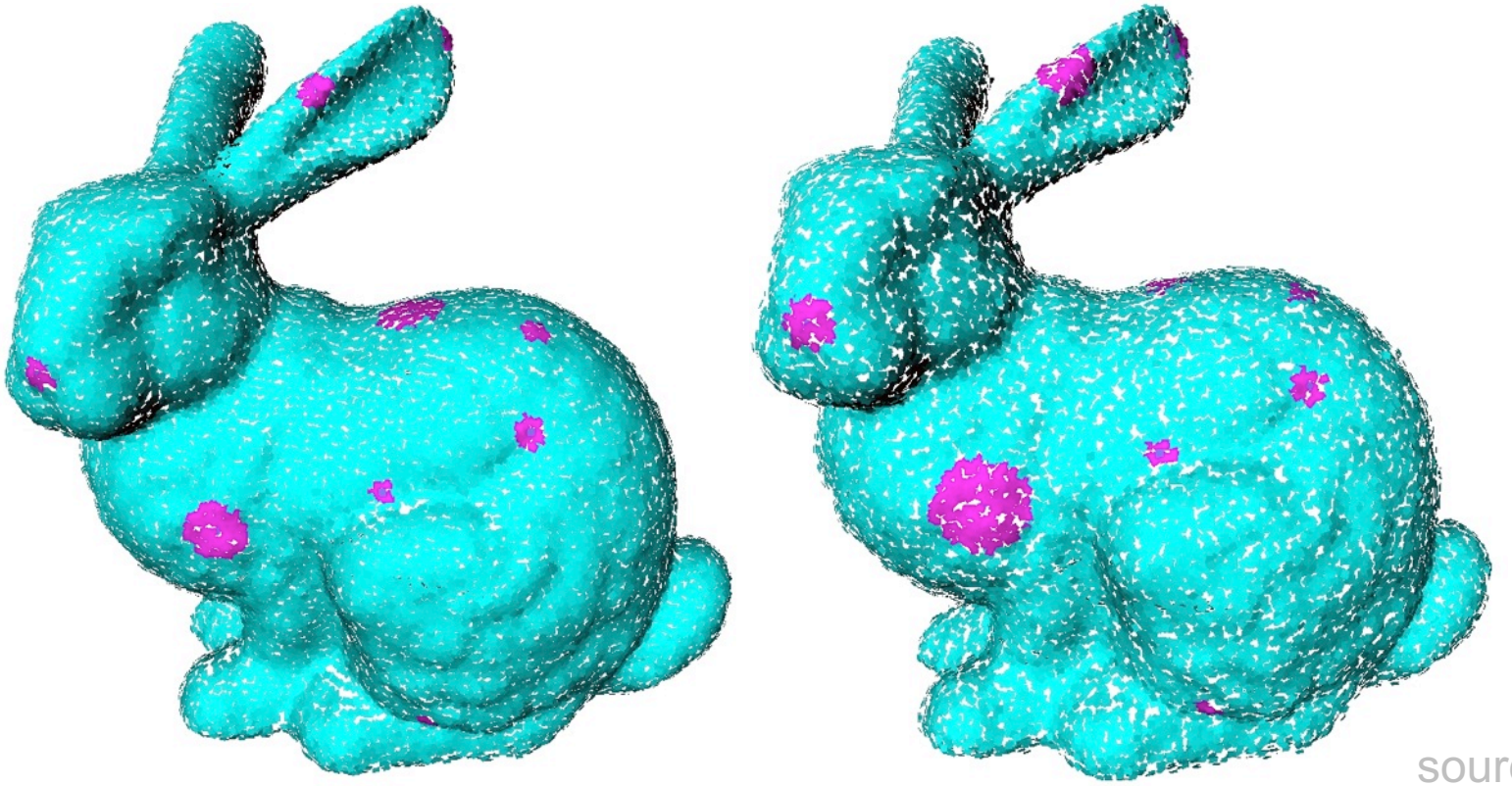
Normal Estimation



source: Mitra et al. ‘04

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

Normal Estimation – Neighborhood Size

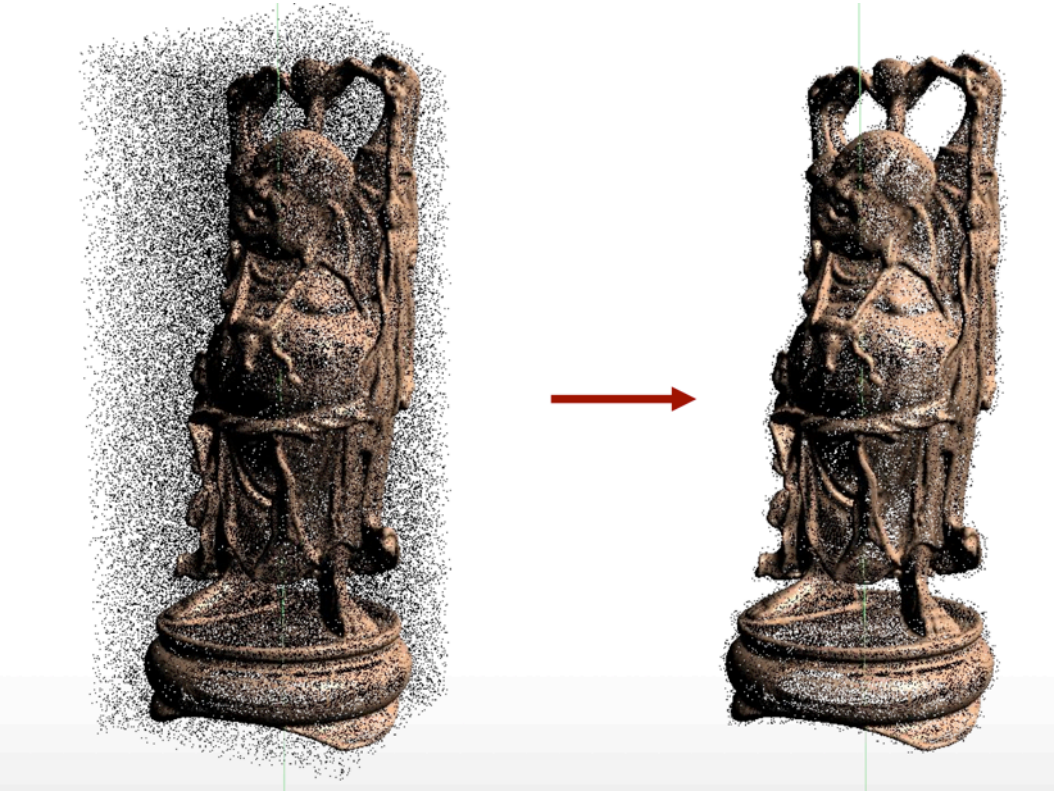


1x noise

2x noise

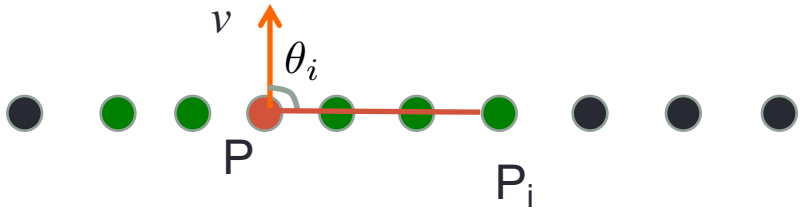
source: Mitra et al. ‘04

Outlier Removal



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

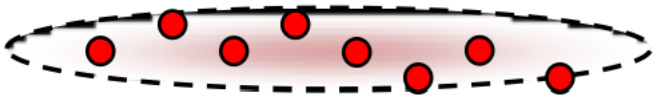
Outlier Estimation



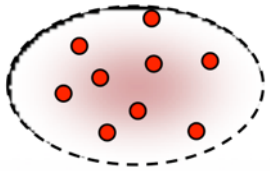
If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{\min}(C) = 0$ and $\lambda_{\max}(C)$ is large.

