Fitting Models to Data

BMVC 2015 Tutorial

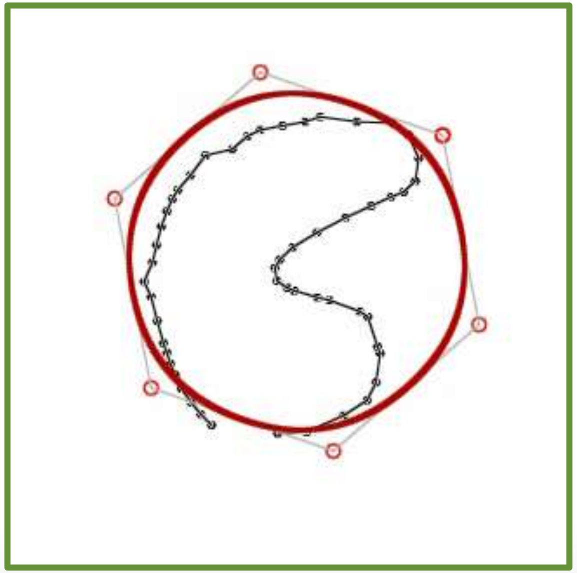
Andrew Fitzgibbon, Microsoft

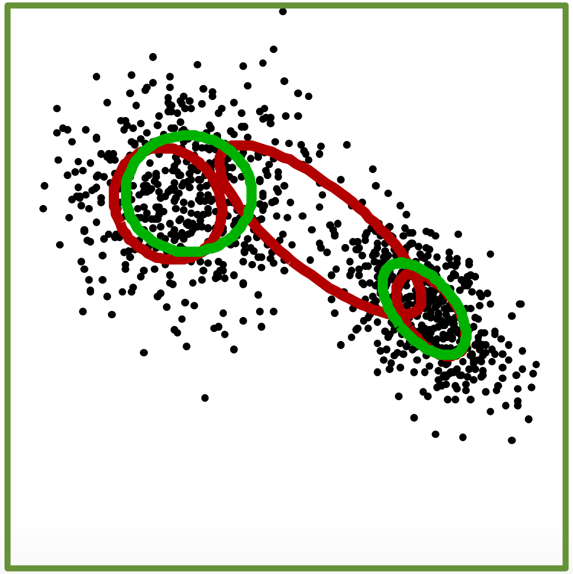
Finding Nemo: Deformable Object Class Modelling using Curve Matching	Mukta Prasad, Andrew Fitzgibbon, Andrew Zisserman, Luc Van Gool	**CVPR ’10**
KinÊtre: **Animating the World with the Human Body**	Jiawen Chen, Shahram Izadi, Andrew Fitzgibbon	**UIST ’12**
What shape are dolphins? Building 3D morphable models from 2D images	Tom Cashman, Andrew Fitzgibbon	**PAMI ’13**
User-Specific Hand Modeling from Monocular Depth Sequences	Jonathan Taylor, Richard Stebbing, Varun Ramakrishna, Cem Keskin, Jamie Shotton, Shahram Izadi, Andrew Fitzgibbon, Aaron Hertzmann	**CVPR ’14**
Real-Time Non-Rigid Reconstruction Using an RGB-D Camera	Michael Zollhöfer, Matthias Nießner, Shahram Izadi, Christoph Rhemann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, Marc Stamminger	**SIGGRAPH ’14**
Learning an Efficient Model of Hand Shape Variation from Depth Images	Sameh Khamis, Jonathan Taylor, Jamie Shotton, Cem Keskin, Shahram Izadi, Andrew Fitzgibbon	**CVPR ’15**
Towards Pointless Structure from Motion: 3D reconstruction from 3D curves	Irina Nurutdinova, Andrew Fitzgibbon	**ICCV ’15**
Secrets of Matrix Factorization: Approximations, Numerics and Manifold Optimization	Je Hyeong Hong, Andrew Fitzgibbon	**ICCV ’15**

PEOPLE

Goal

LEARN HOW TO SOLVE HARD VISION PROBLEMS,
USING TOOLS THAT MAY APPEAR INELEGANT,
BUT ARE MUCH SMARTER THAN THEY LOOK.


Curve/surface fitting


Parameter estimation

"Bundle adjustment"
(Video from our friends at Google)

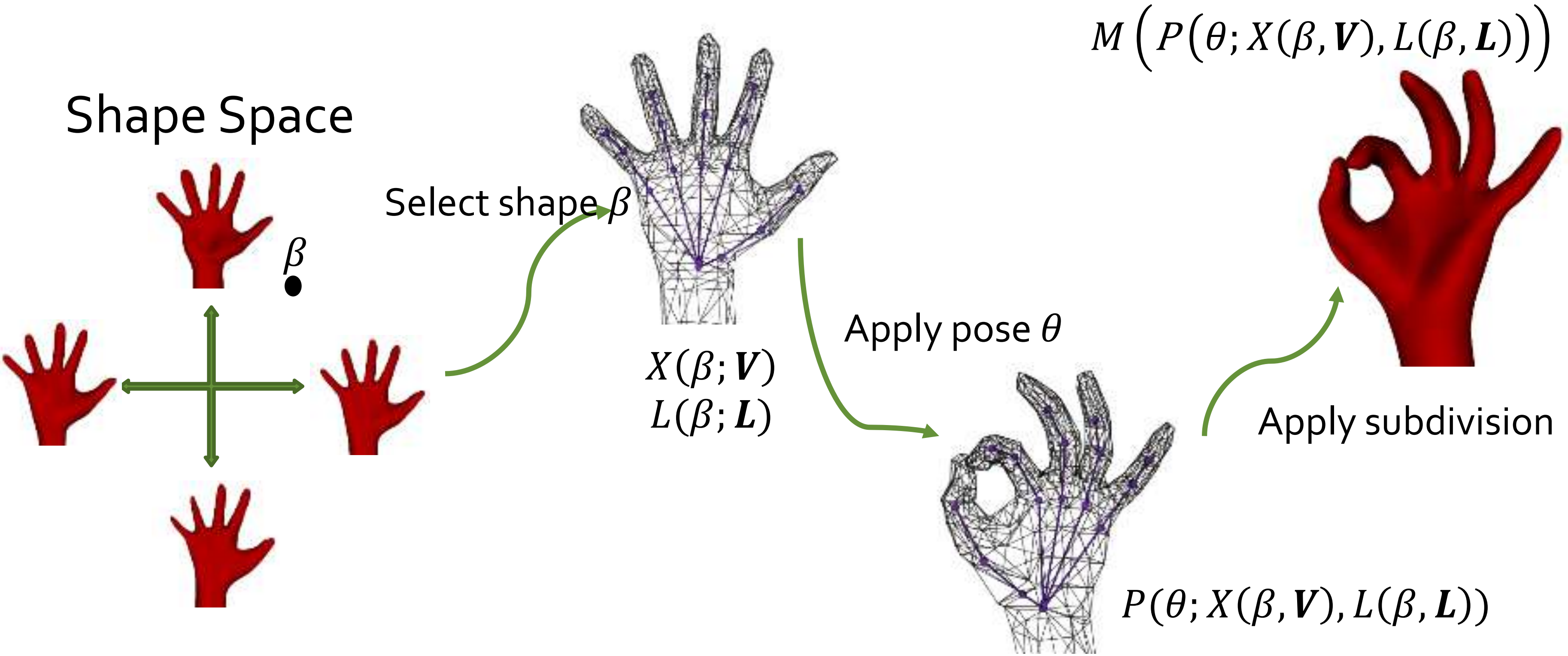
APPLICATIONS

Blanz & Vetter
Siggraph 1999

FITTING 3D SHAPE BASES

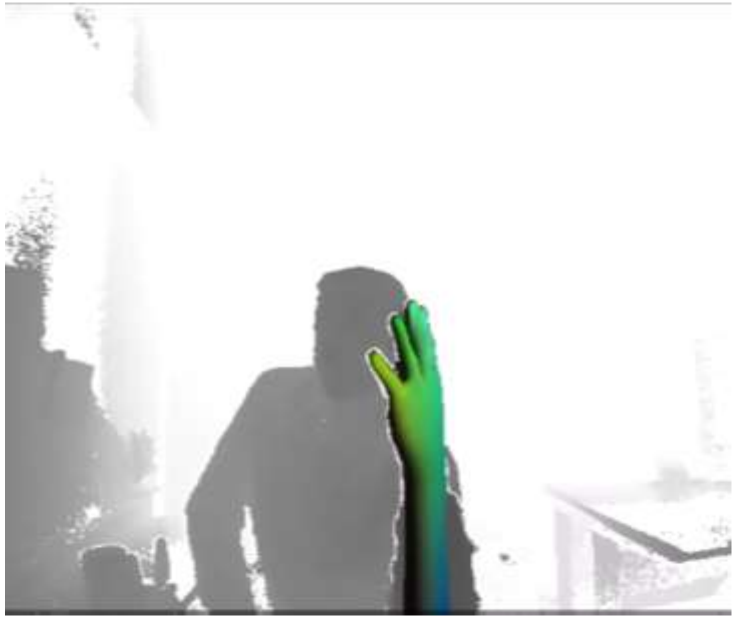
Anguelov *et al.*
Siggraph 2005

FITTING 3D SHAPE BASES


$M\left(P(\theta; X(\beta, \boldsymbol{V}), L(\beta, \boldsymbol{L}))\right)$
Shape Space
β
Select shape β
$X(\beta; \boldsymbol{V})$
$L(\beta; \boldsymbol{L})$
Apply pose θ
Apply subdivision
$P(\theta; X(\beta, \boldsymbol{V}), L(\beta, \boldsymbol{L}))$

Learning an Efficient Model of Hand Shape Variation from Depth Images
Khamis et al, CVPR15

WHAT DO I MEAN BY SHAPE?



FITTING HANDS TO 3D DATA

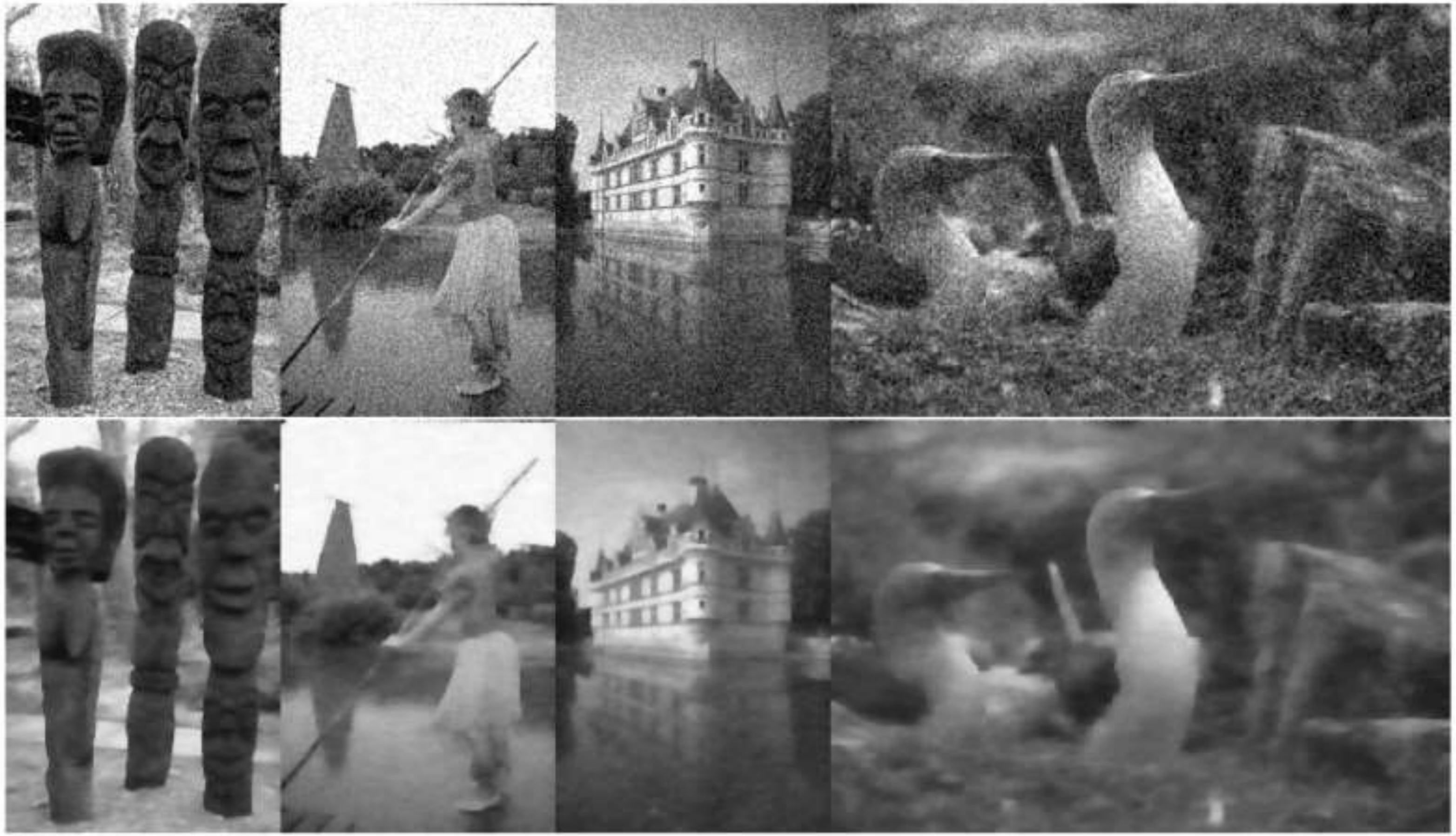
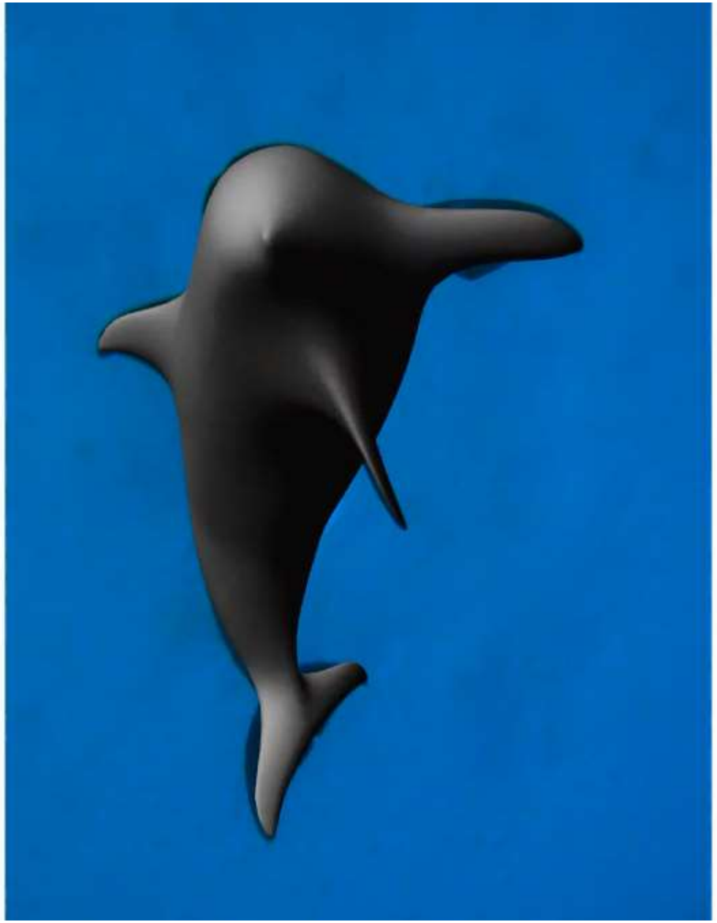


IMAGE DENOISING

[STRANDMARK & AGARWAL, 2014, arXiv:1403.5590]

