3D Surface Reconstruction

2017 Fall

National Cheng Kung University

Instructor: Min-Chun Hu 胡敏君

Parts of the slides are from

"Efficient Simplification of Point-Sampled Surfaces"

by Mark Pauly, Markus Gross, Leif Kobbelt

Parts of the slides are from the course

"INF555 Digital Representation and Analysis of Shapes"

by Luca Castelli Aleardi and Maks Ovsjanikov

http://www.enseignement.polytechnique.fr/informatique/INF555/

Parts of the slides are from the 9th ACCV,

Xi'an China, Sep 25-27 2009

by Nader Salman, INRIA

http://www.cgal.org

Data Representation

Curves and Surfaces

Limitation of Using Polygon Representation

- Inherently an approximation
  - Planar facets and silhouettes
  - Otherwise, it needs a very large numbers of polygons
- Fixed resolution
- No natural parameterization
  - Deformation is relatively difficult
  - Hard to extract information like curvature or to keep smoothness

Figures from MIT EECS 6.837, Durand and Cutler

Subdivision

- Subdividing a polygon can alleviate the problem of polygonal mesh representation.
- E.g. Loop's subdivision
  - Split a triangle into four smaller ones.
  - Choose locations of new vertices by weighted average of the original neighbor vertices.

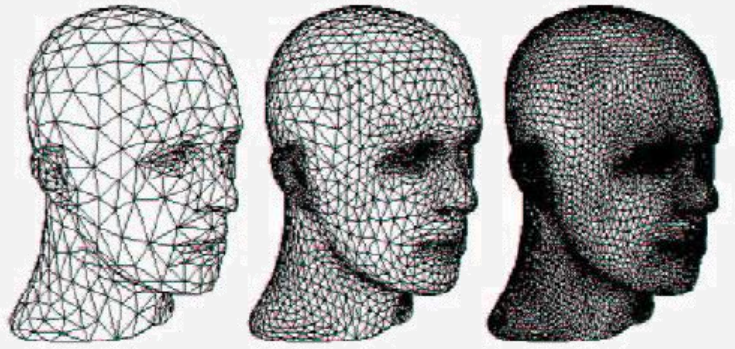
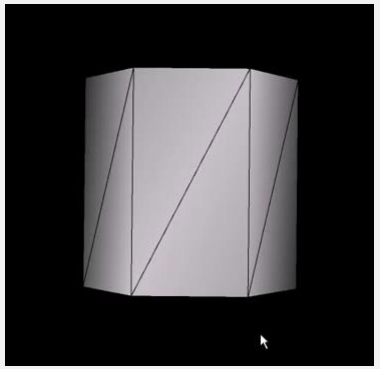
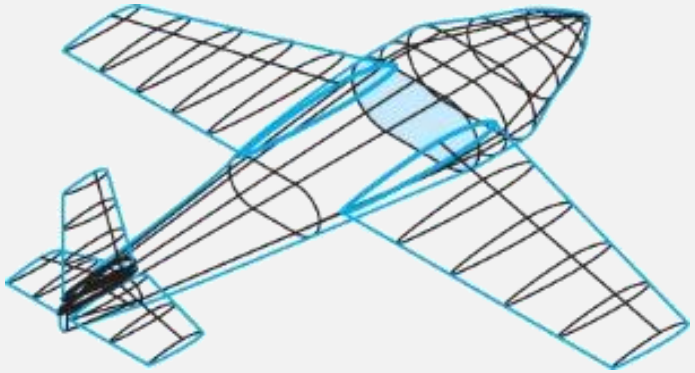


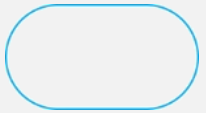
Figure from Zorin & Schroeder
SIGGRAPH 99 Course Notes



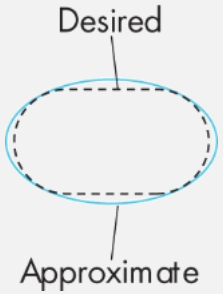
Design Criteria

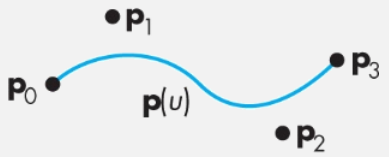
- How to build an airplane using flexible strips of wood?



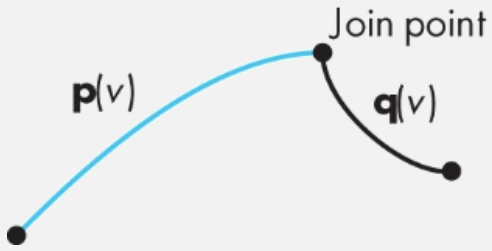


Desired cross-section curve





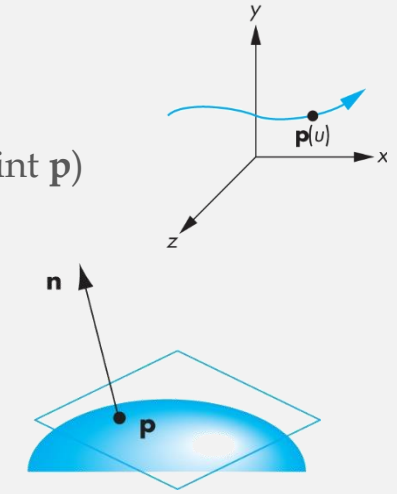
The approximate curve can be determined by the control/data points.



Avoid derivative discontinuity at the join point

Representation of Curves and Surface

- Explicit Representation:
  - $y = f(x)$ or $x = g(y)$
  - No guarantee that either form exists for a given curve (e.g., vertical line and circle)
- Implicit Representation:
  - $f(x, y) = 0$
    - Line: $ax + by + cz + d = 0$
    - Circle: $x^2 + y^2 + z^2 - r^2 = 0$
  - Does represent all lines and circles, but difficult to obtain all points on the curve/surface
- Parametric Representation:
  - Curve: $\mathbf{p}(u) = [x(u) \quad y(u) \quad z(u)]^T$
    - $\frac{d\mathbf{p}(u)}{du} = \left[\frac{dx(u)}{du} \quad \frac{dy(u)}{du} \quad \frac{dz(u)}{du}\right]^T$: the velocity with which the curve is traced out (tangent direction at point $\mathbf{p}$)
  - Surface: $\mathbf{p}(u, v) = [x(u, v) \quad y(u, v) \quad z(u, v)]^T$
    - $\frac{\partial \mathbf{p}}{\partial u} = \left[\frac{\partial x(u,v)}{\partial u} \quad \frac{\partial y(u,v)}{\partial u} \quad \frac{\partial z(u,v)}{\partial u}\right]^T$ and $\frac{\partial \mathbf{p}}{\partial v} = \left[\frac{\partial x(u,v)}{\partial v} \quad \frac{\partial y(u,v)}{\partial v} \quad \frac{\partial z(u,v)}{\partial v}\right]^T$: tangent plane at point $\mathbf{p}$
    - $\mathbf{n} = \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial u}$: normal direction
  - Most flexible and robust form for computer graphics, but not unique

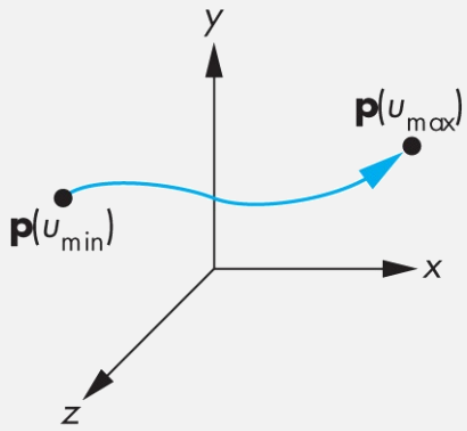


Why Parametric Curves?

- Intended to provide the generality of polygon meshes but with fewer parameters for smooth surfaces
- Faster to create a curve, and easier to edit an existing curve
- Easier to animate than polygon meshes
- Normal vectors and texture coordinates can be easily defined everywhere

Parametric Cubic Polynomial Curves

- We can use polynomial functions to form curves:
  - $\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \cdots \boldsymbol{c_n}u^n$
  - degree of freedom: n+1
  - cubic polynomial curve: $\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \boldsymbol{c_3}u^3 = \mathbf{u}^T\mathbf{c}$
    - Low freedom, but sufficient to produce the desired shape in a small region
  - $0 \le u \le 1$



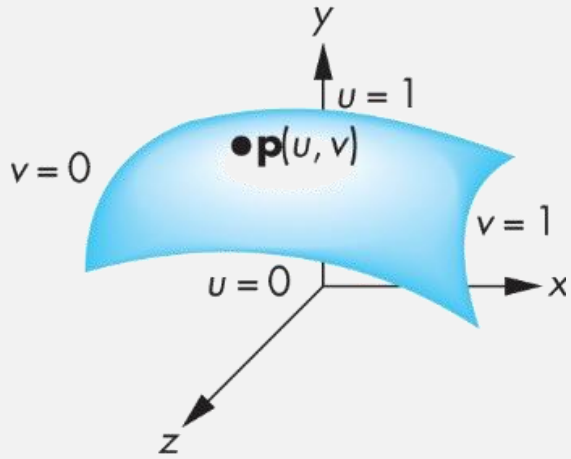
- The problem is how to efficiently find out the coefficient $\boldsymbol{c_i}$
- Least square curve fitting:

$$x(u) = c_{x0} + c_{x1}u + c_{x2}u^2 + c_{x3}u^3$$
$$y(u) = c_{y0} + c_{y1}u + c_{y2}u^2 + c_{y3}u^3$$
$$z(u) = c_{z0} + c_{z1}u + c_{z2}u^2 + c_{z3}u^3$$

➡ Need 4 points to solve 12 unknowns

Parametric Polynomial Surfaces

- We can use polynomial functions to form curves:
  - $$\mathbf{p}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} = \sum_{i=0}^{n} \sum_{j=0}^{m} u^i v^j \boldsymbol{c_{ij}} = \mathbf{u}^T \mathbf{C} \mathbf{v}$$



  - Must specify $3(n+1)(m+1)$ coefficients to determine a particular surface $\mathbf{p}(u,v)$
  - Always take $n = m$ and let u and v vary over the rectangle $0 \leq u, v \leq 1$
  - Any surface patch can be viewed as a collection of curves generated by holding either u or v constant and varying the other

Least Square Curve Fitting

$$\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \boldsymbol{c_3}u^3$$

- Given 4 control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$, and $\mathbf{p}_3$
  - Assume the 4 points are with equally spaced values u:

$$\left.\begin{aligned}
\mathbf{p}_0 &= \mathbf{p}(0) = \mathbf{c}_0 \\
\mathbf{p}_1 &= \mathbf{p}\left(\frac{\mathbf{1}}{\mathbf{3}}\right) = \mathbf{c}_0 + \frac{1}{3}\mathbf{c}_1 + \left(\frac{\mathbf{1}}{\mathbf{3}}\right)^2 \mathbf{c}_2 + \left(\frac{\mathbf{1}}{\mathbf{3}}\right)^3 \mathbf{c}_3 \\
\mathbf{p}_2 &= \mathbf{p}\left(\frac{\mathbf{2}}{\mathbf{3}}\right) = \mathbf{c}_0 + \frac{2}{3}\mathbf{c}_1 + \left(\frac{\mathbf{2}}{\mathbf{3}}\right)^2 \mathbf{c}_2 + \left(\frac{\mathbf{2}}{\mathbf{3}}\right)^3 \mathbf{c}_3 \\
\mathbf{p}_3 &= \mathbf{p}(\mathbf{1}) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3
\end{aligned}\right\} \Rightarrow \mathbf{P} = \mathbf{Ac}$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{3} & \left(\frac{\mathbf{1}}{\mathbf{3}}\right)^2 & \left(\frac{\mathbf{1}}{\mathbf{3}}\right)^3 \\ 1 & \frac{2}{3} & \left(\frac{\mathbf{2}}{\mathbf{3}}\right)^2 & \left(\frac{\mathbf{2}}{\mathbf{3}}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix}$$

nonsingular

$$\mathbf{M_I} = \mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix} \Rightarrow \mathbf{c} = \mathbf{M_I P}$$

Interpolating Geometry Matrix

$$\boldsymbol{c}_k = \begin{bmatrix} \mathbf{c}_{kx} \\ \mathbf{c}_{ky} \\ \mathbf{c}_{kz} \end{bmatrix}$$

Cubic Interpolating Curves

- Rather than deriving a single interpolating curve of degree m for all the points, derive a set of cubic interpolating curves.
- If each segment is derived by letting u varying equally over the interval [0,1], then the matrix $\mathbf{M_I}$ is the same for each segment.
- Derivatives at the joint points will not be continuous.

Blending Functions

$$\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \boldsymbol{c_3}u^3 = \mathbf{u}^T\mathbf{c} = \mathbf{u}^T\mathbf{M_I}\mathbf{P} = \mathbf{b}(u)^T\mathbf{P}$$

$$\mathbf{b}(u) = \begin{bmatrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{bmatrix}$$

Blending Polynomials

$$\mathbf{p}(u) = \mathbf{b}(u)^T\mathbf{P} = b_0(u)\mathbf{p}_0 + b_1(u)\mathbf{p}_1 + b_2(u)\mathbf{p}_2 + b_3(u)\mathbf{p}_3 = \sum_{i=0}^{3} b_i(u)\mathbf{p}_i$$

- The polynomials blend together the individual contributions of each control point and enable us to see the effect of a given control point on the entire curve.

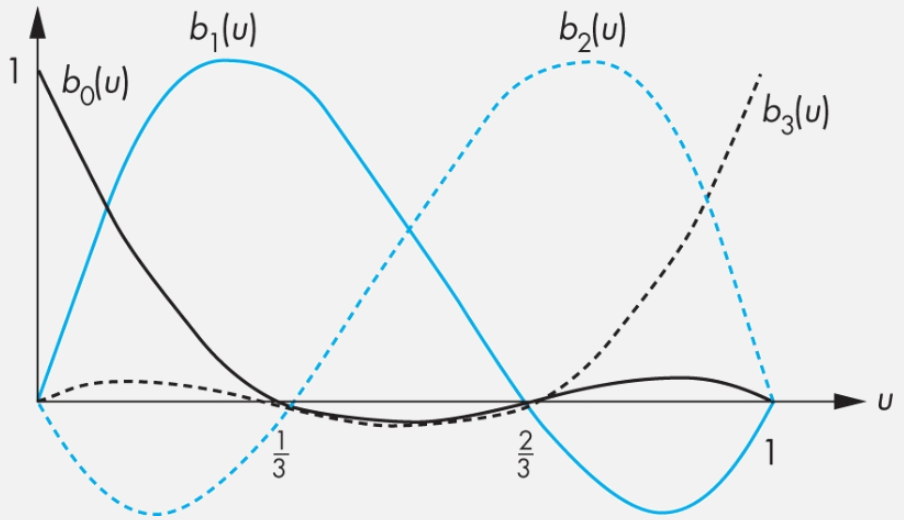
Blending Functions (Cont.)

$$b_0(u) = -\frac{9}{2}\left(u - \frac{1}{3}\right)\left(u - \frac{2}{3}\right)(u - 1)$$

$$b_1(u) = \frac{27}{2}u\left(u - \frac{2}{3}\right)(u - 1)$$

$$b_2(u) = -\frac{27}{2}u\left(u - \frac{1}{3}\right)(u - 1)$$

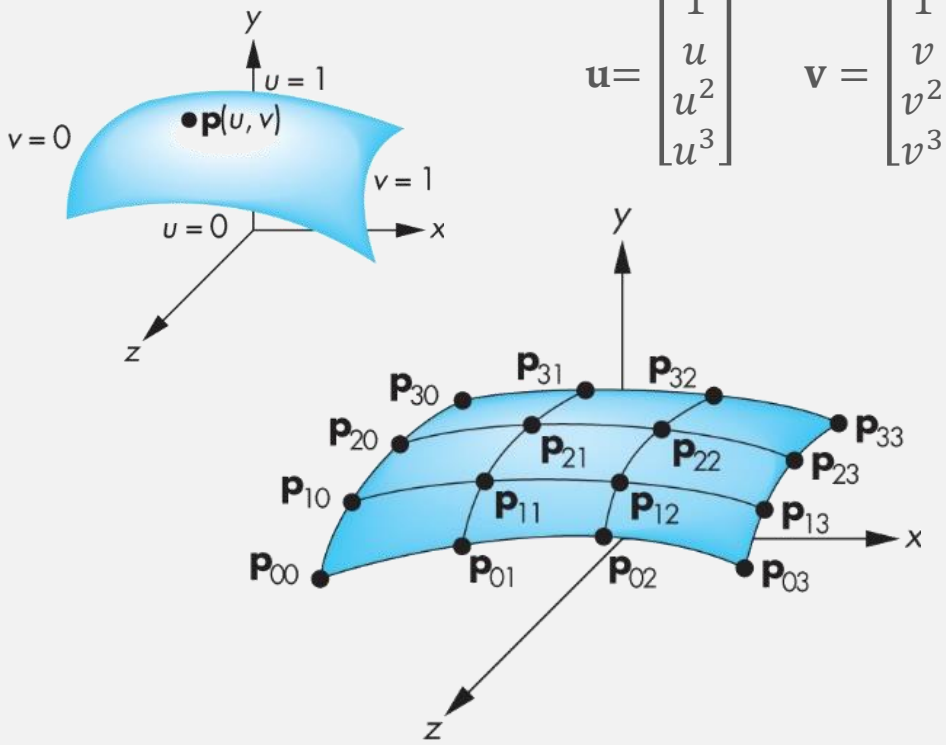
$$b_3(u) = \frac{9}{2}u\left(u - \frac{1}{3}\right)\left(u - \frac{2}{3}\right)$$



Bicubic Surface Patch

$$\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} u^i v^j c_{ij} = \mathbf{u}^T\mathbf{C}\mathbf{v}$$ ➡ Need 16 control points to solve the 4x4 unknows in the $\mathbf{C}$ matrix

$$\mathbf{u} = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}$$



- Rather than writing down and solving 16 equations, we consider the curve at $v = 0$ that interpolates $\mathbf{p}_{00}, \mathbf{p}_{10}, \mathbf{p}_{20}$, and $\mathbf{p}_{30}$:

$$\mathbf{p}(u,0) = \mathbf{u}^T\mathbf{M}_\mathbf{I}\begin{bmatrix} \mathbf{p}_{00} \\ \mathbf{p}_{10} \\ \mathbf{p}_{20} \\ \mathbf{p}_{30} \end{bmatrix} = \mathbf{u}^T\mathbf{C}\mathbf{v} = \mathbf{u}^T\mathbf{C}\begin{bmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

- Each value of $v = \frac{1}{3}, \frac{2}{3}, 1$ defines a interpolating curve with the similar form

Bicubic Surface Patch (Cont.)

$$\mathbf{p}(u,0) = \mathbf{u}^T\mathbf{M_I}\begin{bmatrix}\mathbf{p}_{00}\\ \mathbf{p}_{10}\\ \mathbf{p}_{20}\\ \mathbf{p}_{30}\end{bmatrix} = \mathbf{u}^T\mathbf{C}\begin{bmatrix}\mathbf{1}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}$$

$\Rightarrow$ $\mathbf{u}^T\mathbf{M_I}\mathbf{P} = \mathbf{u}^T\mathbf{C}\mathbf{A}^T$ $\quad$ $\mathbf{A}$: inverse of $\mathbf{M_I}$

Consider 4 curves at
4 different v values
(16 equations)

$\Rightarrow$ $\mathbf{C} = \mathbf{M_I}\mathbf{P}(\mathbf{A}^T)^{-1} = \mathbf{M_I}\mathbf{P}\mathbf{M_I}^T$

$\Rightarrow$ $\mathbf{p}(u,v) = \mathbf{u}^T\mathbf{C}\mathbf{v} = \mathbf{u}^T\mathbf{M_I}\mathbf{P}\mathbf{M_I}^T\mathbf{v}$

- Represented in blending functions: $\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)b_j(v)\,\mathbf{p}_{ij}$

Hermite Curves

- A Hermite curve is a curve for which the user provides:
  - The endpoints of the curve:

    $\mathbf{p}_0 = \mathbf{p}(0) = \mathbf{c}_0$

    $\mathbf{p}_3 = \mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3$

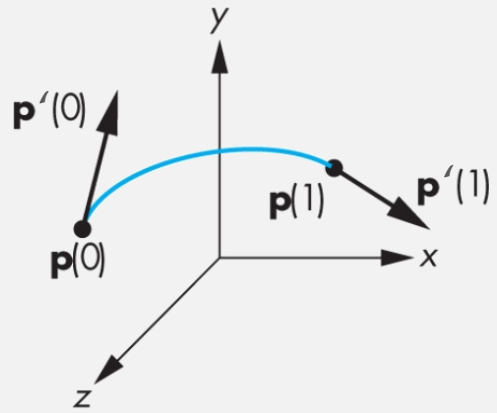
  - The derivatives of the curve at the endpoints:

    $\mathbf{p'}_0 = \mathbf{p}'(0) = \mathbf{c}_1$

    $\mathbf{p'}_3 = \mathbf{p}'(1) = \mathbf{c}_1 + 2\mathbf{c}_2 + 3\mathbf{c}_3$

$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$$

$$\mathbf{p}'(u) = \mathbf{c}_1 + 2u\mathbf{c}_2 + 3u^2\mathbf{c}_3$$



$$\Rightarrow \quad \mathbf{Q} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_3 \\ \mathbf{p'}_0 \\ \mathbf{p'}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c}$$

Hemite Geometry Matrix

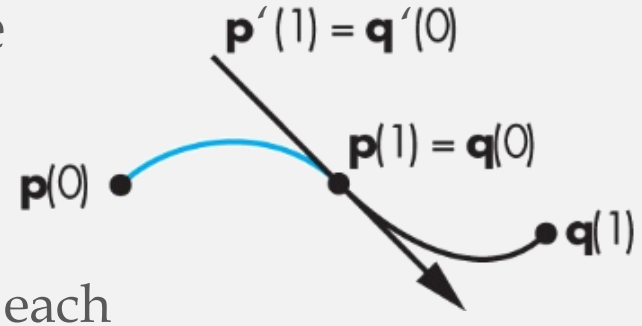
$$\mathbf{M_H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & 2 & 1 & 1 \end{bmatrix}$$

$$\Rightarrow \quad \mathbf{c} = \mathbf{M_H Q}$$

$$\Rightarrow \quad \mathbf{p}(u) = \mathbf{u}^T \mathbf{c} = \mathbf{u}^T \mathbf{M}_H \mathbf{Q}$$

Hermite Curves (Cont.)

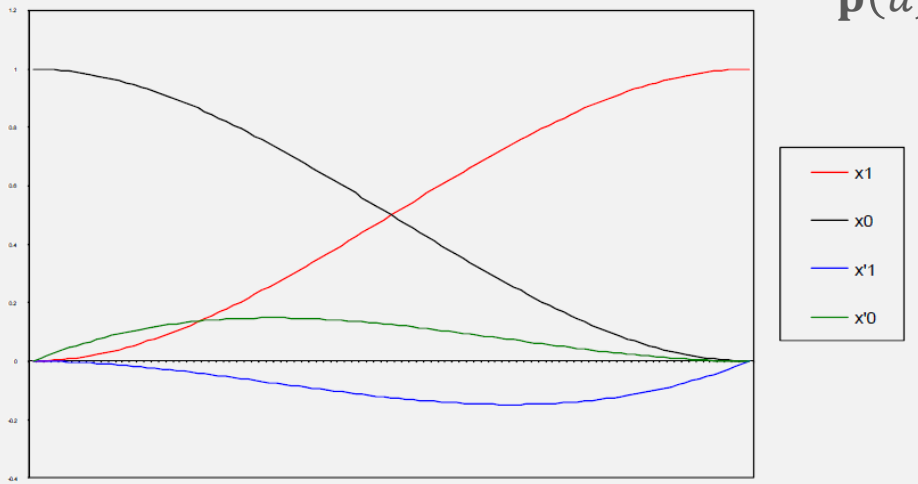
- Both the resulting function and the first derivative are continuous over all segments
- Represented in blending functions:
  - A point on a Hermite curve is obtained by weighted blending each control point and tangent vector.



$$\mathbf{p}(u) = \mathbf{u}^T\mathbf{c} = \mathbf{u}^T\mathbf{M}_H\mathbf{Q}$$

$$\mathbf{p}(u) = \mathbf{b}(u)^T\mathbf{Q}$$

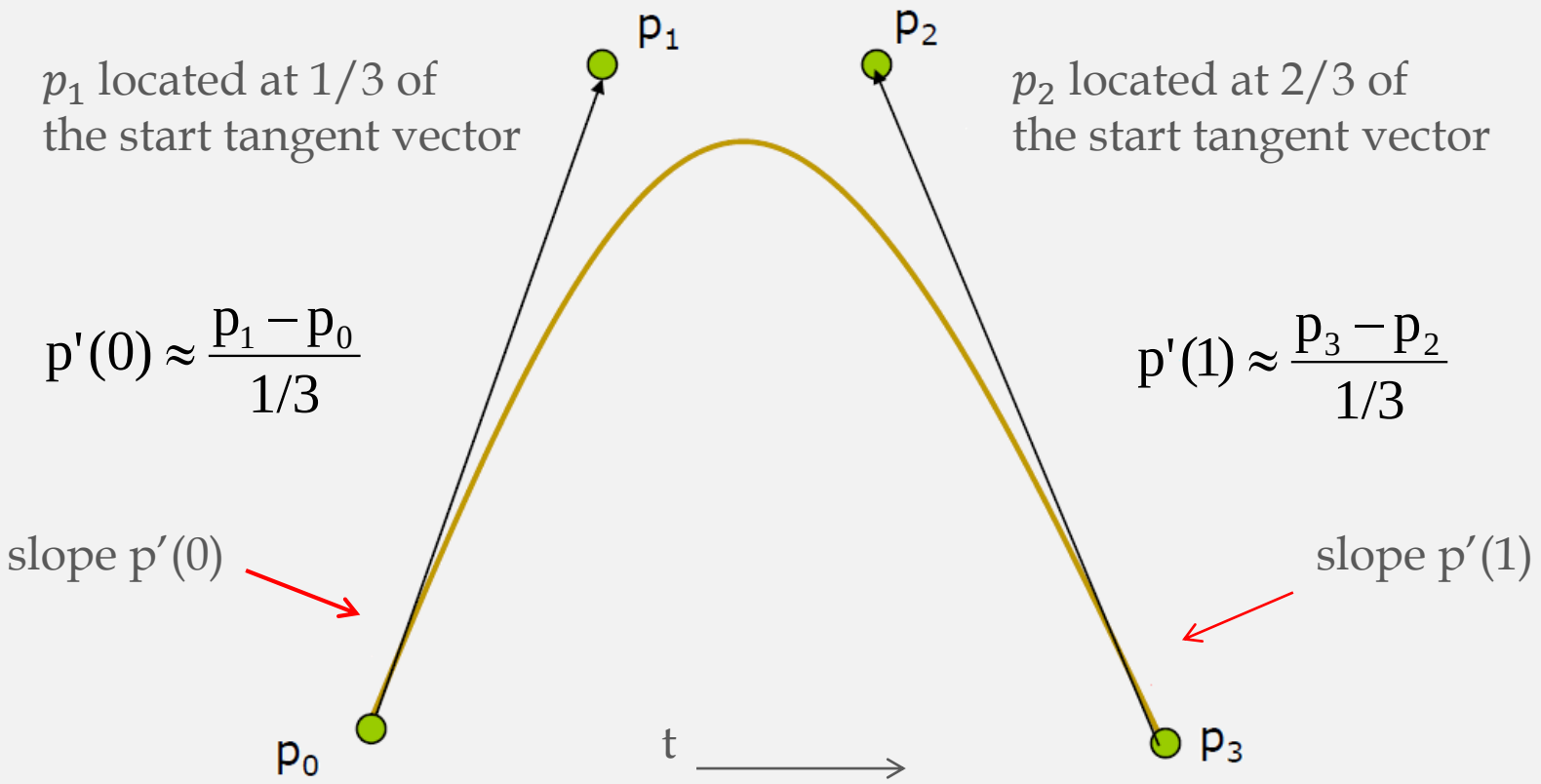
$$\mathbf{b}(u) = \mathbf{M}_H{}^T\mathbf{u} = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}$$



Weights of each component

Bezier Curves

- Two control points define endpoints, and two points control the tangents.



Bezier Curves (Cont.)

- The endsite conditions are the same
  - $P(0) = P_0 = c_0$
  - $P(1) = P_3 = c_0 + c_1 + c_2 + c_3$
- Approximating derivative conditions
  - $P'(0) = 3(P_1 - P_0) = c_1$
  - $P'(1) = 3(P_3 - P_2) = c_1 + 2c_2 + 3c_3$
- Replacing the original Hermite matrix.

$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$$

$$\mathbf{c} = \mathbf{M_B P} \qquad \mathbf{M_B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Bezier Geometry Matrix

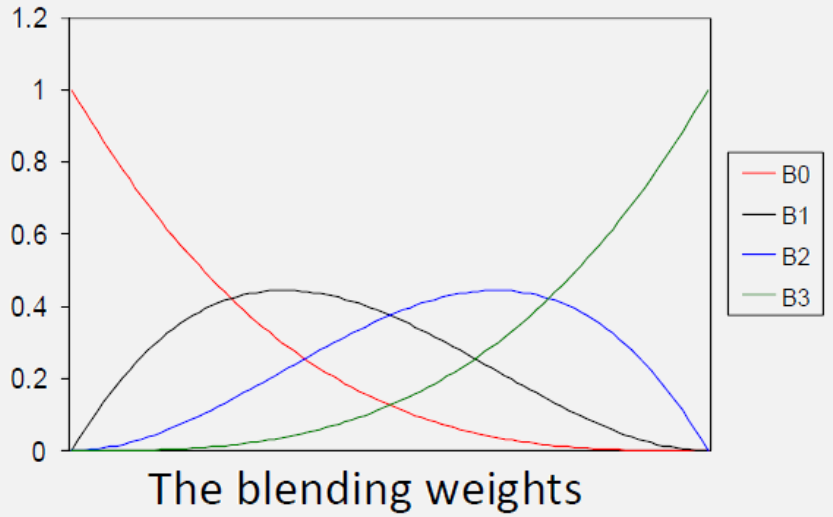
$$\mathbf{p}(u) = \mathbf{u}^T \mathbf{c} = \mathbf{u}^T \mathbf{M_B P}$$

Bezier Curves (Cont.)