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

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

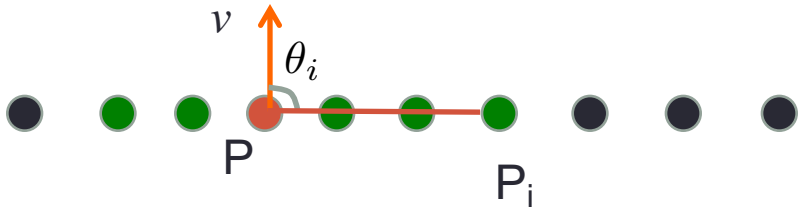


$\frac{\lambda_1}{\lambda_2}$ small



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

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.

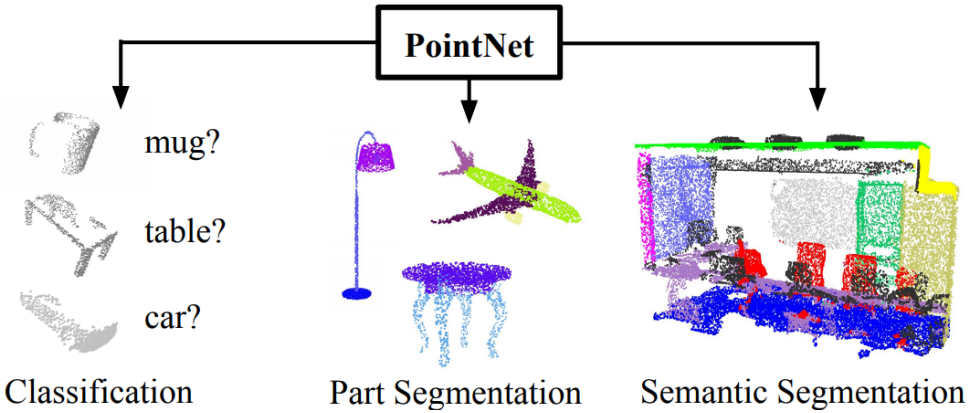
If points are scattered randomly, then: $\lambda_{\max}(C) \approx \lambda_{\min}(C)$.

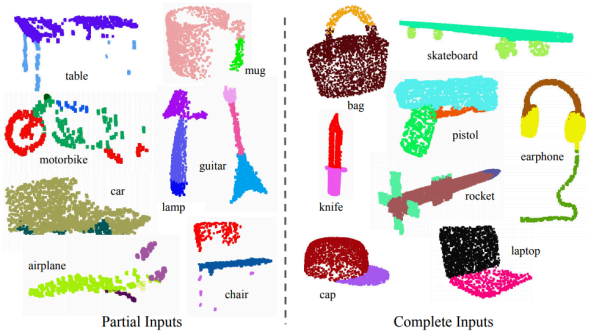
Thus, can remove points where $\frac{\lambda_1}{\lambda_2} > \epsilon$ for some threshold.

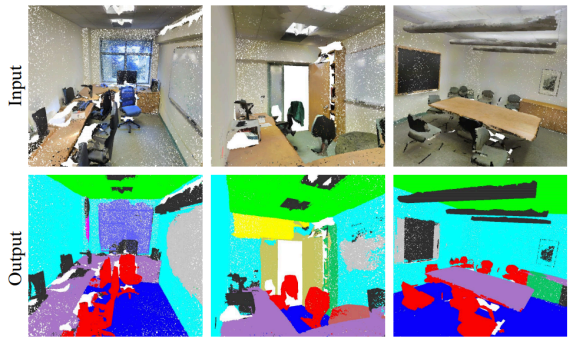
In 3d we expect two zero eigenvalues, so use $\frac{\lambda_2}{\lambda_3} > \epsilon$ for some threshold.

Data-Driven Approaches

Learning-based methods for analyzing point clouds





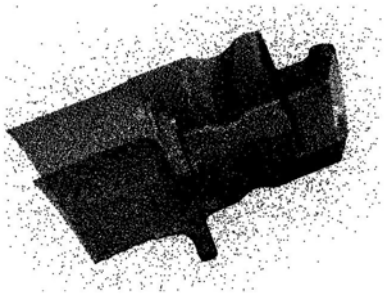


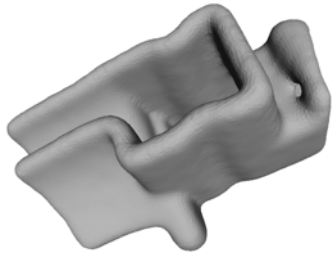
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas, CVPR 2017

3d Point Cloud Processing

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

1. Shape scanning (acquisition)
2. If have multiple scans, align them.
3. Smoothing – remove local noise and outliers.
4. Estimate surface normals.
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh

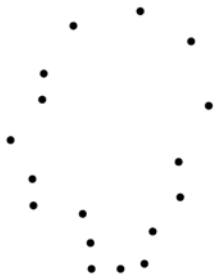


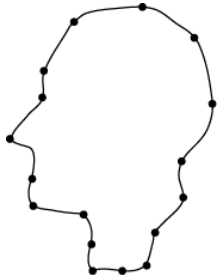


3d Point Cloud Reconstruction

Main Goal:

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



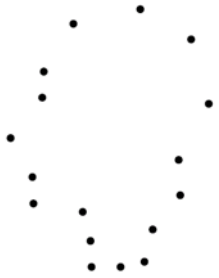


PCD → Reconstruction algorithm → curve/ surface

3d Point Cloud Reconstruction

Main Problem:

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

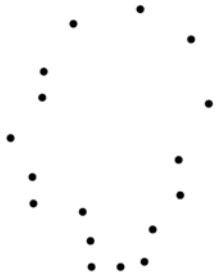


PCD → Reconstruction algorithm → curve/ surface

3d Point Cloud Reconstruction

Main Problem:

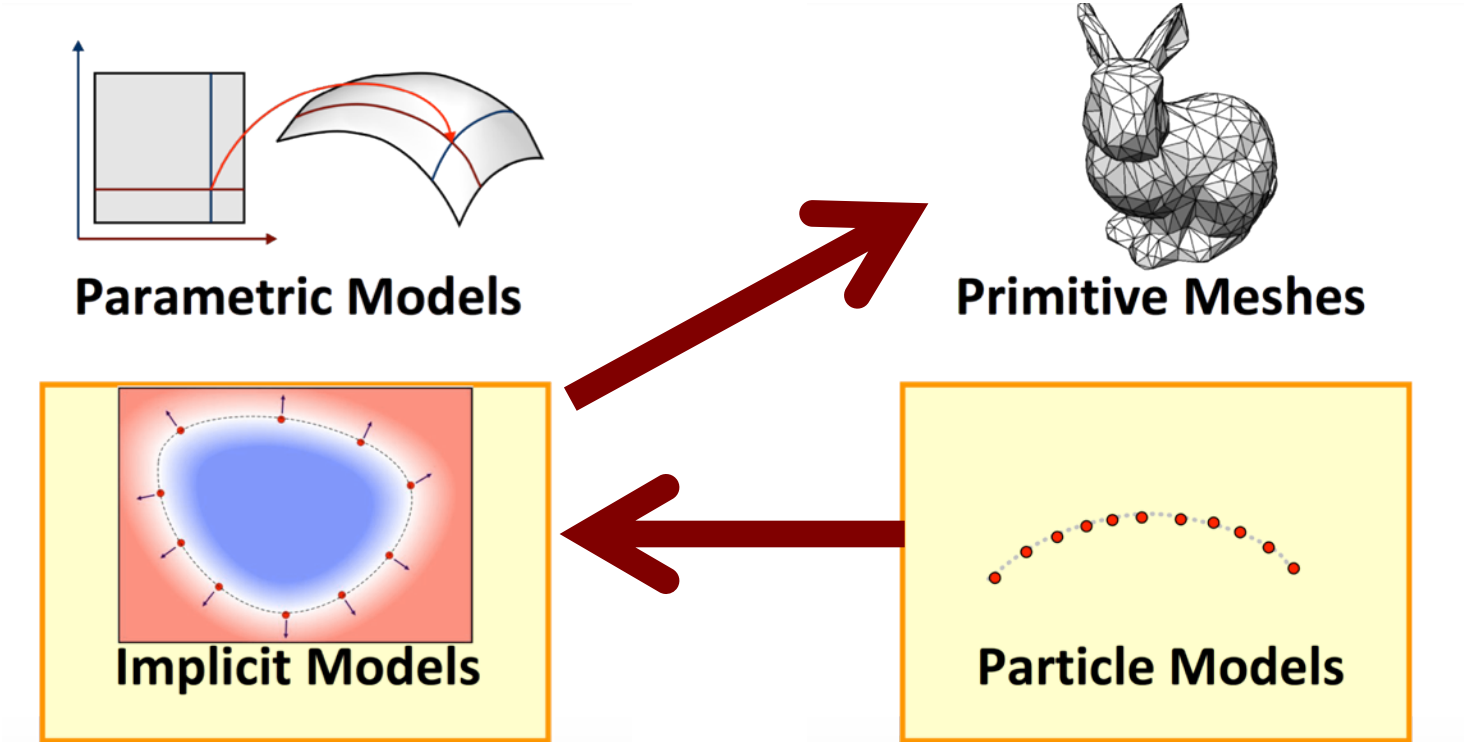
Data is **unstructured.** E.g. in 2D the points are not **ordered.**
Inherently **ill-posed** (aka difficult) problem.



PCD → Reconstruction algorithm → curve/ surface

3d Point Cloud Reconstruction

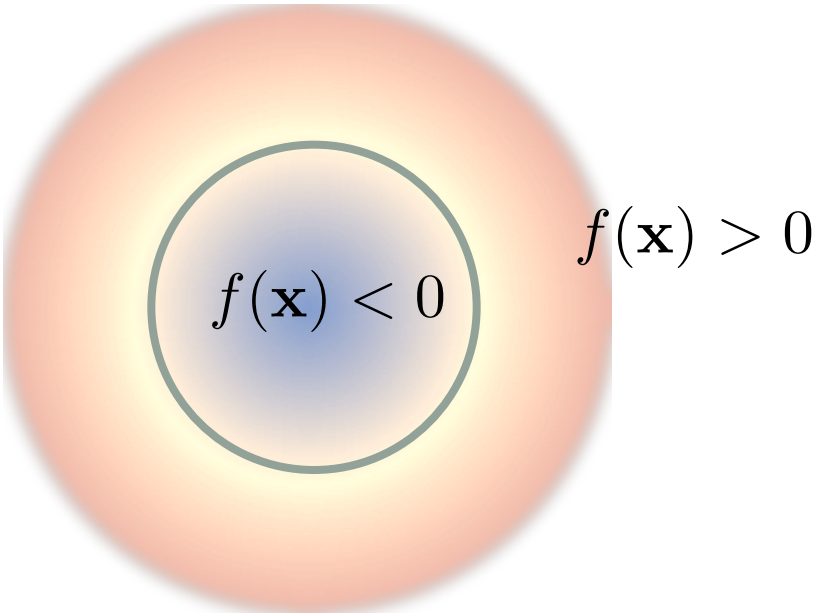
Today: Reconstruction through Implicit models.



Implicit surfaces

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

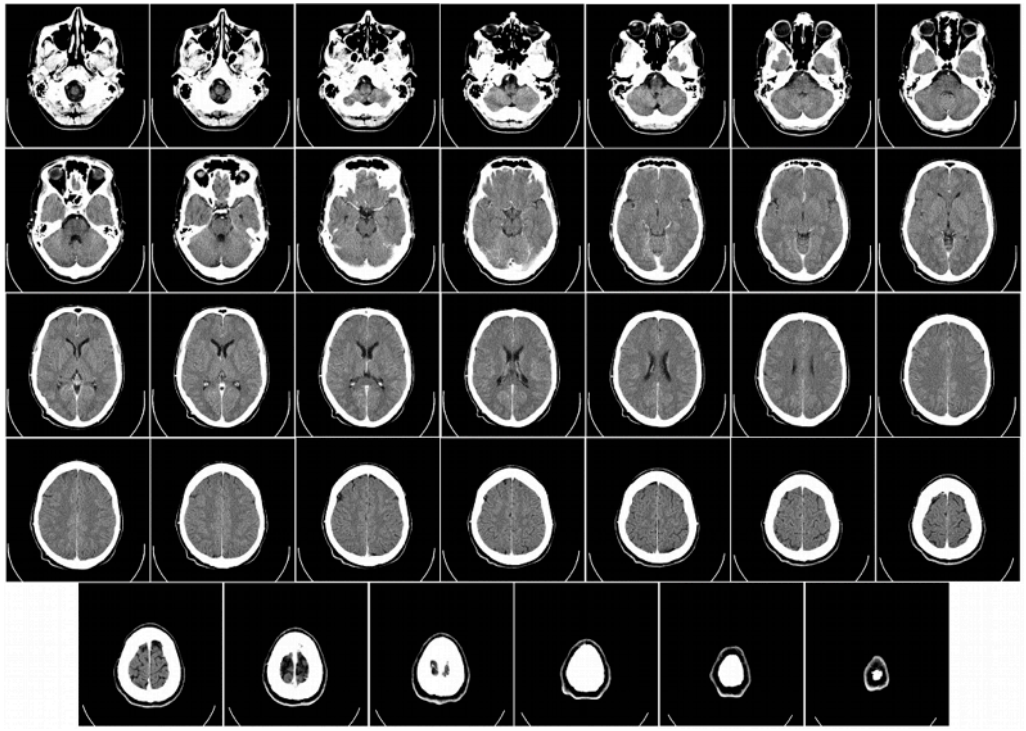
$$\{\mathbf{x}, \text{s.t.} f(\mathbf{x}) = 0\}$$