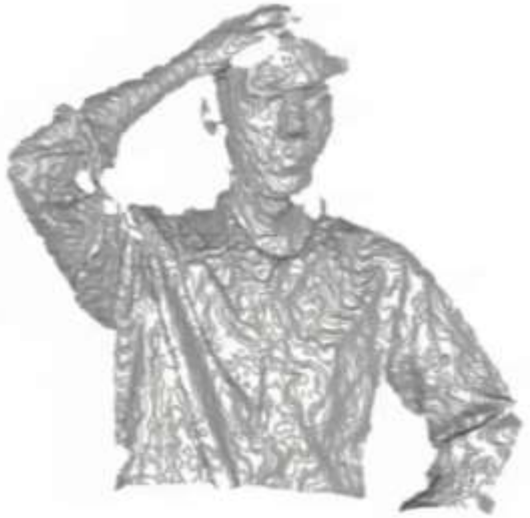


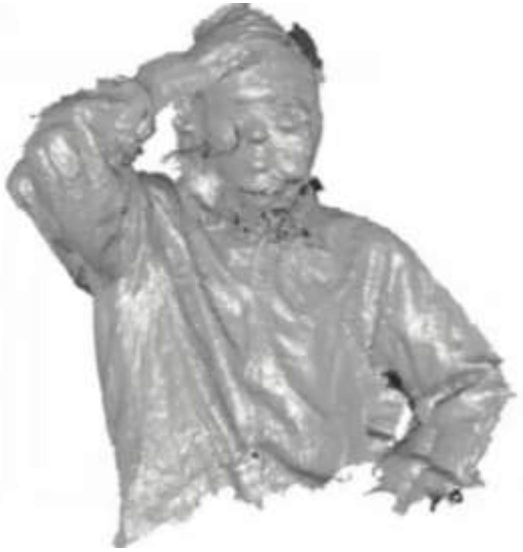
FITTING SUBDIVISION SURFACES TO 2D DATA

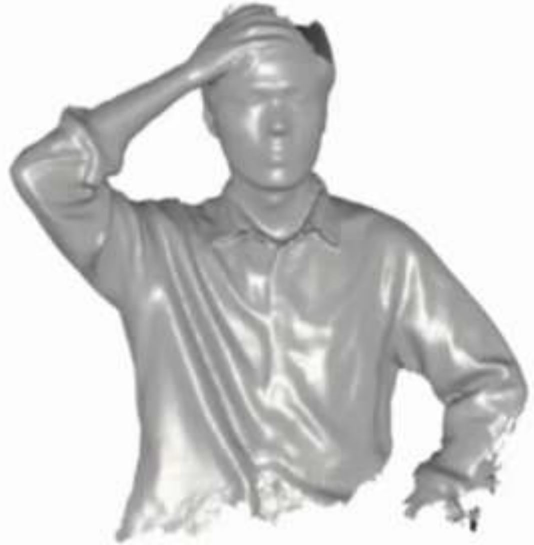
Microsoft

FITTING POLYGON MESHES TO VIDEO

Microsoft

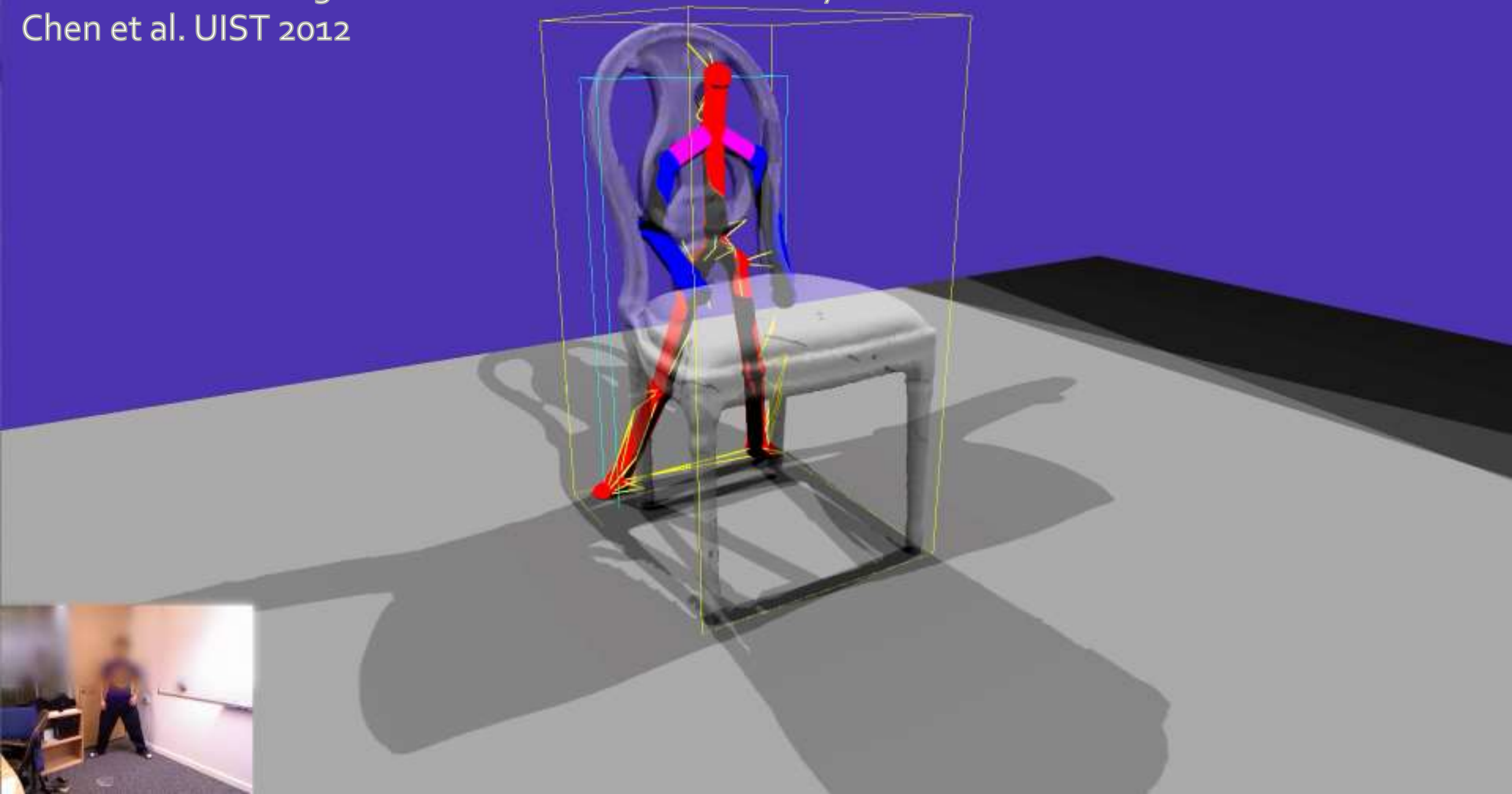

Input Kinect Stream


KinectFusion

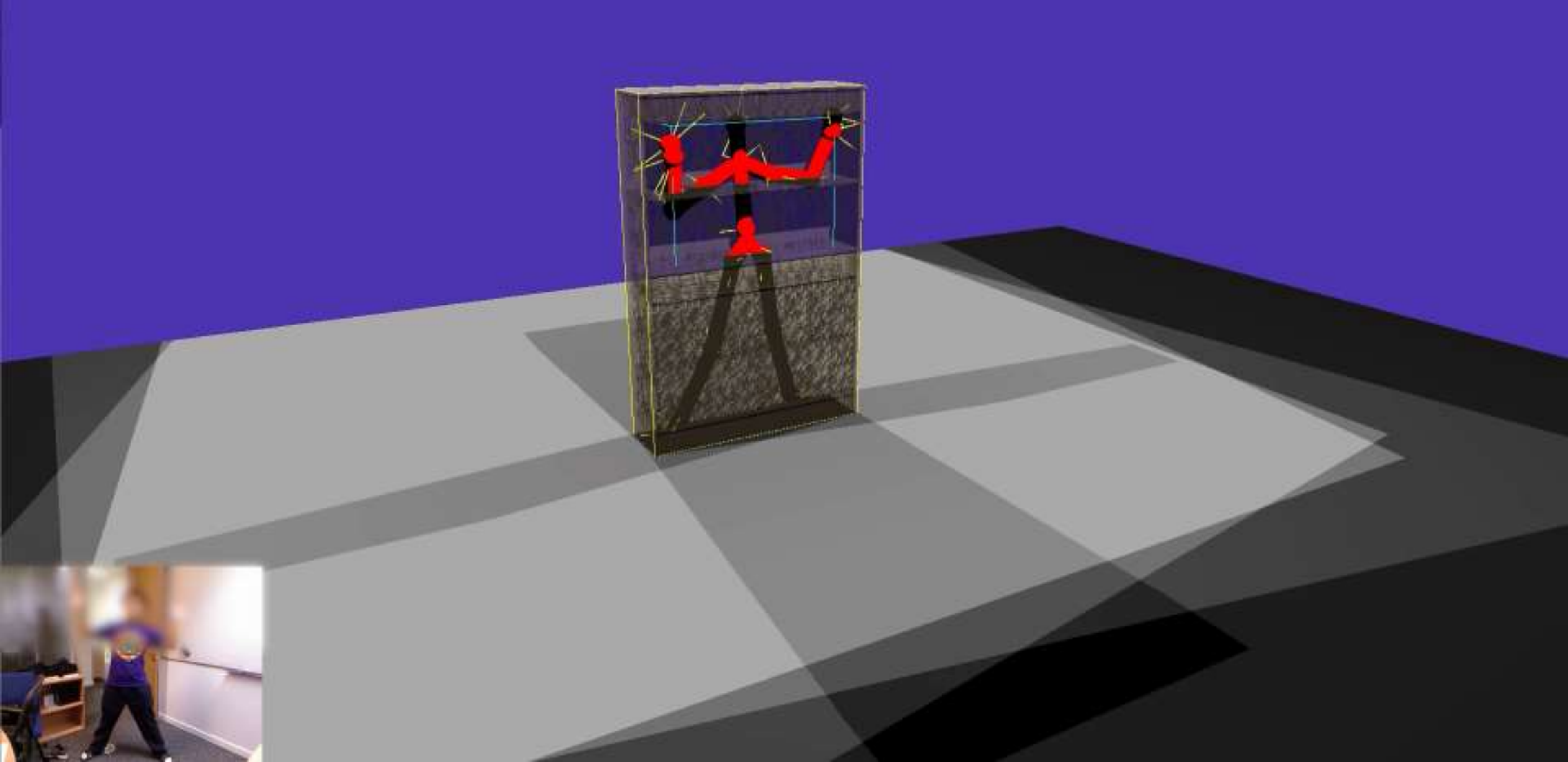

Deformable Fusion

[3D Scanning Deformable Objects with a Single RGBD Sensor, Dou et al, CVPR15]

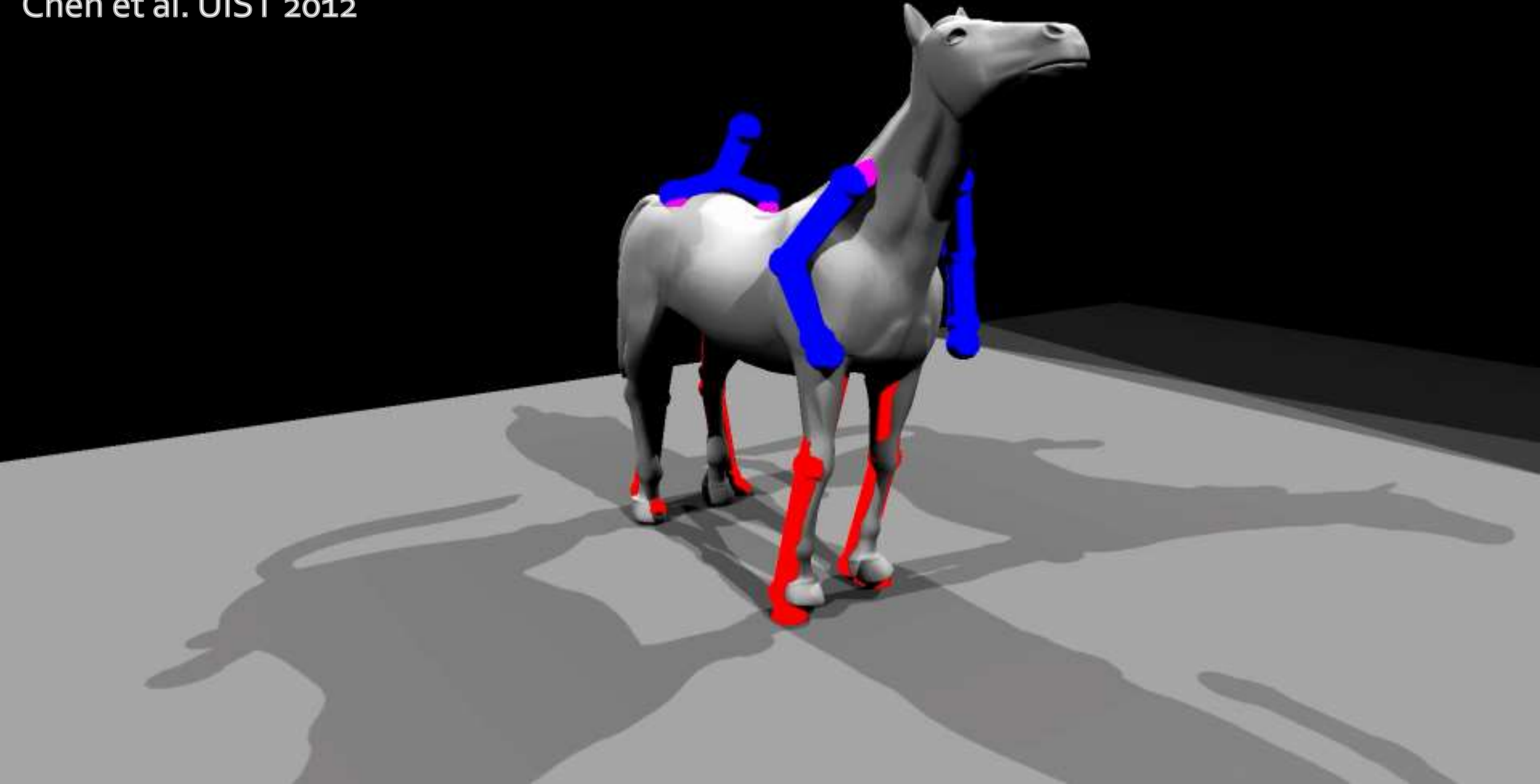
"KinEtre: Animating the World with the Human Body "
Chen et al. UIST 2012



“KinEtre: Animating the World with the Human Body ˆ
Chen et al. UIST 2012

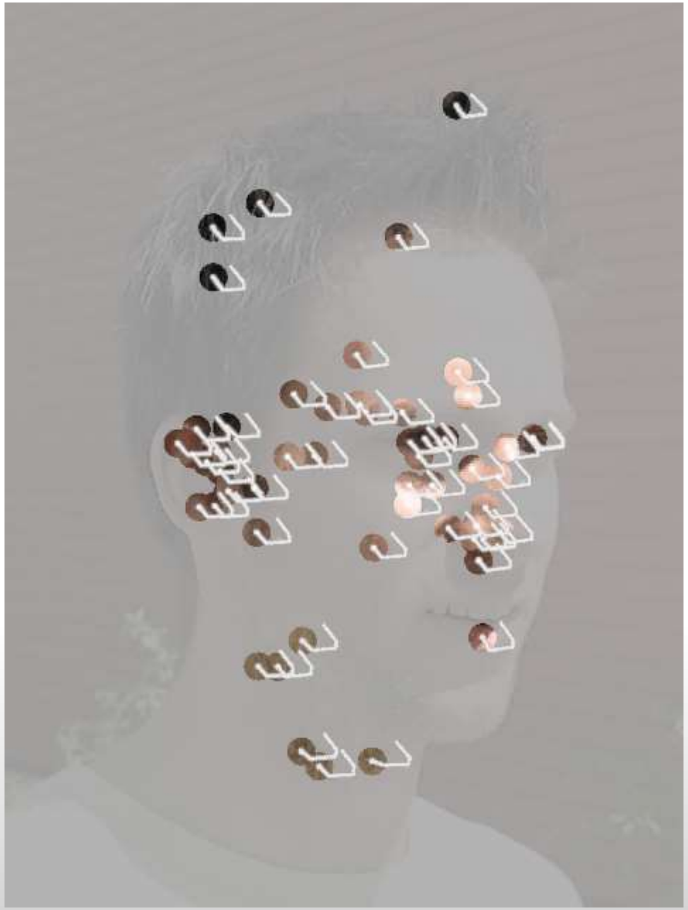


"KinEtre: Animating the World with the Human Body ˆ
Chen et al. UIST 2012

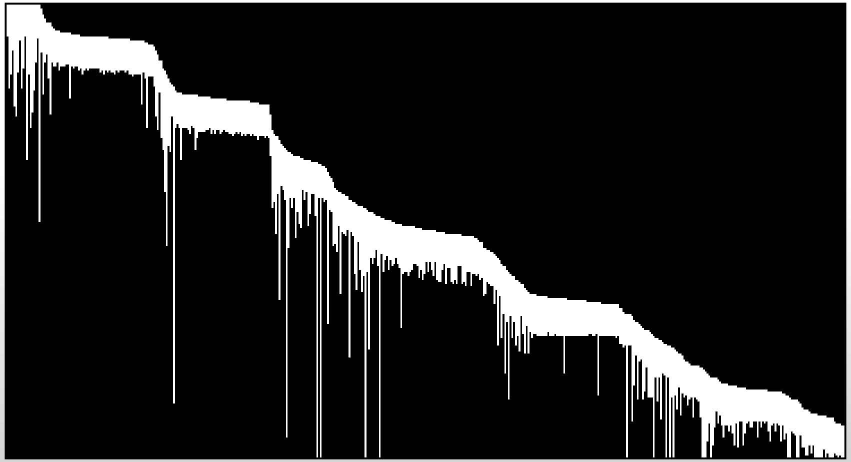


REALTIME MESH FITTING TO 3D

Microsoft



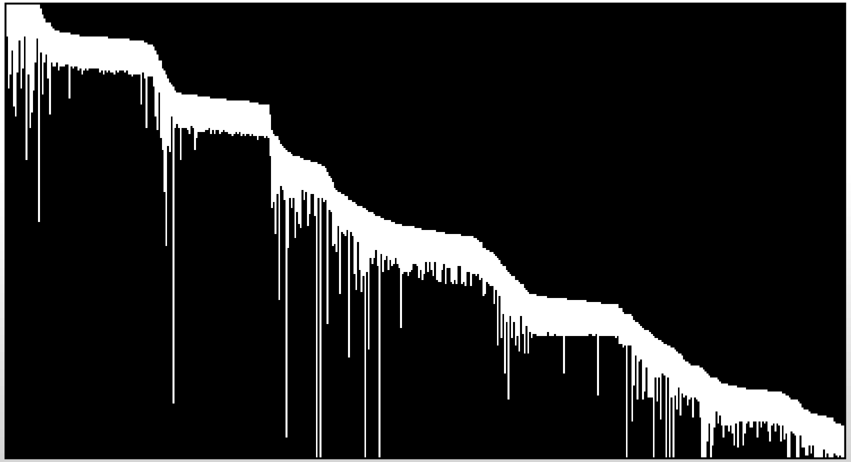
$$\begin{array}{c} \text{Time} \\ \downarrow \end{array} \begin{bmatrix} \boldsymbol{w}_{11} & \boldsymbol{w}_{12} & \cdots & \boldsymbol{w}_{1n} \\ \boldsymbol{w}_{21} & \boldsymbol{w}_{22} & \cdots & \boldsymbol{w}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{T1} & \boldsymbol{w}_{T2} & \cdots & \boldsymbol{w}_{Tn} \end{bmatrix}$$