- Represented in blending functions:
  - A point on a Bezier curve is obtained by weighted blending each control point

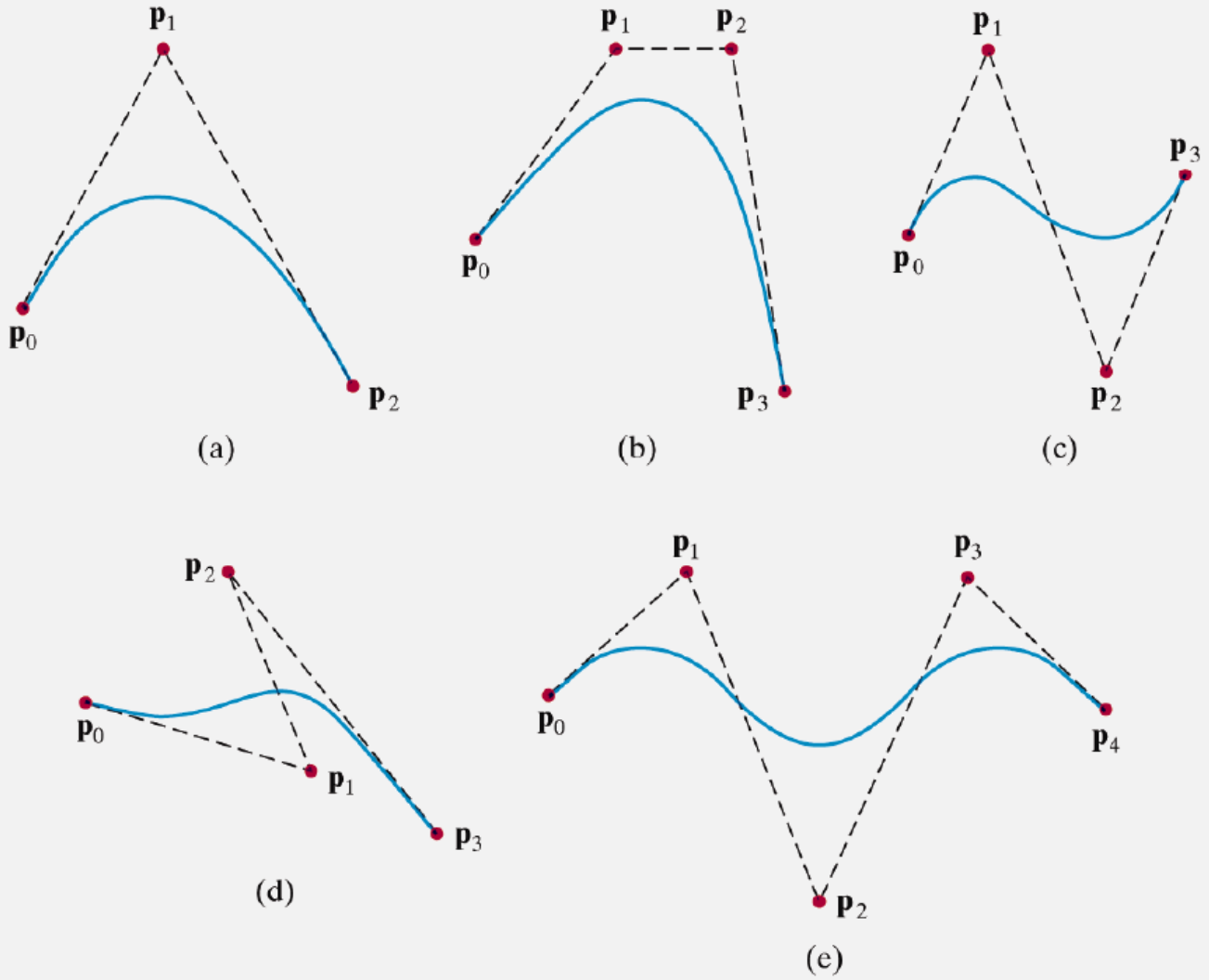
$$\mathbf{p}(u) = \mathbf{b}(u)^T \mathbf{P}$$

$$\mathbf{b}(u) = \mathbf{M_B}^T \mathbf{u} = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u)^2 \\ u^3 \end{bmatrix}$$



The blending weights

Examples of Bezier curves

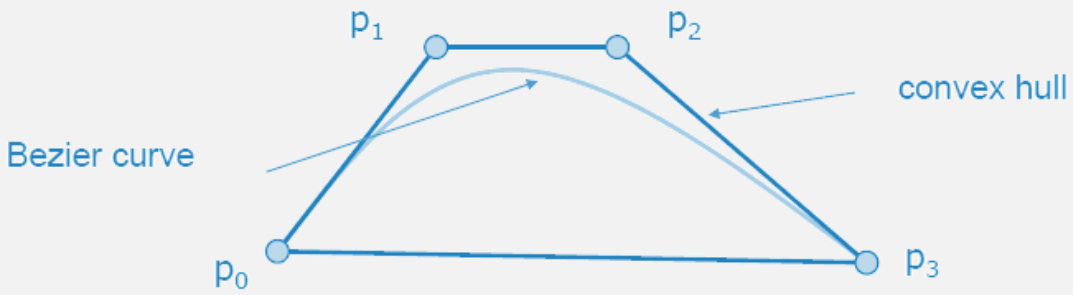


Bernstein Polynomials

- The blending functions of cubic Bezier curves are a special case of the Bernstein polynomials

$$b_{kd}(u) = \frac{d!}{k!(d-k)!} u^k (1-u)^{d-k}$$

- These polynomials give the blending polynomials for any degree Bezier form
  - All the zeros are either at u=0 or at u=1
  - For any degree they all sum to 1: $\sum_{i=0}^{d} b_{id}(u) = 1$
  - They are all between 0 and 1 inside [0,1]
  - Bezier curves lie in the convex hull of their control points



Bezier Patch (Cont.)

- Bezier curves can be extended to surfaces {from u to (u,v)}.

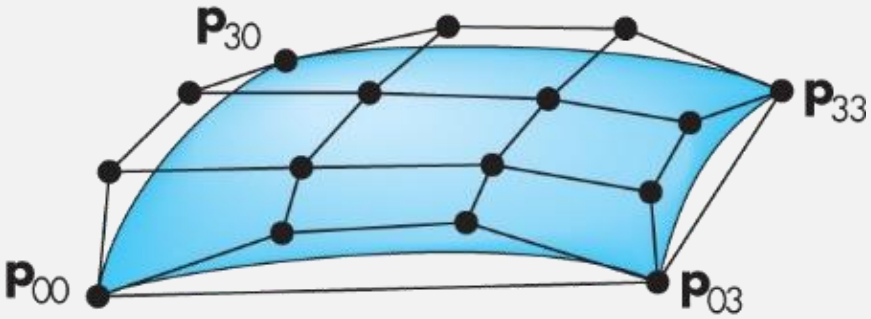
$$\mathrm{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u) b_j(v)\,\mathbf{p}_{ij} = \mathbf{u}^T \mathbf{M}_B \mathbf{P} \mathbf{M}_B^T \mathbf{v}$$

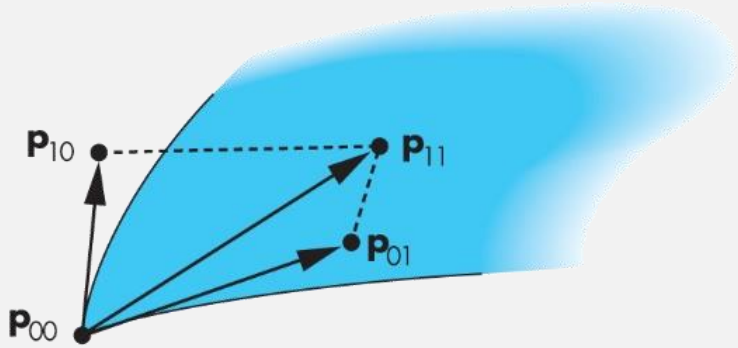
$$\mathbf{p}(0,0) = \mathbf{p}_{00}$$

$$\frac{\partial \mathbf{p}}{\partial u}(0,0) = 3(\mathbf{p}_{10} - \mathbf{p}_{00})$$

$$\frac{\partial \mathbf{p}}{\partial v}(0,0) = 3(\mathbf{p}_{01} - \mathbf{p}_{00})$$

$$\frac{\partial^2 \mathbf{p}}{\partial u \partial v}(0,0) = 9(\mathbf{p}_{00} - \mathbf{p}_{01} + \mathbf{p}_{10} - \mathbf{p}_{11})$$

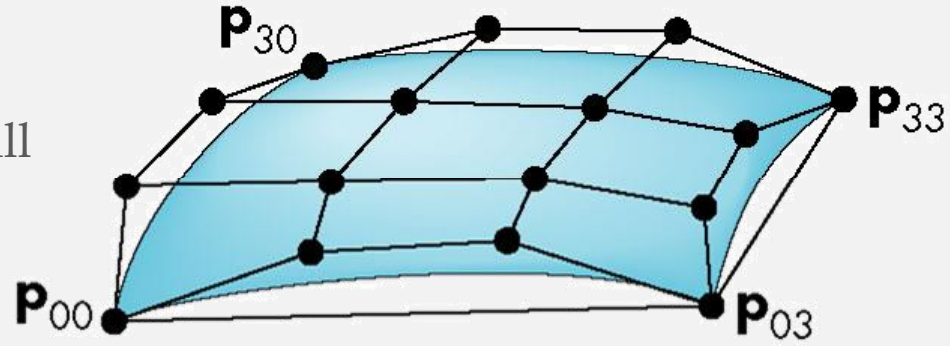




Twist (a measure of the tendency of the patch to divert from being flat)

Matrix Form of Bezier Patch

Patch lies in convex hull



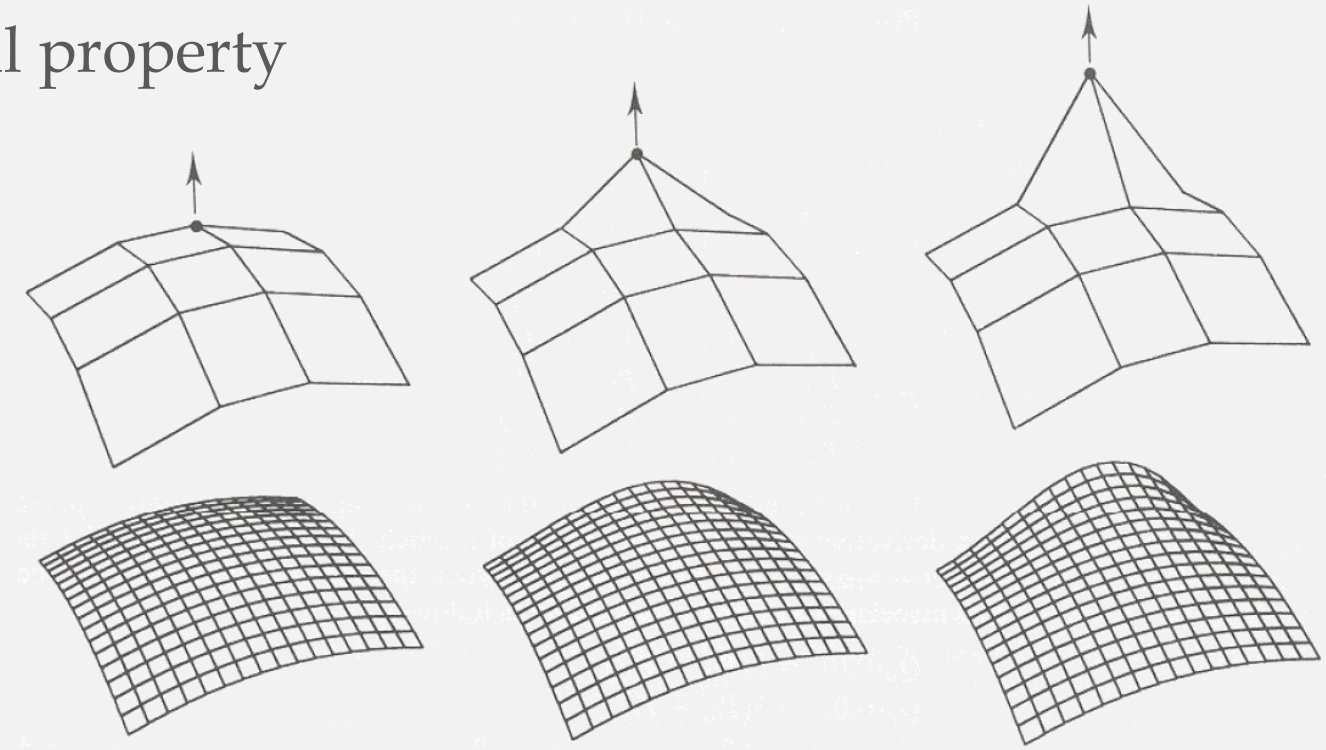
$$\mathrm{p}(u,v) = \sum_{j=0}^{n}\sum_{k=0}^{n} p_{jk} B_j^n(u) B_k^n(v)$$

$$= \mathrm{U}^{\mathrm{T}} \cdot \mathrm{M} \cdot \mathrm{G} \cdot \mathrm{M}^{\mathrm{T}} \cdot \mathrm{V}$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{00} & p_{10} & p_{20} & p_{30} \\ p_{01} & p_{11} & p_{21} & p_{31} \\ p_{02} & p_{12} & p_{22} & p_{32} \\ p_{03} & p_{13} & p_{23} & p_{33} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Bezier Patches

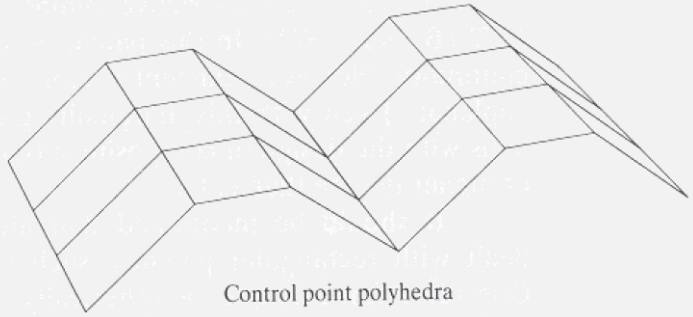
- Interpolates four corner points
- Convex hull property

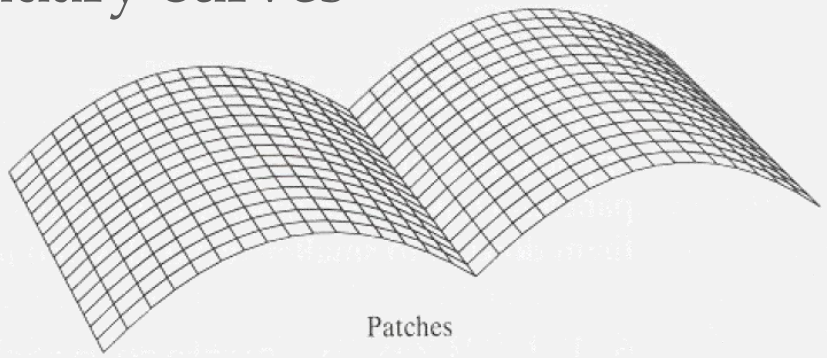


Watt, 3D Graphics

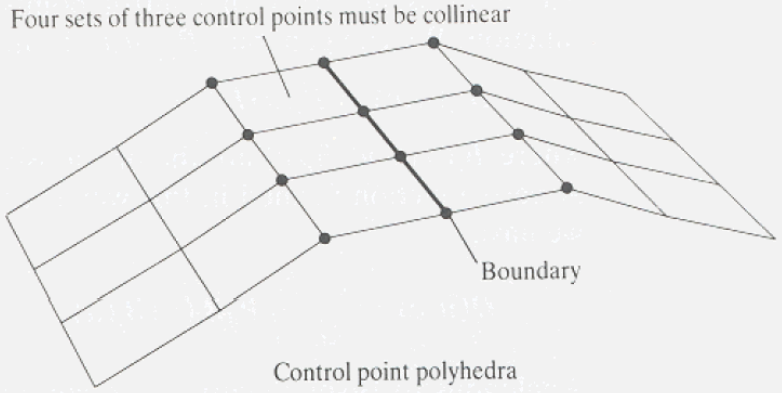
Bezier Surfaces

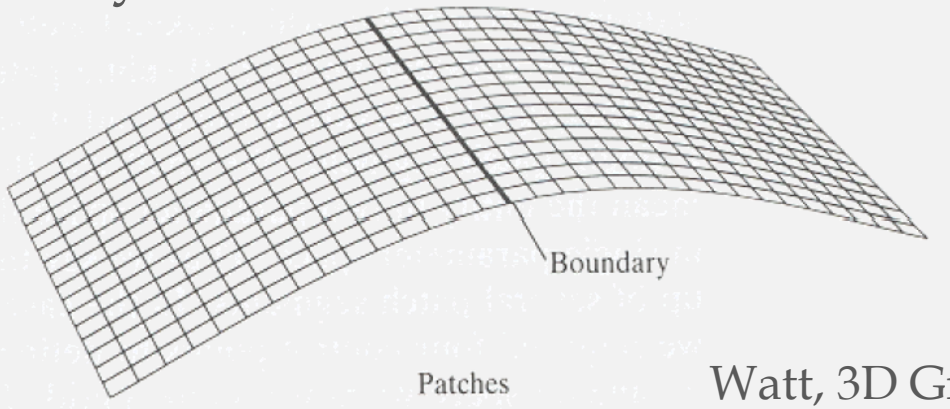
- C0 continuity requires aligning boundary curves





- C1 continuity requires aligning boundary curves and derivatives





Watt, 3D Graphics

Bezier Curve Subdivision

- Subdividing control polylines
  - produces two new control polylines for each half of the curve
  - defines the same curve
  - all control points are closer to the curve

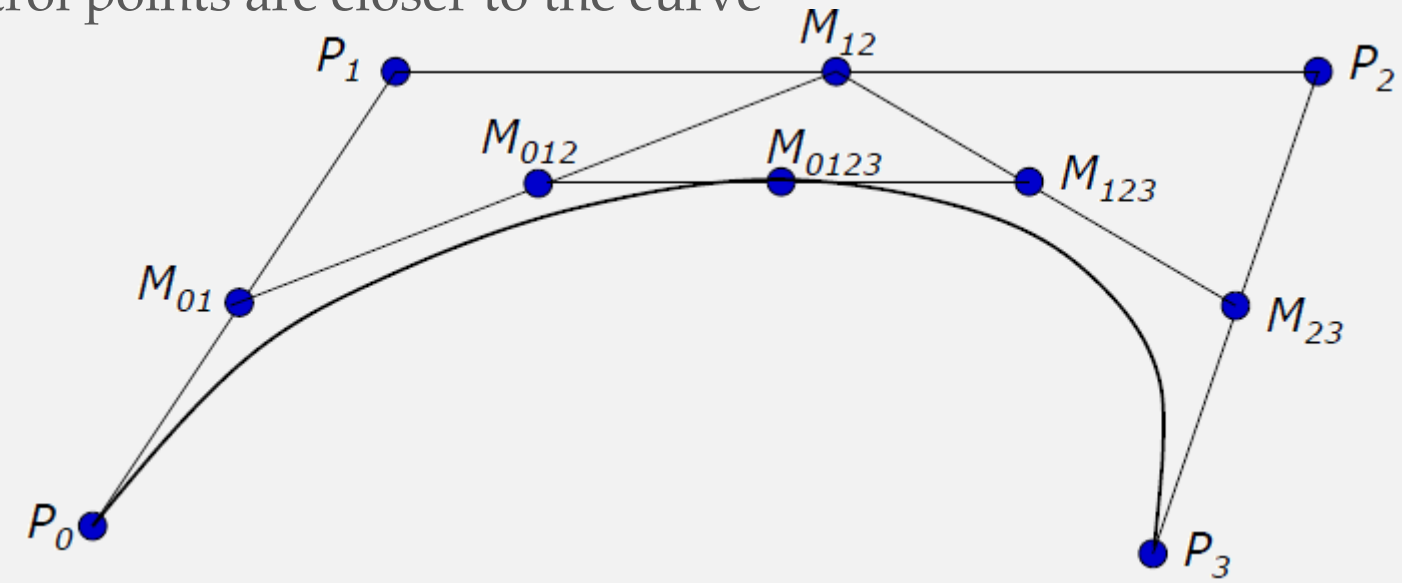
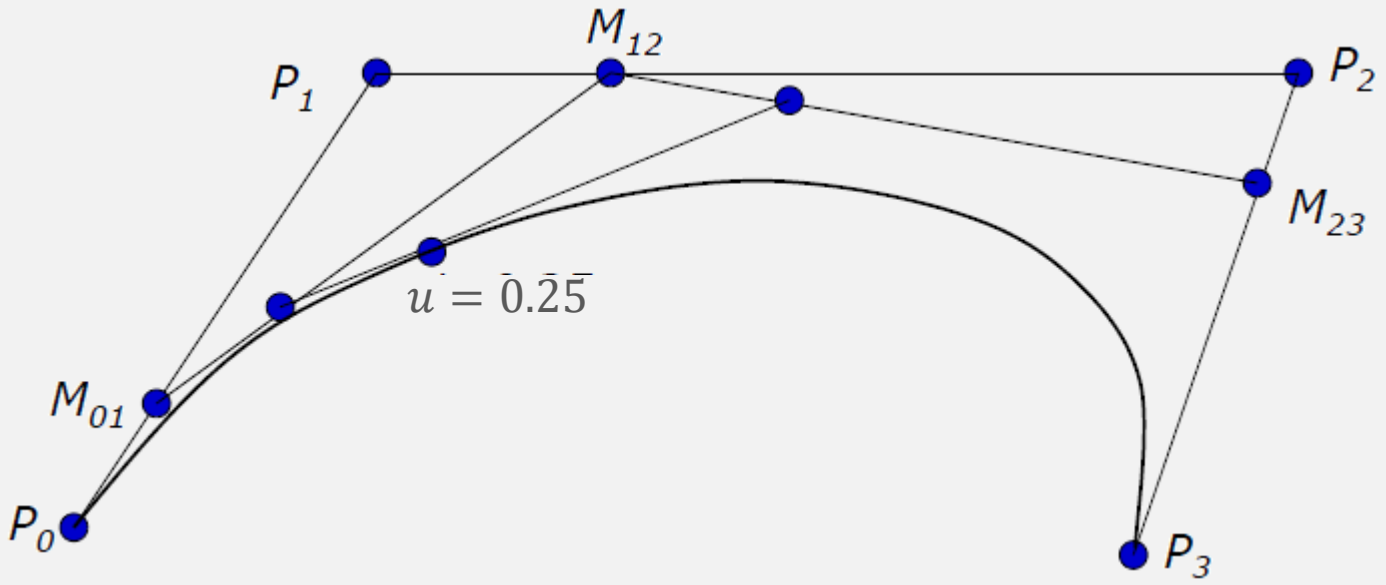


Figure from Prof. S.Chenney, Computer Graphics coursenote, Univ. Wisconsin

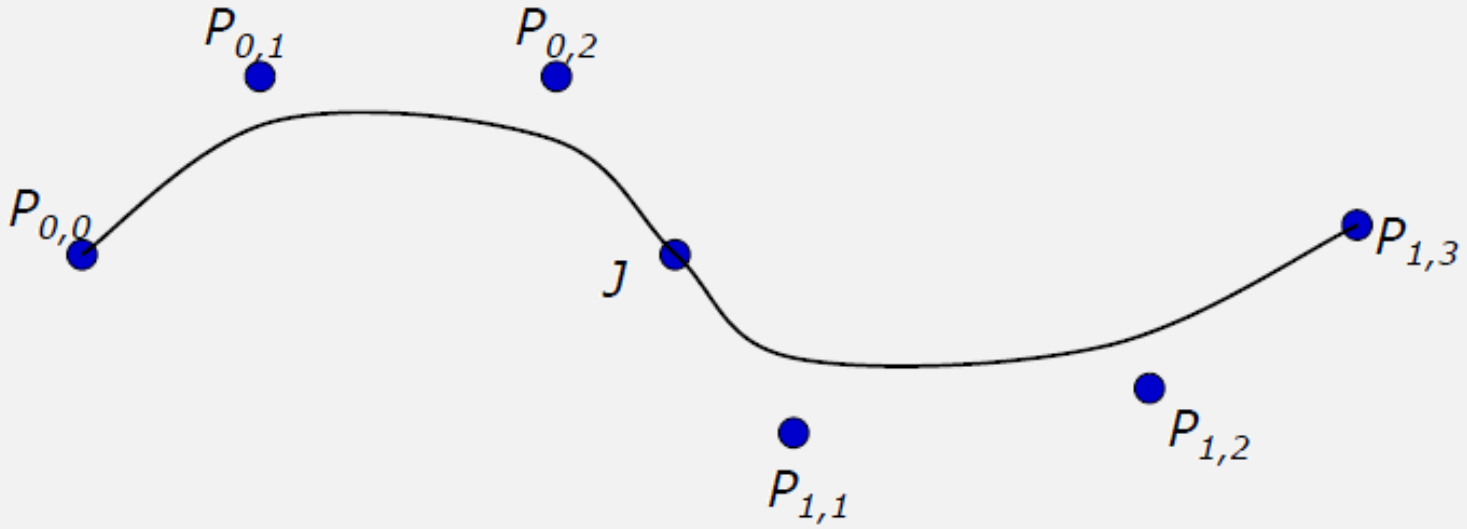
de Casteljau's Algorithm

- You can find the point on a Bezier curve for any parameter value u by subdivision
  - Eg. u=0.25



Bezier Continuity

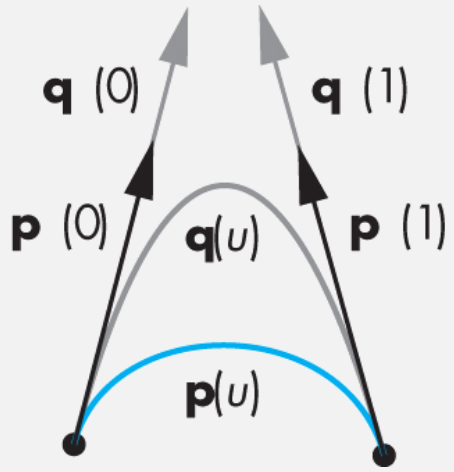
- We can make a long curve by concatenating multiple short Bezier curves.

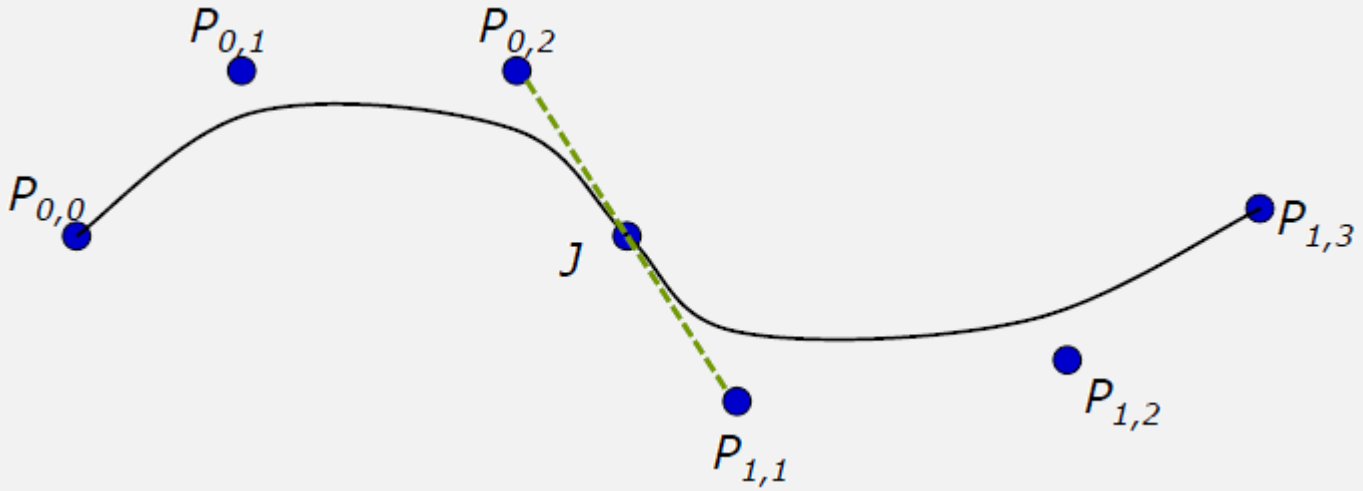


- How to keep the continuity?

Continuity Properties

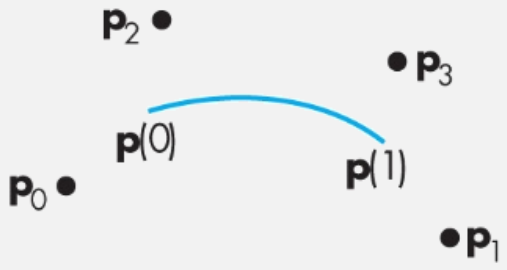
- C^0 continuous :curve/surface has no breaks
- G^1 continuous : tangent at joint has same direction
- C^1 continuous : tangent at join has same direction and magnitude
- C^n continuous : curve/surface through nth derivative is continuous





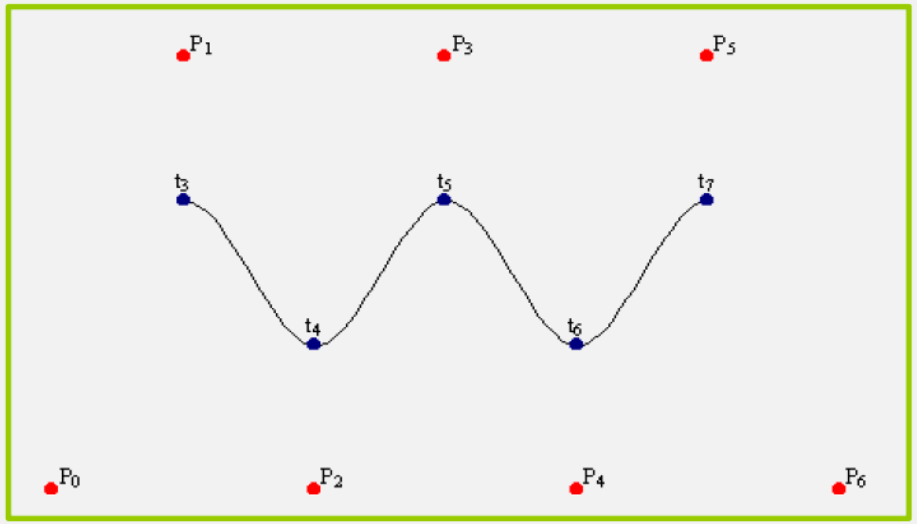
B-splines

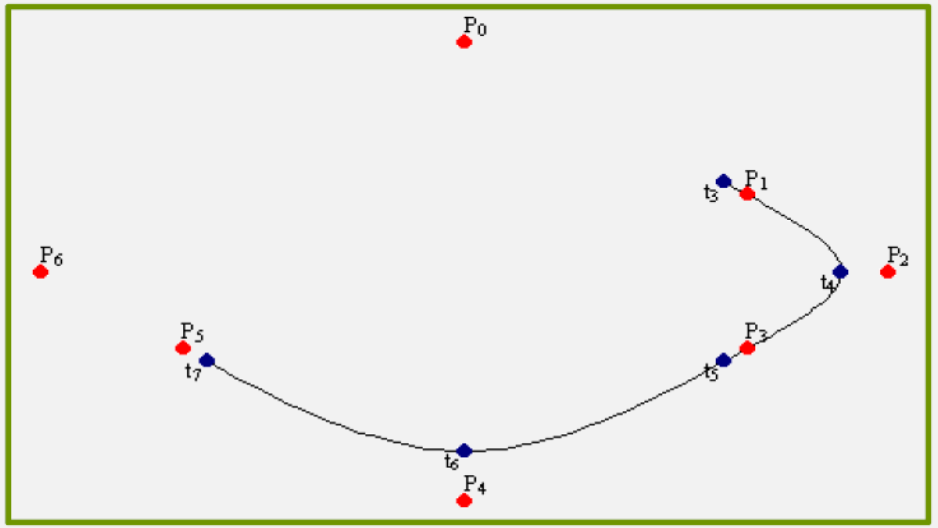
- How to reach both C^2 continuity and local controllability ?
  - Slightly loose the endpoint constraints.
  - B-splines do not interpolate any of control points.