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

Implicit surfaces

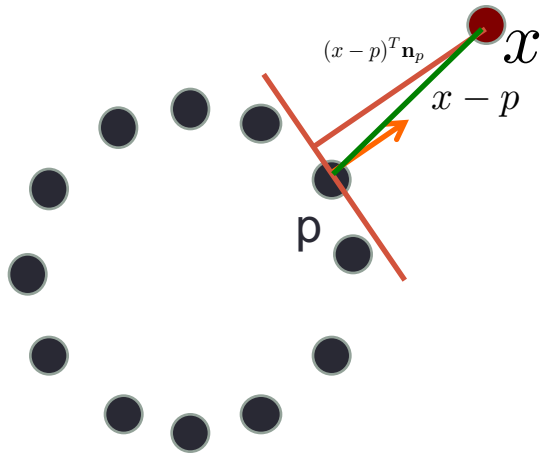
Some 3d scanning technologies (e.g. CT, MRI) naturally produce implicit representations



CT scans of human brain

Implicit surfaces

Converting from a point cloud to an implicit surface:

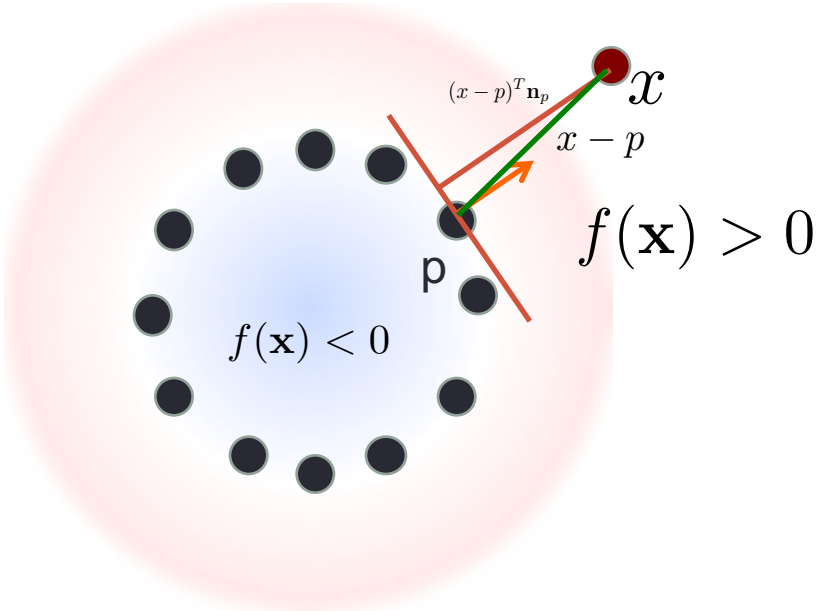


Simplest method:

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.

Implicit surfaces

Converting from a point cloud to an implicit surface:

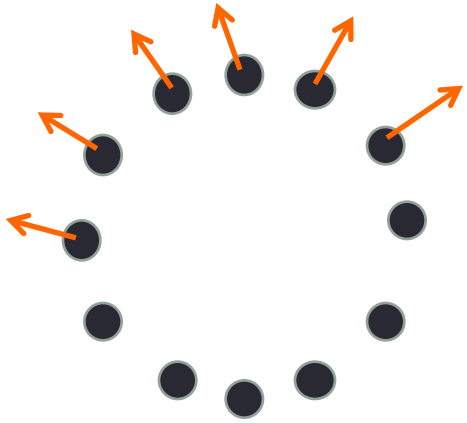


Simplest method:

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.

Implicit surfaces

Converting from a point cloud to an implicit surface:



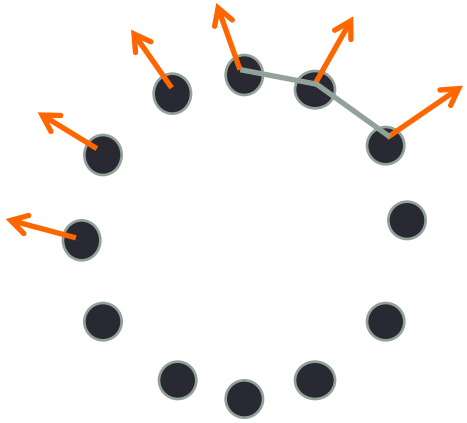
Simplest method:

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**

Implicit surfaces

Converting from a point cloud to an implicit surface:



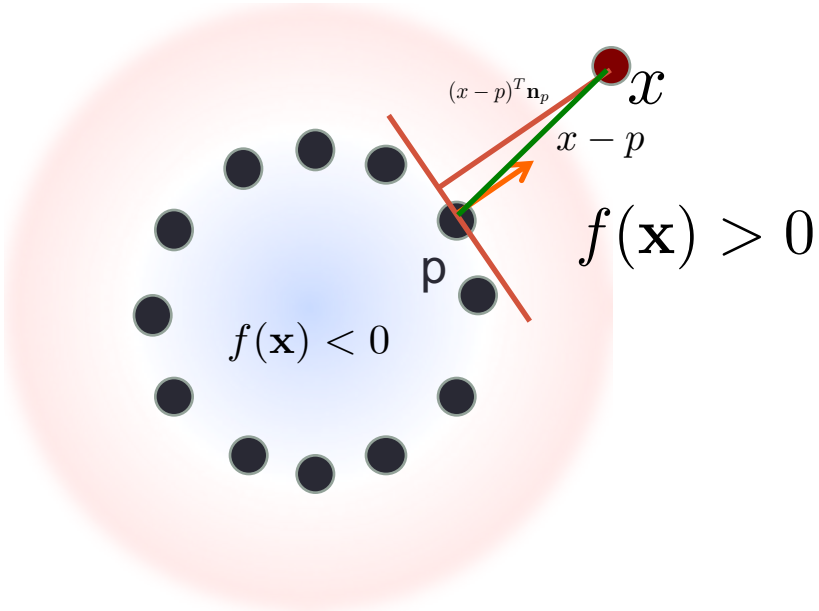
Simplest method:

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:

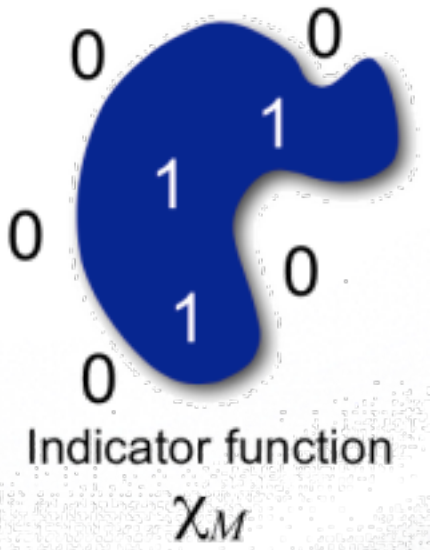
1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

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

Poisson Surface Reconstruction

Main Goal: construct an indicator function of a surface

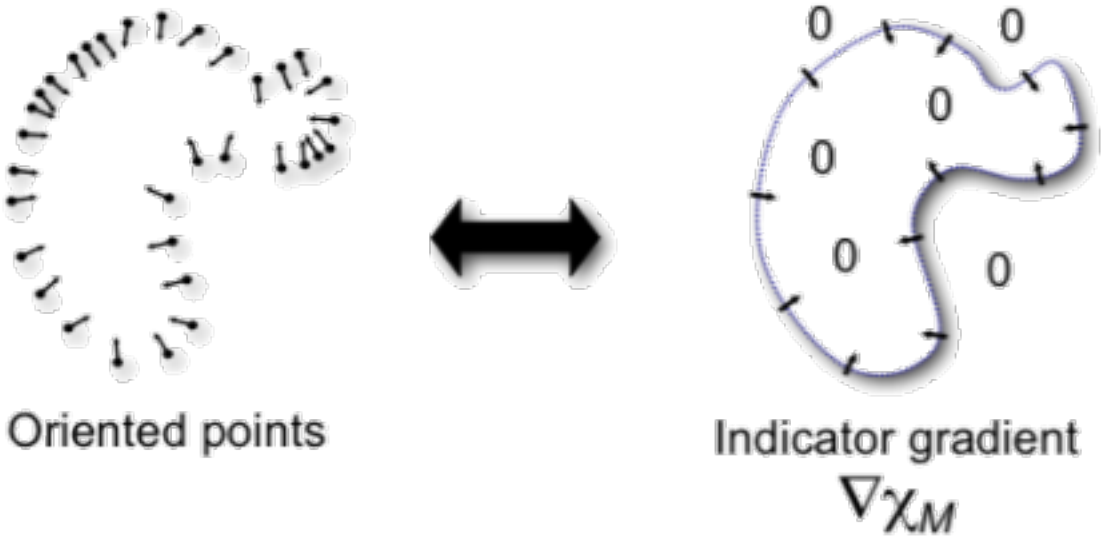
$$\chi_M(p) = \begin{cases} 1 & \text{if } p \in M \\ 0 & \text{if } p \notin M \end{cases}$$



M. Kazhdan, M. Bolitho, H. Hoppe *Poisson Surface Reconstruction,* Proc. SGP 2006

Poisson Surface Reconstruction

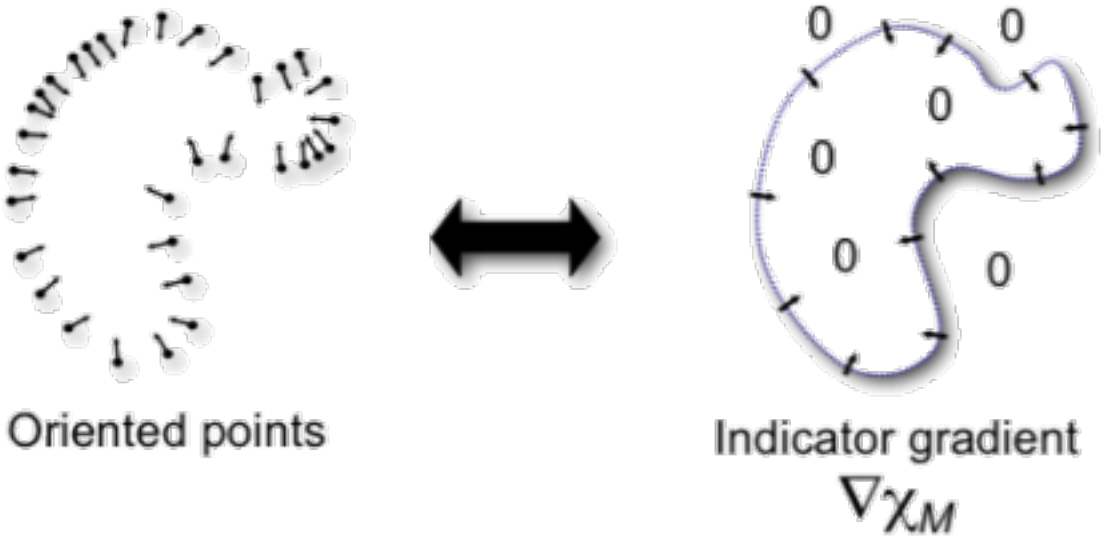
Observation:



The gradient of the indicator function should agree with the vector field of oriented points.

Poisson Surface Reconstruction

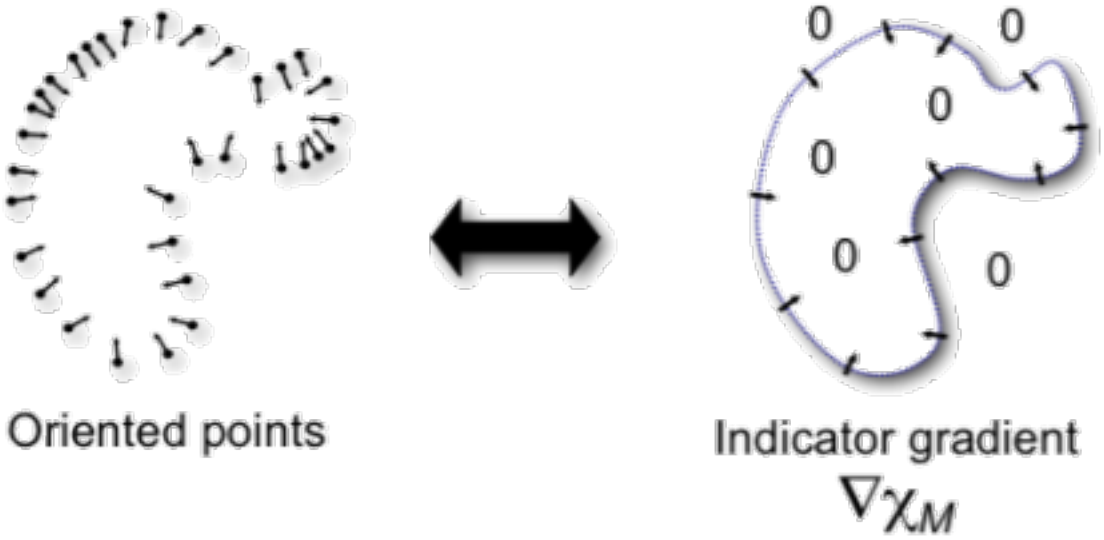
Solving for the indicator function:



$$\min_{\chi} \|\nabla\chi - V\|$$

Poisson Surface Reconstruction

Solving for the indicator function:

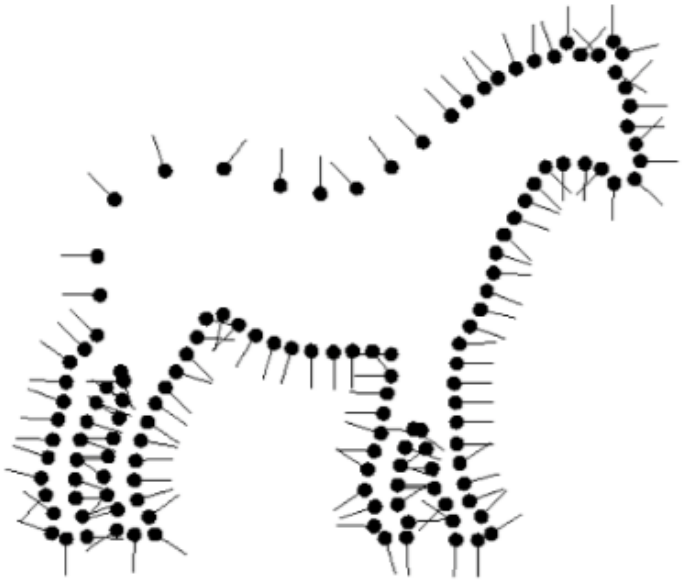


Or, applying the divergence operator:

$$\min_{\chi} \|\Delta\chi - \mathrm{div}(V)\|$$

Poisson Surface Reconstruction

Given the indicator function – extract the isosurface.



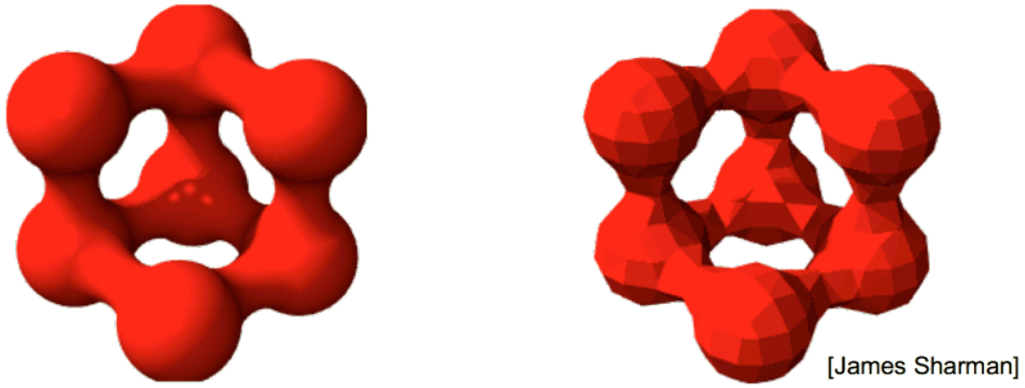
M. Kazhdan, M. Bolitho, H. Hoppe *Poisson Surface Reconstruction,* Proc. SGP 2006

Marching Cubes

Converting from implicit to explicit representations.

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

Create a triangle mesh that approximates the surface.



Lorensen and Cline, SIGGRAPH ‘87

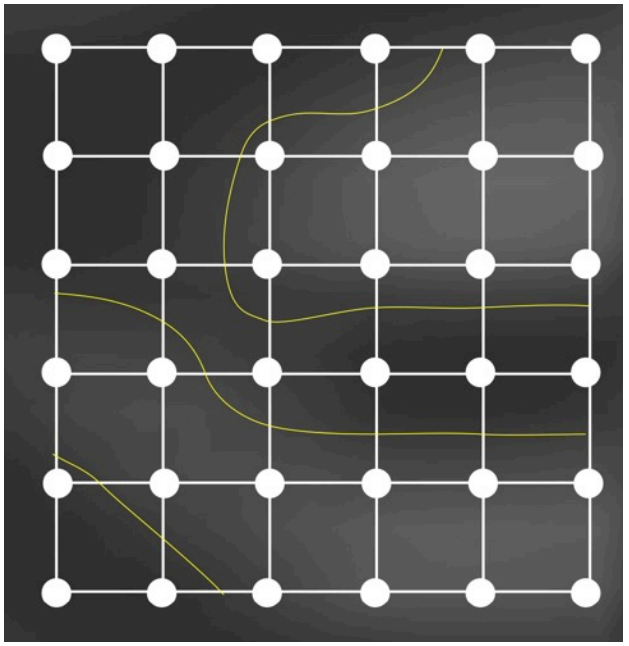
One of the most cited computer graphics papers of all time.

Marching Squares (2D)

Given a function: $f(x)$

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

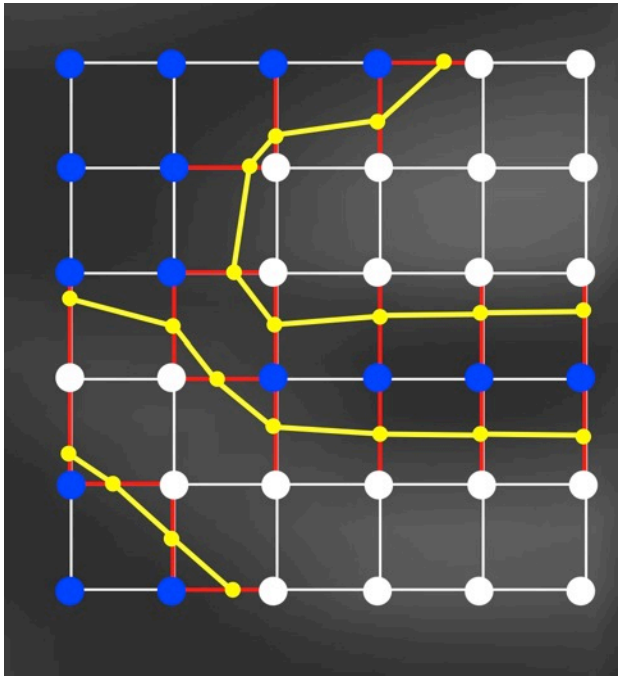
1. Discretize space.
2. Evaluate $f(x)$ on a grid.