NONRIGID STRUCTURE FROM MOTION

Microsoft

$$\begin{bmatrix} \mathbf{w}_{11} & \mathbf{w}_{12} & \cdots & \mathbf{w}_{1n} \\ \mathbf{w}_{21} & \mathbf{w}_{22} & \cdots & \mathbf{w}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{w}_{T1} & \mathbf{w}_{T2} & \cdots & \mathbf{w}_{Tn} \end{bmatrix}$$



- **Affine rigid**: linear embedding into $\mathbb{R}^3$, solved with Wiberg / bundle adjustment
- **Perspective rigid**: (slightly) nonlinear embedding into $\mathbb{R}^3$ solved with bundle adjustment
- **Nonrigid**: linear embedding into $\mathbb{R}^{3K}$, [with nonlinear constraints]
- **Kernel nonrigid/Trajectory bases**: nonlinear/basis function embedding into $\mathbb{R}^k$
- **Unwrap mosaic**: nonlinear embedding into $\mathbb{R}^2$

NONRIGID STRUCTURE FROM MOTION

Microsoft

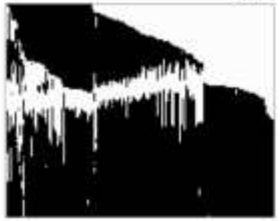
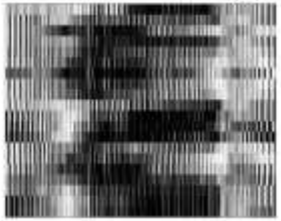
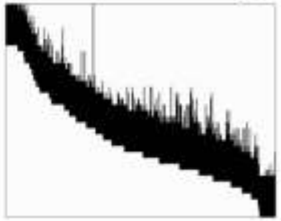
"Unwrap Mosaics"

Microsoft

240 × 167
30% missing

20 × 2944
42% missing

72 × 319
72% missing

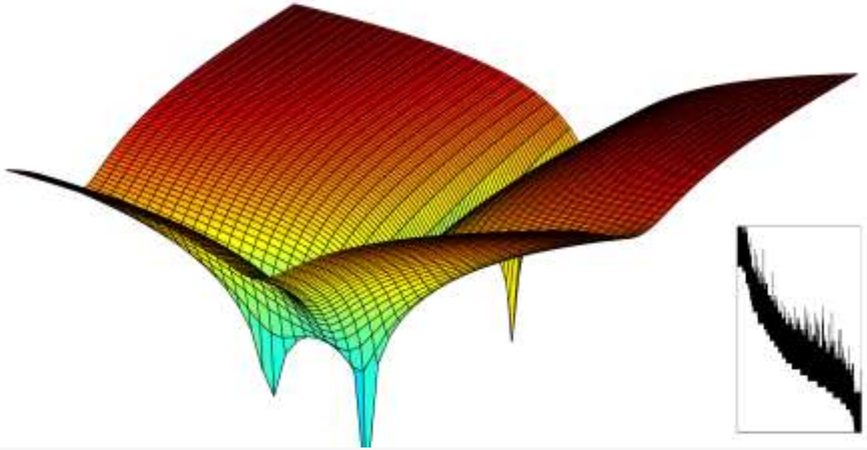
Task: Given noisy observation $\mathbf{M}$, and weight matrix $\mathbf{W}$, compute

$$\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \left\| \mathbf{W} \odot (\mathbf{M} - \mathbf{A}\mathbf{B}^\top) \right\|_F^2$$

where $\mathbf{R} = \mathbf{P} \odot \mathbf{Q} \Leftrightarrow r_{ij} = p_{ij} q_{ij}$.

2D slice through $\log \epsilon(\mathbf{A}, \mathbf{B})$ where

$$\epsilon(\mathbf{A}, \mathbf{B}) := \left\| \mathbf{W} \odot (\mathbf{M} - \mathbf{A}\mathbf{B}^\top) \right\|_F^2$$



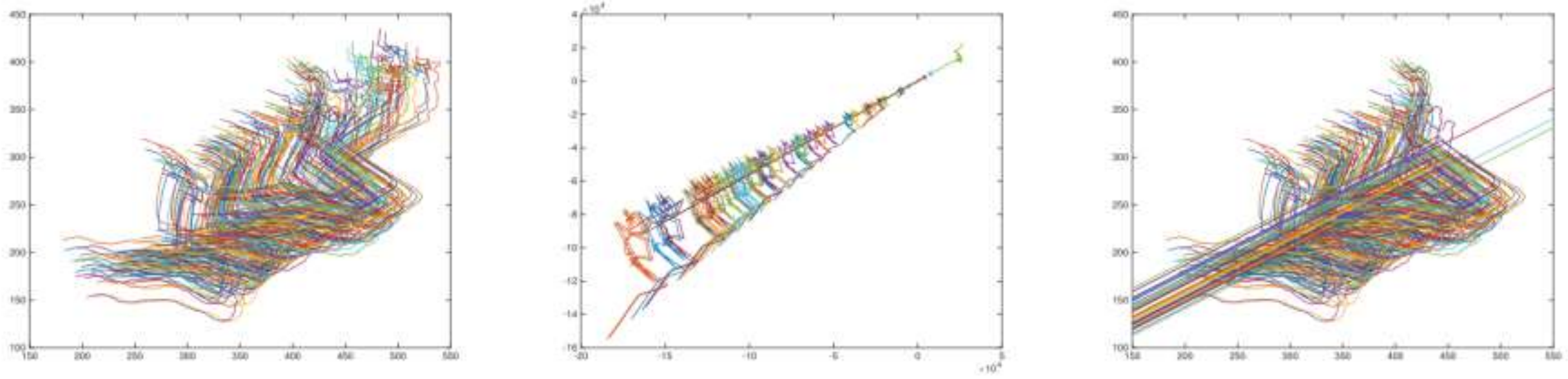
MATRIX FACTORIZATION [HONG & F., ICCV 15]

Algorithm	Framework	Manifold retraction
ALS [5]	RW3 (ALS)	None
PowerFactorization [5, 27]	RW3 (ALS)	q-factor
LM-S [8]	Newton + ⟨Damping⟩	`orth` (replaced by q-factor)
LM-S$_{GN}$ [9, 13]	RW1 (GN) + ⟨Damping⟩ (DRW1 equiv.)	`orth` (replaced by q-factor)
LM-M [8]	Reduced$_r$ Newton + ⟨Damping⟩	`orth` (replaced by q-factor)
LM-M$_{GN}$ [8]	Reduced$_r$ RW1 (GN) + ⟨Damping⟩	`orth` (replaced by q-factor)
Wiberg [18]	RW2 (Approx. GN)	None
Damped Wiberg [19]	RW2 (Approx. GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	None
CSF [13]	RW2 (Approx. GN) + ⟨Damping⟩ (DRW2 equiv.)	q-factor
RTRMC [4]	Projected$_p$ Newton + {Regularization} + ⟨Trust Region⟩	q-factor
LM-S$_{RW2}$	RW2 (Approx. GN) + ⟨Damping⟩ (DRW2 equiv.)	q-factor
LM-M$_{RW2}$	Reduced$_r$ RW2 (Approx. GN) + ⟨Damping⟩	q-factor
DRW1	RW1 (GN) + ⟨Damping⟩	q-factor
DRW1P	RW1 (GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	q-factor
DRW2	RW2 (Approx. GN) + ⟨Damping⟩	q-factor
DRW2P	RW2 (Approx. GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	q-factor

$$\tfrac{1}{2}\mathtt{H}^* = \mathtt{P}_r{}^\top \Big(\tilde{\mathtt{V}}^{*\top}(\mathtt{I}_p - [\tilde{\mathtt{U}}\tilde{\mathtt{U}}^\dagger]_{RW2})\tilde{\mathtt{V}}^* + [\mathtt{K}_{mr}^\top \mathtt{Z}^*(\tilde{\mathtt{U}}^\top\tilde{\mathtt{U}})^{-1}\mathtt{Z}^{*\top}\mathtt{K}_{mr}]_{RW1} \times [-1]_{FN} \\ + [\mathtt{K}_{mr}^\top \mathtt{Z}^*\tilde{\mathtt{U}}^\dagger\tilde{\mathtt{V}}^*\mathtt{P}_p + \mathtt{P}_p\tilde{\mathtt{V}}^{*\top}\tilde{\mathtt{U}}^{\dagger\top}\mathtt{Z}^{*\top}\mathtt{K}_{mr}]_{FN} + \langle \alpha\mathtt{I}_r \otimes \mathtt{U}\mathtt{U}^\top\rangle_P + \langle\lambda\mathtt{I}_{mr}\rangle\Big)\mathtt{P}_r$$

MATRIX FACTORIZATION [HONG & F., ICCV 15]

Microsoft



(a) Best known minimum (0.3228) (b) Second best solution (0.3230) (c) Second best, zoomed to image

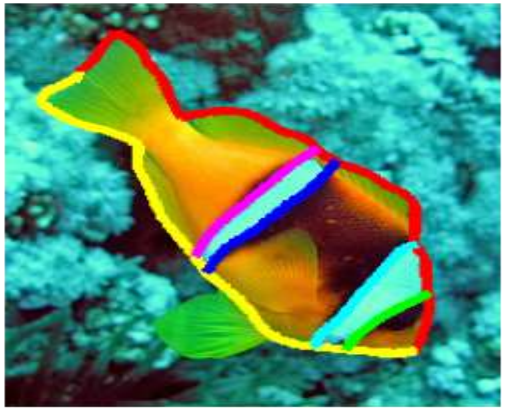
Figure 1: Illustration that a solution with function value just .06% above the optimum can have significantly worse extrapolation properties. This is a reconstruction of point trajectories in the standard "Giraffe" sequence. Even when zooming in to eliminate gross outliers (not possible for many reconstruction problems), it is clear that numerous tracks have been incorrectly reconstructed.