B-spline Curves

- Start with a sequence of control points
- Select four from middle of sequence $(p_{i-2}, p_{i-1}, p_i, p_{i+1})$
- Bezier and Hermite goes between p_{i-2} and p_{i+1}
- B-Spline doesn't interpolate (touch) any of them but approximates the curve by going through p_{i-2} and p_{i+1}.





Figures from CG lecture note, U. Virginia

B-spline Curves (Cont.)

$$\mathbf{p}(0) = \mathbf{q}(1) = \frac{1}{6}(\mathbf{p}_{i-2} + 4\mathbf{p}_{i-1} + \mathbf{p}_i)$$

$$\mathbf{p}'(0) = \mathbf{q}'(1) = \frac{1}{2}(\mathbf{p}_i - \mathbf{p}_{i-2})$$

(for symmetric)

Since $\mathbf{p}(u) = \mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 u^2 + \mathbf{c}_3 u^3 = u^T \mathbf{c}$

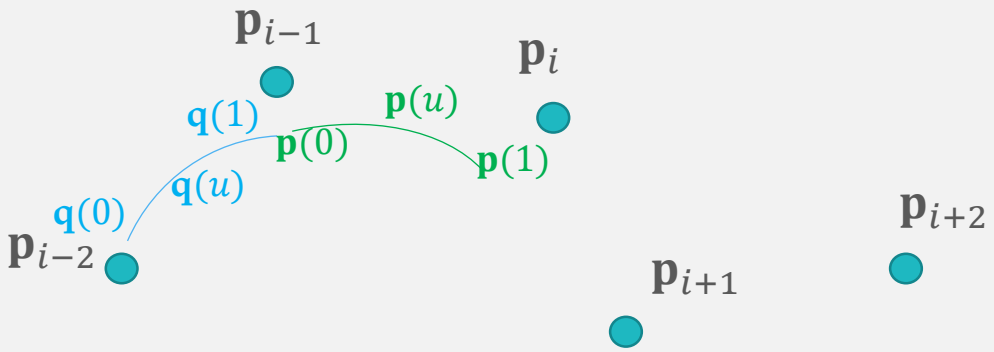
$$\mathbf{p}(0) = \mathbf{c}_0 = \frac{1}{6}(\mathbf{p}_{i-2} + 4\mathbf{p}_{i-1} + \mathbf{p}_i)$$

$$\mathbf{p}'(0) = \mathbf{c}_1 = \frac{1}{2}(\mathbf{p}_i - \mathbf{p}_{i-2})$$

$$\mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 = \frac{1}{6}(\mathbf{p}_{i-1} + 4\mathbf{p}_i + \mathbf{p}_{i+1})$$

$$\mathbf{p}'(1) = \mathbf{c}_1 + 2\mathbf{c}_2 + 3\mathbf{c}_3 = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$$

$\mathbf{p}_{i-3}$



$$\Rightarrow \mathbf{P} = \mathbf{A}\mathbf{c}$$

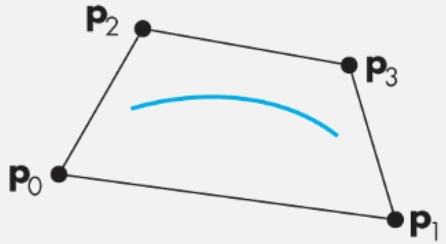
$$\mathbf{M}_S = \mathbf{A}^{-1} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad \Rightarrow \mathbf{c} = \mathbf{M}_S\mathbf{P}$$

The Blending Weights

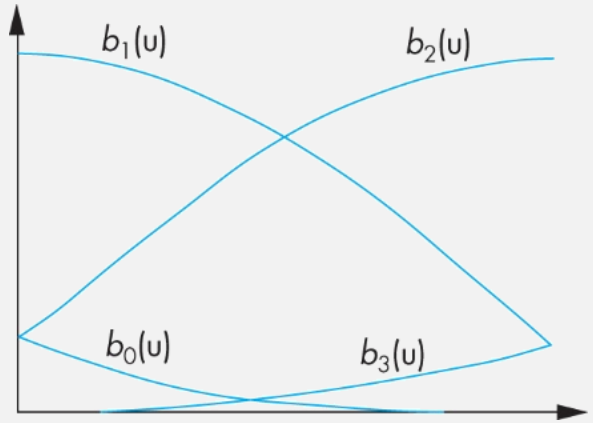
$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3 = \mathbf{u}^T \mathbf{c} = \mathbf{u}^T \mathbf{M_S P} = \mathbf{b}(u)^T \mathbf{P}$$

$$\mathbf{b}(u) = \begin{bmatrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{bmatrix} = \mathbf{M_S}^T \mathbf{u} = \frac{1}{6} \begin{bmatrix} (1-u)^3 \\ 4 - 6u^2 + 3u^2 \\ 1 + 3u + 3u^2 - 3u^3 \\ u^3 \end{bmatrix}$$

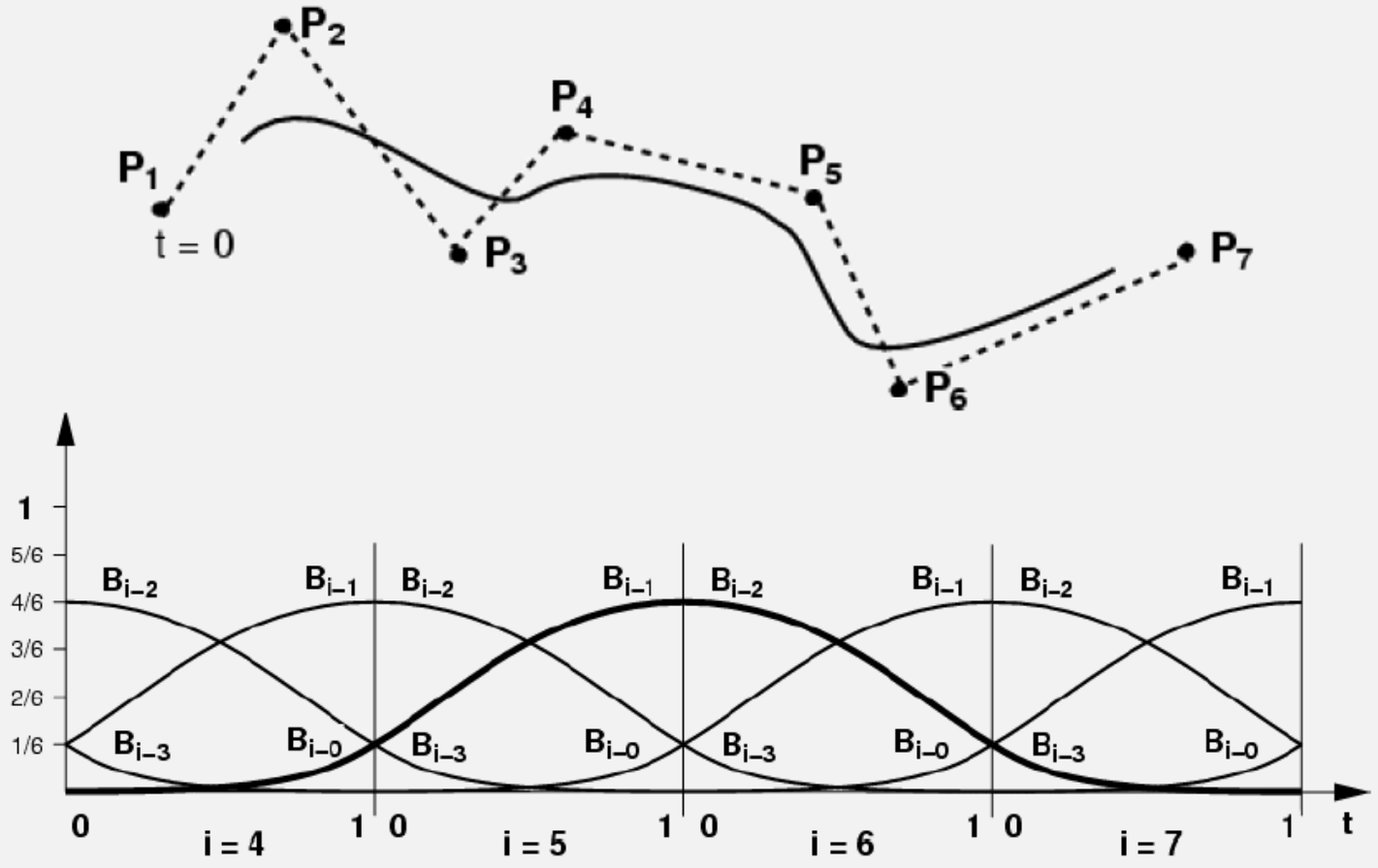
$$\sum_{i=0}^{3} b_i(u) = 1 \quad \text{and} \quad 0 < b_i(u) < 1 \ \text{ in the interval } 0 < u < 1$$



The curve must lie in the convex hull



The Blending Weights (Cont.)



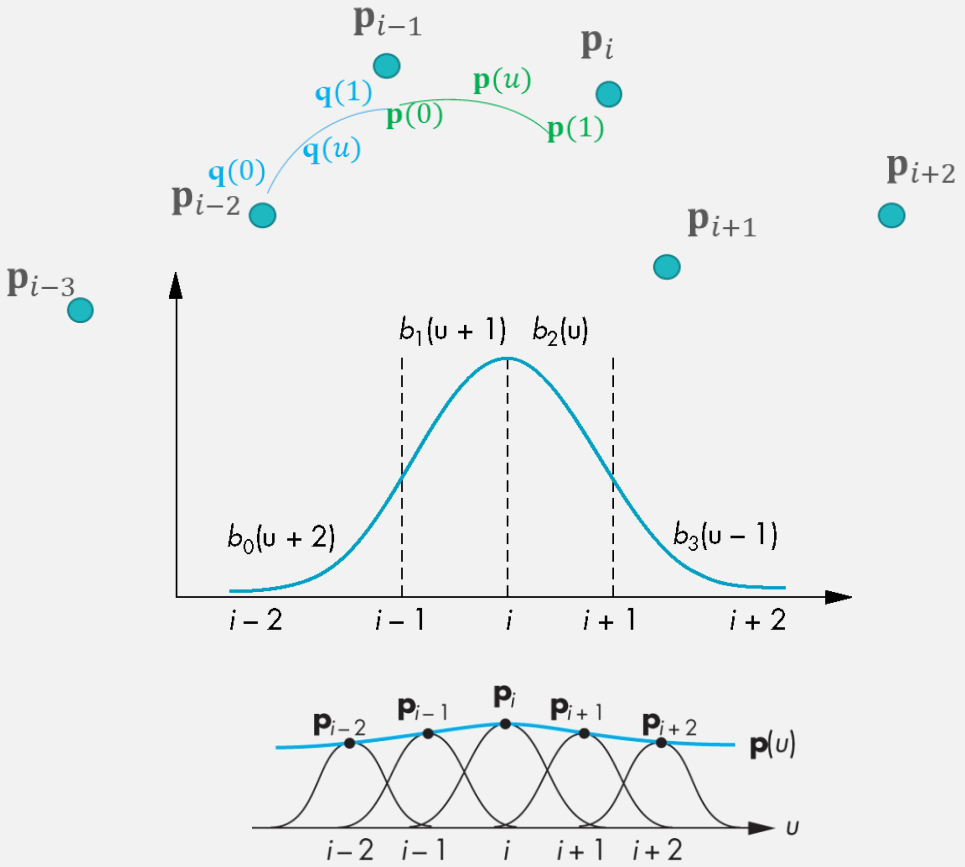
Figures from MIT EECS 6.837, Durand and Cutler

Basis Functions

- From the perspective of a single control point, each control point contributes to the spline in four adjacent intervals.

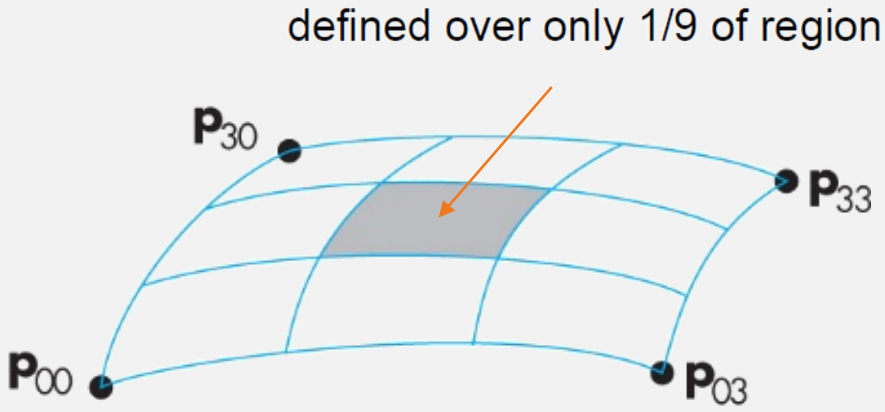
$$\mathbf{p}(u) = \sum_{i=1}^{m-1} B_i(u)\mathbf{p}_i$$

$$B_i(u) = \begin{cases} 0 & u < i-2 \\ b_0(u+2) & i-2 \le u < i-1 \\ b_1(u+1) & i-1 \le u < i \\ b_2(u) & i \le u < i+1 \\ b_3(u-1) & i+1 \le u < i+2 \\ 0 & u \ge i+2 \end{cases}$$



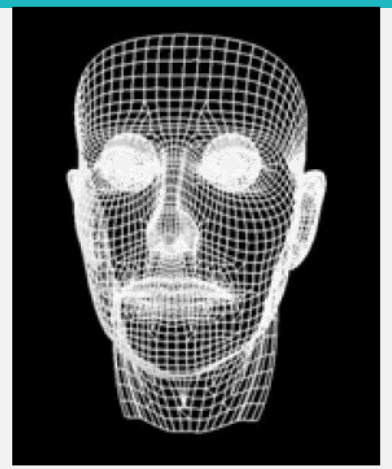
B-Spline Surface (Patch)

$$p(u,v) = \sum_{i=0}^{3}\sum_{k=0}^{3} b_i(u) b_j(v) p_{ij} = u^T \mathrm{M}_s \mathrm{P} \mathrm{M}_S^T v$$

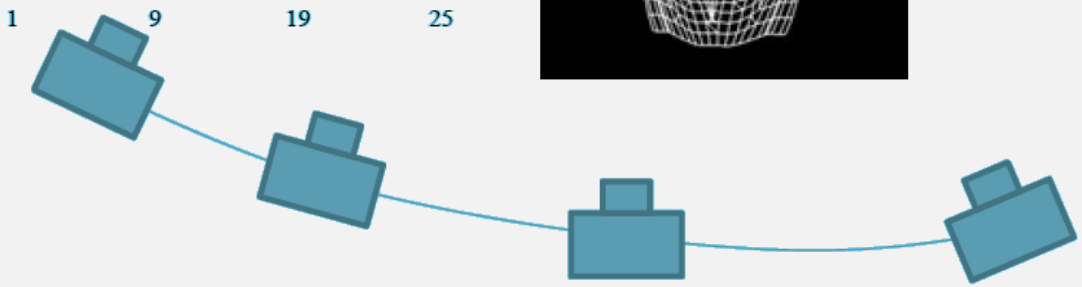


Applications of Splines and Surfaces

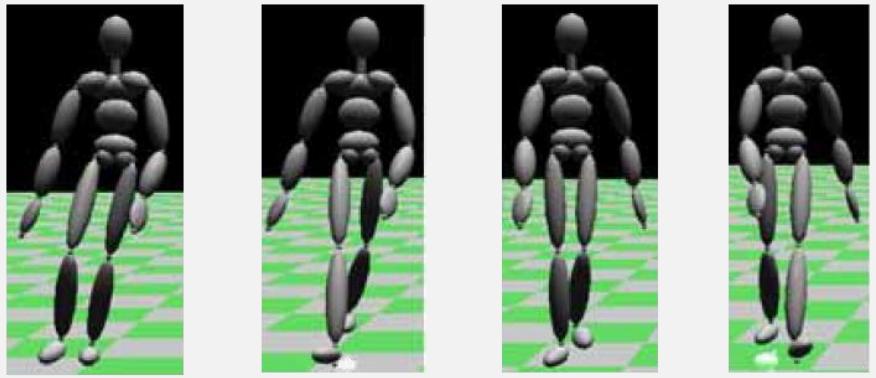
- Modeling and editing 3D objects.



- Smooth paths (e.g. camera views)



- Key-frame animation.



- etc….

How to render polynomial curves and surfaces ?

- Simplest method
  - Use Horner's method to evaluate polynomials (3 multiplications/evaluation for cubic)

$$p(u)=c_0+u(c_1+u(c_2+uc_3))$$

  - For equally spaced $\{u_k\}$, define finite differences

$$\Delta^{(0)}p(u_k)=p(u_k) \qquad \Delta^{(1)}p(u_k)=p(u_{k+1})-p(u_k) \qquad \Delta^{(m+1)}p(u_k)=\Delta^{(m)}p(u_{k+1})-\Delta^{(m)}p(u_k)$$

  - Build a **Finite Difference Table** for p(u)
  - Starting at the bottom, we can work up generating new values for the polynomial

$$\Delta^{(m-1)}p(u_{k+1})=\Delta^{(m)}p(u_k)+\Delta^{(m-1)}p(u_k)$$

t	0	1	2	3	4	5
p	1	7	23	55	109	191
$\Delta^{(1)}$p	6	16	32	54	82	
$\Delta^{(2)}$p	10	16	22	28		
$\Delta^{(3)}$p	6	6	6			

Constant

$p(u)=1+3u+2u^2+u^3$

How to render polynomial curves and surfaces ?

- Simplest method
  - Use Horner's method to evaluate polynomials (3 multiplications/evaluation for cubic)

$$p(u)=c_0+u(c_1+u(c_2+uc_3))$$

  - For equally spaced $\{u_k\}$, define finite differences

$$\Delta^{(0)}p(u_k)=p(u_k) \qquad \Delta^{(1)}p(u_k)=p(u_{k+1})-p(u_k) \qquad \Delta^{(m+1)}p(u_k)=\Delta^{(m)}p(u_{k+1})-\Delta^{(m)}p(u_k)$$

  - Build a **Finite Difference Table** for p(u)
  - Starting at the bottom, we can work up generating new values for the polynomial

$$\Delta^{(m-1)}p(u_{k+1})=\Delta^{(m)}p(u_k)+\Delta^{(m-1)}p(u_k)$$

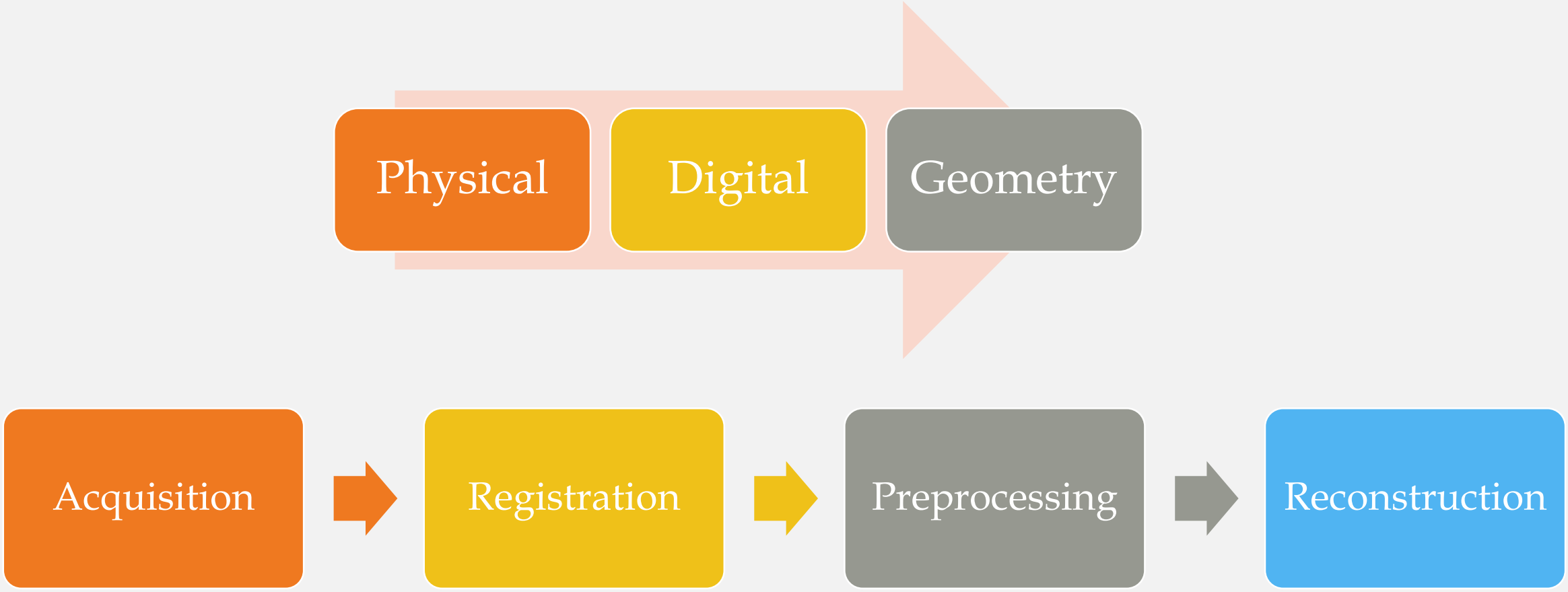
t	0	1	2	3	4	5
$\mathbf{p}$	1	7	23	55 →	109 →	191
$\Delta^{(1)}\mathbf{p}$	6	16	32 →	54 →	82	
$\Delta^{(2)}\mathbf{p}$	10	16 →	22 →	28		
$\Delta^{(3)}\mathbf{p}$	6 →	6 →	6			

$p(u)=1+3u+2u^2+u^3$

3D Reconstruction

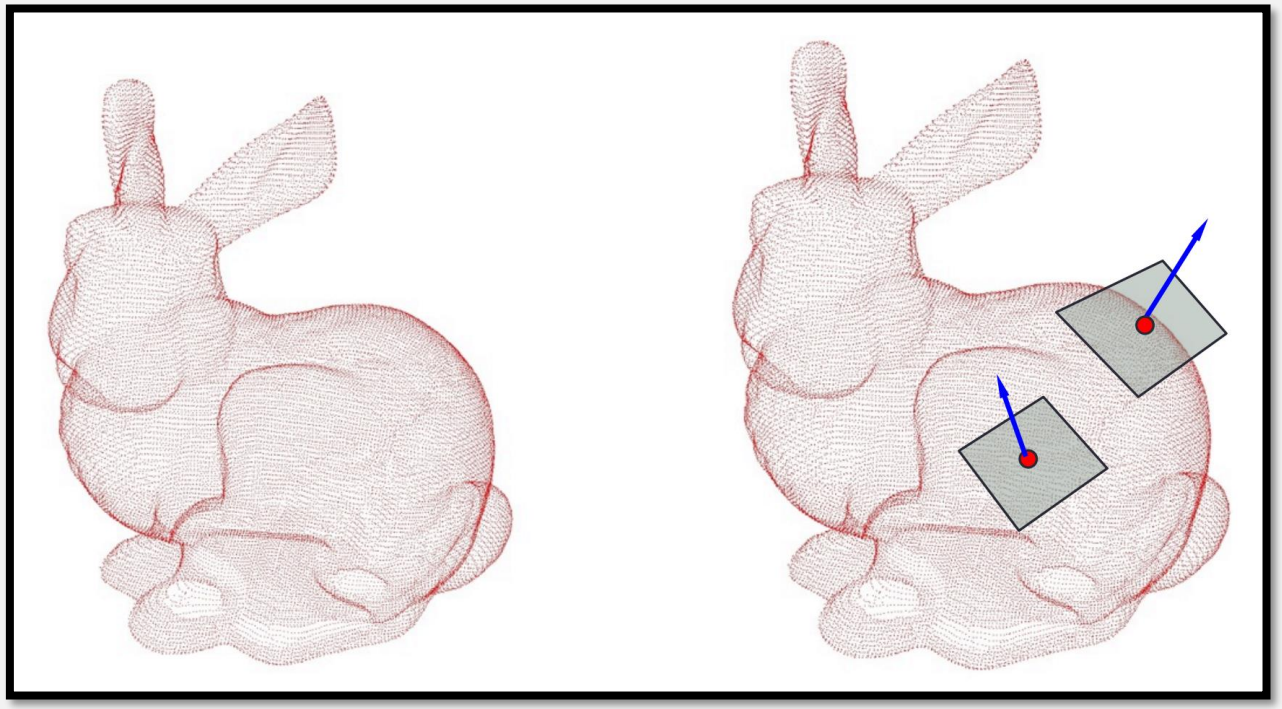
Pipeline

Common Pipeline



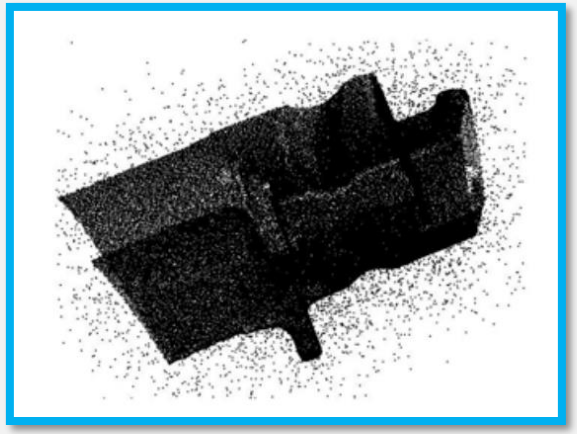
Point Clouds

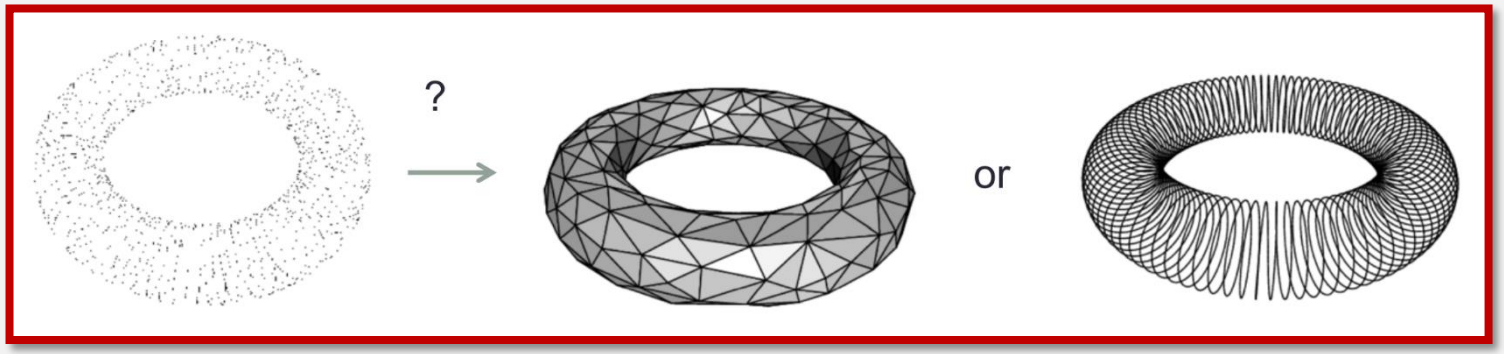
- Simplest representation: only points, no connectivity
- Collection of (x,y,z) coordinates, possibly with normals
- Points with orientation are called **surfels**



Limitations of Point Clouds

- Noise and outliers
- No simplification or subdivision
- No direct smooth rendering
- No topological information
- Weak approximation power





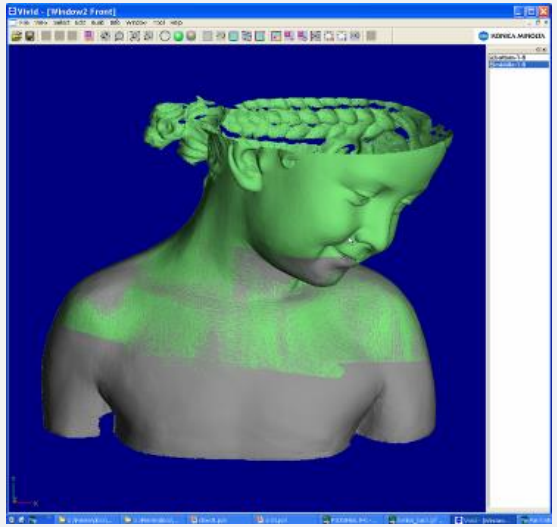
Why Point Cloud?