Marching Squares (2D)

Given a function: $f(x)$

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

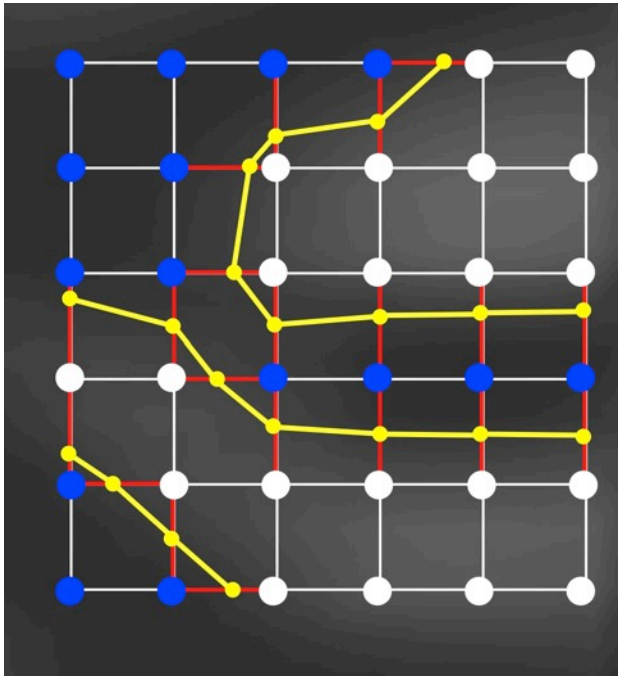


1. Discretize space.
2. Evaluate $f(x)$ on a grid.
3. Classify grid points (+-)
4. Classify grid edges
5. Compute Intersections
6. Connect Intersections

Marching Squares (2D)

Connecting the intersections:

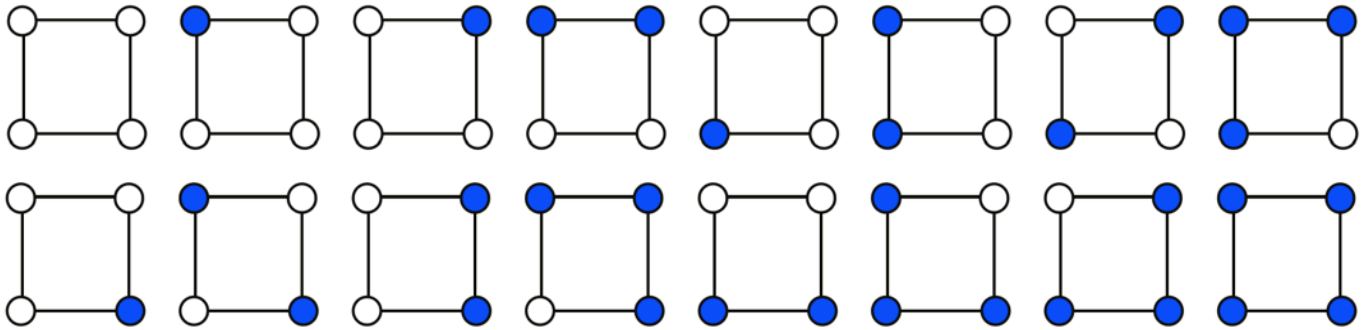
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



Marching Squares (2D)

Connecting the intersections:

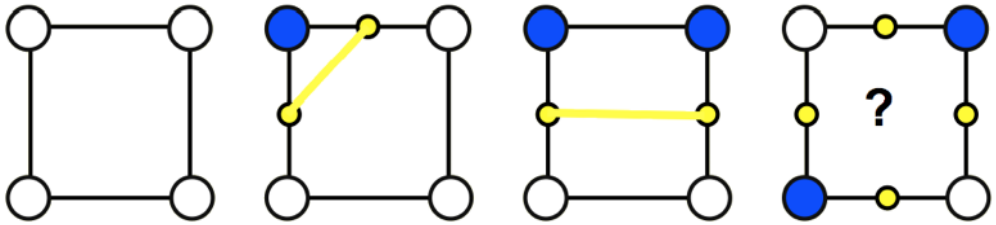
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections



Marching Squares (2D)

Connecting the intersections:

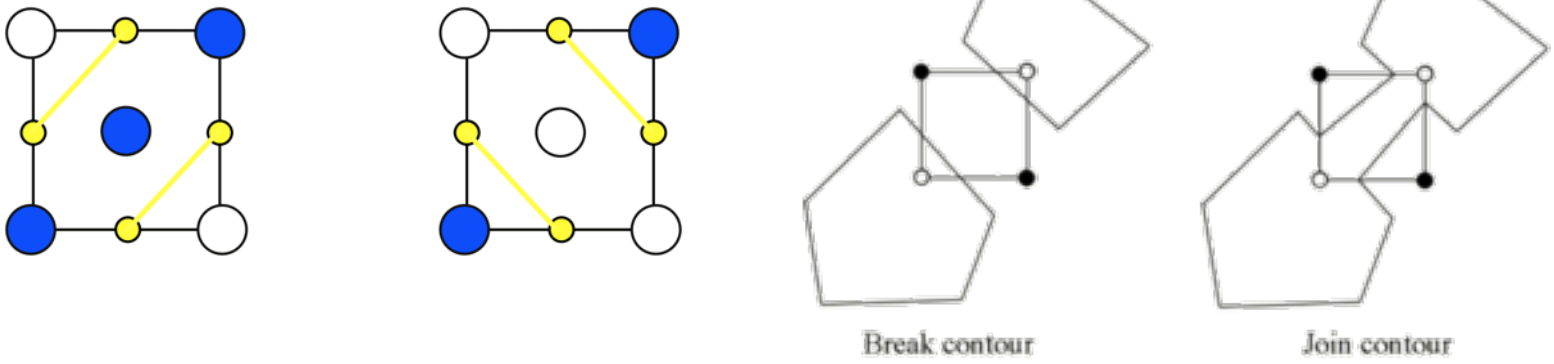
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections



Marching Squares (2D)

Connecting the intersections:

Ambiguous cases:



2 options:

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

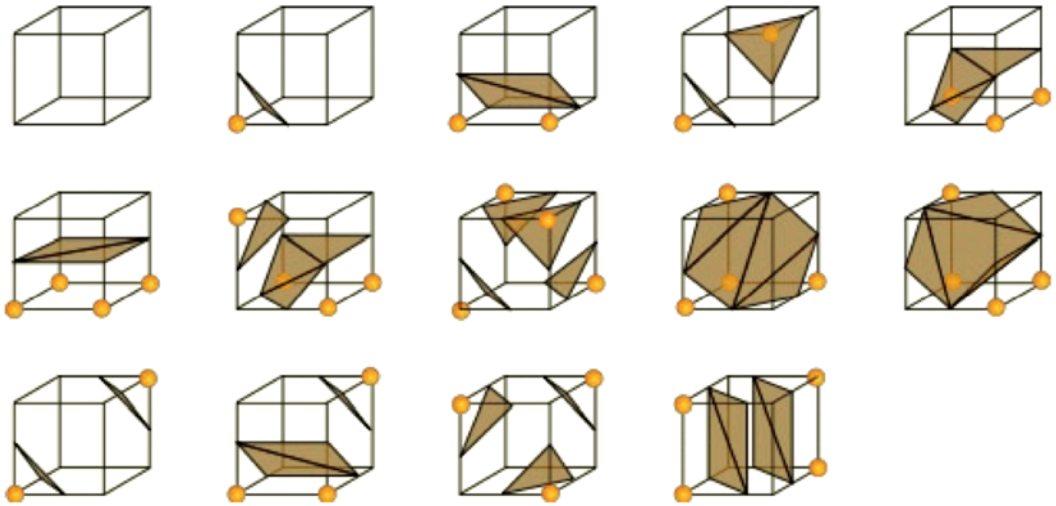
Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 after symmetries