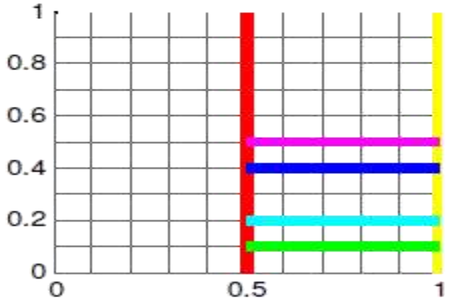
MYTH: YOU DON'T NEED TO OPTIMIZE FAR

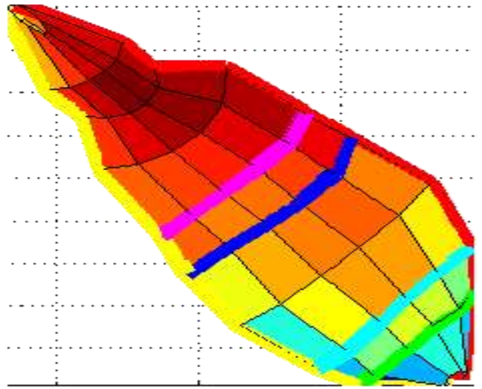
BUT POINTS ARE TOO EASY…

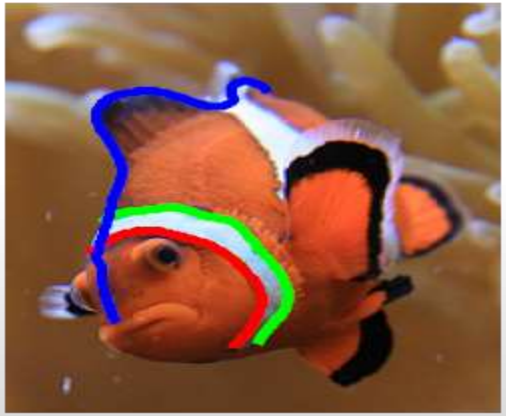
clownfish

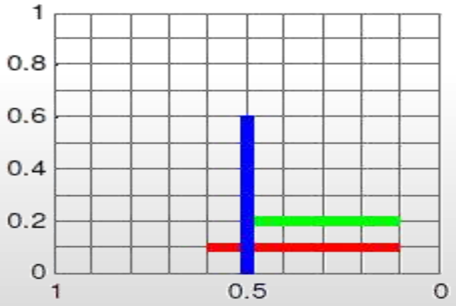
OBJECT CATEGORY MODELS

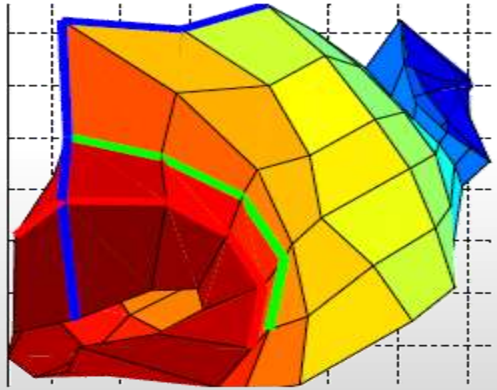




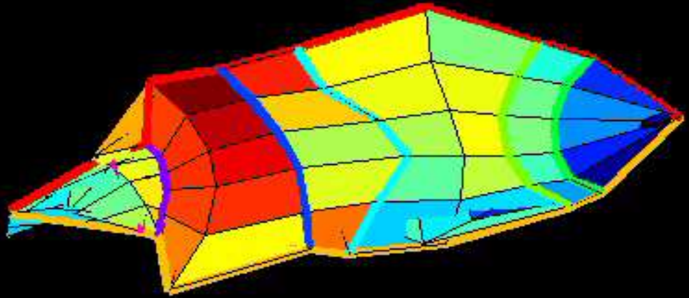




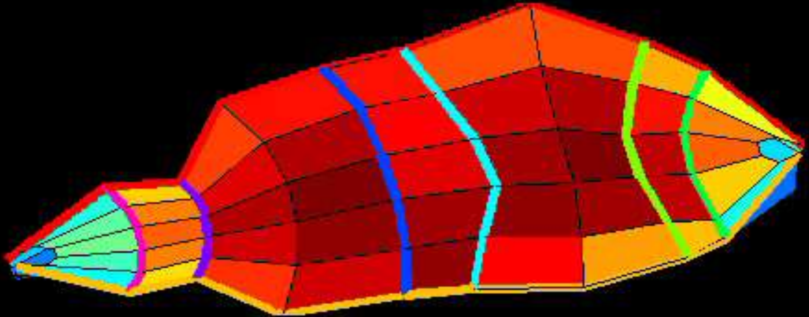




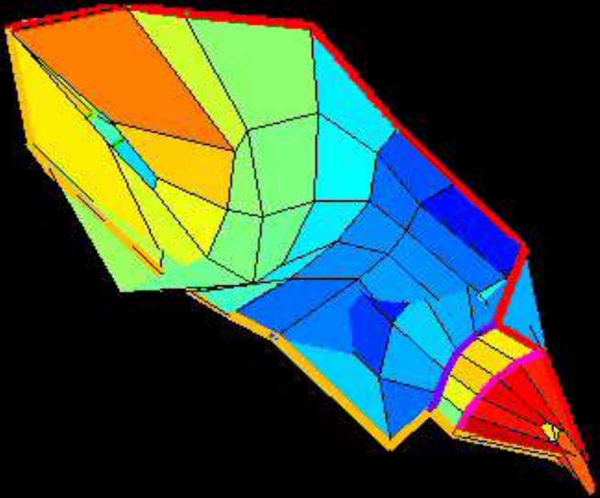
SHAPE FROM CURVES



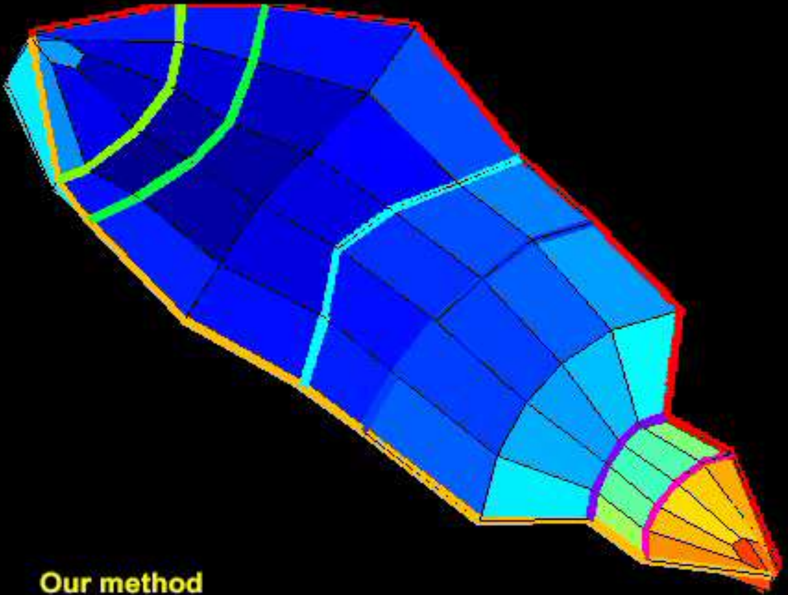
NRSfM



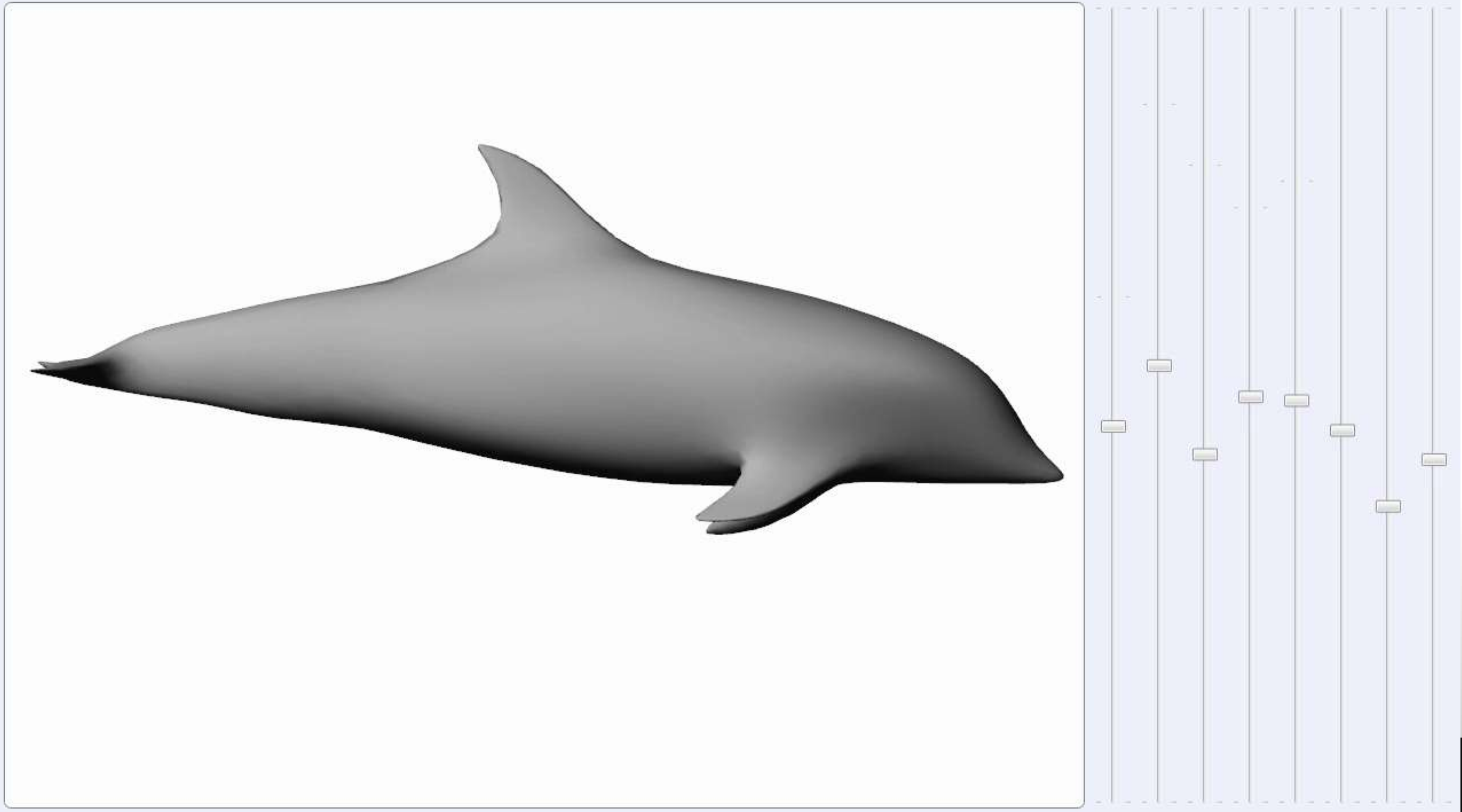
Our method



NRSfM

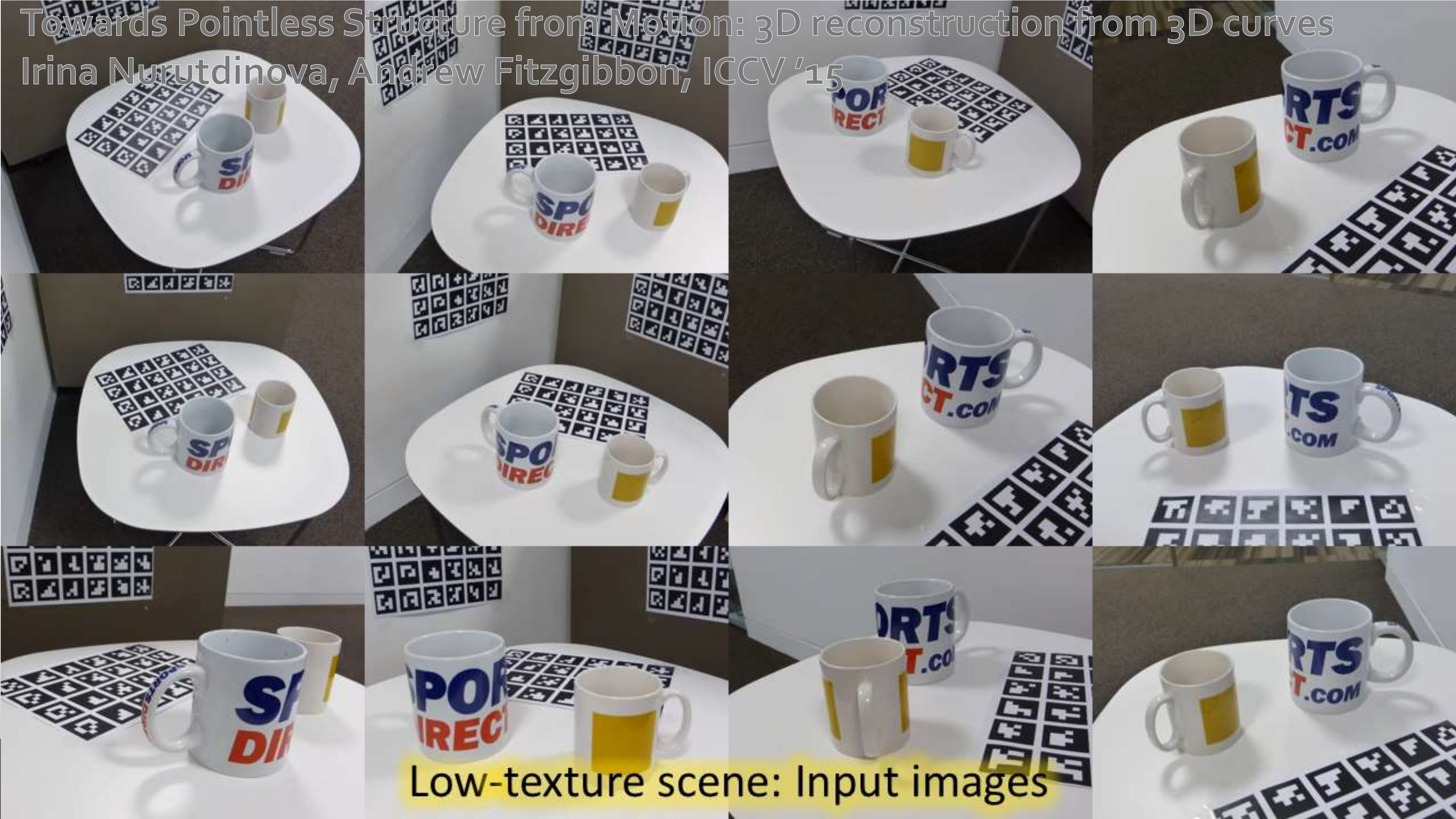


Our method



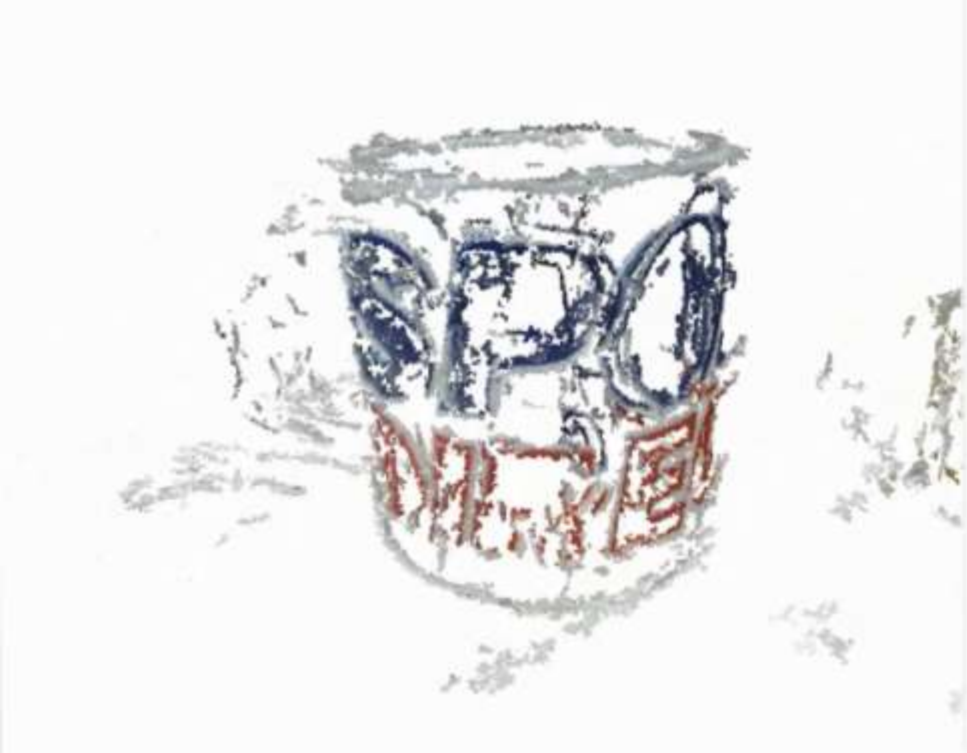
Towards Pointless Structure from Motion: 3D reconstruction from 3D curves

Irina Nurutdinova, Andrew Fitzgibbon, ICCV '15



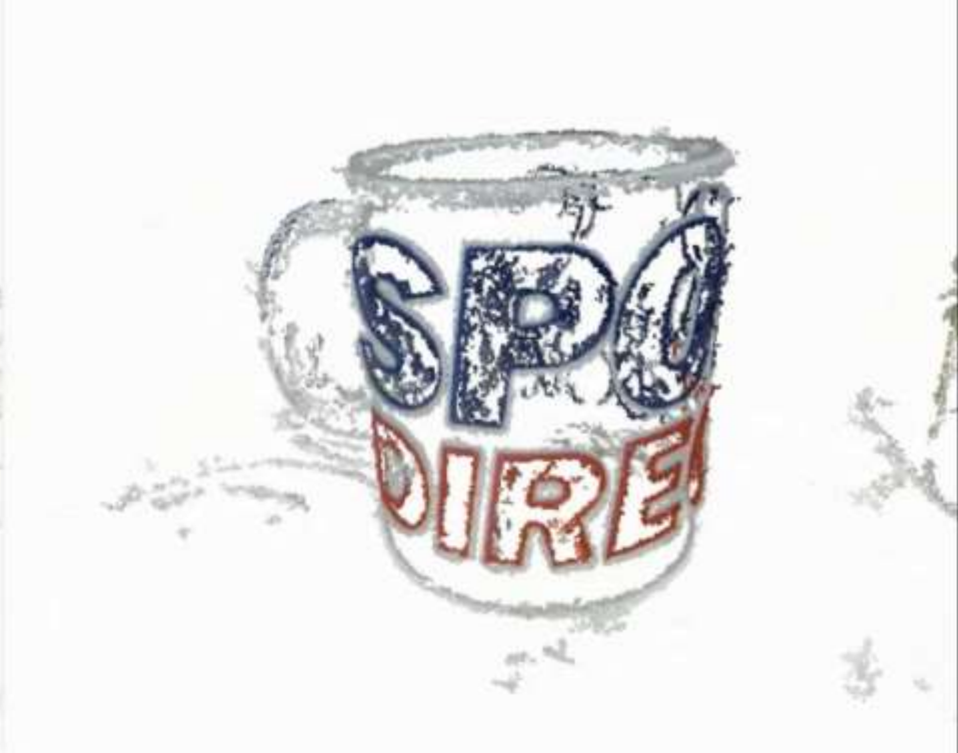
Low-texture scene: Input images

Towards Pointless Structure from Motion: 3D reconstruction from 3D curves
Irina Nurutdinova, Andrew Fitzgibbon, ICCV '15



[Zoom]

Dense reconstruction (PMVS) using cameras estimated from points only



[Zoom]

Dense reconstruction (PMVS) using cameras estimated from points and curves

- The shape from silhouette problem, even for multiple images of the same structure, was not adequately solved before
- Why?
  1. The discovery of the fundamental matrix and closed form solutions to various geometry problems revolutionized computer vision…
  2. …and distracted us from easy problems like this one.
- Behind every "closed form" solution (ellipse fitting, F+radial), there's a perfectly good nonlinear minimization solution you could have used instead
  - unless you are in the extreme speed domain [see Kukelova et al]

Write energy describing the image collection

$$\sum_{f=1}^{F} E_{\mathrm{data}}\left(I_f, \boldsymbol{\theta}_f\right) + E_{\mathrm{reg}}\left(\boldsymbol{\theta}_f, \boldsymbol{\theta}_{\mathrm{core}}\right)$$

Where:

$\boldsymbol{\theta}_f$ are (unknown) parameters of surface model in frame f

$\boldsymbol{\theta}_{\mathrm{core}}$ are (unknown) parameters of some shape model (e.g. linear combination) and E_{reg} measures distance, e.g. ARAP

And optimize it using Levenberg-Marquardt

- (i.e. any Newton-like algorithm, making maximum use of problem structure)

FOR EACH TASK, THE METHOD IS THE SAME

Microsoft

- So, you can do lots of things by “fitting models to data”.

- How do you do it right?

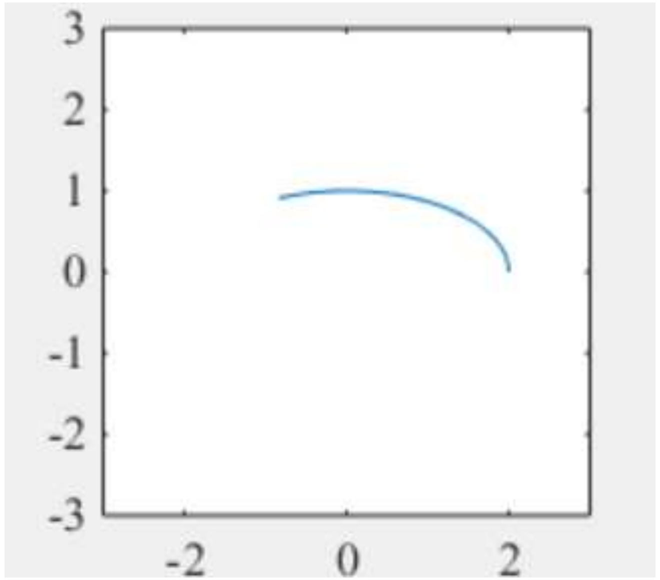
- Let’s look at some examples.