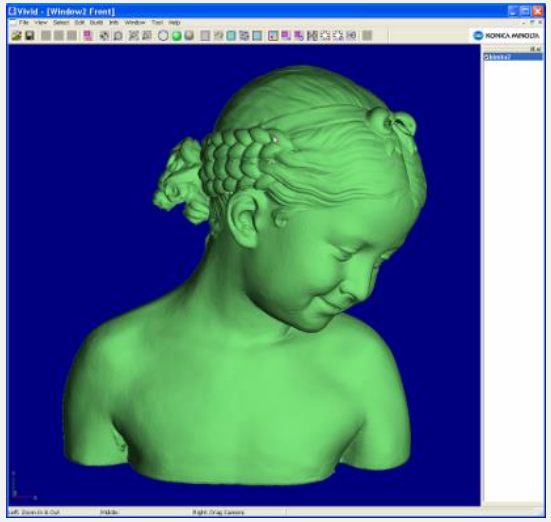
- That's the only thing available
  - Nearly all 3d scanning devices produce point clouds
- Sometimes, easier to handle (especially in hardware)

Common Pipeline (Example 1)

Acquisition → Registration → Preprocessing → Reconstruction

laser range data

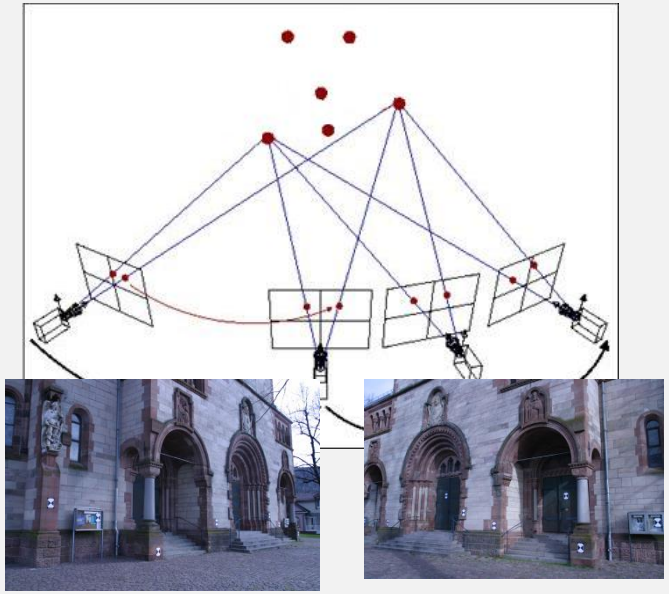




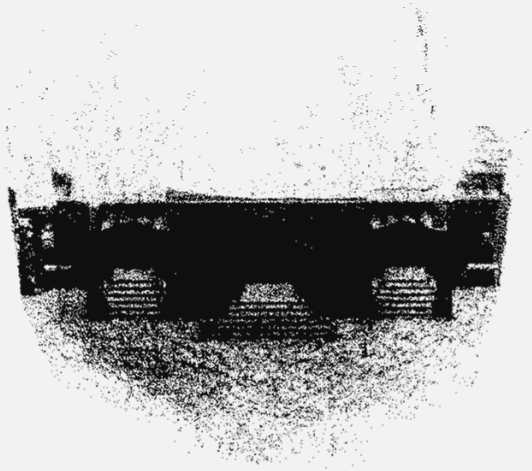
(simplification, denoising, smoothing, …)

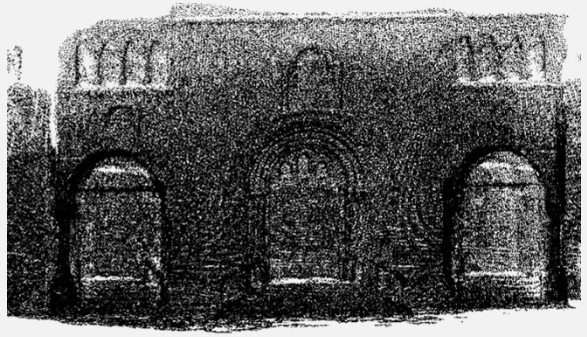
Common Pipeline (Example 2)

Acquisition & Calibration → Point Cloud Generation → Preprocessing → Reconstruction



multi-view passive stereo





(simplification, denoising, smoothing, …)

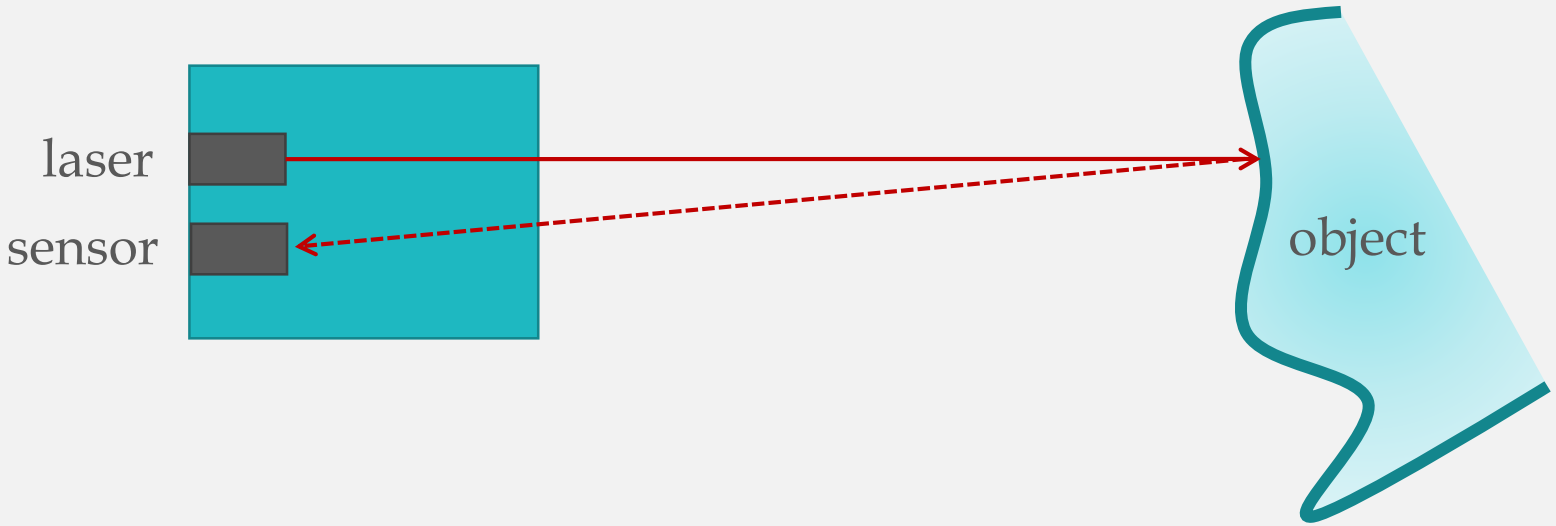
Major Types of 3d Scanners

- 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 pulse of laser
2. Capture the reflection
3. Measure the time it took to come back

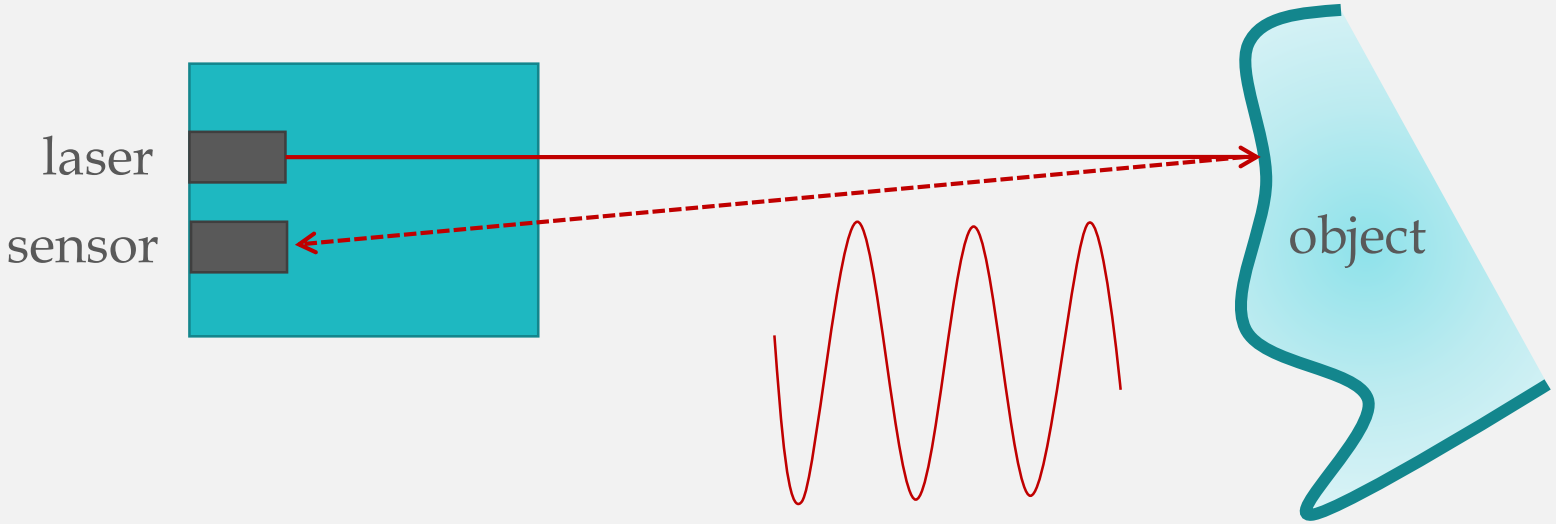
$$D = \frac{cT}{2} \qquad c = speed\ of\ light\ \approx 299792458\ m/s$$



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

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

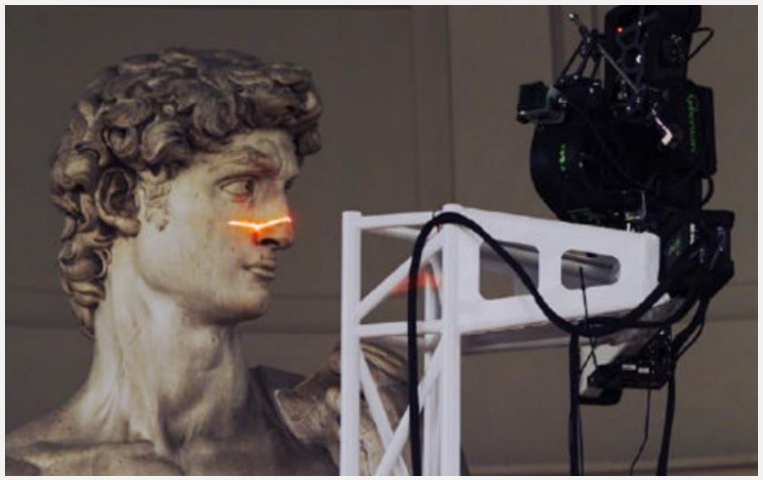


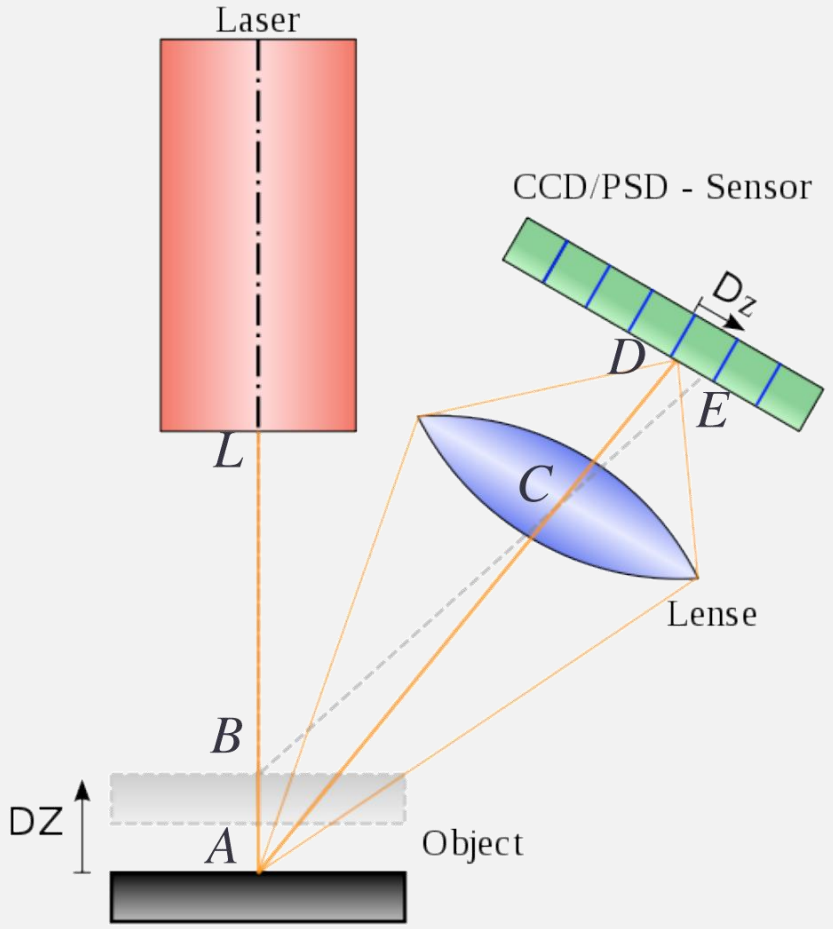
$$D = \frac{cT}{2} = \frac{c}{2f}\frac{\phi}{2\pi}$$

Triangulation-based Approaches

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

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



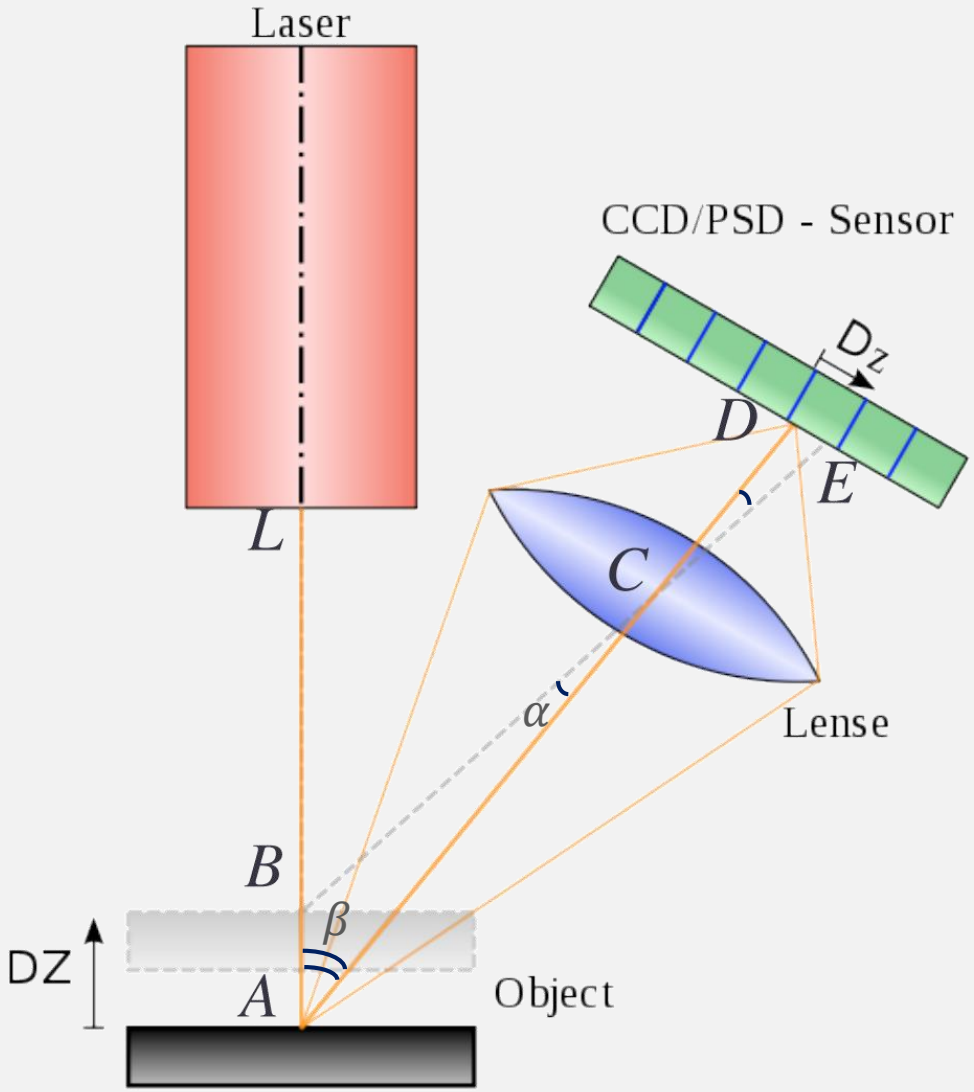


Triangulation-based Approaches (Cont.)

- Use Dz, CD and ∠CDE, compute ∠ DCE = α
- Use α, β, and AC = AD – CD, compute BC, AB
- The depth LB = LA - AB

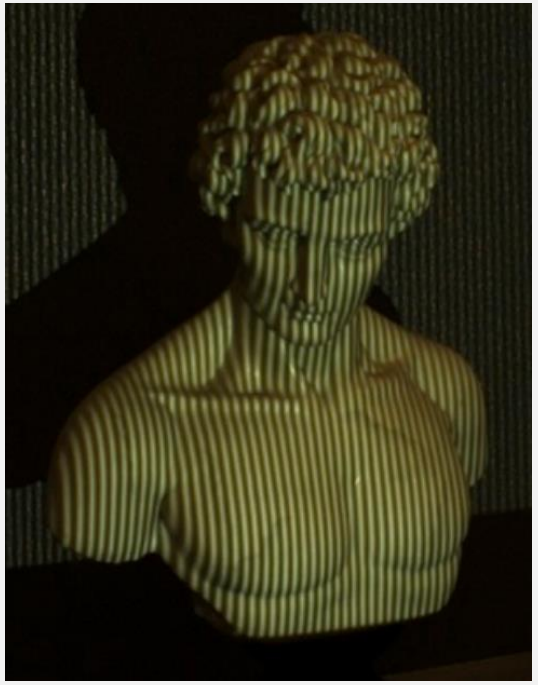
If well-calibrated, can lead to extremely accurate depth measurements.

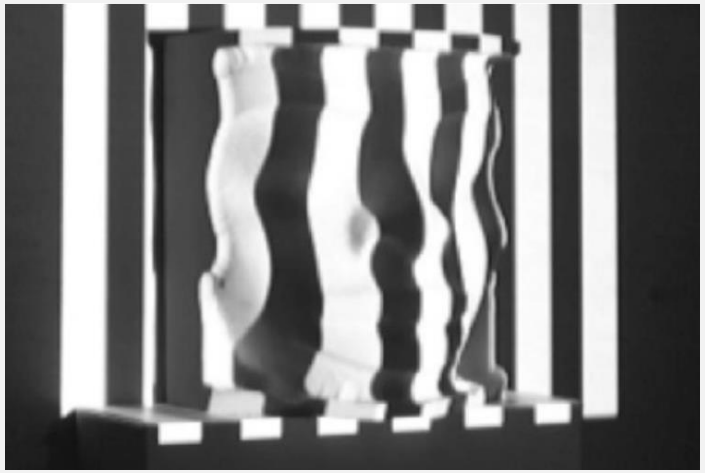
Main problem: slow and expensive !



Structured-Light Scanners

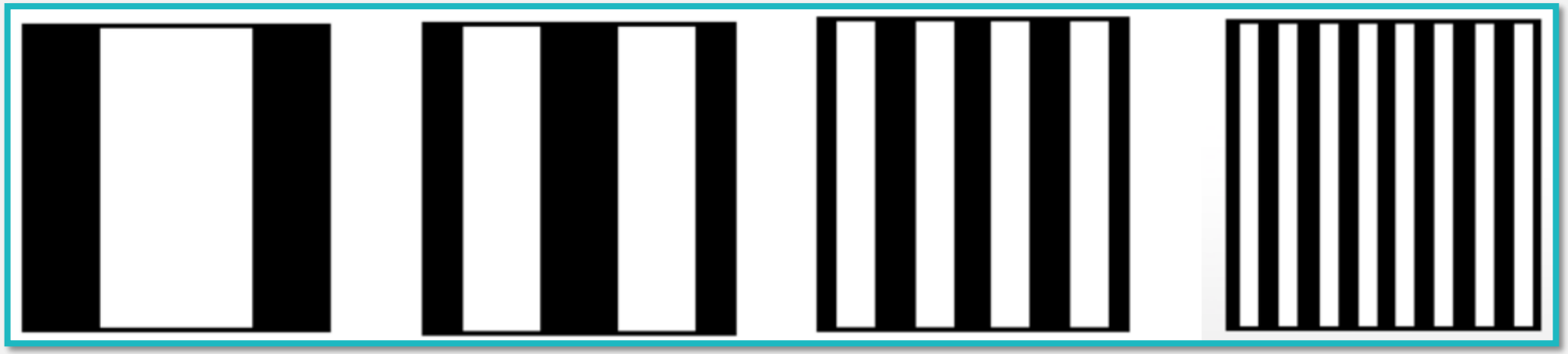
- Same general idea as Triangulation based scanner
- Replace laser with projector (Project stripes instead of sheets)
- Challenge: Need to identify which (input/output) lines correspond





Structured-Light Scanners (Cont.)

- Project **multiple stripes** to identify the position of a point

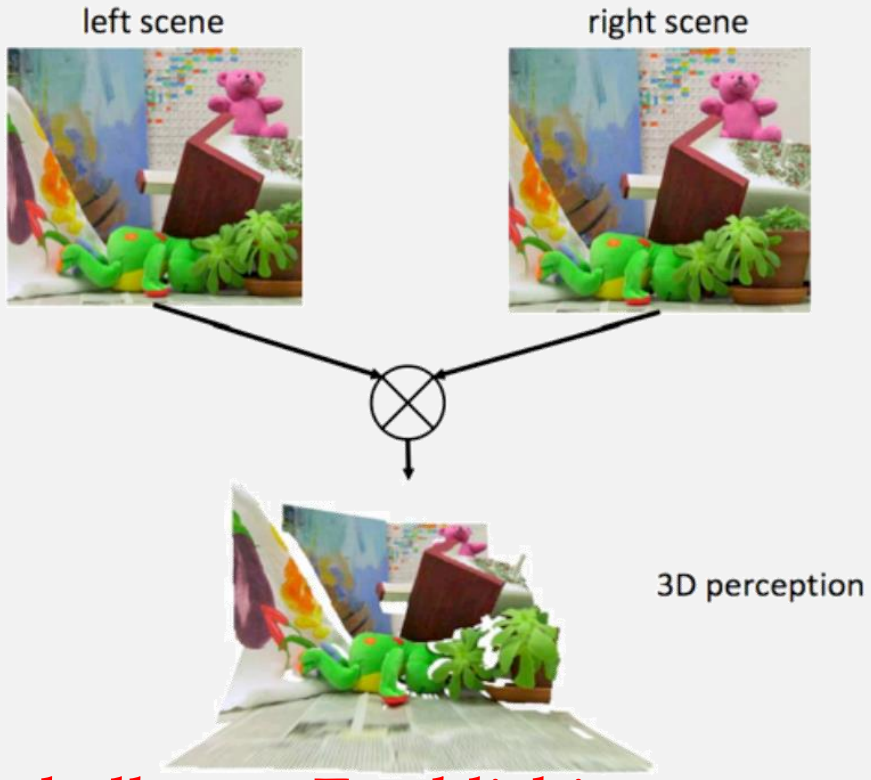


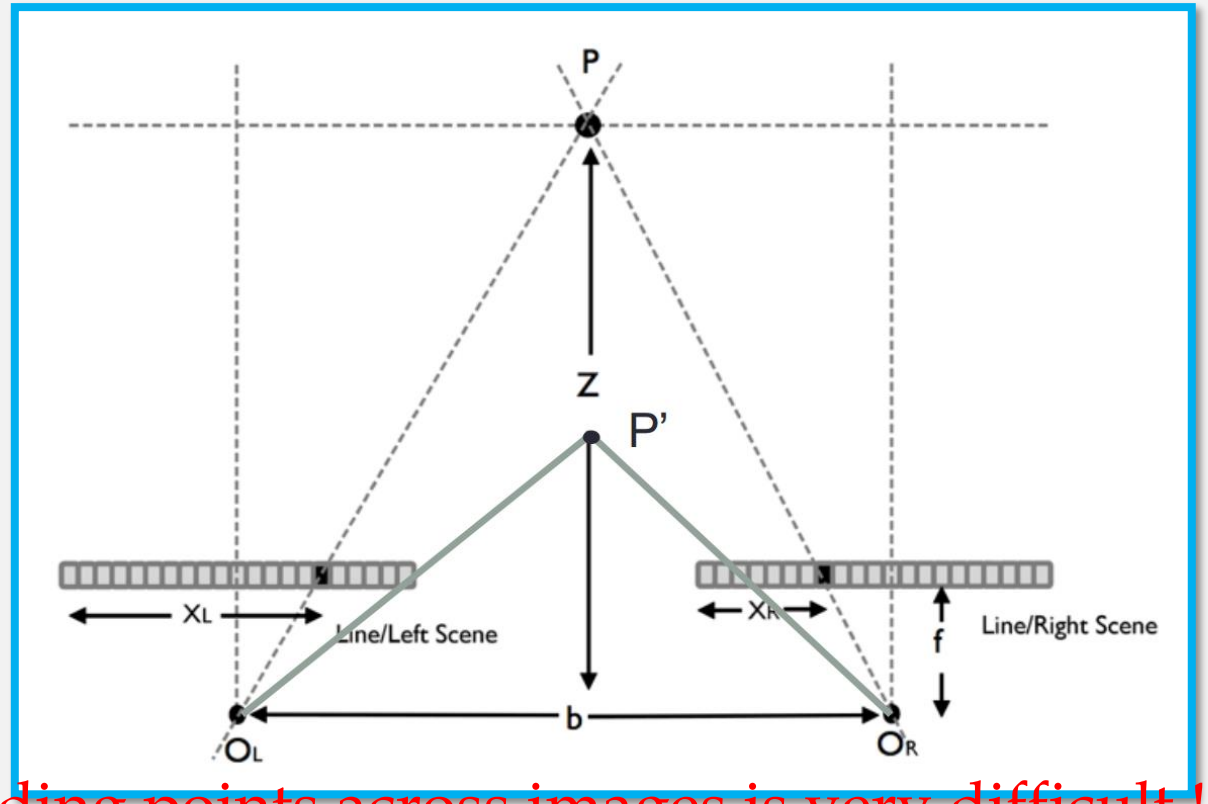
log(N) projections are sufficient to identify N stripes

Advantage: cost and speed
Disadvantage: need controlled conditions & projector calibration

Computer Vision Based Methods (1)

- Depth from stereo:
  - Given 2 images, shift in the x-axis is related to the depth



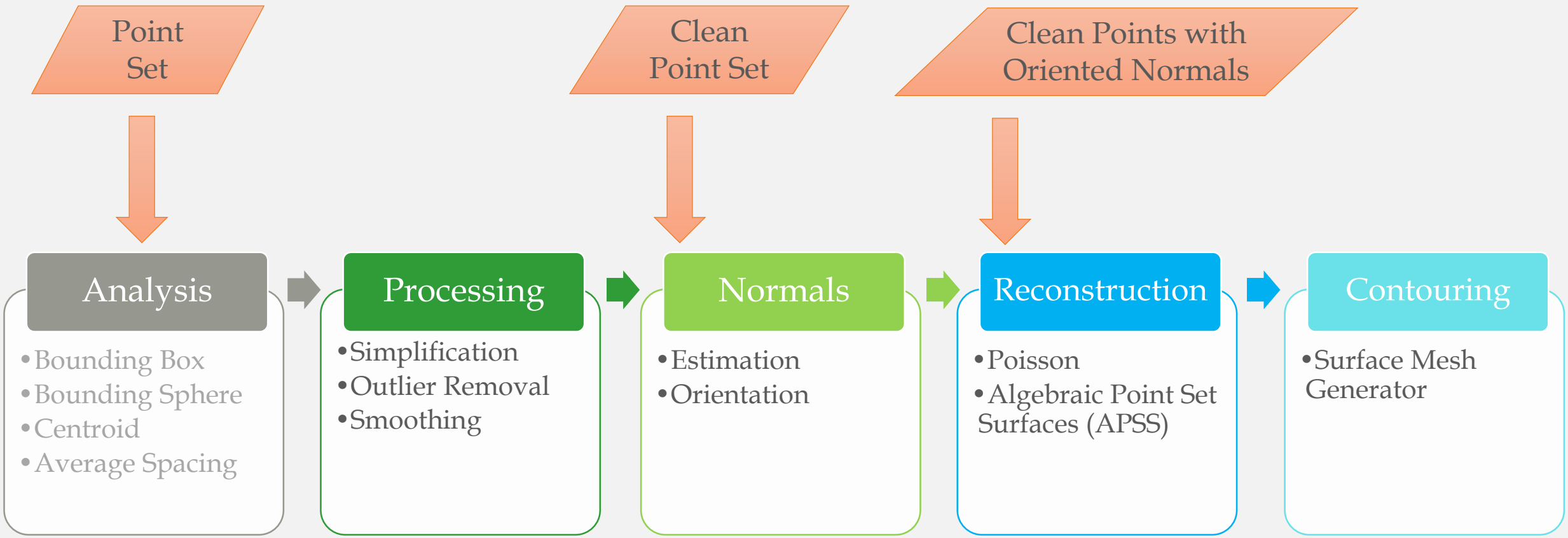


Main challenge: Establishing corresponding points across images is very difficult !

Computer Vision Based Methods (2)

- Depth from blur:
  - Depth can be approximated by detecting how blurry part of the image is (with known focal length)

Entering the Pipeline



Local Surface Analysis

- Cloud of point samples describes underlying (manifold) surface
- We need:
  - fast estimation of tangent plane and curvature

  ➔Principal Component Analysis (PCA) of local neighborhood
  - mechanisms for locally approximating the surface

  ➔Moving Least Square (MLS) approach

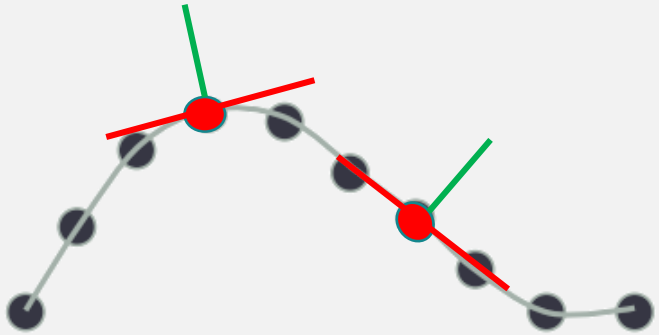
Normals

- Estimation (no orientation)
  - KNN + PCA (fit a plane)
  - KNN + jet fitting (better for noisy data)

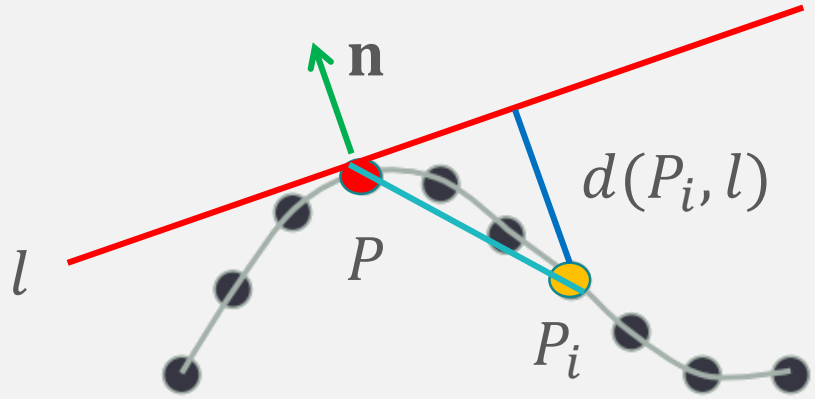
- Orientation
  - KNN + BGL MST (Minimum Spanning Tree) [Hoppe 92]

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.