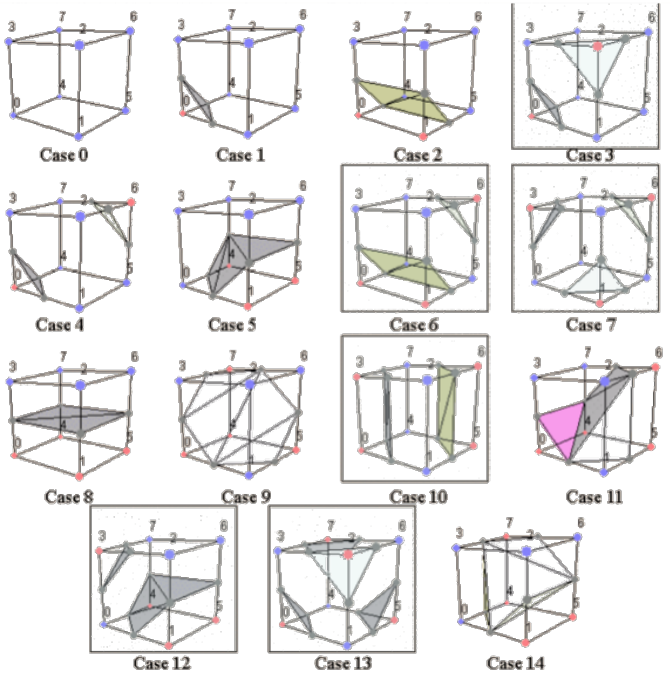
Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

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



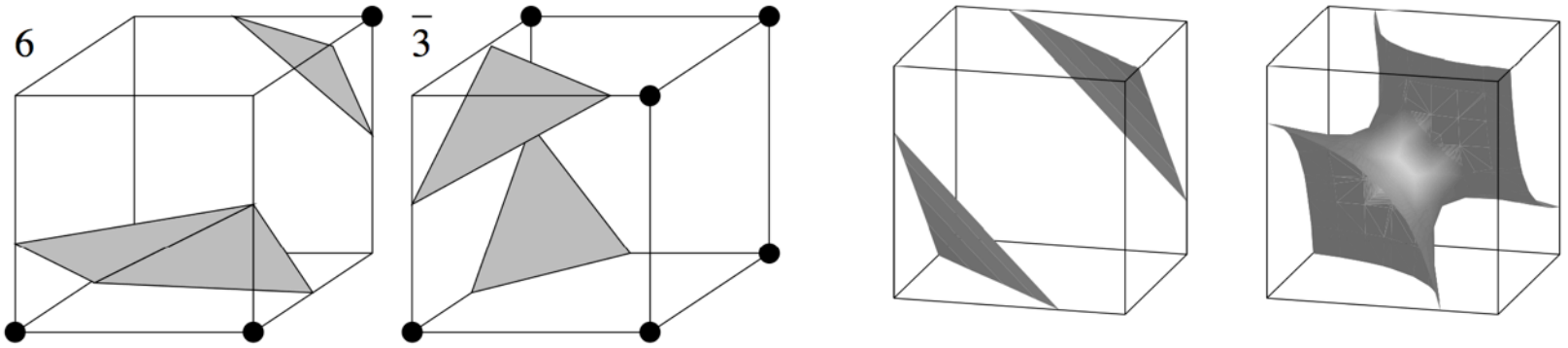
Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

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



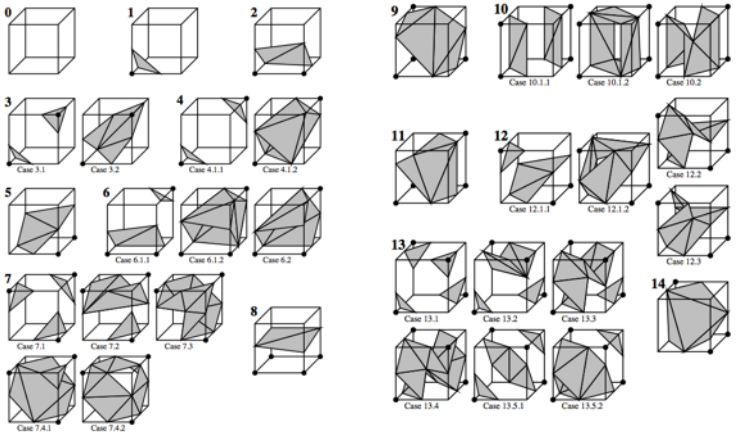
Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 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.
- 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.

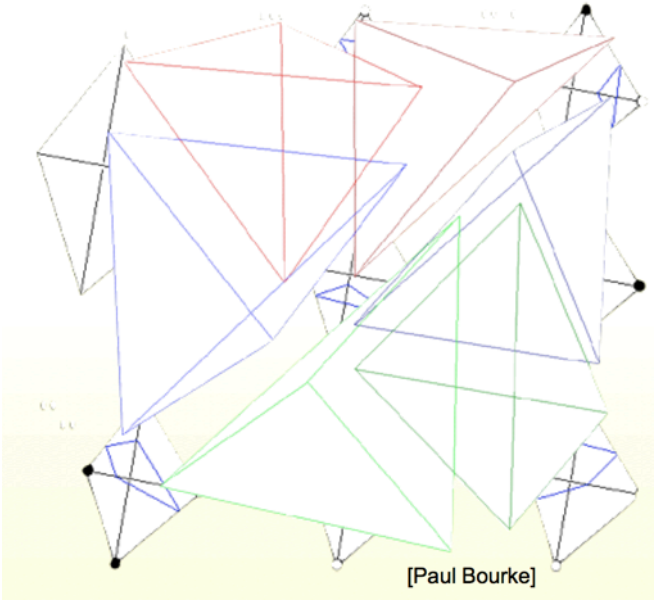
Marching Cubes – Extensions

Marching Tetrahedra.

Instead of cubes (grid) use **Tetrahedra.**

- 6 Tetrahedra per voxel
- 16 cases (8 after symmetry)
- Up to 2 triangles per tet
- **No ambiguities.**

Can be used when input is discretized as tetrahedra.



Regularised marching tetrahedra: improved iso-surface extraction, G.M. Treece, R.W. Prager, A.H. Gee, Computers & Graphics, 1999.

Conclusions

Wide variety of 3d scanning techniques.

Reconstruction is a difficult, highly data-dependent problem.

Majority of reconstruction methods require normal information.

Marching cubes: classical method for converting an implicit to explicit representation.

Many extensions to:

- Improve topology
- Handle sharp features
- Improve quality