EXAMPLE: SHAPE FITTING

```
t = 0:.01:2;

plot(cos(t)*2, sin(t));
```

SHAPE QUIZ

```
t = 0:.01:2;

plot(cos(t)*2, sin(t));
```

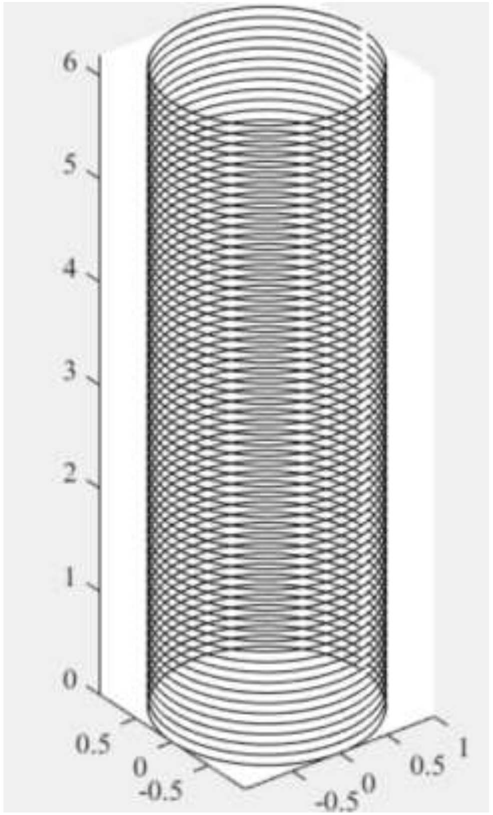


SHAPE QUIZ

```
>> u = 0:.1:2*pi; v= 0:.1:2*pi;
>> l = ones(size(v));
>> u = u'*l;
>> v = l'*v;
>> plot3(cos(u), sin(u), v, 'k.')
```

SHAPE QUIZ

```
>> u = 0:.1:2*pi; v= 0:.1:2*pi;
>> l = ones(size(v));
>> u = u'*l;
>> v = l'*v;
>> plot3(cos(u), sin(u), v, 'k.')
```

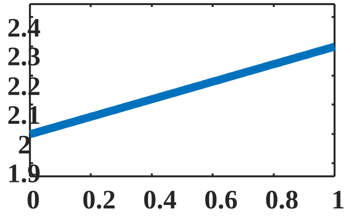


SHAPE QUIZ

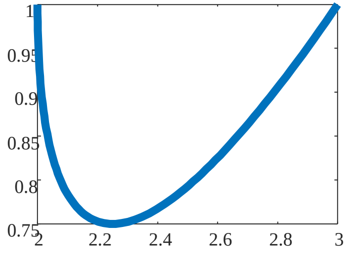
What is a shape?

- Functions
- Curves
- Surfaces

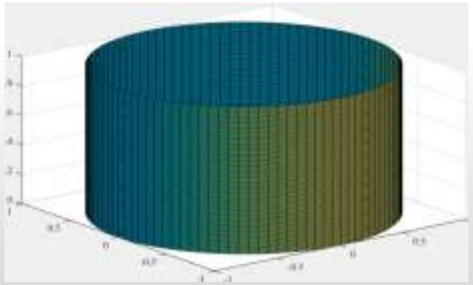
```
function y(x::Real)::Real = .3*x + 2
```



```
function C(t::Real)::Point2D =
        Point2D(t^2 + 2, t^2 - t + 1)
```



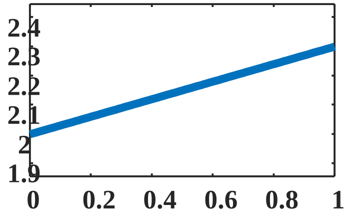
```
function S(u::Real, v::Real)::Point3D =
        Point3D(cos(u), sin(u), v)
```



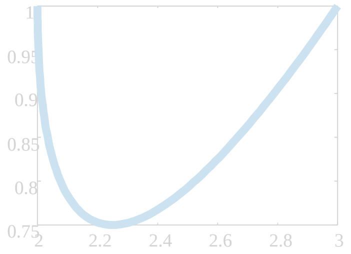
A SHAPE IS A FUNCTION

Microsoft

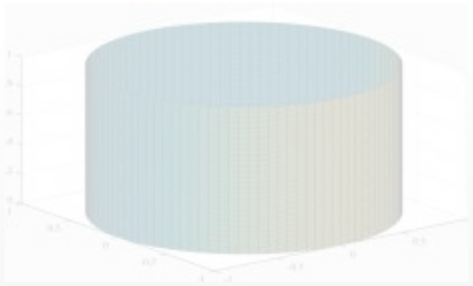
```
function y(x::Real)::Real = .3*x + 2
```



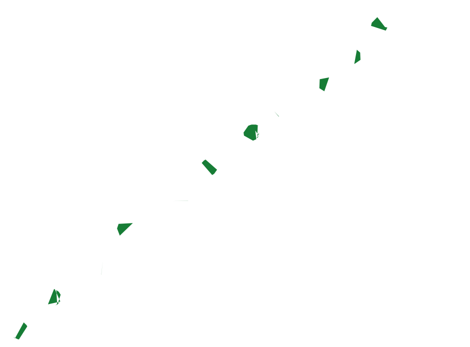
```
function C(t::Real)::Point2D =
      Point2D(t^2 + 2, t^2 - t + 1)
```

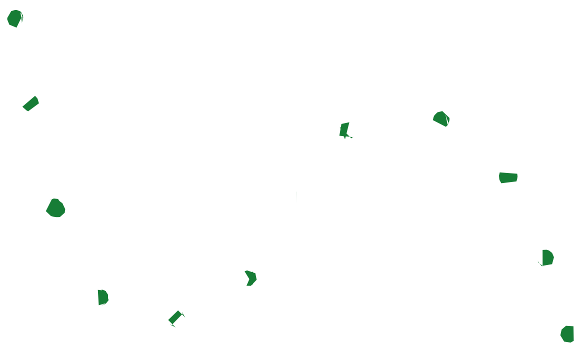


```
function S(u::Real, v::Real)::Point3D =
      Point3D(cos(u), sin(u), v)
```

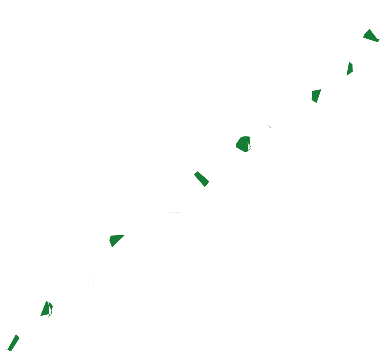


A SHAPE IS A FUNCTION

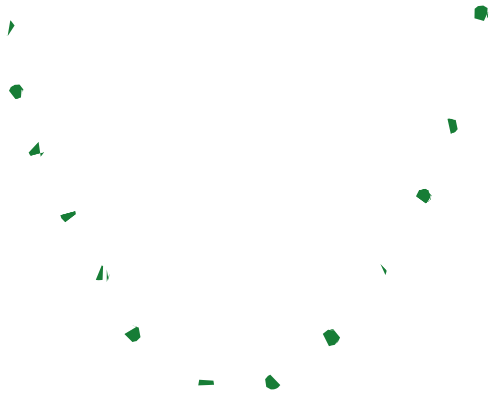




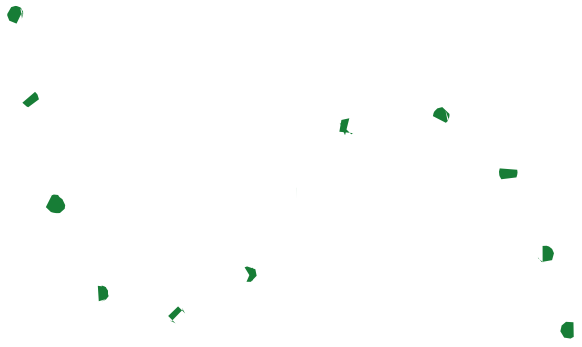
SHAPES DESCRIBE DATA



$y = ax + b$

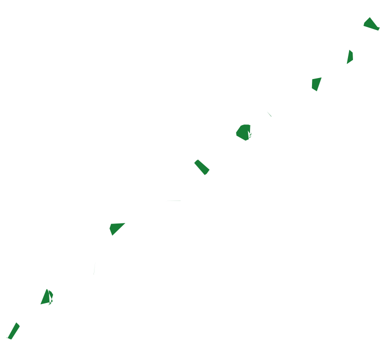


$y = ax^2 + bx + c$

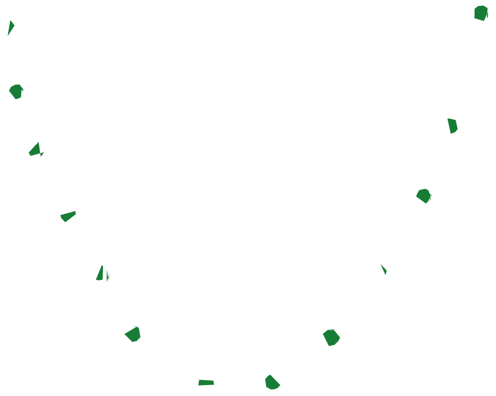


$y = ax^3 + bx^2 + cx + d$

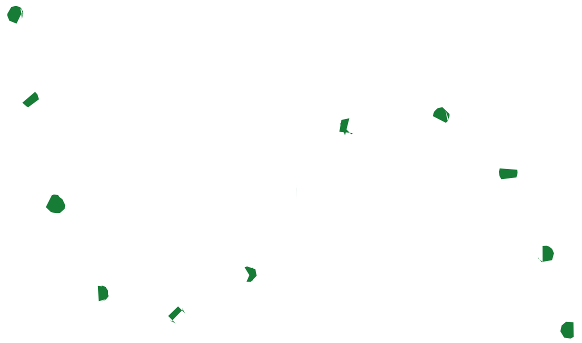
SHAPES DESCRIBE DATA



$y = ax + b$

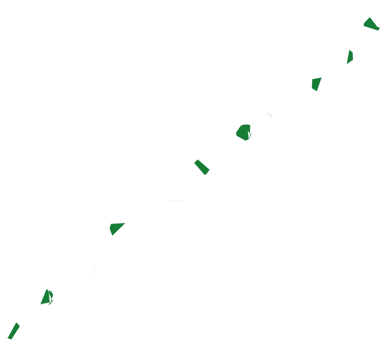


$y = ax^2 + bx + c$

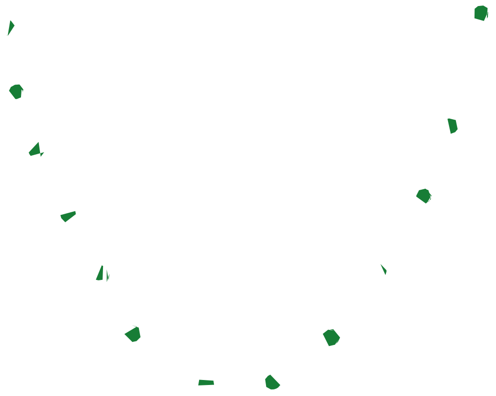


$y = \sin(x) + ax + b$

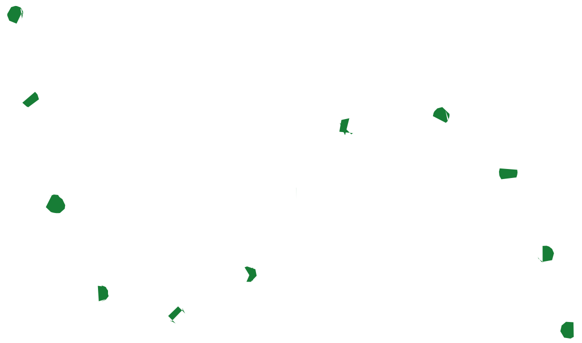
SHAPES DESCRIBE DATA



$y = ax + b$



$y = ax^2 + bx + c$



$y = \text{if } x < a$

$\quad (x - b)^2 + c$

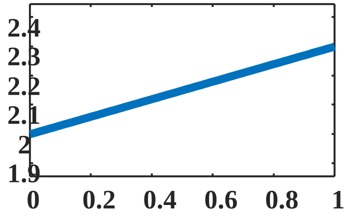
else

$\quad -(x - d)^2 + e$

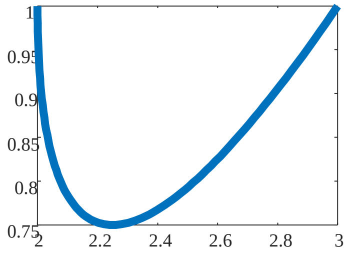
SHAPES DESCRIBE DATA

Microsoft

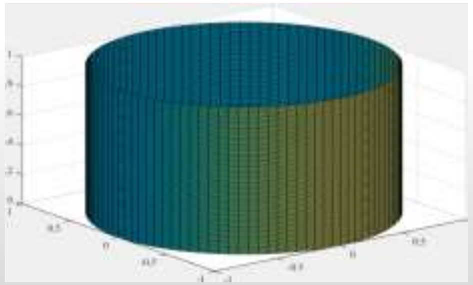
```
function y(x::Real)::Real = .3*x + 2
```



```
function C(t::Real)::Point2D =
        Point2D(t^2 + 2, t^2 - t + 1)
```

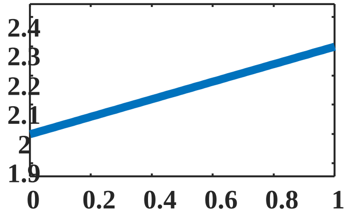


```
function S(u::Real, v::Real)::Point3D =
        Point3D(cos(u), sin(u), v)
```

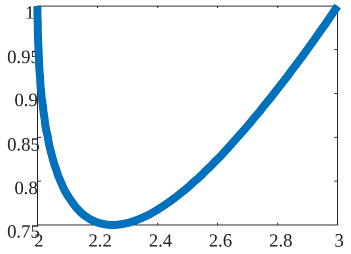


A SHAPE IS A FUNCTION

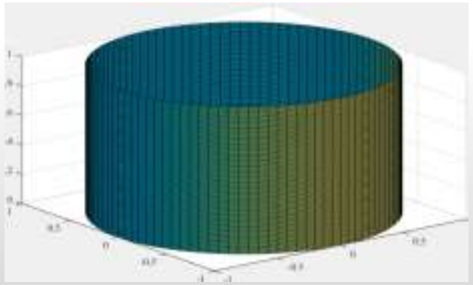
```
function y(x::Interval)::Real = .3*x + 2
```



```
function C(t::Interval)::Point2D =
        Point2D(t^2 + 2, t^2 - t + 1)
```

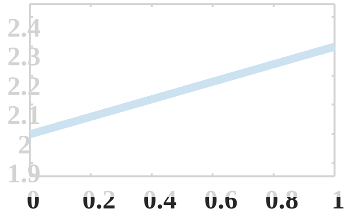


```
function S(u::Interval, v::Real)::Point3D =
        Point3D(cos(u), sin(u), v)
```

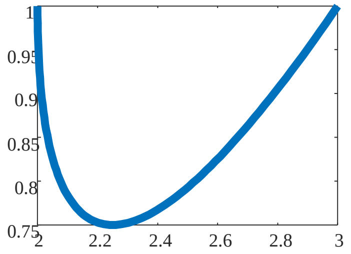


FUNCTIONS OVER DOMAINS Ω

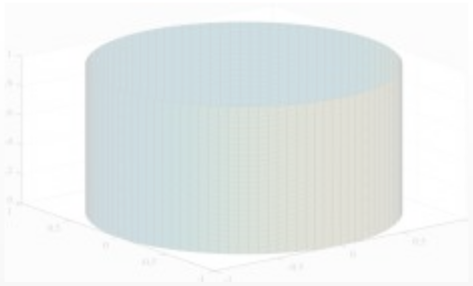
```
function y(x::Interval)::Real = .3*x + 2
```



```
function C(t::Interval)::Point2D =
        Point2D(t^2 + 2, t^2 - t + 1)
```



```
function S(u::Interval, v::Real)::Point3D =
        Point3D(cos(u), sin(u), v)
```

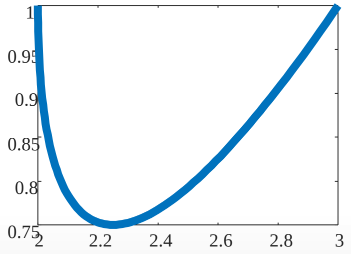


CURVES

Microsoft

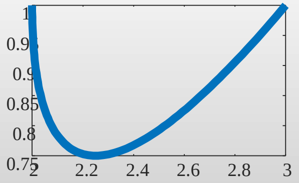
```
abstract Curve {
    method eval(t::Interval)::Point2D
};
```

```
abstract Curve {
      method eval(t::Interval)::Point2D
};


type Conic < Curve {
      eval(t) =
            Point2D(t^2 + 2, t^2 - t + 1)
};
```