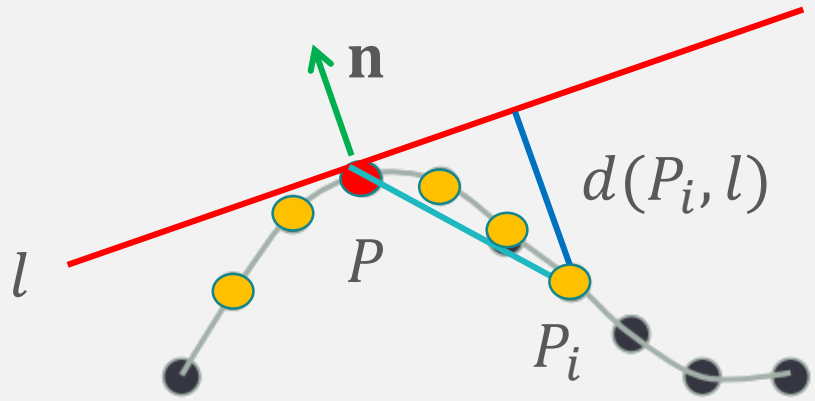
Normal Estimation (Cont.)



- Given line l through P with normal $\mathbf{n}$, for another point P_i :

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

Normal Estimation (Cont.)



- Find the **n** minimizing $\sum_{i=1}^{k} d(P_i, l)^2$ for a set of k points (k nearest neighbors of P)

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

Normal Estimation (Cont.)

- Use Lagrange Multiplier:

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

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

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

$$\Rightarrow C\mathbf{n} = \lambda \mathbf{n}$$

Normal Estimation (Cont.)

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

➔The normal **n** must be an eigenvector of the matrix ***C***.

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

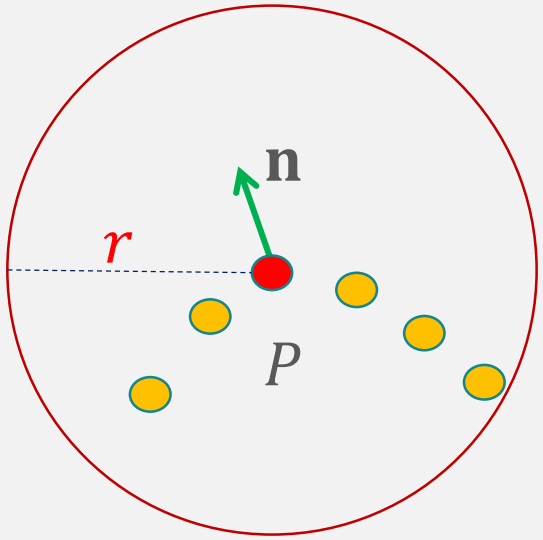
➔ $\mathbf{n}_{opt}$ must be the eigenvector corresponding to the smallest eigenvalue of ***C***.

Normal Estimation (Cont.)

- Can be done by 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** should be the eigenvector corresponding to the smallest eigenvalue of C
- Variant:
  - Use
$$C = \sum_{i=1}^{k} (P_i - \bar{P})(P_i - \bar{P})^T, \qquad \bar{P} = \frac{1}{k}\sum_{i=1}^{k} P_i$$

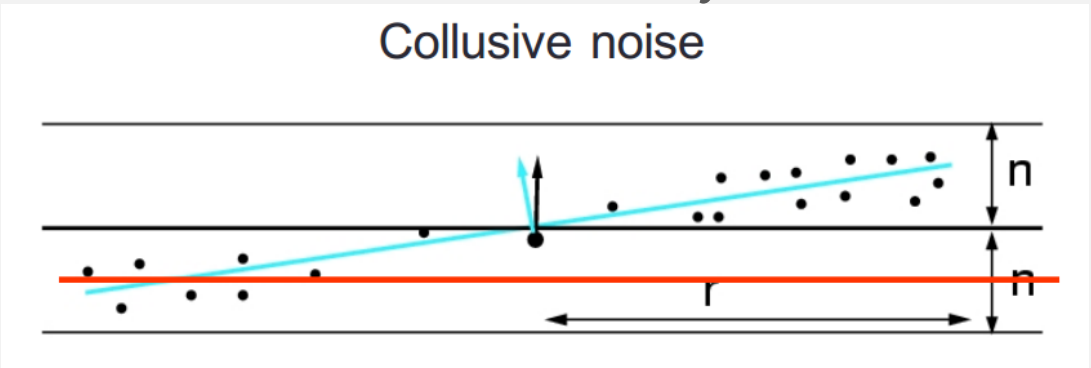
Parameter Selection

- 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?

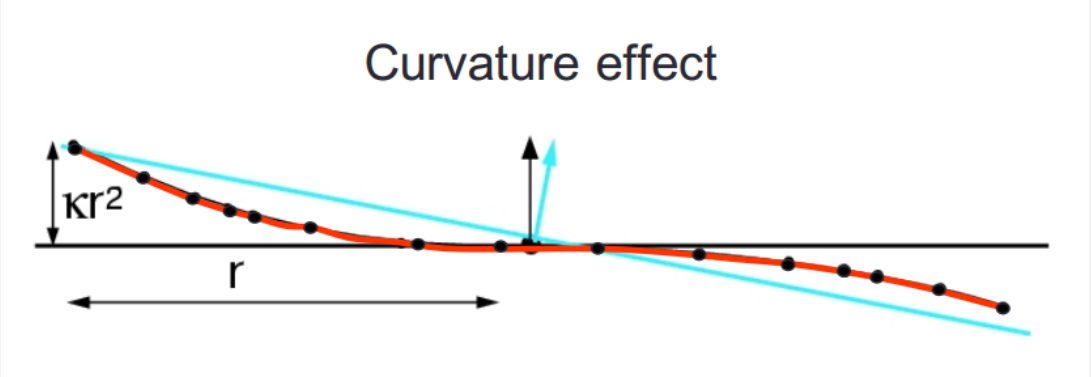


Parameter Selection (Cont.)

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

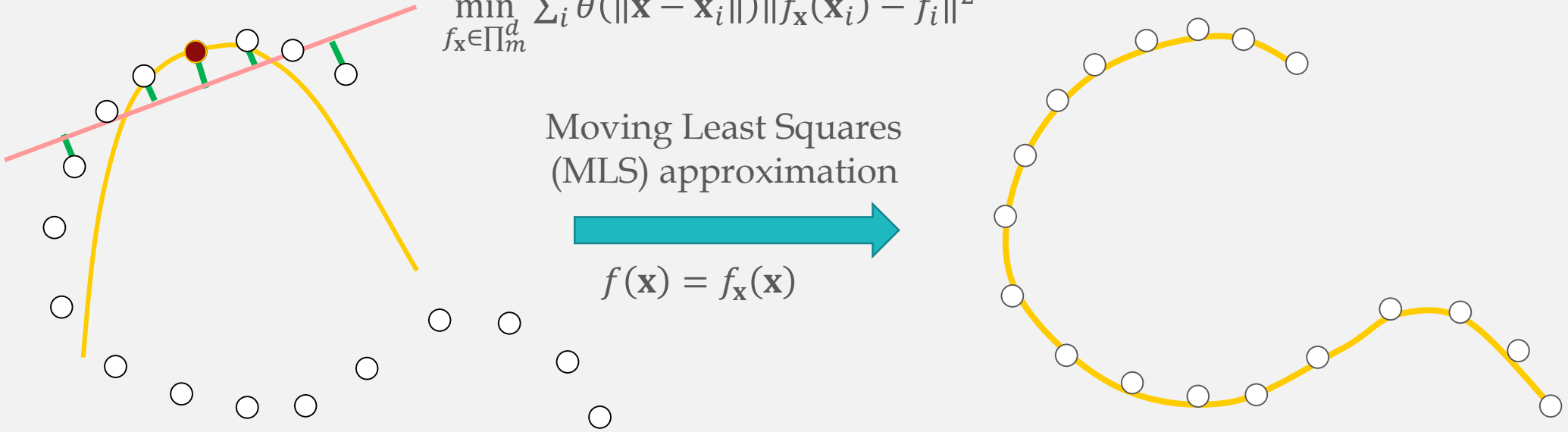


- Due to curvature, large r can lead to estimation bias:



Moving Least Squares (MLS)

- Idea: Locally approximate surface with polynomial
  - Compute the weighted least-squares polynomial function $f_{\mathbf{x}}$



Point Cloud Simplification

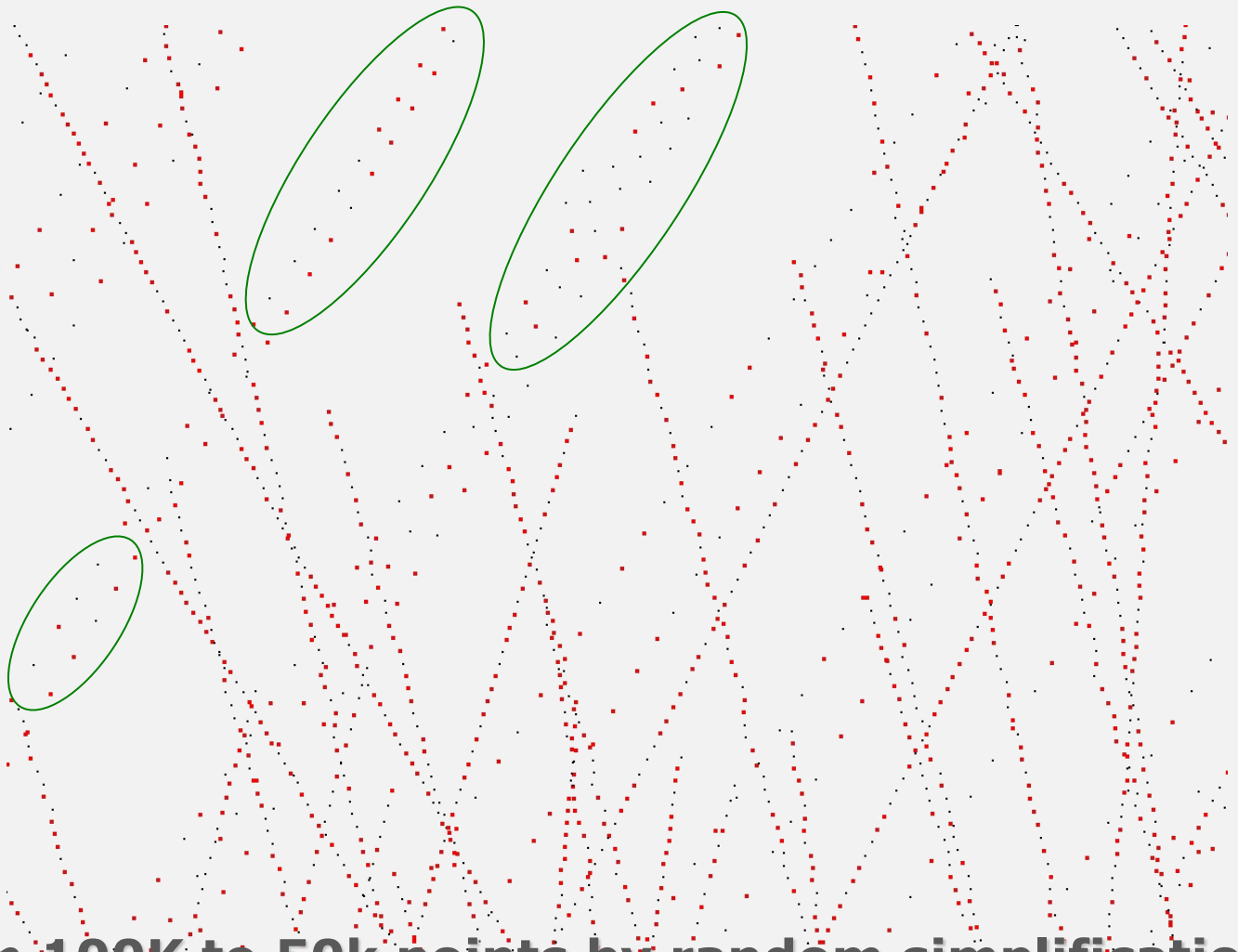
- Point cloud simplification can be used to
  - reduce the complexity of geometric models early in the 3d content creation pipeline
  - create surface hierarchies

- References
  - Mark Pauly *et al.* "Efficient Simplification of Point Sampled Surfaces"
  - Mark Alexa et al. "Point Set Surfaces"
  - Levin D. "Mesh-independent surface interpolation"

Surface Simplification Methods

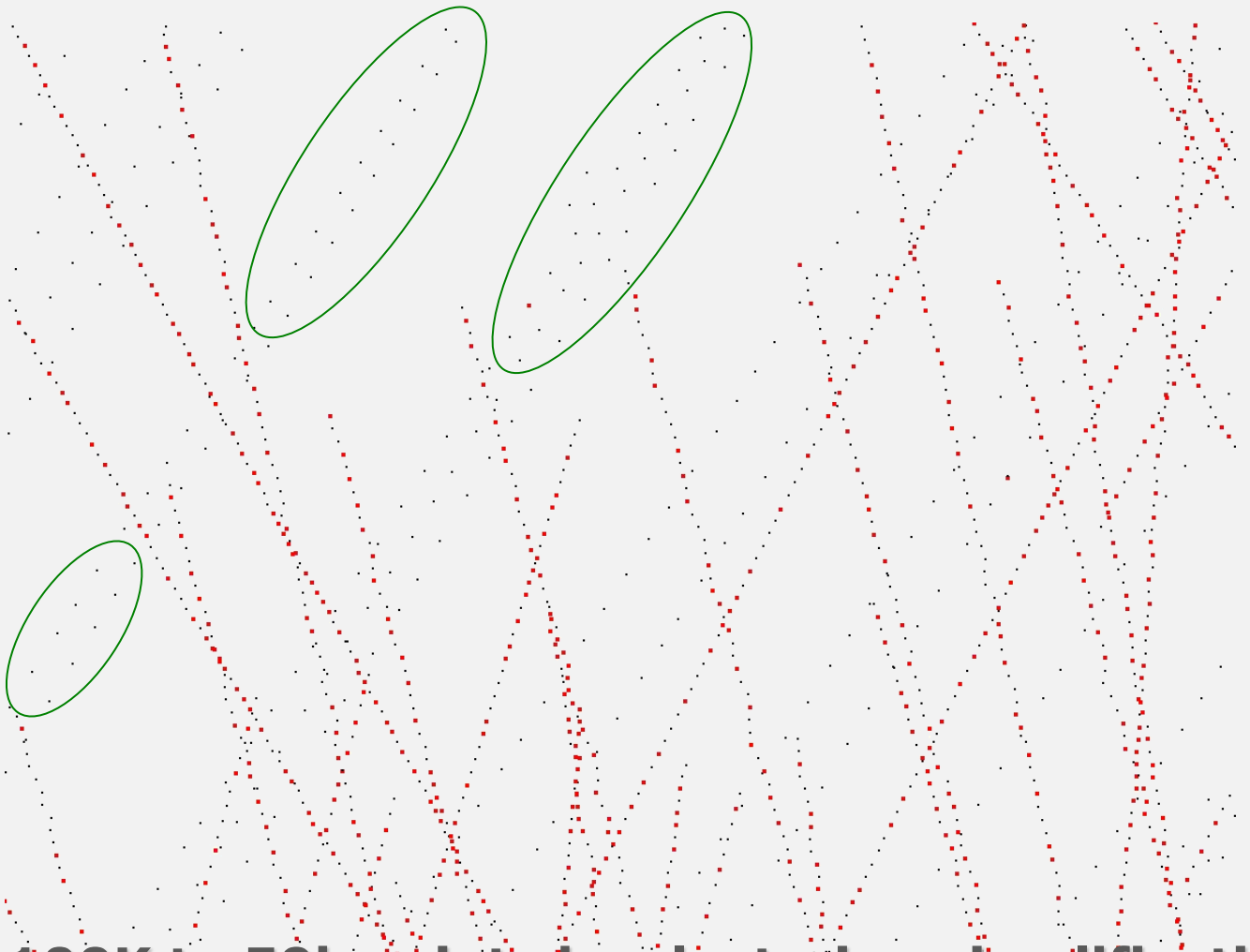
- Random simplification
- Incremental clustering
- Hierarchical clustering
- Iterative simplification
- Particle simulation

Random Simplification Example



from 100K to 50k points by random simplification

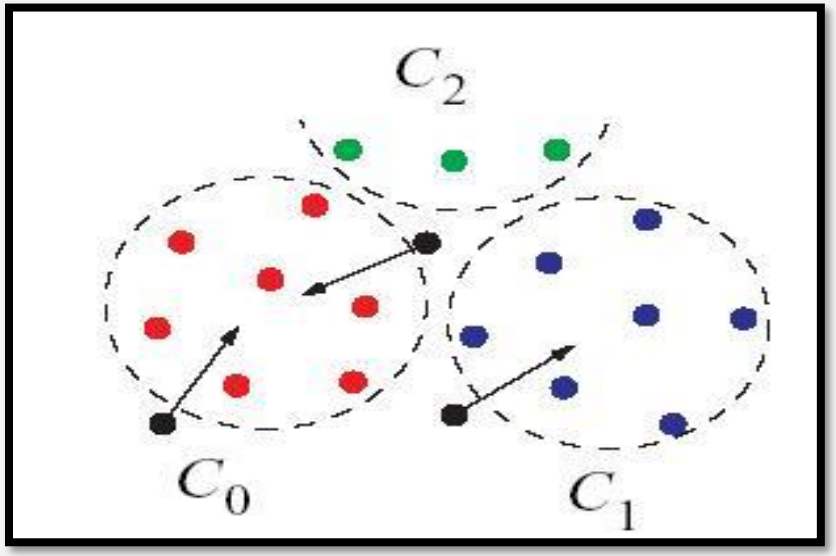
Clustering Simplification Example



from 100K to 50k points by clustering simplification

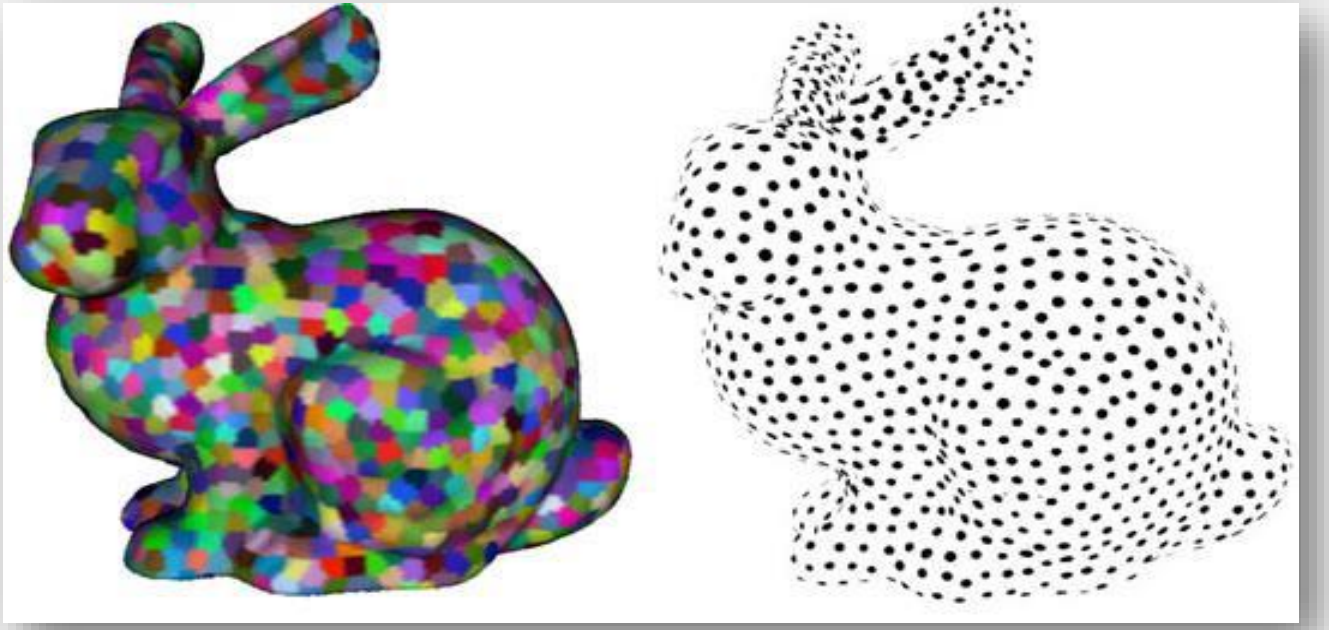
Incremental Clustering

- Clustering by growing regions
  - Start with a random seed point
  - Successively add nearest points to cluster until cluster reaches desired maximum size
- The growth of clusters can also be limited by surface variation and in that way the curvature adaptive clustering is achieved
- Incremental growth leads to some fragmentation.
  - Stray samples need to be added to closest clusters at the end of the run.



Example of Incremental Clustering

- Each cluster is replaced by its centroid



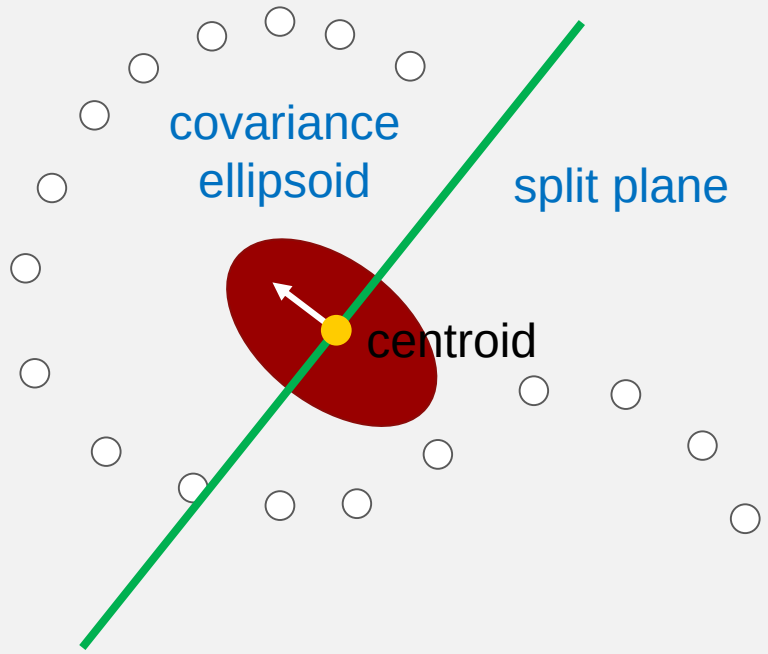
Original model: 34,384 points

Simplified model: 1,000 pts

Hierarchical Clustering

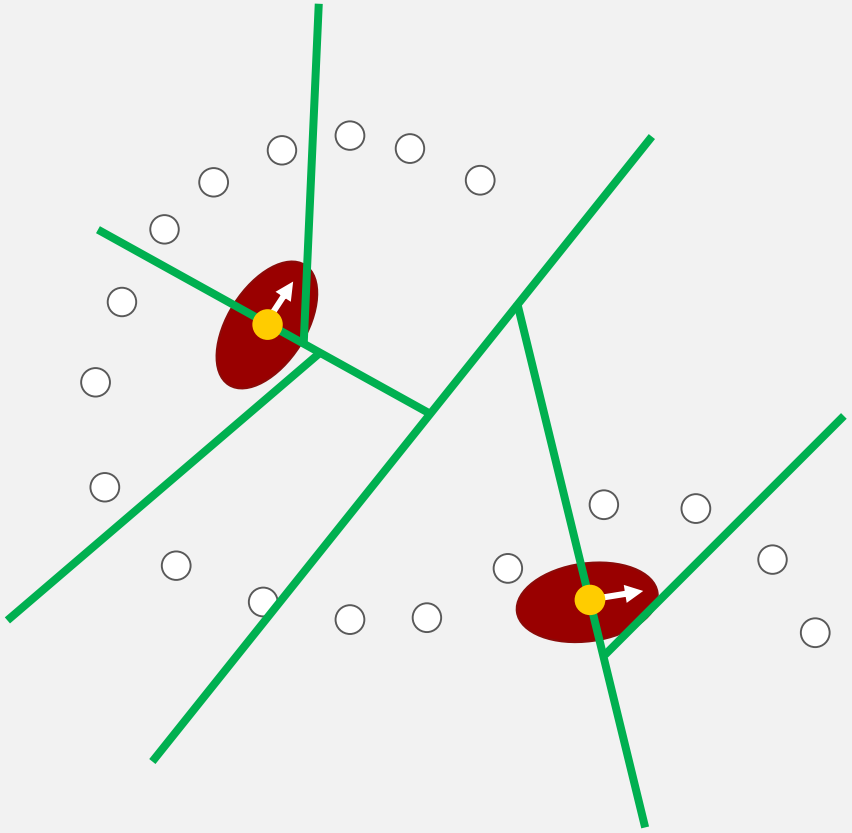
- Top-down approach using binary space partition
- Recursively split the point cloud if
  - Size is larger than a user-specified threshold or
  - Surface variation is above maximum threshold
- Split plane defined by centroid and axis of greatest variation
- Replace clusters by centroid

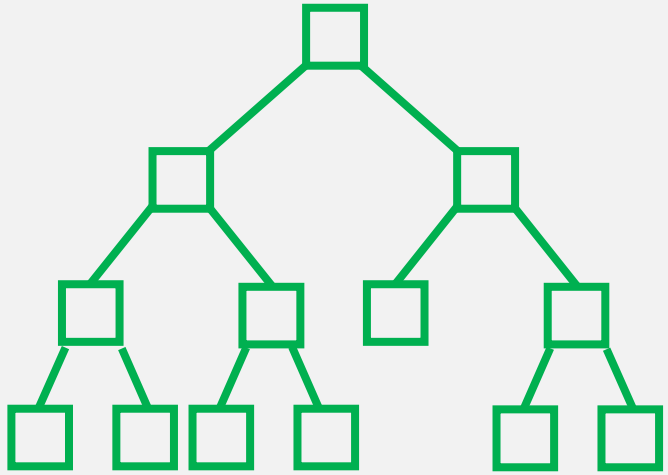
2D Example of Hierarchical Clustering



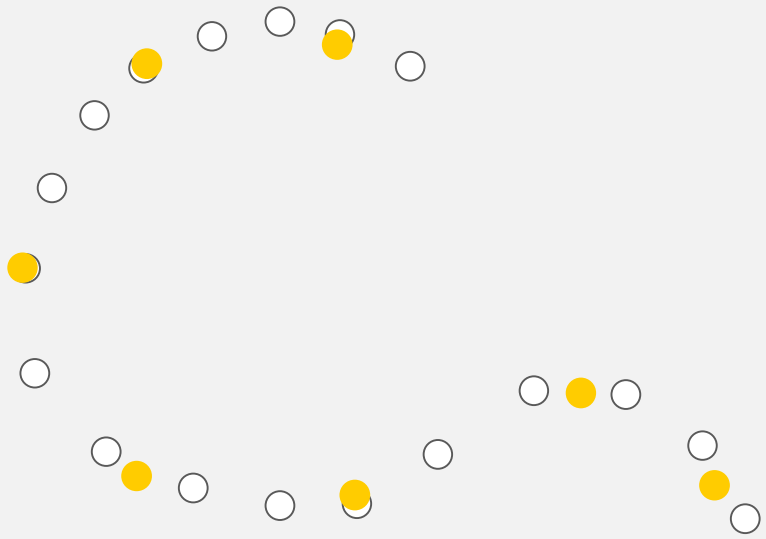
root

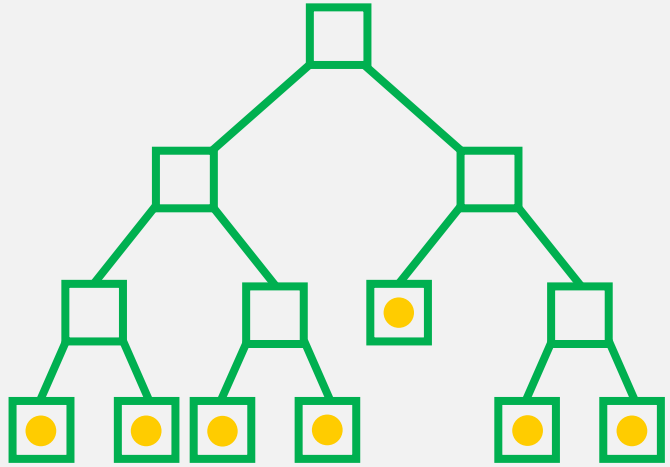
2D Example of Hierarchical Clustering



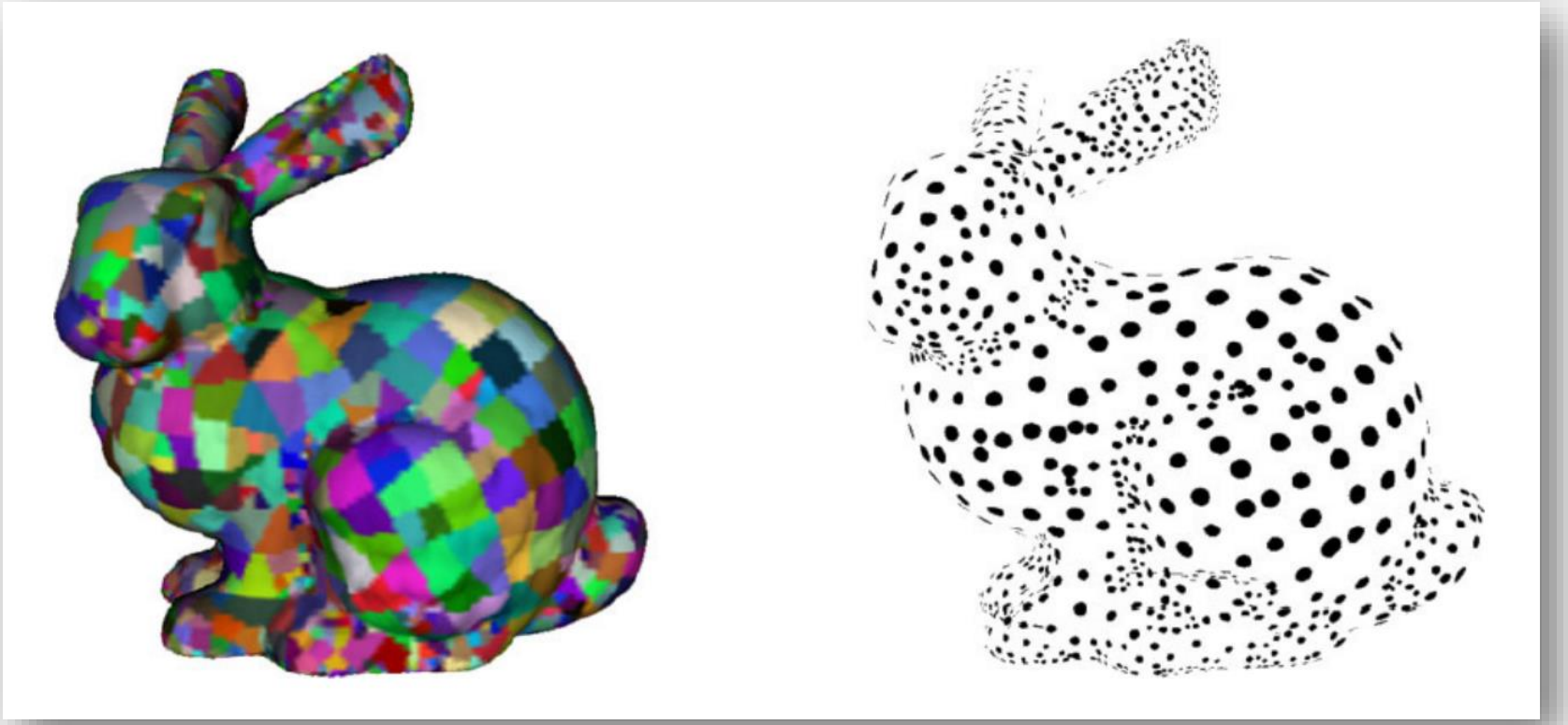


2D Example of Hierarchical Clustering





Example of Hierarchical Clustering



Original model with color-coded clusters
(34,384 points)