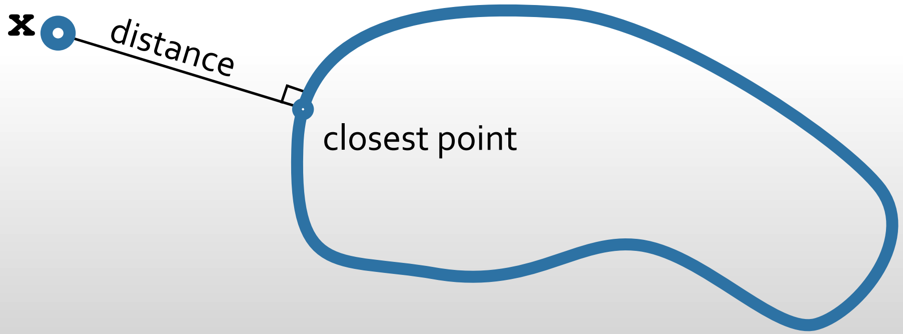
PARAMETERIZED SHAPES

```
abstract Curve {
      method eval(t::Interval)::Point2D
};
type Conic < Curve {
      θ::Real[]; // Shape parameters
      eval(t) =
        Point2D( θ[0]*t^2 + θ[1]*t + θ[2],
                 θ[3]*t^2 + θ[4]*t + θ[5] )
};
Conic([1,0,2,1,-1,1])
```



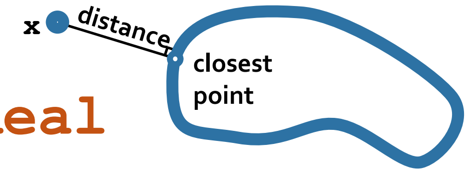
PARAMETERIZED SHAPES

```
abstract Curve {
      method eval(t::Interval)::Point2D

      method distance(x::Point2D)::Real
      method closest_point(x::Point2D)::Point2D
};
```



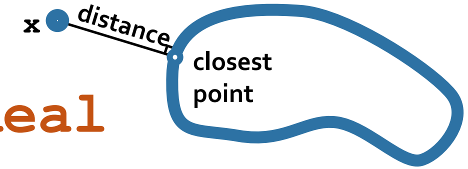
OTHER "METHODS"

```
abstract Curve {
      method eval(t::Interval)::Point2D


      method distance(x::Point2D)::Real
      method closest_point(x::Point2D)::Point2D
};


distance(x) = norm(x - this.closest_point(x))
```



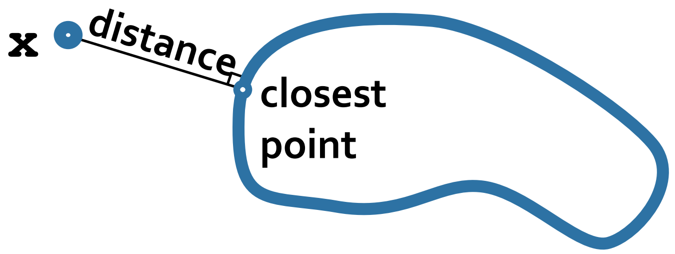
OTHER "METHODS"

```
abstract Curve {
      method eval(t::Interval)::Point2D


      method distance(x::Point2D)::Real
      method closest_point(x::Point2D)::Point2D
};


distance(x) =
      minimize(λ(t) norm(this.eval(t) - x), 0.0)
```



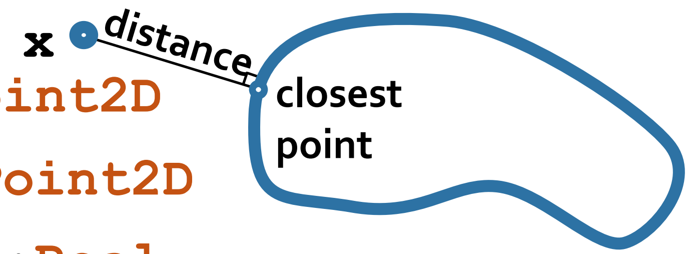
OTHER “METHODS”

```
abstract Curve {
        method eval(t::Interval)::Point2D
        method distance(x::Point2D)::Real
        ...
function f(t) = norm(this.eval(t) − x)^2
distance(x) =  minimize(f, Interval::Min)


function minimize(f, t)
    while not converged
      t -= α * f′(t)   // Compute derivative
```



OTHER “METHODS”

```
abstract Curve {
        method eval(t::Interval)::Point2D
        method eval′(t::Interval)::Point2D
        method distance(x::Point2D)::Real
        method closest_point(x::Point2D)::Point2D
};
```



$$\begin{aligned} y = \text{if } t < a \\ (t-b)^2 + c \\ \text{else} \\ f(t-d)^2 + e \end{aligned}$$

$$\begin{aligned} y' = \text{if } t < a \\ 2(t-b) \\ \text{else} \\ 2f(t-d) \end{aligned}$$

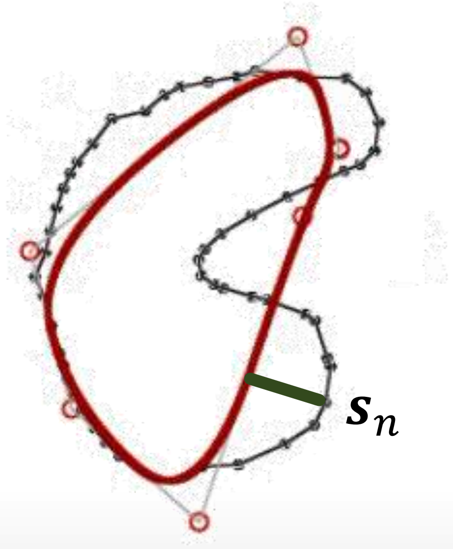
OTHER "METHODS"

Microsoft

Shape, meet thy data

Sum-of-min problems

$$\min_{\theta} \sum_{n=1}^{N} C(\theta).\mathrm{closest\_point}(\boldsymbol{s_n})$$



Microsoft

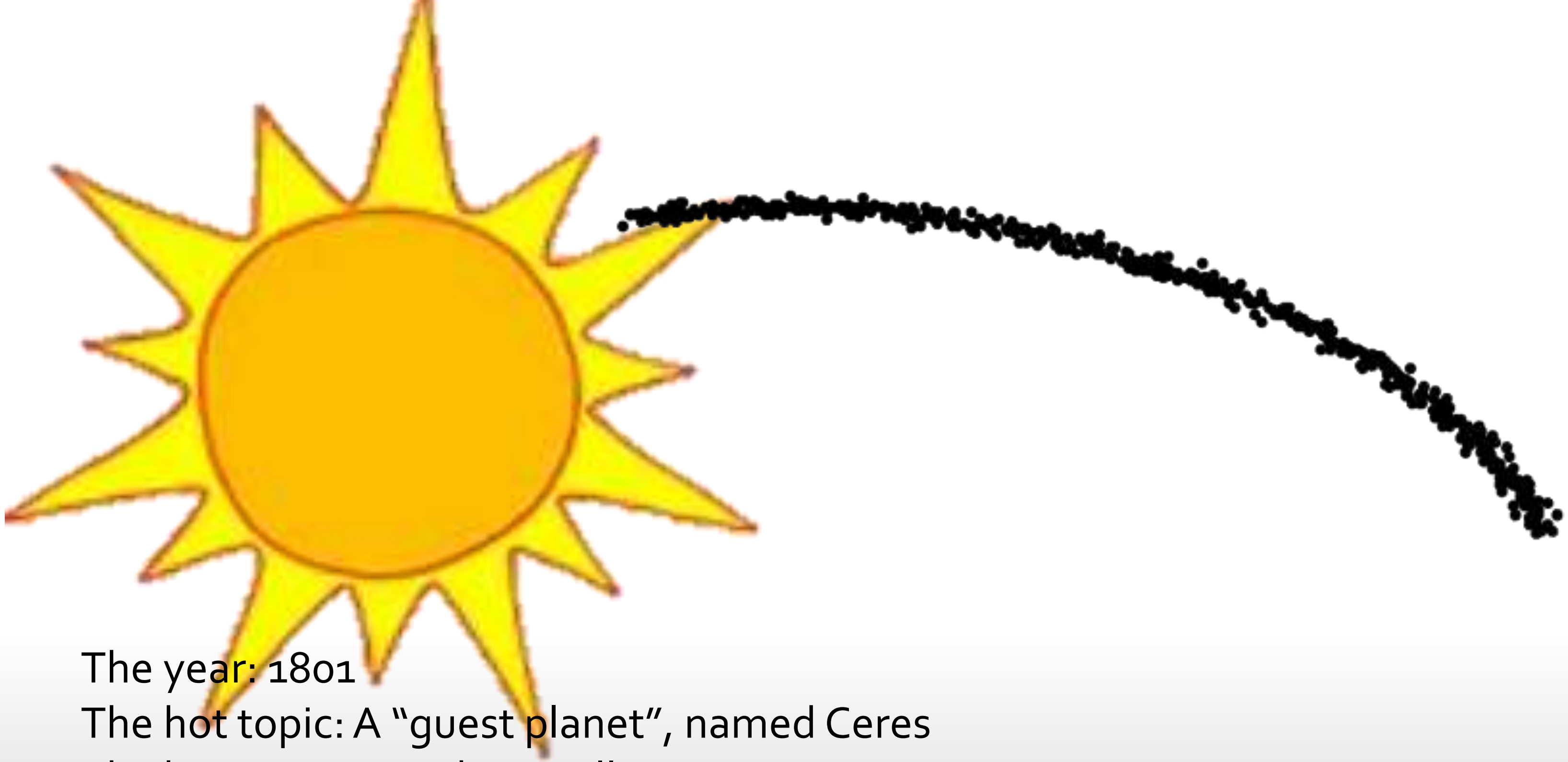
AN EXEMPLARY PROBLEM

“Based on a true story”, not necessarily historically accurate

Note well: this problem is a good proxy for much more realistic problems:

1. Stereo camera calibration
2. Multiple-camera bundle adjustment
3. Surface fitting, e.g. subdivision surfaces to range data, realtime hand tracking
4. Matrix completion
5. Image denoising.

[Inspired by Neil Lawrence’s professorial inaugural]

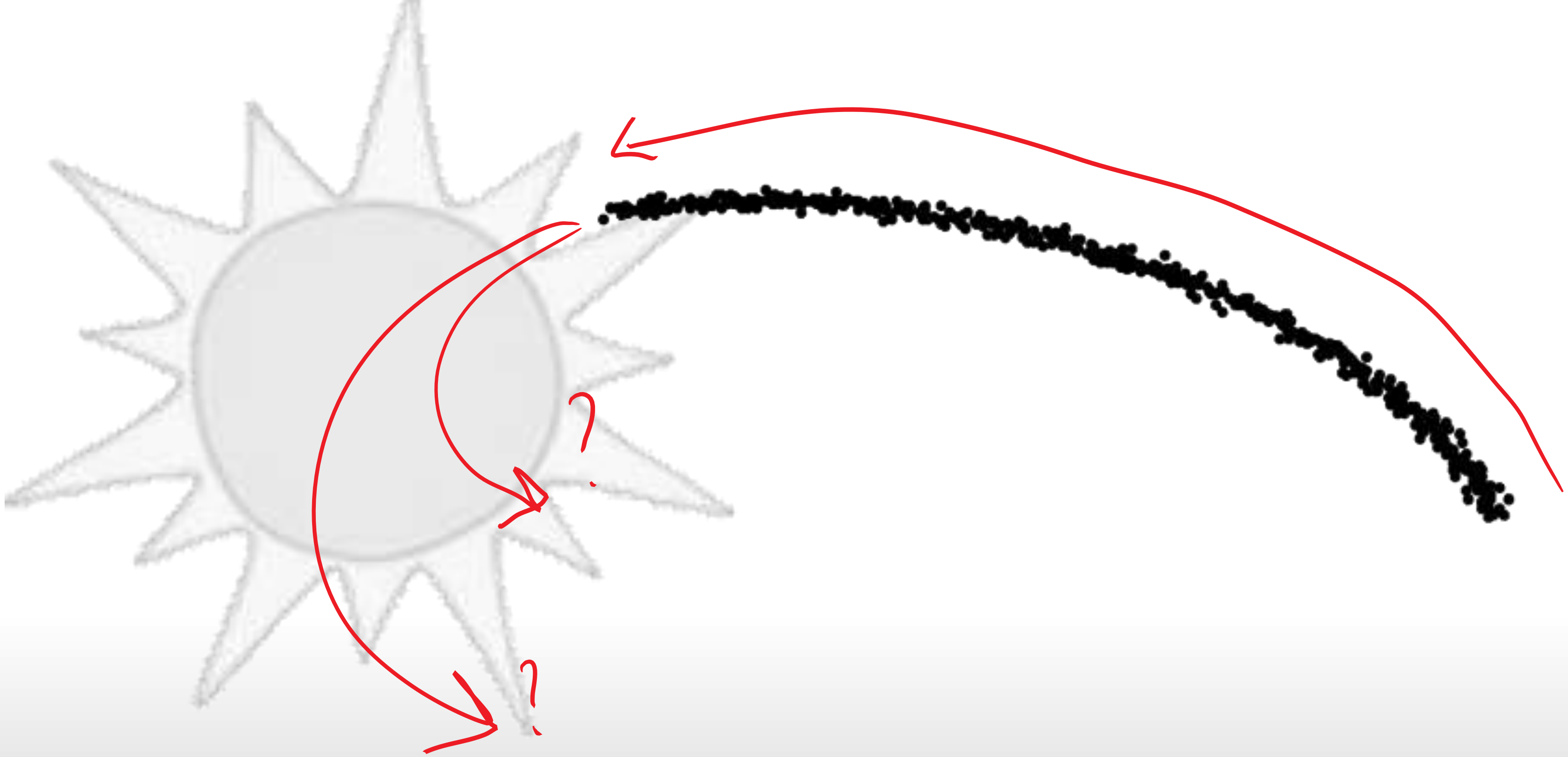


The year: 1801

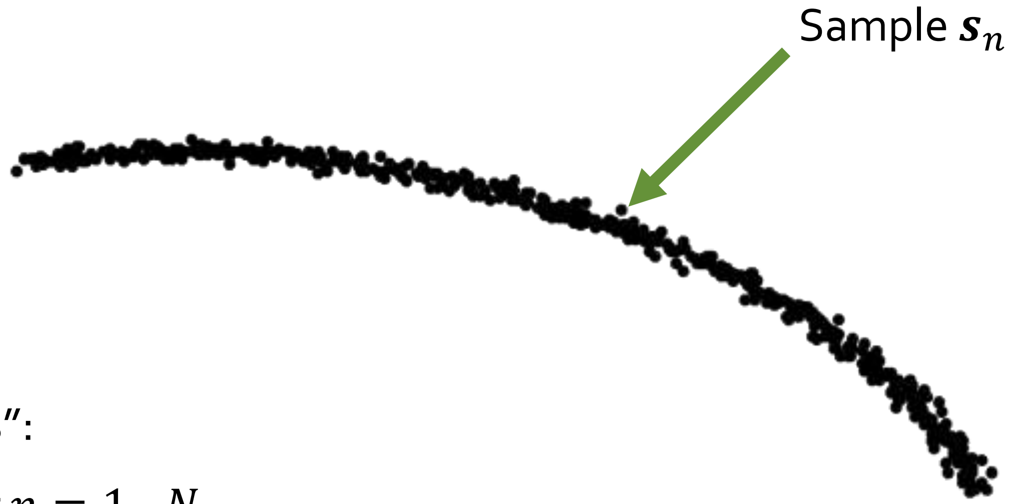
The hot topic: A "guest planet", named Ceres

The big question: Where will it reappear?