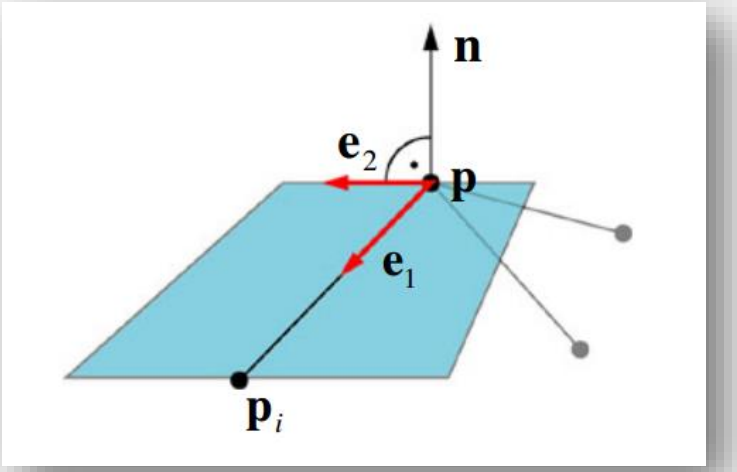
Simplified model
(1,000 points)

Iterative Simplification

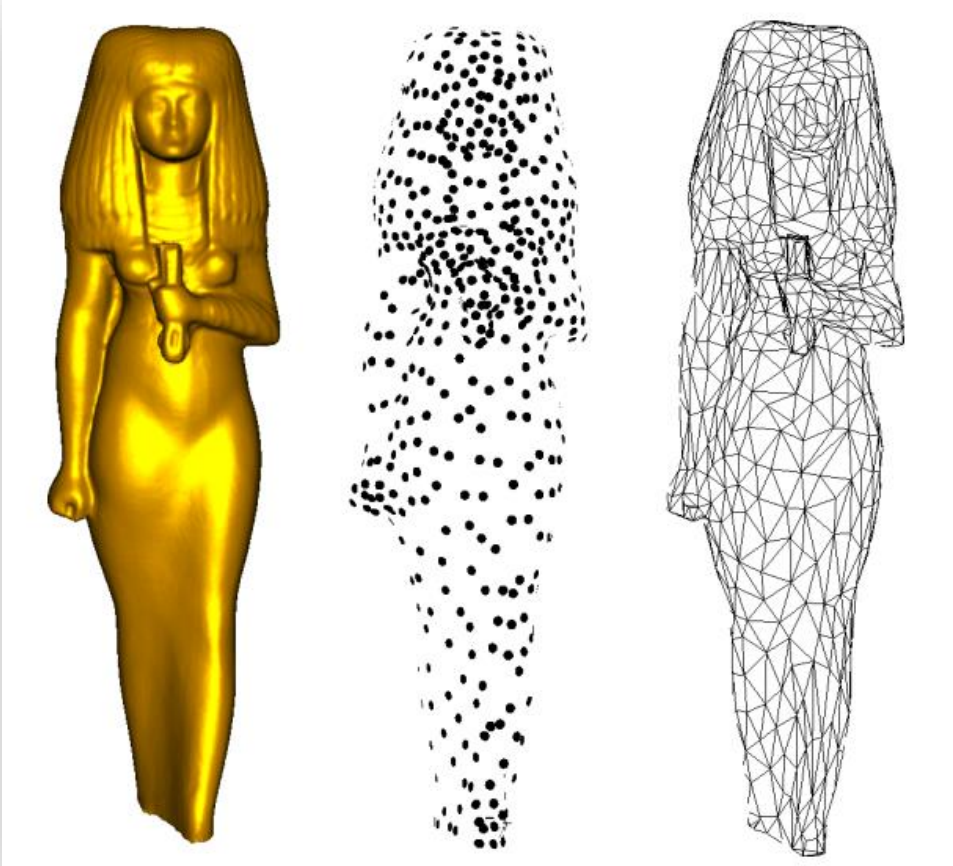
- Iteratively contracts point pairs
  - Each contraction reduces the number of points by one
- Contractions are arranged in priority queue according to quadric error metric
- Quadric measures cost of contraction and determines optimal position for contracted sample
- Equivalent to QSlim except for definition of approximating planes

Iterative Simplification

- Quadric measures the squared distance to a set of planes defined over edges of neighborhood
  - Plane spanned by vectors $e_1 = p_i - p$ and $e_2 = e_1 \times n$

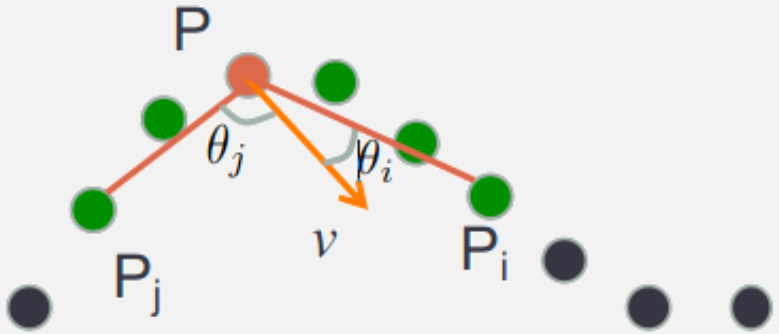


Example of Iterative Simplification



Outlier Removal

- Goal : Remove points that do not lie close to a surface



The covariance matrix $C = \sum_{i=1}^{k}(P_i - P)(P_i - P)^T$

For any vector v, the Releigh quotient:

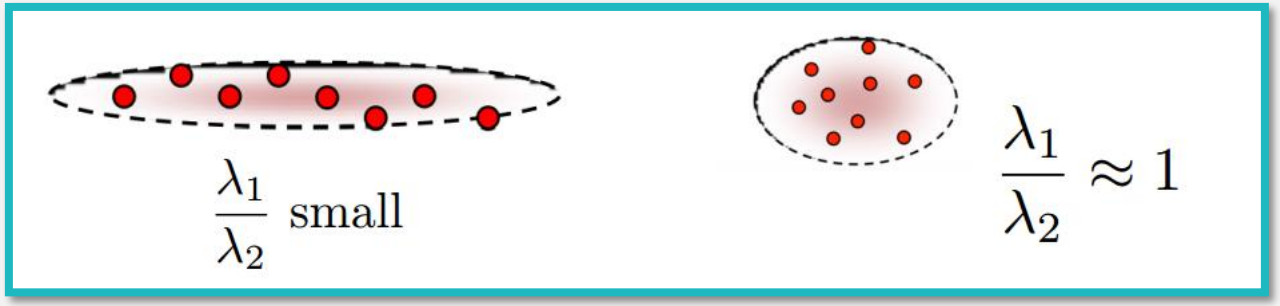
$$\frac{v^T C v}{v^T v} = \sum_{i=1}^{k}\left((P_i - P)^T v\right)^2 = \sum_{i=1}^{k}(\|P_i - P\|\cos\theta_i)^2 \quad if\ \|v\| = 1$$

Outlier Removal (Cont.)

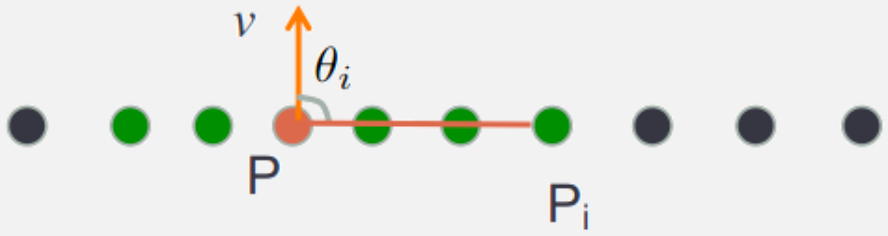
- If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{min}(C) = 0$ and $\lambda_{max}(C)$ is large

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

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



Outlier Removal (Cont.)

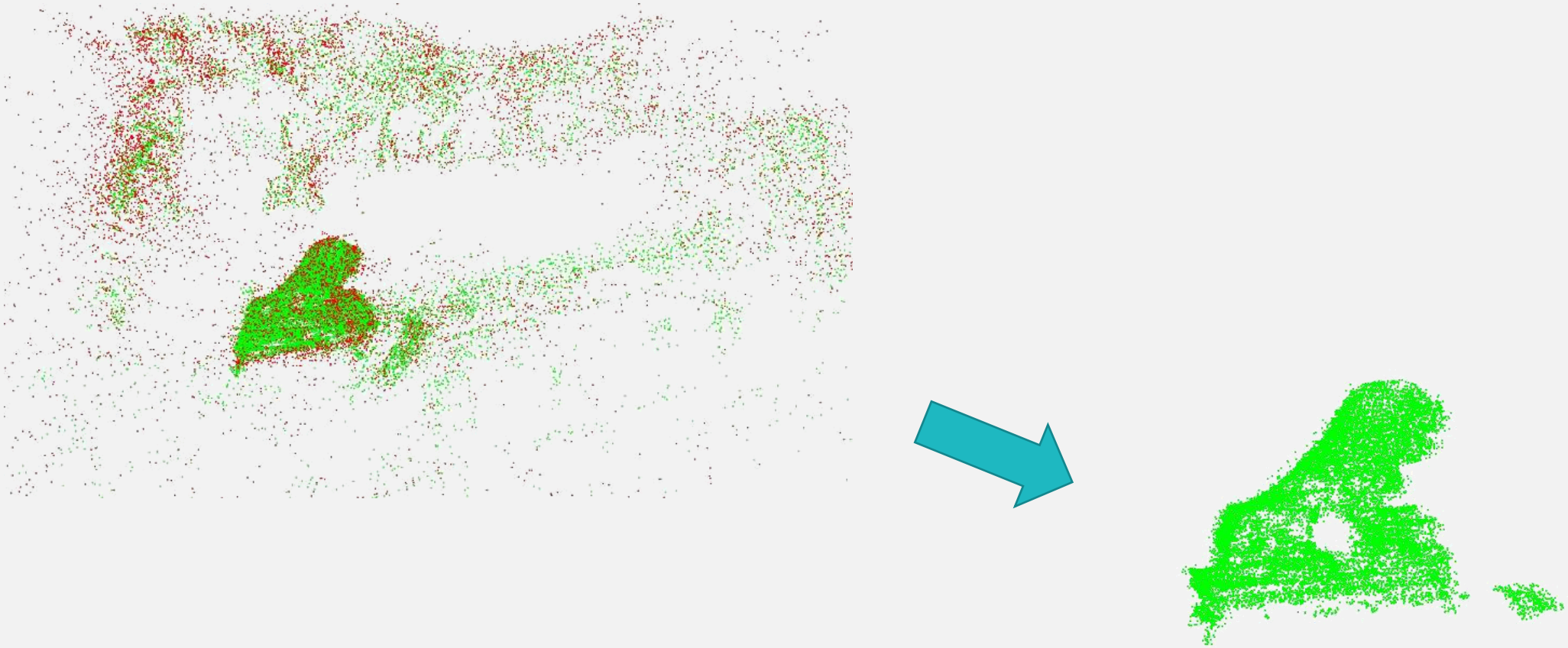


- If all the points are on a line, then $\min_v \frac{v^T C v}{v^T v} = \lambda_{min}(C) = 0$ and $\lambda_{max}(C)$ is large

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

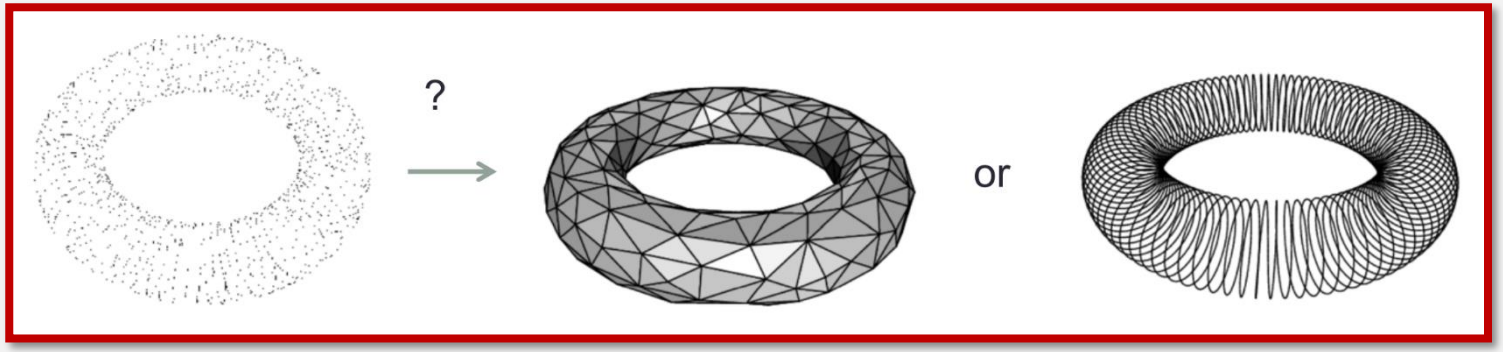
- If points are scattered randomly, then $\lambda_{max}(C) \approx \lambda_{min}(C)$
- Thus, can remove points where $\frac{\lambda_1}{\lambda_2} > \epsilon$ for some threshold
- In 3D we expect two zero eigenvalues, so use $\frac{\lambda_2}{\lambda_3} > \epsilon$ for for some threshold

Outlier Removal Example

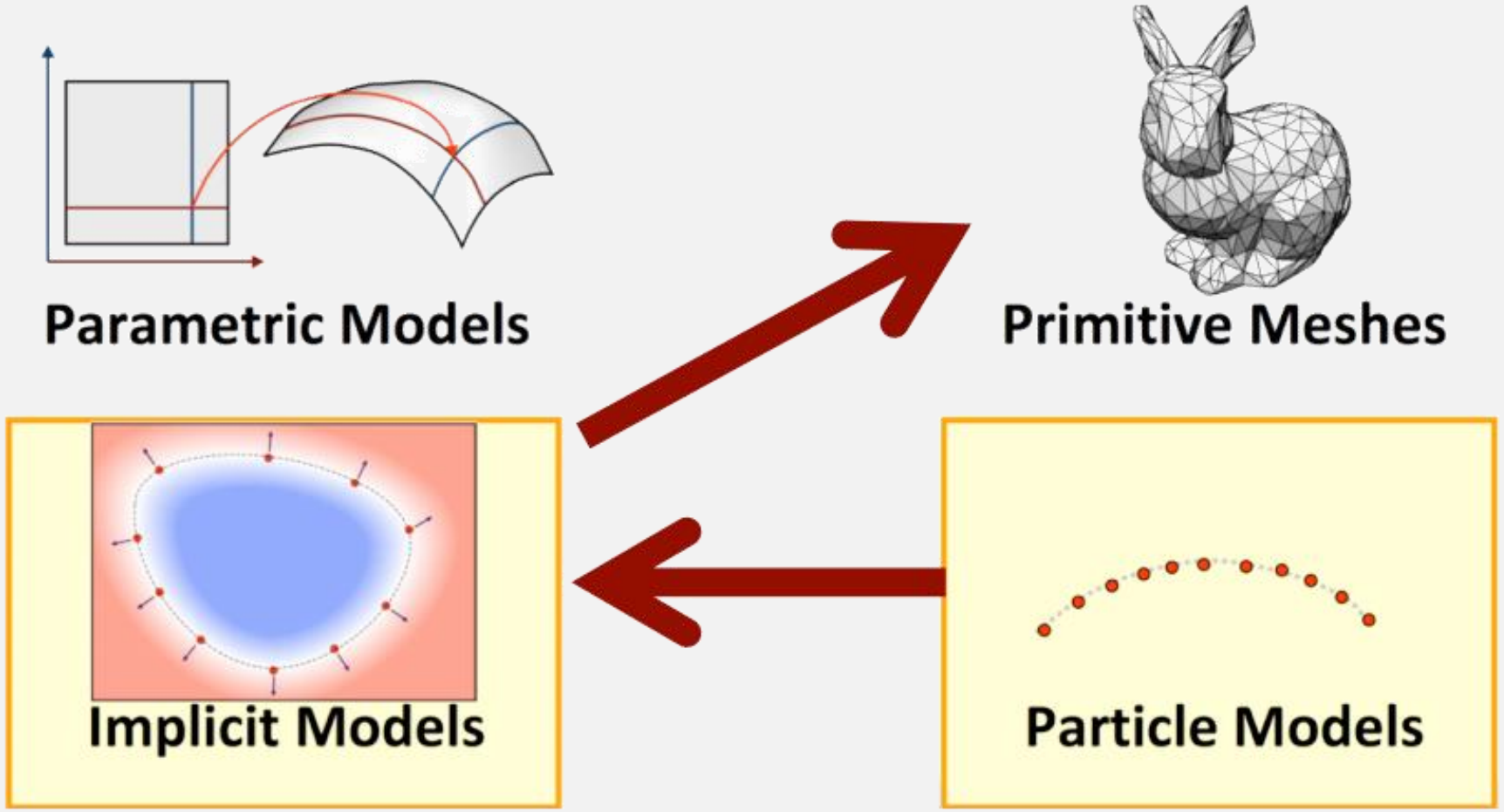


Surface Reconstruction

- Goal: Construct a polygonal (e.g. triangle mesh) representation of the point cloud
- Main Problem: Data is **unstructured**
  - E.g. the points are not ordered
- An inherently **ill-posed** (a.k.a. difficult) problem

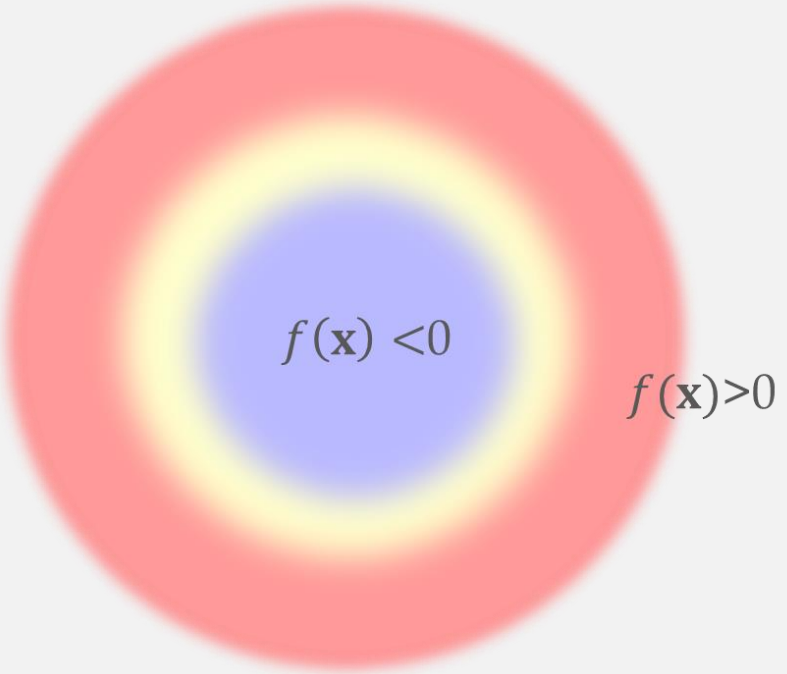


Reconstruction Through Implicit Models



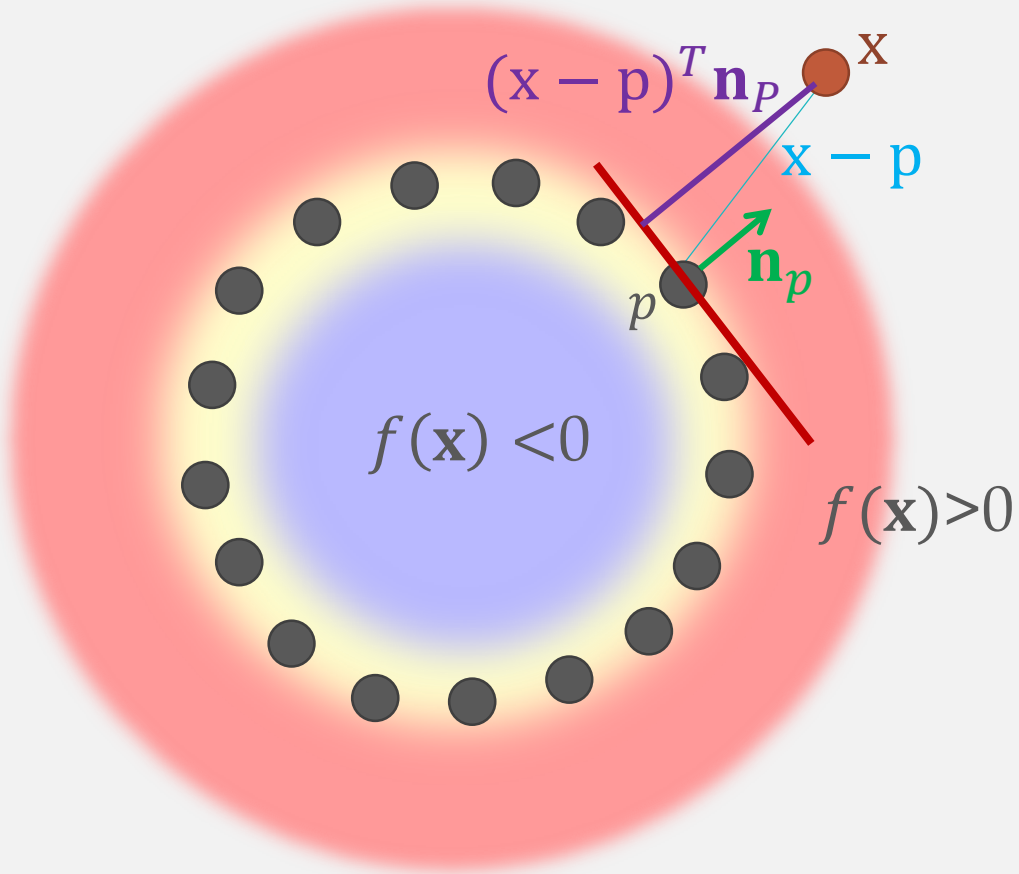
Implicit Surfaces

- Given a function f(**x**), the surface is defined as $\{\mathbf{x}, \text{s. t. } f(\mathbf{x}) = 0\}$



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

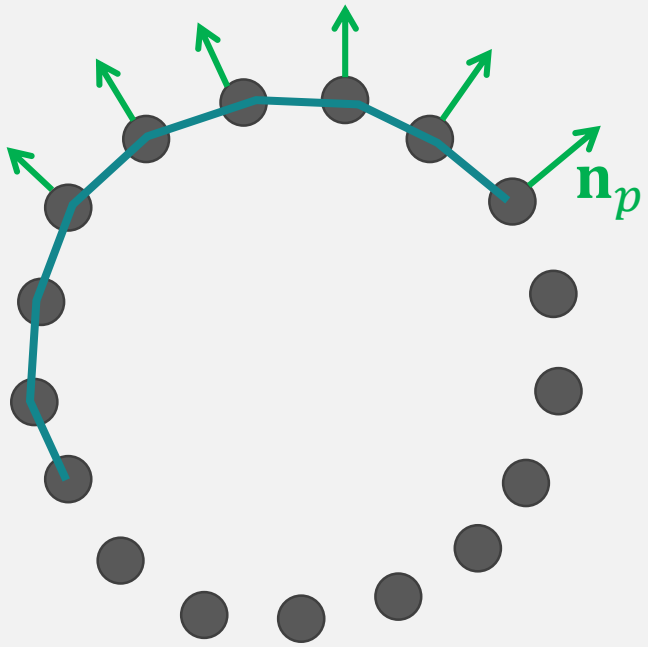
Implicit Surfaces (Cont.)



Set $f(\mathbf{x}) = (\mathrm{x}-\mathrm{p})^T\mathbf{n}_P$

= Signed distance to the tangent plane

Implicit Surfaces (Cont.)



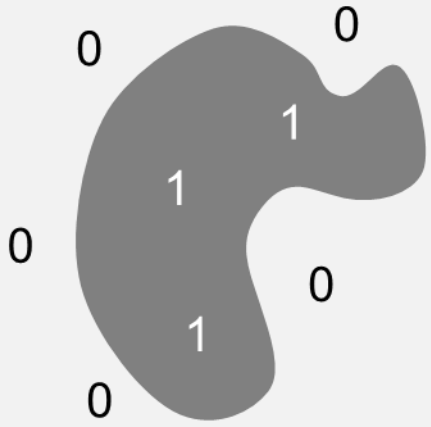
$$\text{Set } f(\mathbf{x}) = (\mathrm{x} - \mathrm{p})^T \mathbf{n}_P$$

- Need consistently oriented normals , but PCA only gives normals **up to orientation**.
- In general, difficult problem, but can try to locally connect points and fix orientations.

Poisson Surface Reconstruction

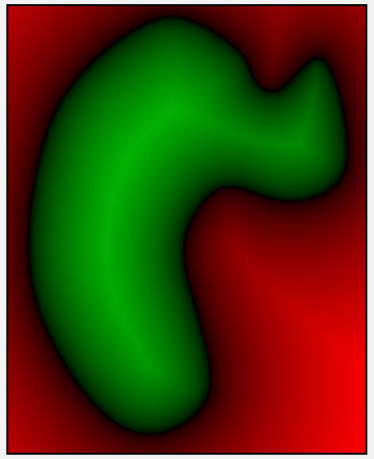
- Poisson surface reconstruction [Kazhdan-Bolitho-Hoppe, SGP 2006]
  - Reconstruct the surface of the model by solving for the **indicator function** of the shape.
  - Isosurface extracted by CGAL surface mesh generator

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



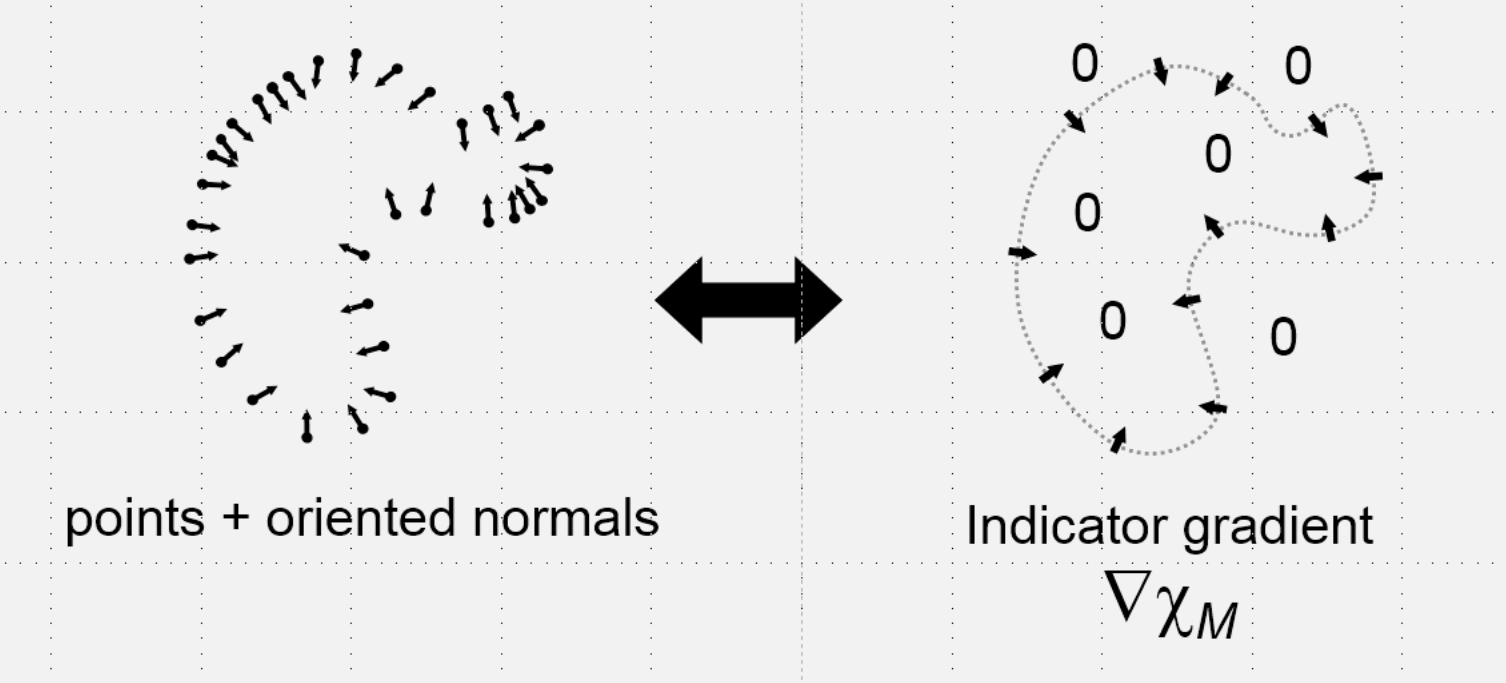
Indicator function

χ_M



Poisson Surface Reconstruction (Cont.)

- There is a relationship between the normal field and gradient of indicator function



Poisson Surface Reconstruction (Cont.)

- Represent the points by a vector field $\vec{V}$
- Find the function χ whose gradient best approximates $\vec{V}$:

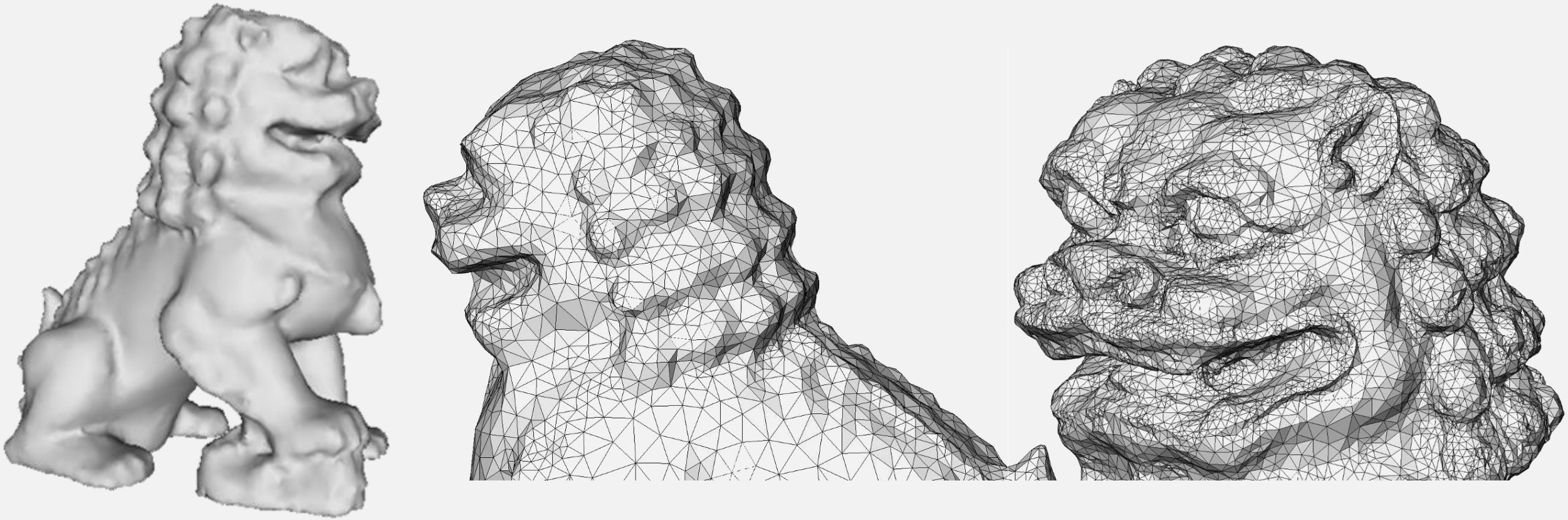
$$\min_{\chi} \left\| \nabla\chi - \vec{V} \right\|$$

- Applying the **divergence operator**, we can transform this into a **Poisson** problem:

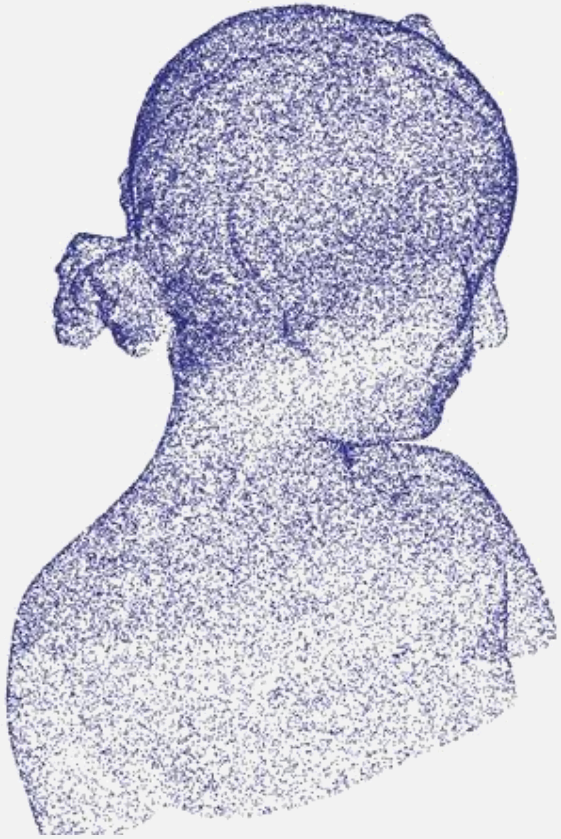
$$\nabla \cdot (\nabla\chi) = \nabla \cdot \vec{V} \quad \Leftrightarrow \quad \Delta\chi = \nabla \cdot \vec{V}$$

- We solve for the Poisson equation onto the vertices of a (refined) 3D Delaunay triangulation [TAUCS linear solver]

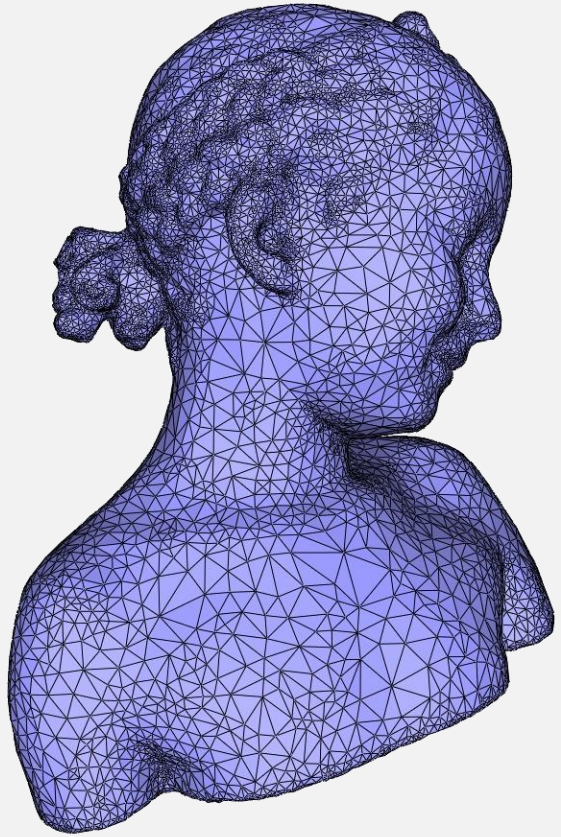
Poisson Surface Reconstruction Example



Poisson Surface Reconstruction Example

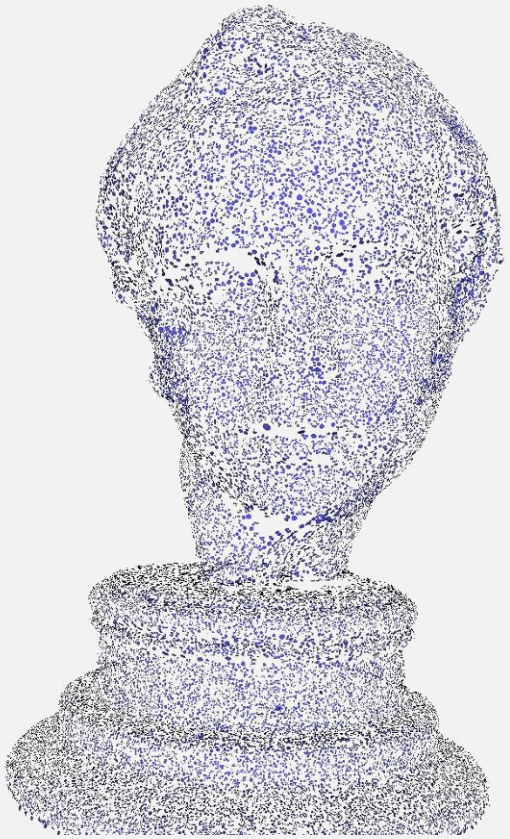


120K points sampled on child statue (Minolta laser scanner)

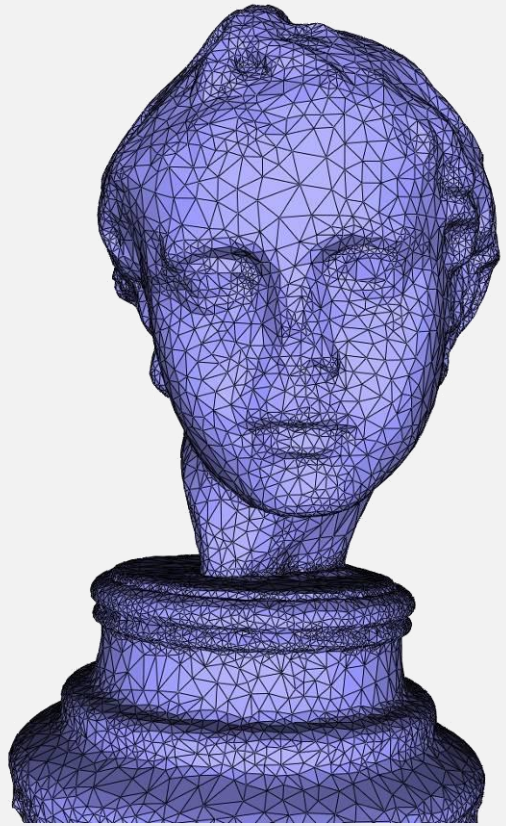


Reconstructed surface

Poisson Surface Reconstruction Example

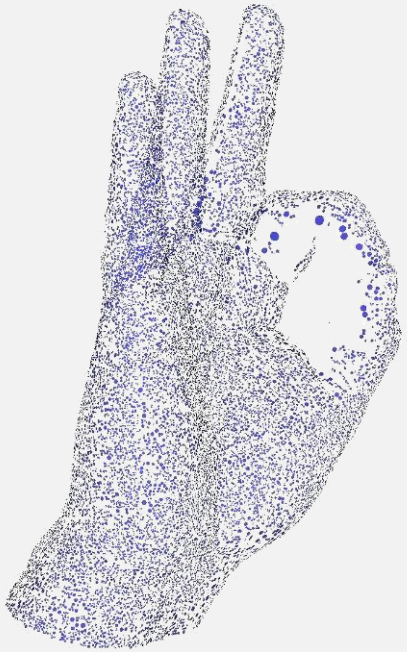


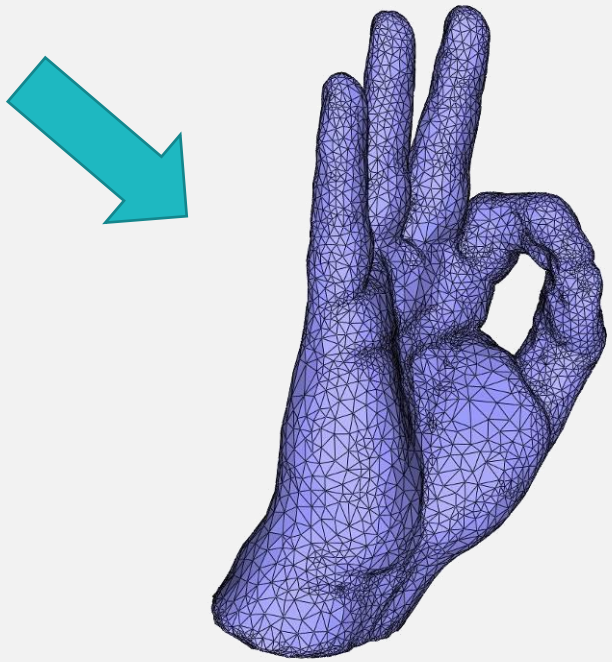
120K points sampled on a statue (Minolta laser scanner)

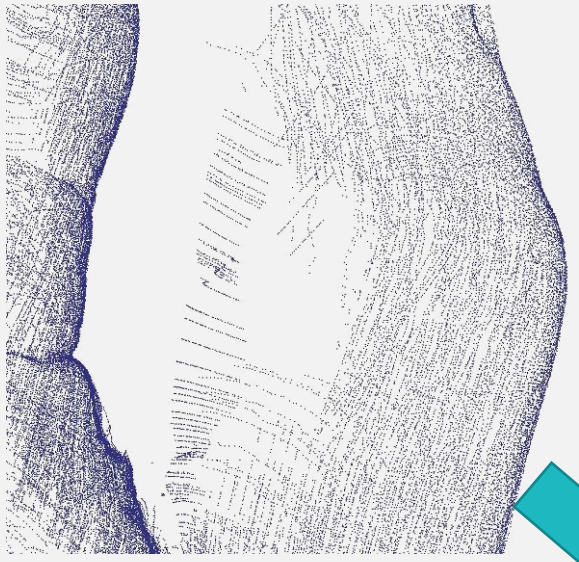


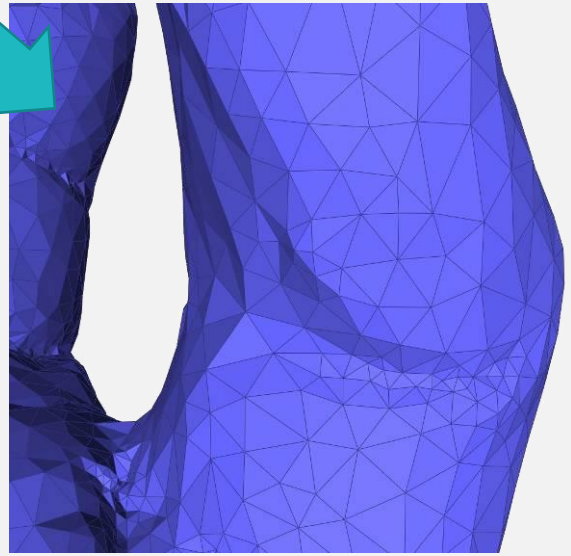
Reconstructed surface

Poisson Surface Reconstruction Example

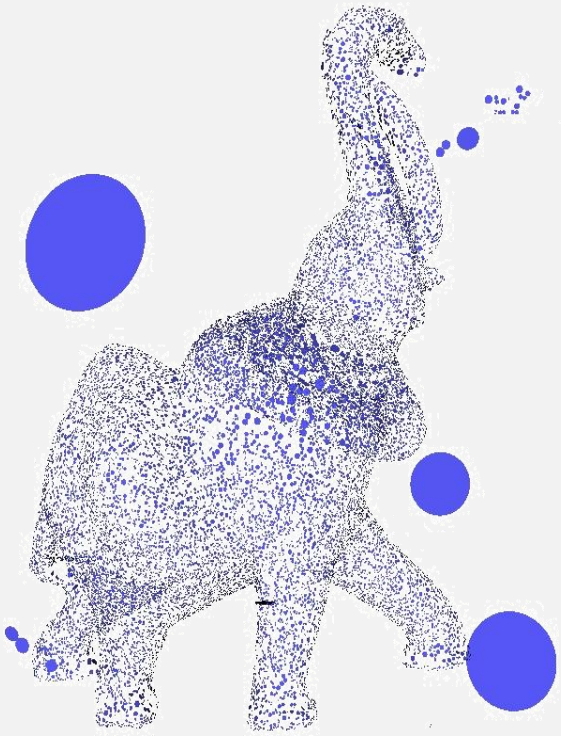


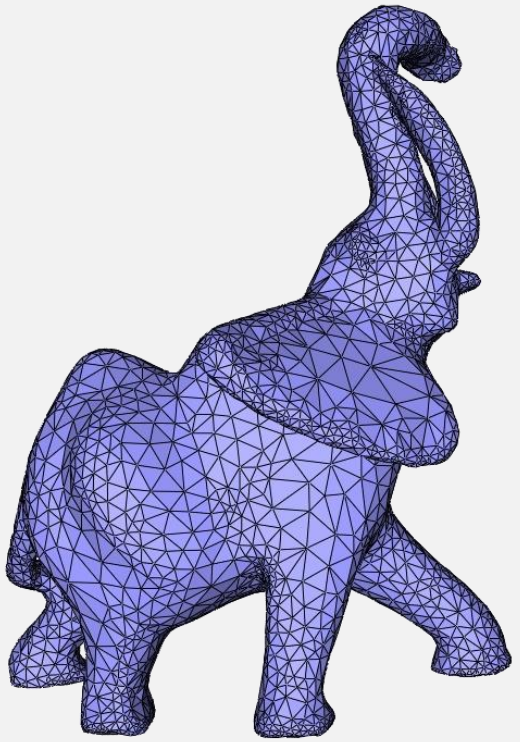






Poisson Surface Reconstruction Example

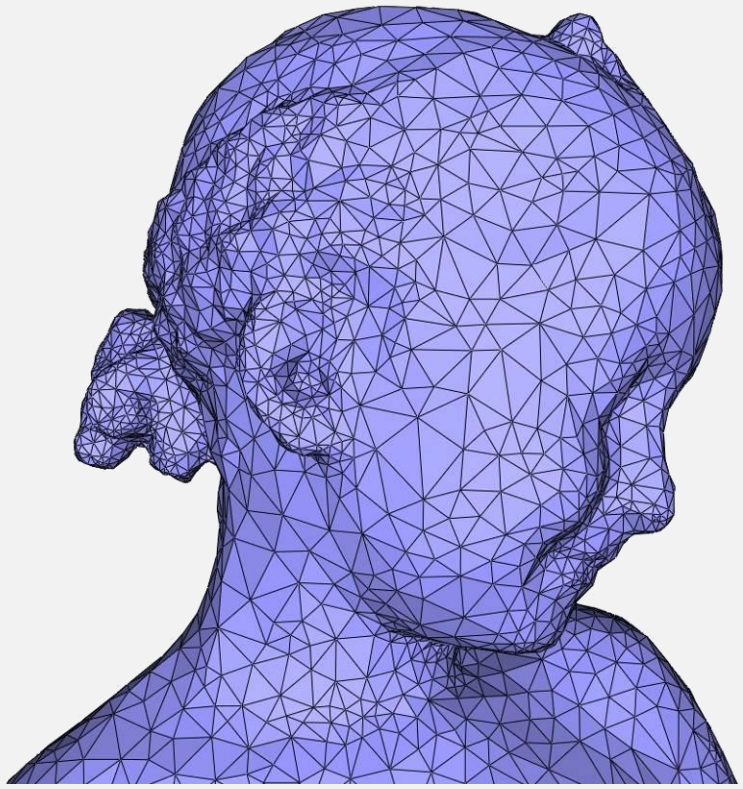


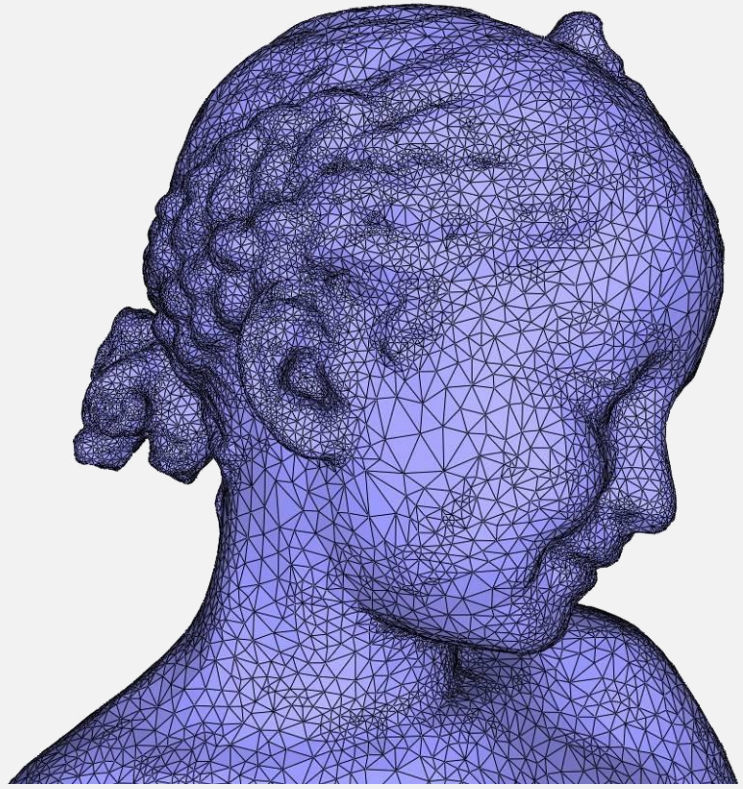


Points with Outliers

Reconstructed surface

Poisson Surface Reconstruction Example

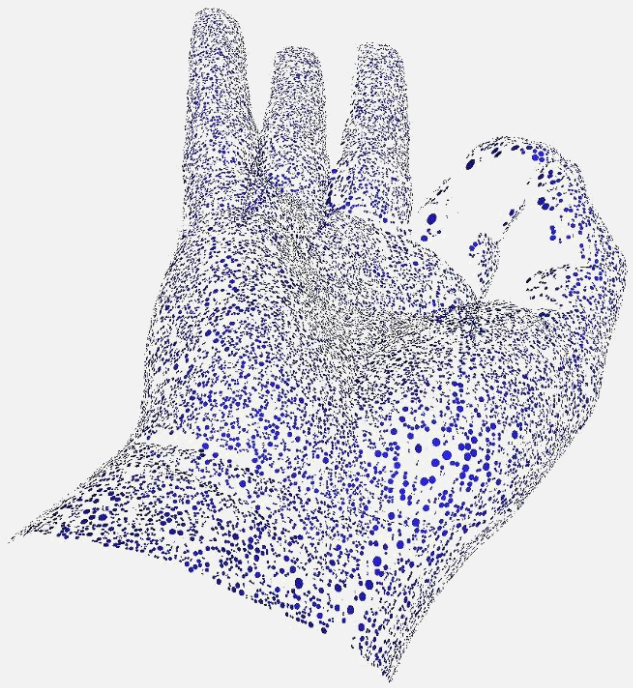




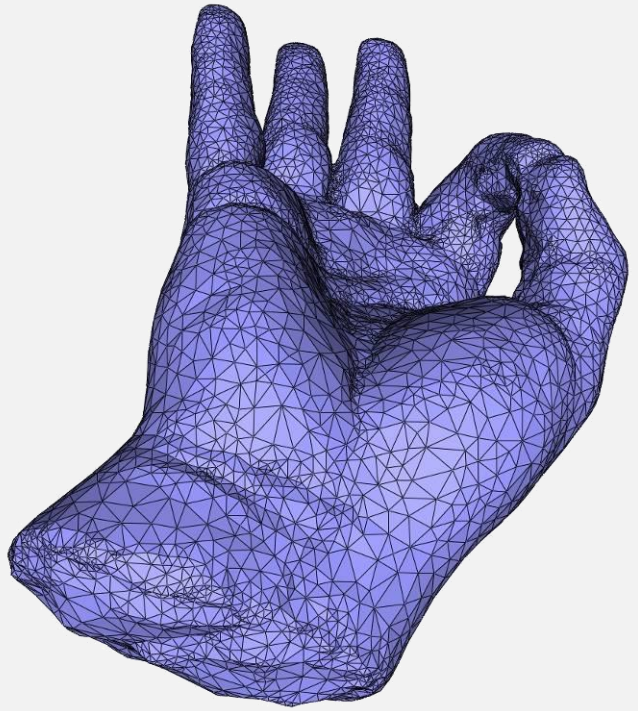
Bimba 120K reconstructed with distance = 0.25*average spacing

Bimba 120K reconstructed with distance = 0.15*average spacing

Poisson Surface Reconstruction Example

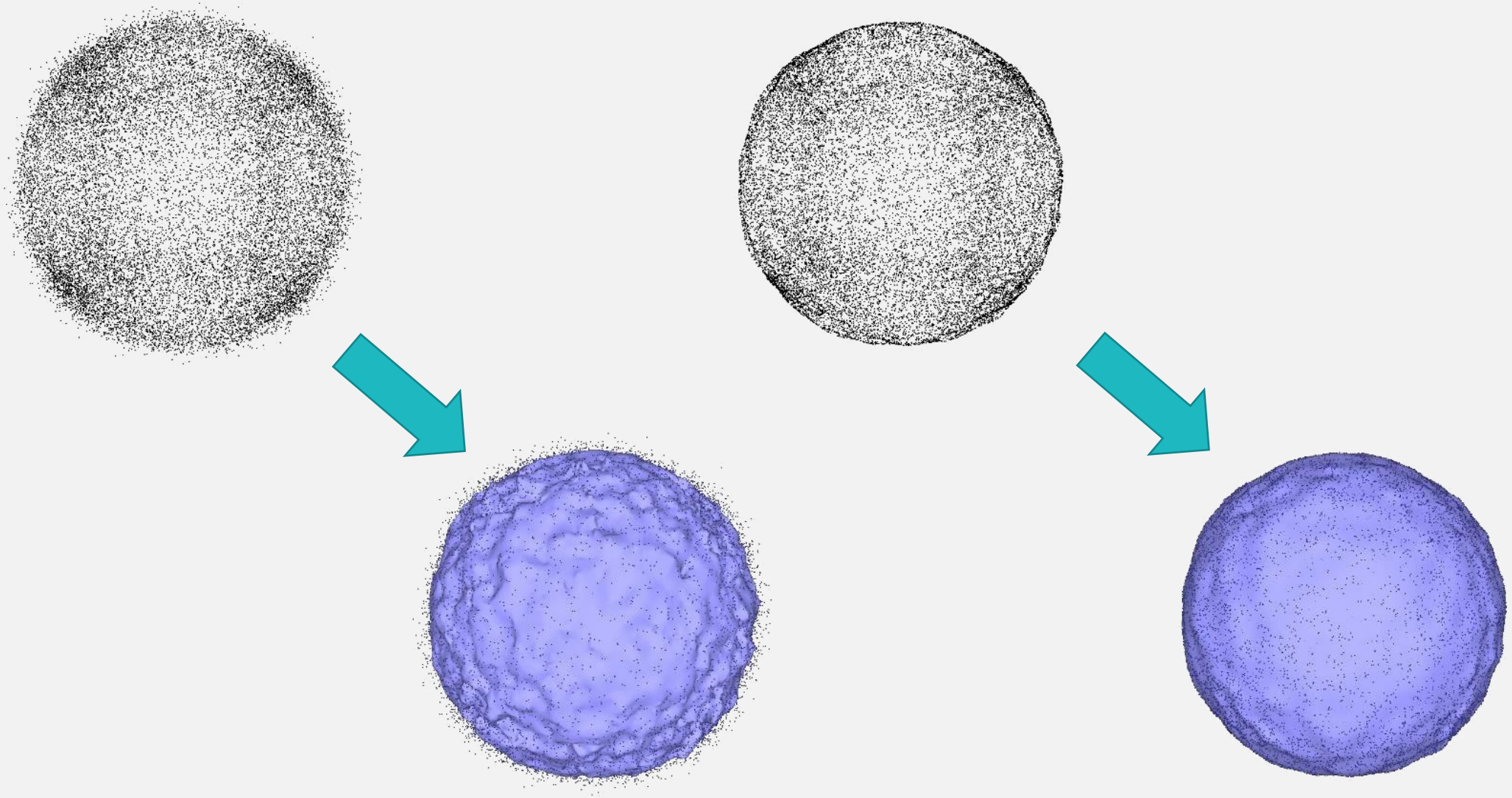


65K points sampled on a hand with no data at the wrist base (Kreon laser scanner)