AN EXEMPLARY PROBLEM



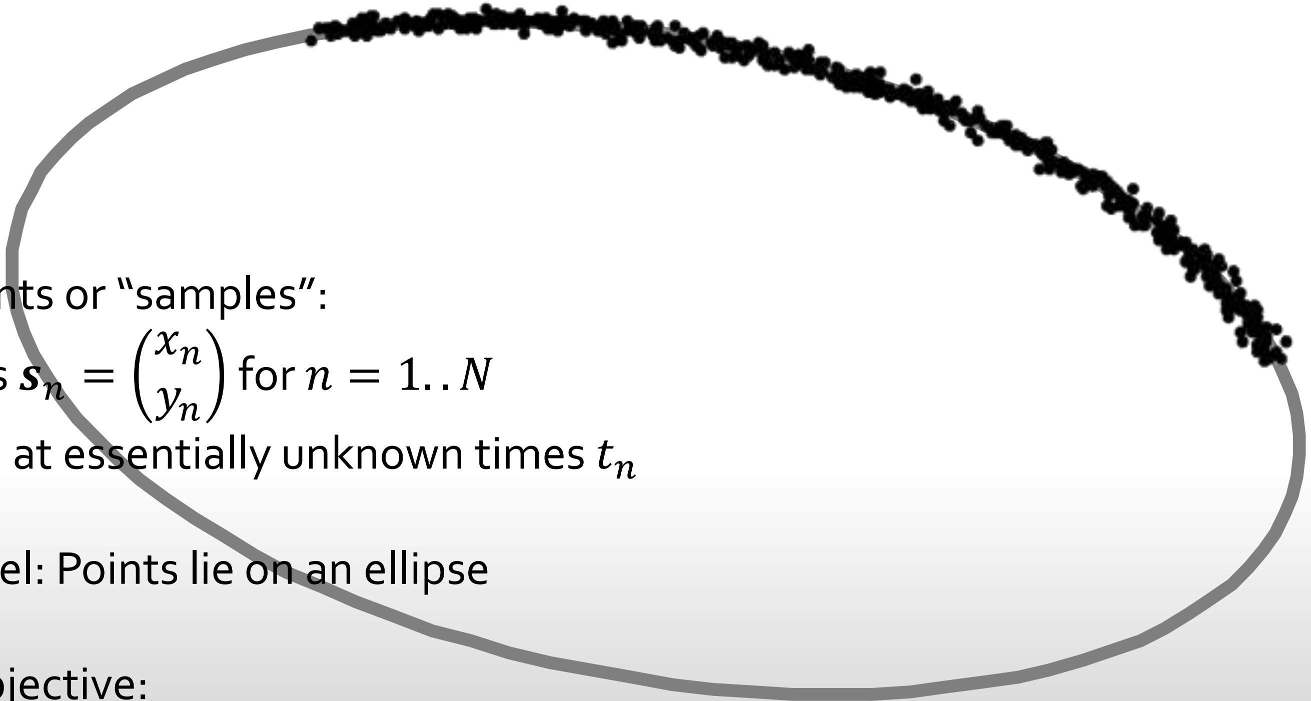
AN EXEMPLARY PROBLEM



Measurements or “samples”:

- 2D points $\boldsymbol{s}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ for $n = 1..N$
- Captured at essentially unknown times t_n

AN EXEMPLARY PROBLEM



Measurements or “samples”:

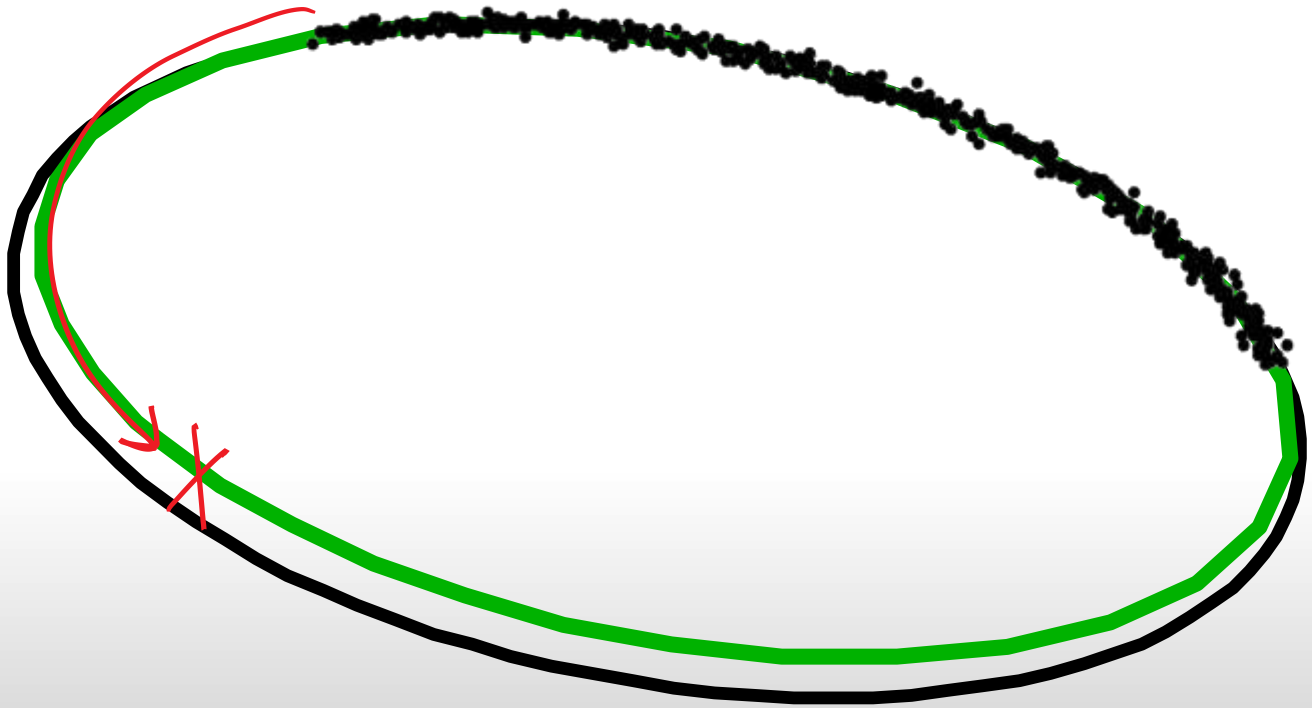
- 2D points $\boldsymbol{s}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ for $n = 1..N$
- Captured at essentially unknown times t_n

Known model: Points lie on an ellipse

Clear(ish) objective:
Estimate the ellipse parameters, intersect with circle of sun, achieve fame

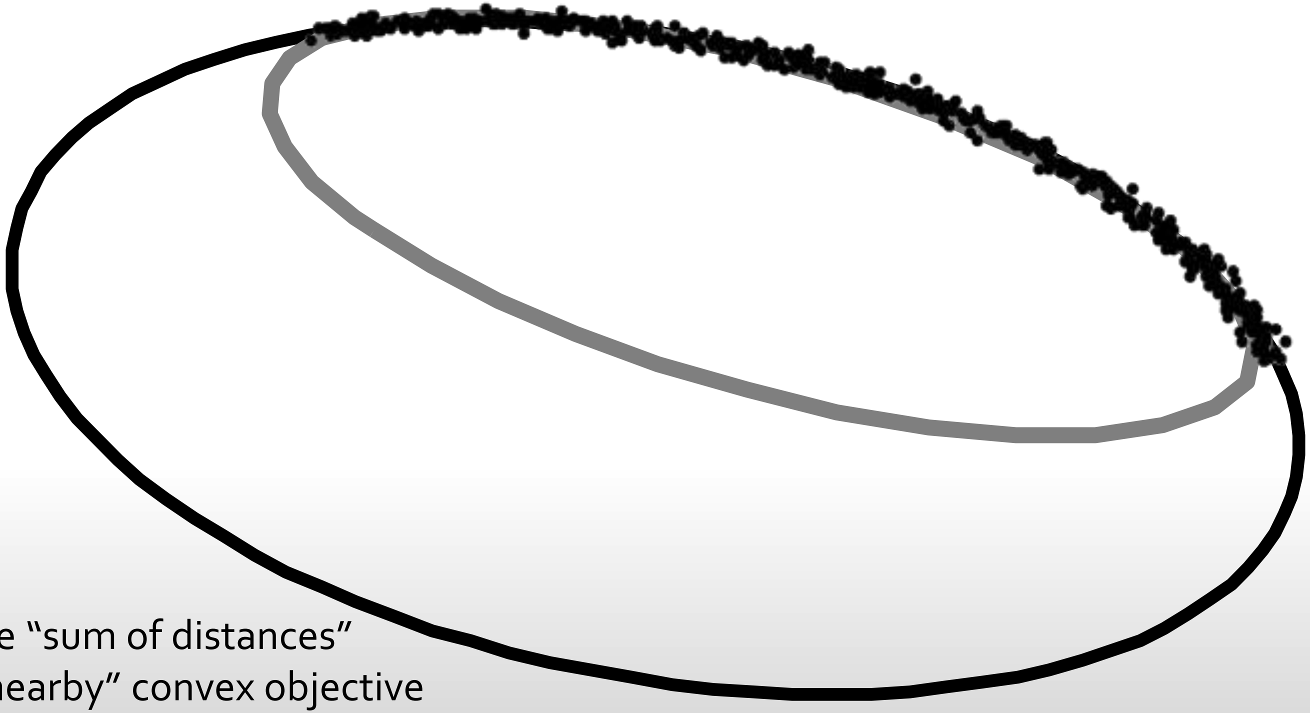
WE KNOW THE EXACT FORM OF THE MODEL

Microsoft



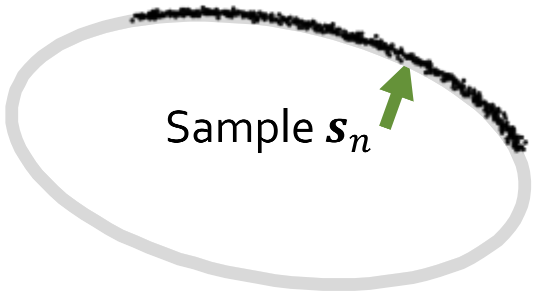
AND ESTIMATING IT WELL GETS US CLOSE…

“Direct least squares fitting of ellipses”
[Fitzgibbon et al, 1999]



Does not minimize “sum of distances” objective, but a “nearby” convex objective

RUNNING AN OFF-THE-SHELF FITTER DOES NOT.



Measurements or "samples":

- 2D points $\boldsymbol{s}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ for $n = 1..N$
- Captured at unknown times t_n

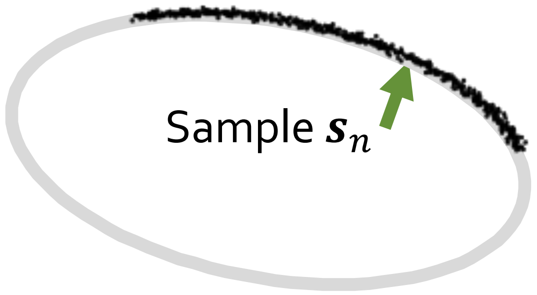
Known model: Points lie on an ellipse

$$\boldsymbol{s}_n = \boldsymbol{c}(t_n; \boldsymbol{\theta}) + Noise$$

$$\boldsymbol{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

WE KNOW THE EXACT FORM OF THE MODEL

Microsoft



$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\boldsymbol{s}_n = \boldsymbol{c}(t_n;\boldsymbol{\theta}) + Noise$$

A parametric description

$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

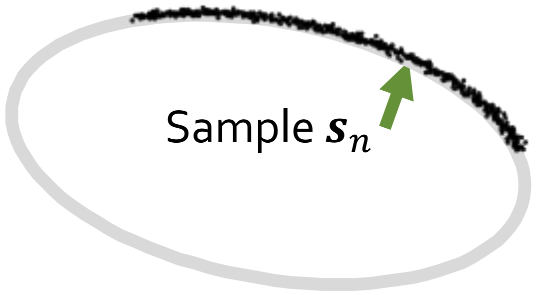
Defines a **curve** (a set of points in $\mathbb{R}^2$)

$$C(\boldsymbol{\theta}) = \{\boldsymbol{c}(t;\boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

Potential confusion: curve parameter t and shape parameter vector $\boldsymbol{\theta}$. This should be ok for this talk.

A PARAMETRIC FUNCTION AND A CURVE

Microsoft



$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\boldsymbol{s}_n = \boldsymbol{c}(t_n;\boldsymbol{\theta}) + Noise$$

$$C(\boldsymbol{\theta}) = \{\boldsymbol{c}(t;\boldsymbol{\theta}) \mid 0 < t \le 2\pi\}$$

All our algorithms will start with a guess of $\boldsymbol{\theta}$ and refine it.

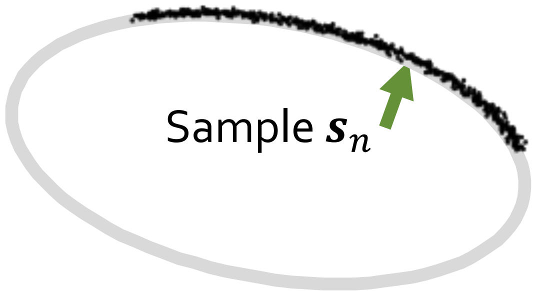
We will often want to think about the *distance* of a sample $\boldsymbol{s}$ from the curve $C(\boldsymbol{\theta})$.

Often, *closest point* is appropriate.
[Others easily handled too.]

$$D(\boldsymbol{s},\boldsymbol{\theta}) := \min_{\boldsymbol{x}\in C(\boldsymbol{\theta})} \|\boldsymbol{s}-\boldsymbol{x}\|^2$$

$$D(\boldsymbol{s},\boldsymbol{\theta}) := \min_{t} \|\boldsymbol{s}-\boldsymbol{c}(t;\boldsymbol{\theta})\|^2$$

DISTANCES AND CLOSEST POINTS



$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\boldsymbol{s}_n = \boldsymbol{c}(t_n;\boldsymbol{\theta}) + Noise$$

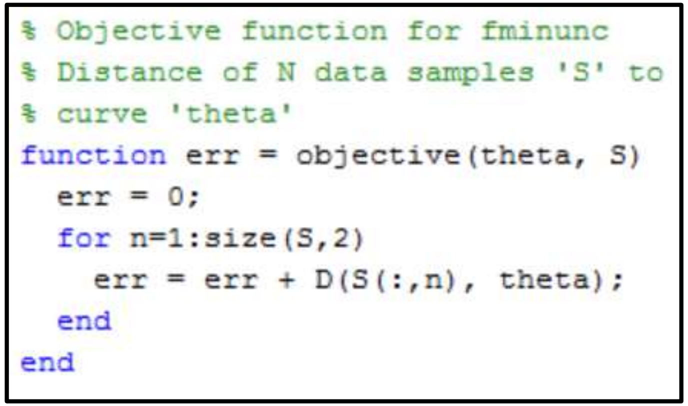
$$C(\boldsymbol{\theta}) = \{\boldsymbol{c}(t;\boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

$$D(\boldsymbol{s},\boldsymbol{\theta}) := \min_t \|\boldsymbol{s} - \boldsymbol{c}(t;\boldsymbol{\theta})\|^2$$

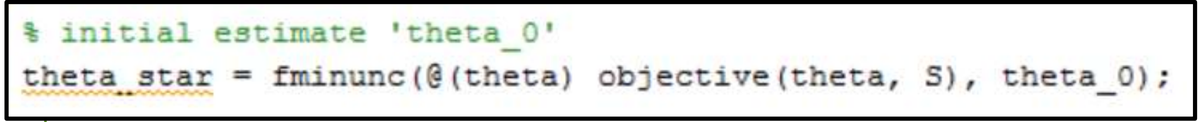
Minimize over all ellipses $\boldsymbol{\theta}$

$$\boldsymbol{\theta}^* := \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_n D(\boldsymbol{s}_n, \boldsymbol{\theta})$$

Just using an off-the-shelf optimizer.



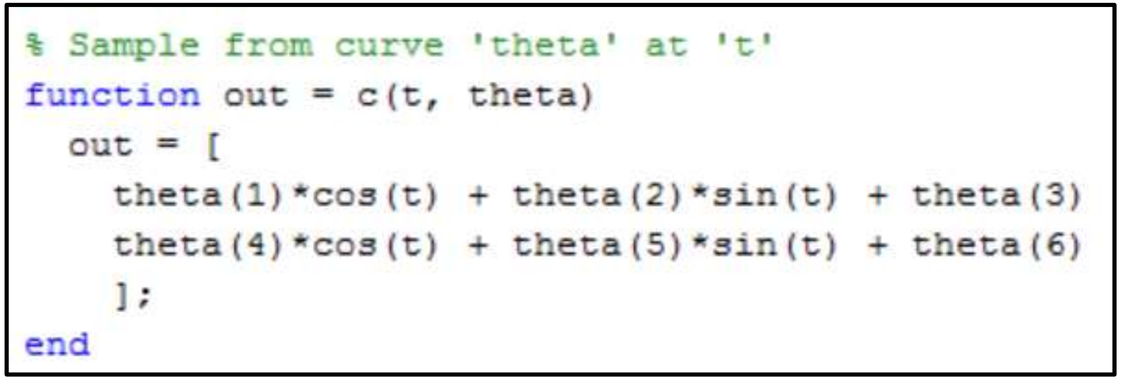
```
% Objective function for fminunc
% Distance of N data samples 'S' to
% curve 'theta'
function err = objective(theta, S)
  err = 0;
  for n=1:size(S,2)
    err = err + D(S(:,n), theta);
  end
end
```



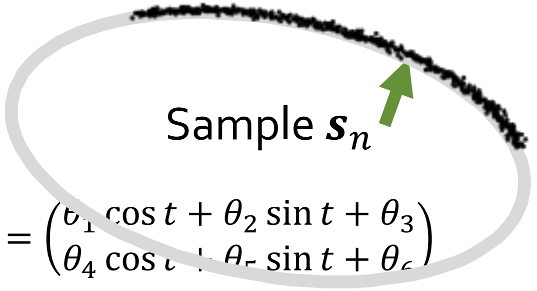
```
% initial estimate 'theta_0'
theta_star = fminunc(@(theta) objective(theta, S), theta_0);
```