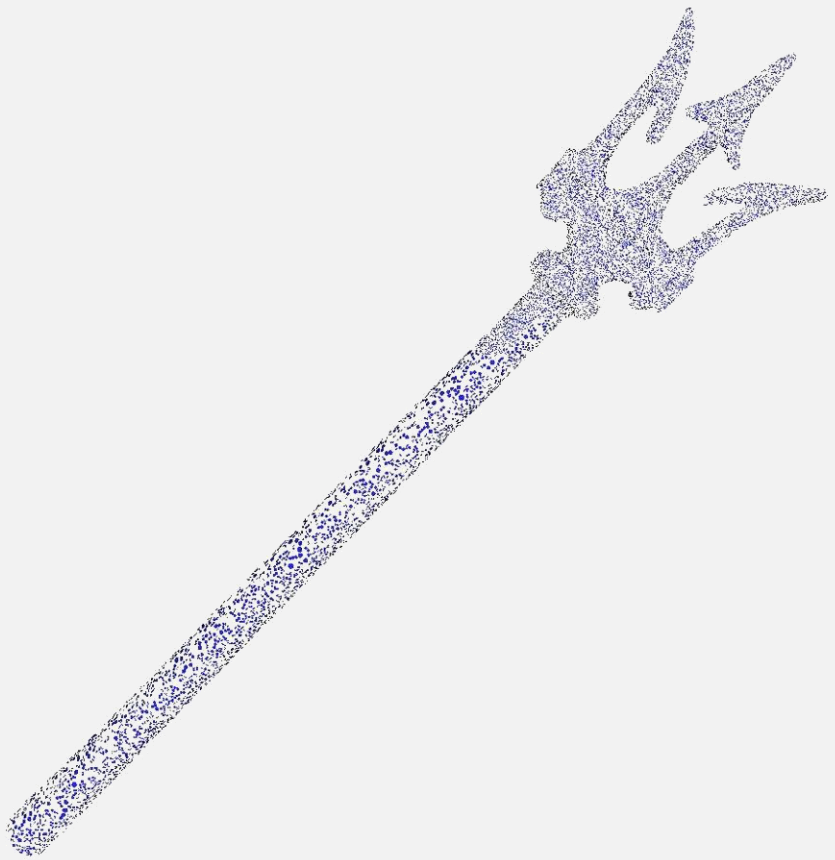


Reconstructed surface

Poisson Surface Reconstruction Example



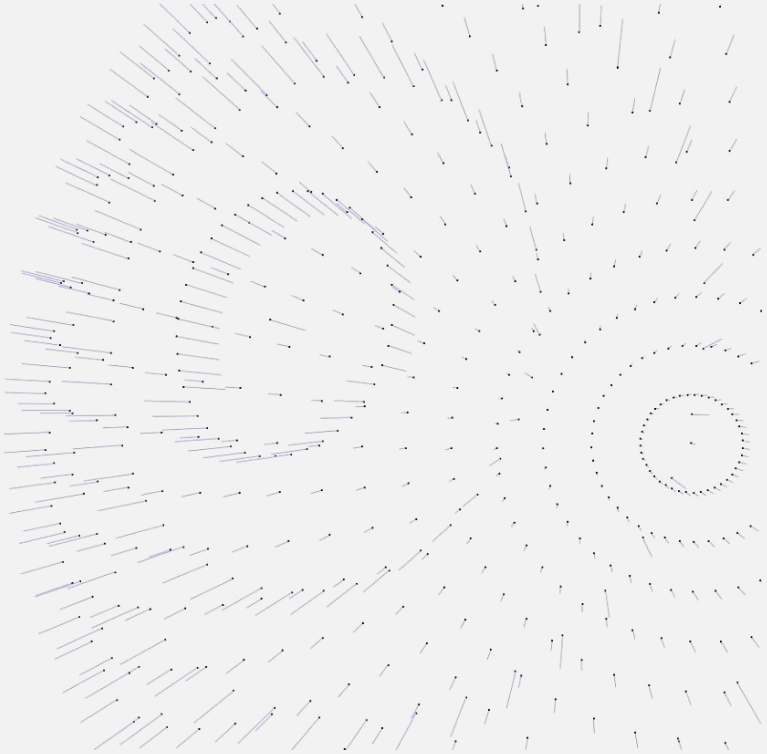
Poisson Surface Reconstruction Example



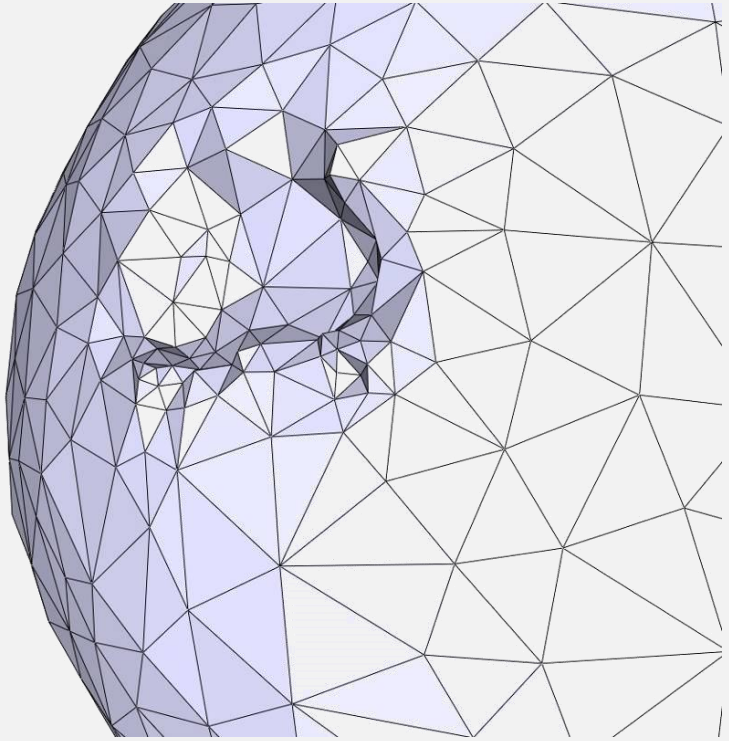
50K points sampled on Neptune trident

Point set simplified to 1K then reconstructed

Poisson Surface Reconstruction Example



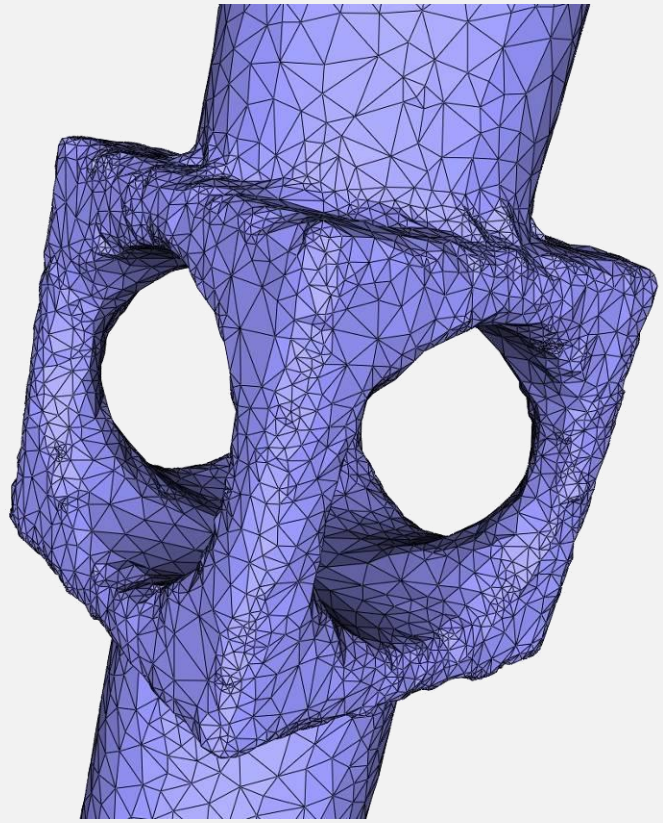
Points sampled on a sphere with flipped normals



Reconstructed surface

Poisson Surface Reconstruction Example

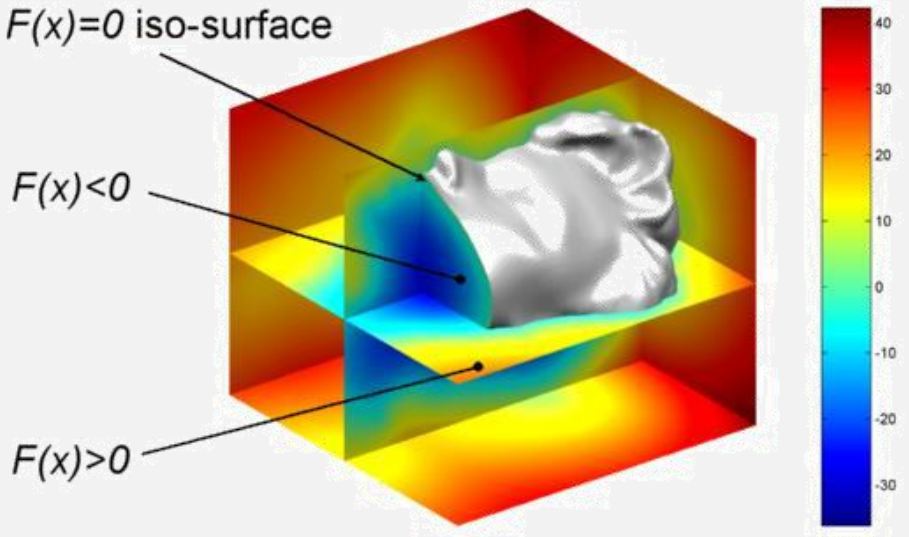
4K points sampled on a mechanical piece with sharp edges



Reconstructed surface

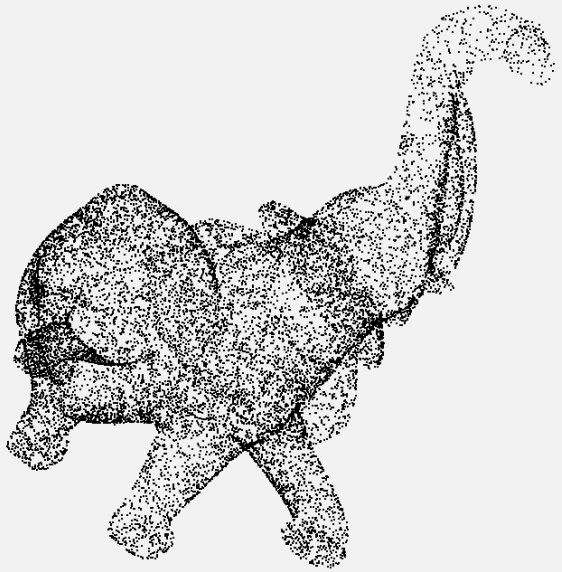
APSS Surface Reconstruction

- Algebraic point set surfaces (APSS) [Guennebaud, SIGGRAPH 2007]
  - Evaluates an implicit function on the fly (~signed distance function)
  - Isosurface extracted by CGAL surface mesh generator
  - Based on **Moving Least Squares fitting** on algebraic spheres

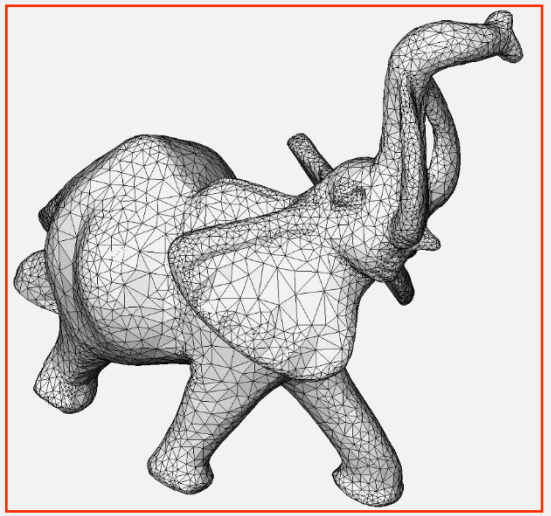


APSS Surface Reconstruction (Cont.)

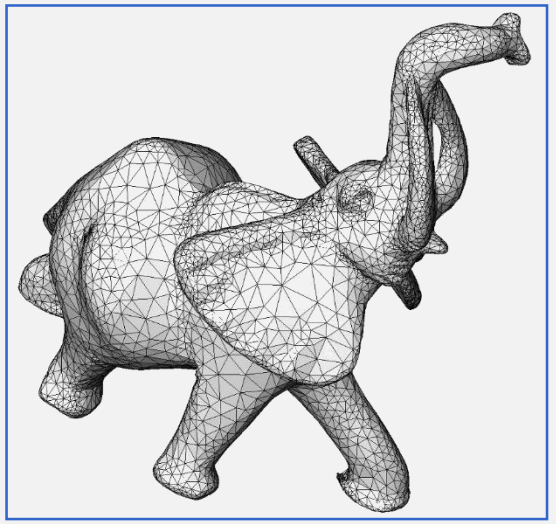
Reconstruction Example



17K points sampled on an elephant (Minolta laser scanner)



reconstructed surface using Poisson

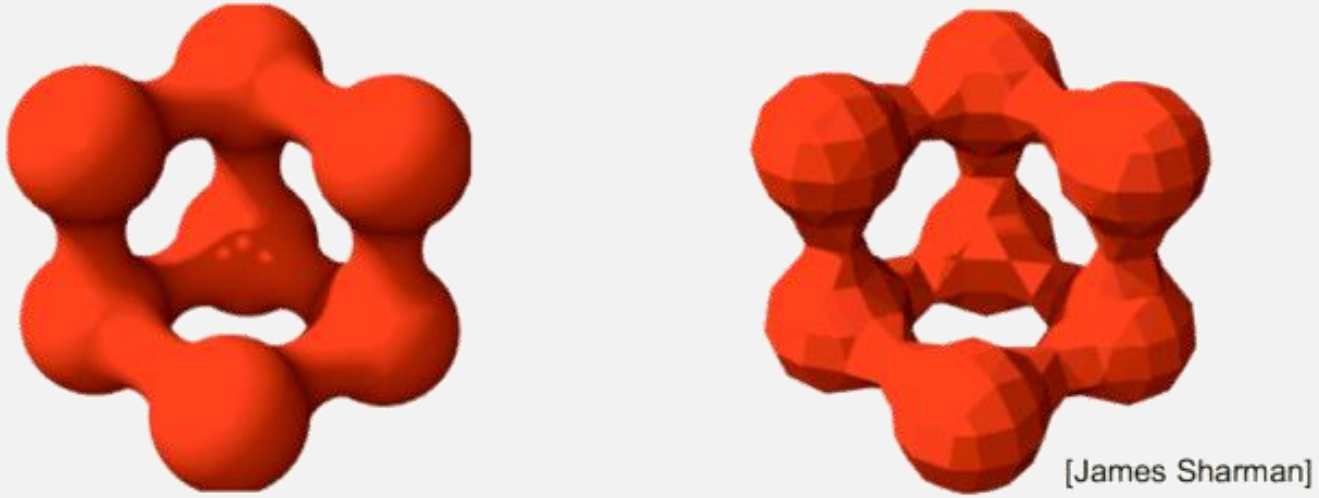


reconstructed surface using APSS (in progress)

Marching Cubes

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

  create a triangle mesh that approximates the surface.

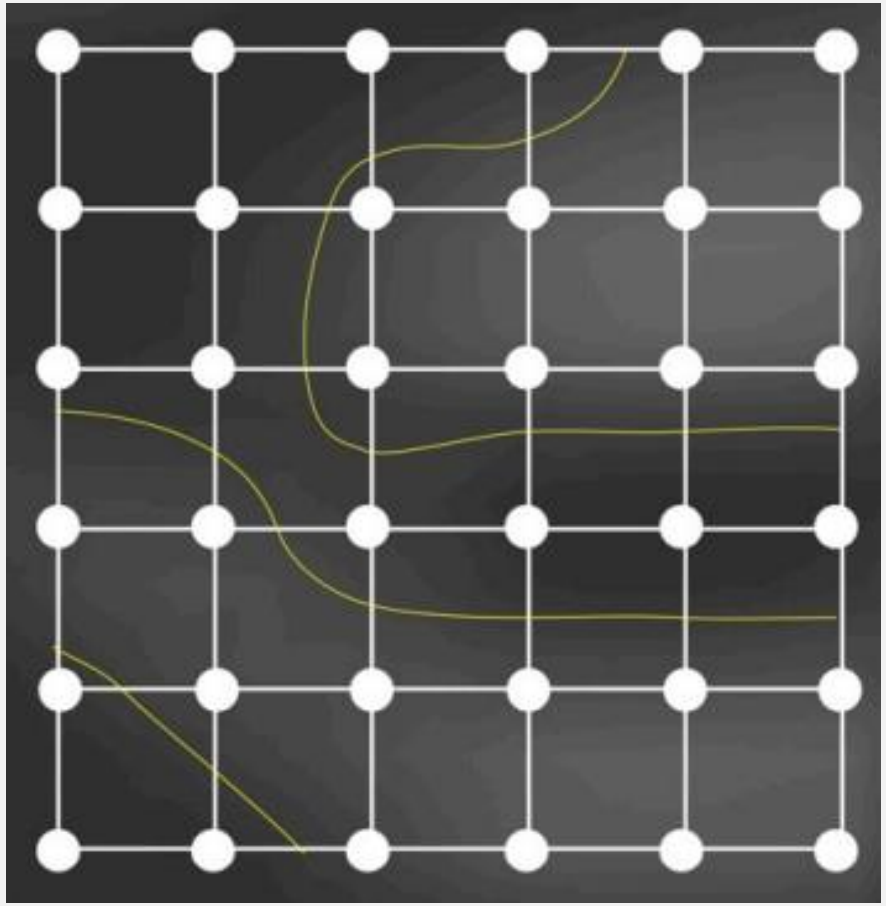


[James Sharman]

Marching Squares (2D)

- Given a function $f(\mathbf{x})$
  - $f(\mathbf{x}) < 0$: inside
  - $f(\mathbf{x}) > 0$: outside

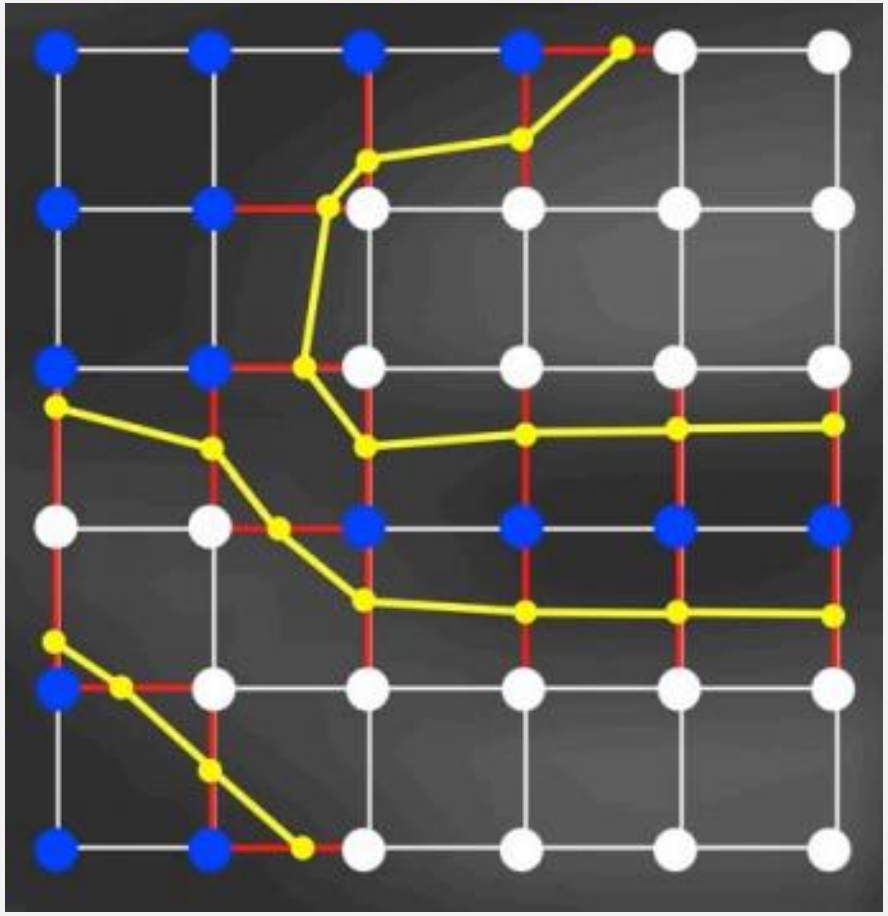
1. Discretize space
2. Evaluate $f(\mathbf{x})$ on a grid



Marching Squares (2D)

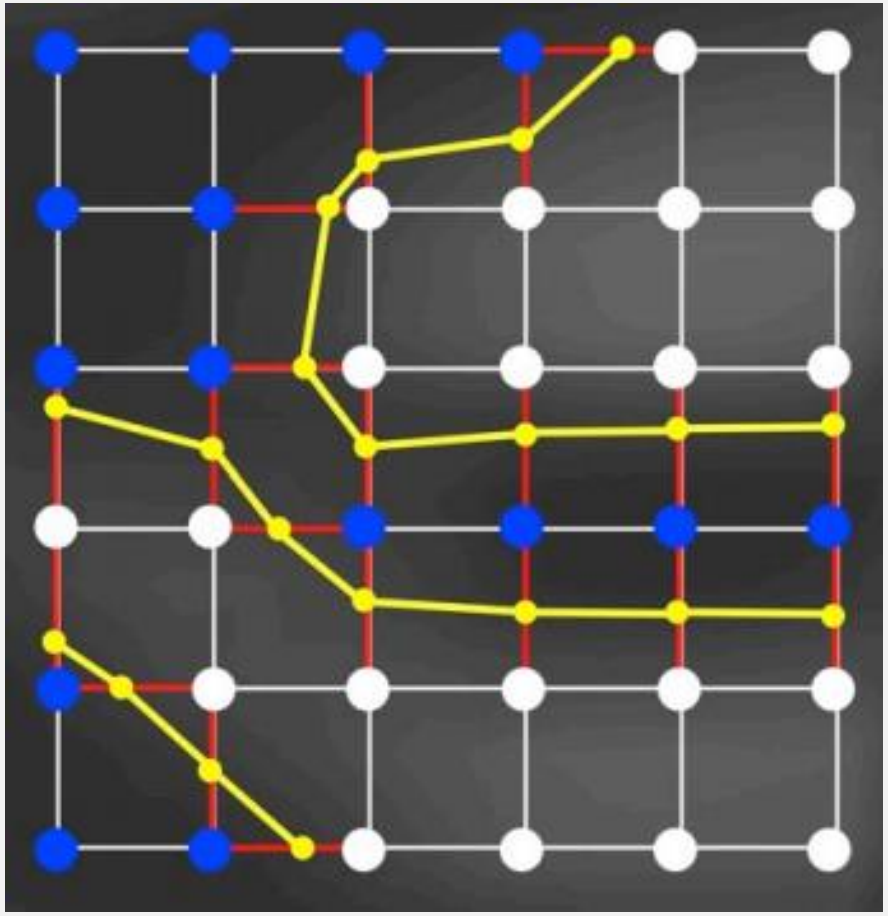
- Given a function $f(\mathbf{x})$
  - $f(\mathbf{x}) < 0$: inside
  - $f(\mathbf{x}) > 0$: outside

1. Discreyize space
2. Evaluate $f(\mathbf{x})$ on a grid
3. Classify grid points (+-)
4. Classify grid edges
5. Compute intersections
6. Connect intersections



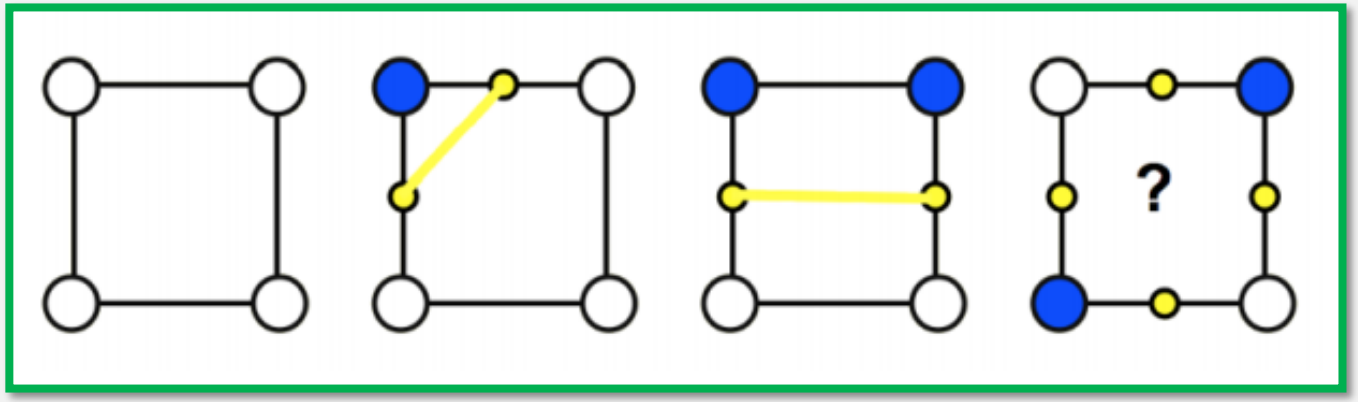
Marching Squares (2D)

- Compute intersections :
  - Edges with a sign switch contain intersections
  - $f(\mathbf{x}_1) < 0\,, f(\mathbf{x}_2) > 0$
  - ➔$f(\mathbf{x}_1 + \boldsymbol{t}(\mathbf{x}_2 - \mathbf{x}_1)) = 0$ for some $0 \leq \boldsymbol{t} \leq \mathbf{1}$
- Connecting 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



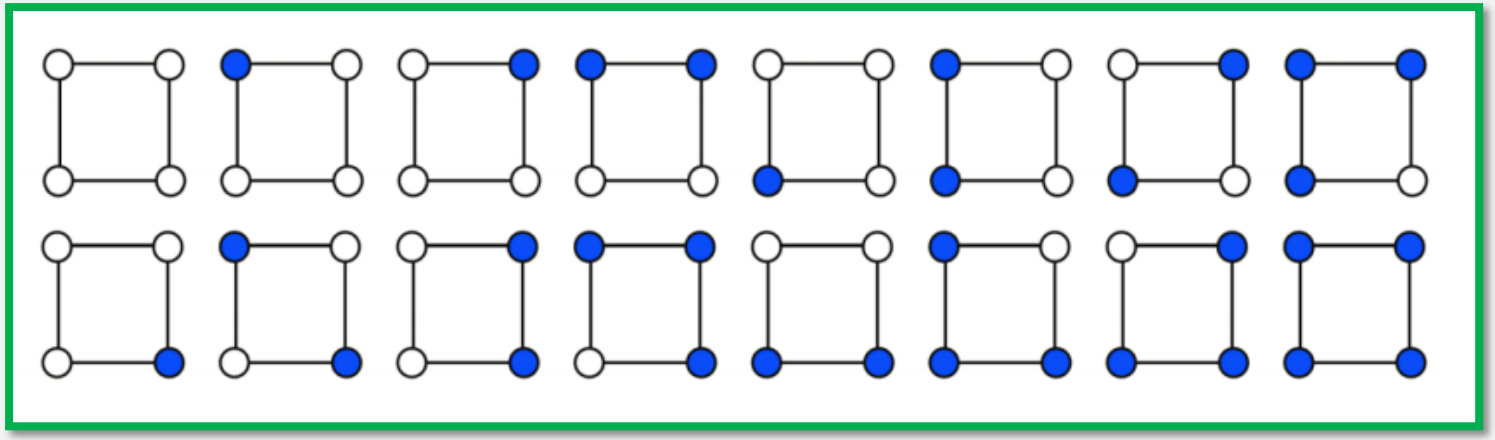
Marching Squares (2D)

- Connecting 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



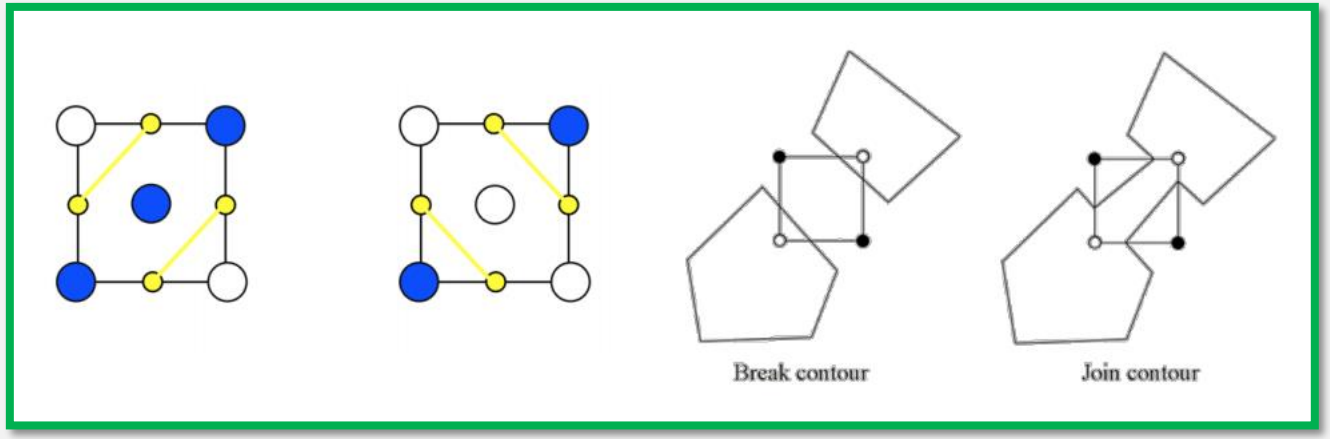
Marching Squares (2D)

- Connecting 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 intersections
  - Ambiguous Cases:



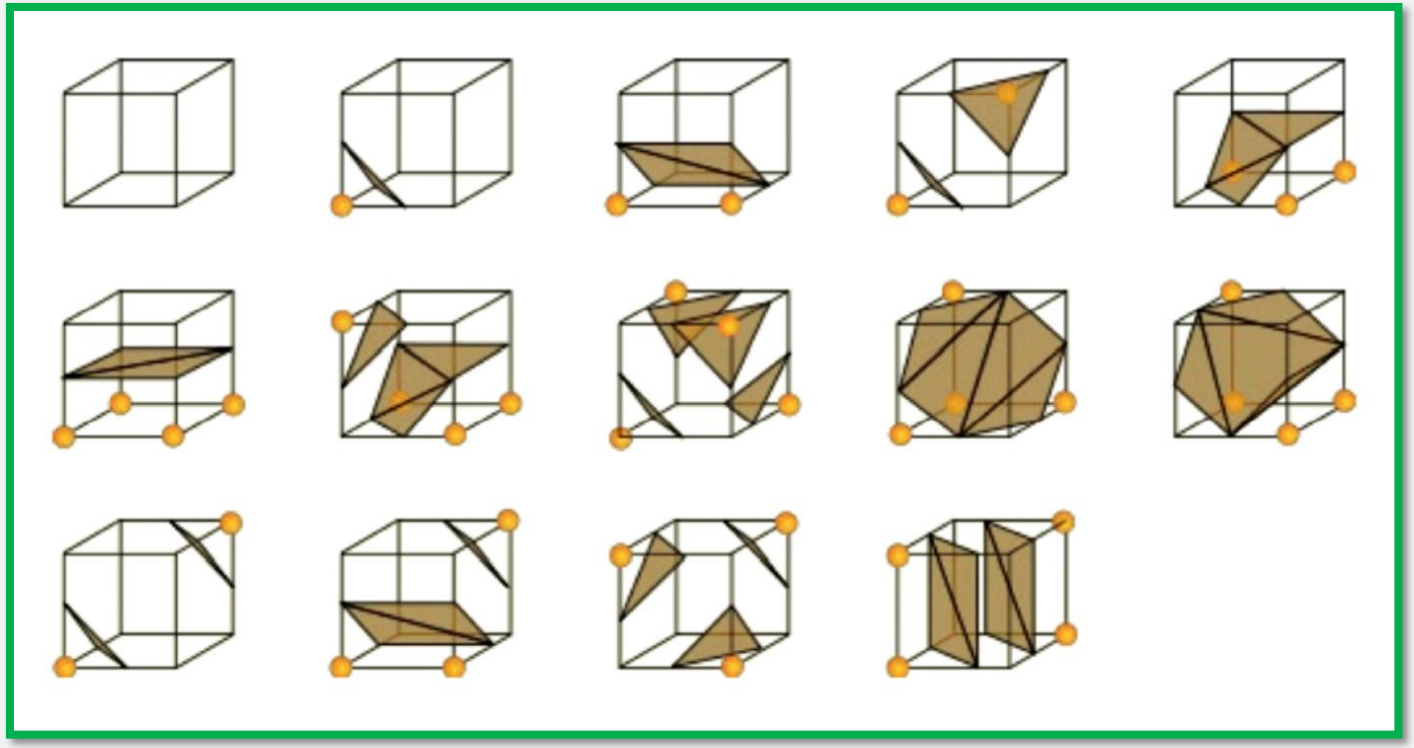
  - Two options:

    1) 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)

- 6 ambiguous cases :