A BETTER ESTIMATE

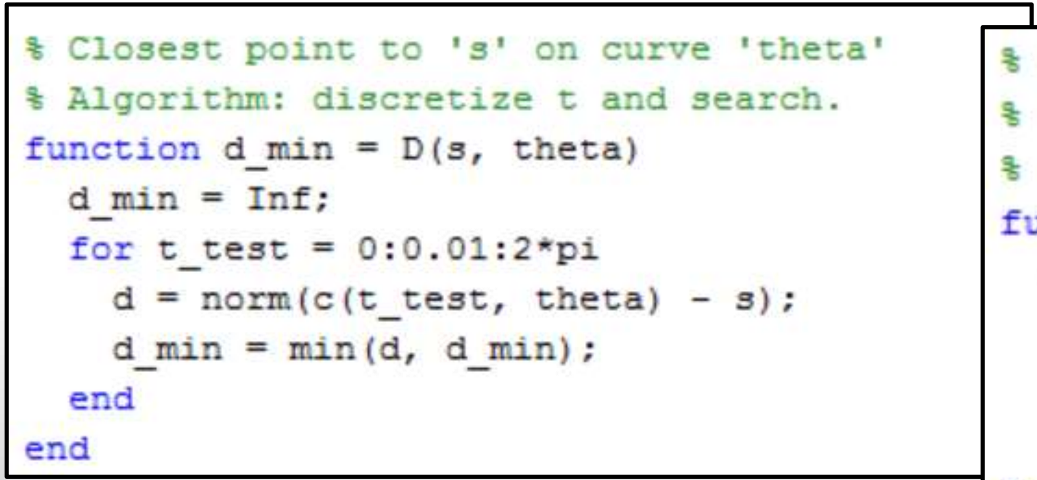
Microsoft



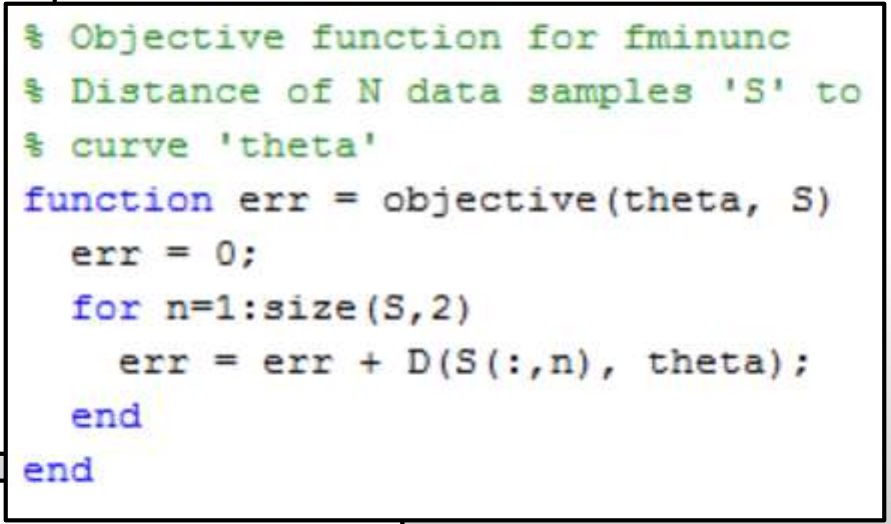
```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
  out = [
    theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
    theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
    ];
end
```



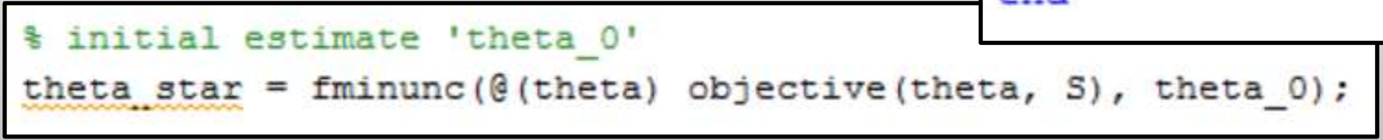
$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix}\theta_1\cos t + \theta_2 \sin t + \theta_3\\ \theta_4 \cos t + \theta_5 \sin t + \theta_6\end{pmatrix}$$



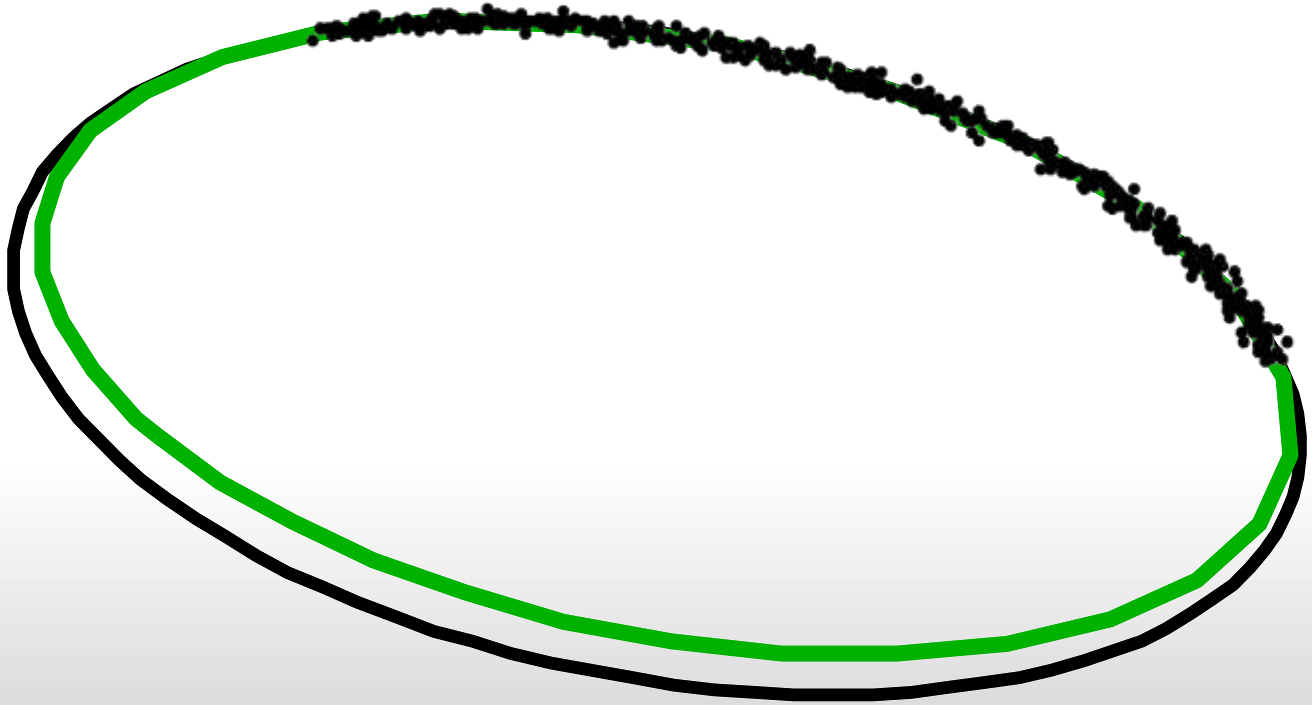
```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
  d_min = Inf;
  for t_test = 0:0.01:2*pi
    d = norm(c(t_test, theta) - s);
    d_min = min(d, d_min);
  end
end
```



```
% Objective function for fminunc
% Distance of N data samples 'S' to
% curve 'theta'
function err = objective(theta, S)
  err = 0;
  for n=1:size(S,2)
    err = err + D(S(:,n), theta);
  end
end
```



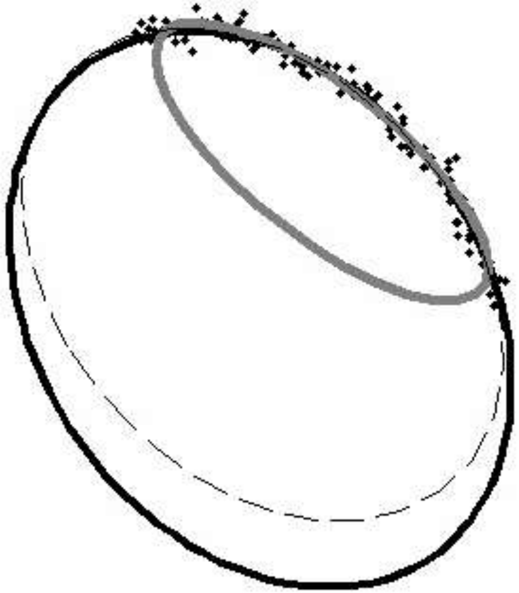
```
% initial estimate 'theta_0'
theta_star = fminunc(@(theta) objective(theta, S), theta_0);
```



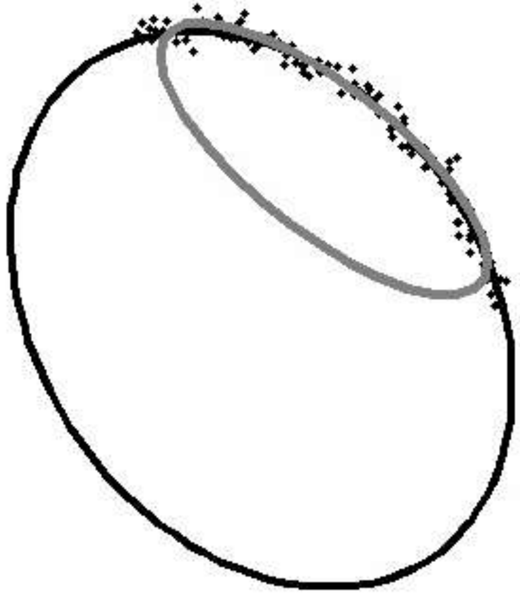
AND ESTIMATING IT WELL GETS US CLOSE…

- We have an accurate solution
  - Certainly better than the “closed form” algorithm, which minimized a “nearby” convex objective.
- All we need to worry about now is speed…
  - If you take 3 weeks to make a prediction, someone else will get the fame.
  - Speed *is* everything. If speed didn’t matter, you would just use random search.
- Strategies to speed it up
  - Attack the inner loop
    - Remove discrete minimization in $D(\boldsymbol{s}, \boldsymbol{\theta})$
  - Analyse the problem again
  - Understand our tools: ‘fminunc’, or whatever we’re using
  - Compute analytic derivatives

SO ARE WE DONE YET?



A slow method



A fast method, slowed down 10x

SPEED RESULTS: SNEAK PREVIEW

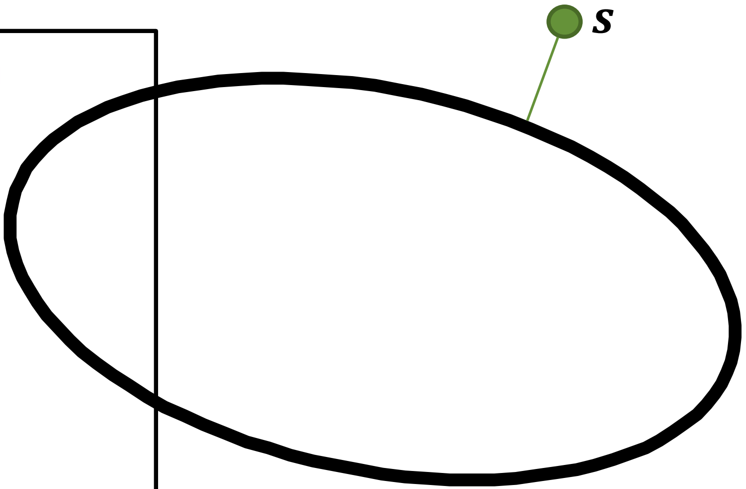
Microsoft

SPEEDUP 1: ATTACK THE INNER LOOP

```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
  out = [
    theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
    theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
en
```

```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
  d_min = Inf;
  for t_test = 0:0.01:2*pi
    d = norm(c(t_test, theta) - s);
    d_min = min(d, d_min);
  end
end
```

```
%
theta_star = fminunc(@(theta) objective(theta, S), theta_0);
```



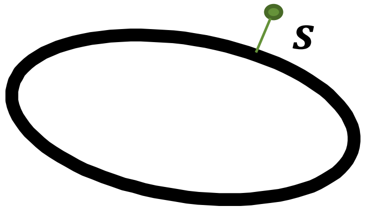
SPEEDUP 1: ATTACK THE INNER LOOP

```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
  out = [
    theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
    theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
    ];
end
```

$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix}\theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6\end{pmatrix}$$

```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
  d_min = Inf;
  for t_test = 0:0.01:2*pi
    d = norm(c(t_test, theta) - s);
    d_min = min(d, d_min);
  end
end
```

$$D(\boldsymbol{s},\boldsymbol{\theta}) = \min_t \|\boldsymbol{s} - \boldsymbol{c}(t;\boldsymbol{\theta})\|^2$$



Define $E(t) = \|\boldsymbol{s} - \boldsymbol{c}(t;\boldsymbol{\theta})\|^2$

Set $\frac{dE}{dt} = 0$

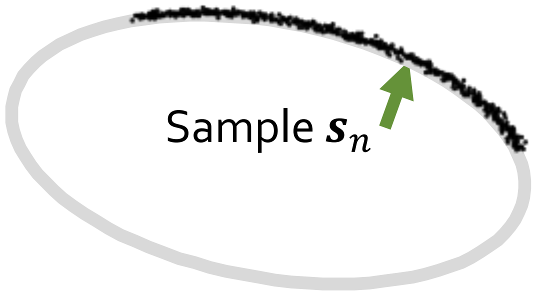
Yields 4th order polynomial, extract 4 roots.

Much cheaper than previous implementation.

SPEEDUP 1: ATTACK THE INNER LOOP

Microsoft

SPEEDUP 2: ANALYSE THE PROBLEM



$$\boldsymbol{c}(t;\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\boldsymbol{s}_n = \boldsymbol{c}(t_n;\boldsymbol{\theta}) + Noise$$

$$C(\boldsymbol{\theta}) = \{\boldsymbol{c}(t;\boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

$$D(\boldsymbol{s},\boldsymbol{\theta}) \coloneqq \min_t \|\boldsymbol{s} - \boldsymbol{c}(t;\boldsymbol{\theta})\|^2$$

$$\boldsymbol{\theta}^* \coloneqq \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_n D(\boldsymbol{s}_n,\boldsymbol{\theta})$$

Minimize over all ellipses $\boldsymbol{\theta}$

$$\sum_n D(\boldsymbol{s}_n,\boldsymbol{\theta}) = \sum_n \min_t \|\boldsymbol{s}_n - \boldsymbol{c}(t;\boldsymbol{\theta})\|^2$$

Notice $\boldsymbol{c}(t;\boldsymbol{\theta})$ is linear in $\boldsymbol{\theta}$, so function is

$$= \sum_n \min_{t_n} \|\boldsymbol{s}_n - A(t_n)\boldsymbol{\theta}\|^2$$

And we can solve in closed form:

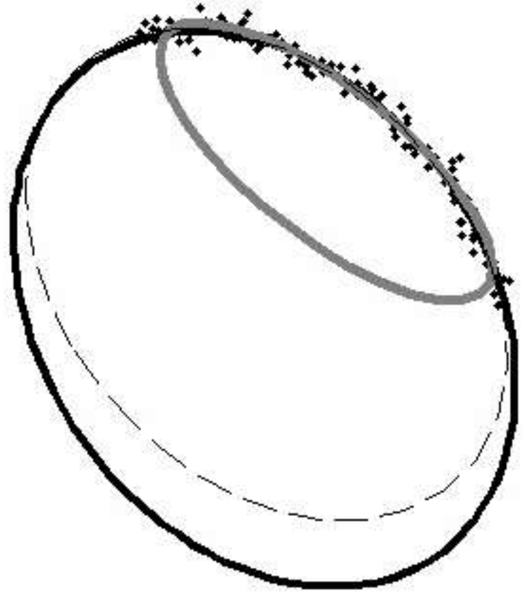
- for $\mathrm{T} = \{t_n\}_{n=1}^N$ given $\boldsymbol{\theta}$. Cost N `RootOf`s.
- and $\boldsymbol{\theta}$ given T. Cost one linear solve.

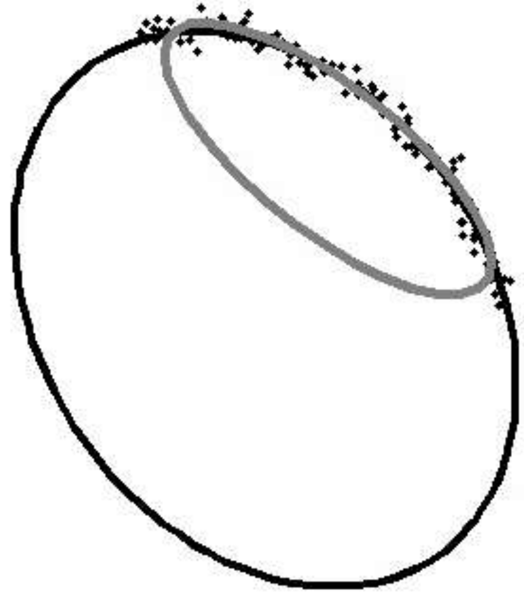
So alternate—"ICP", "EM", "Block Coordinate Descent"

LOOK AT THE PROBLEM AGAIN

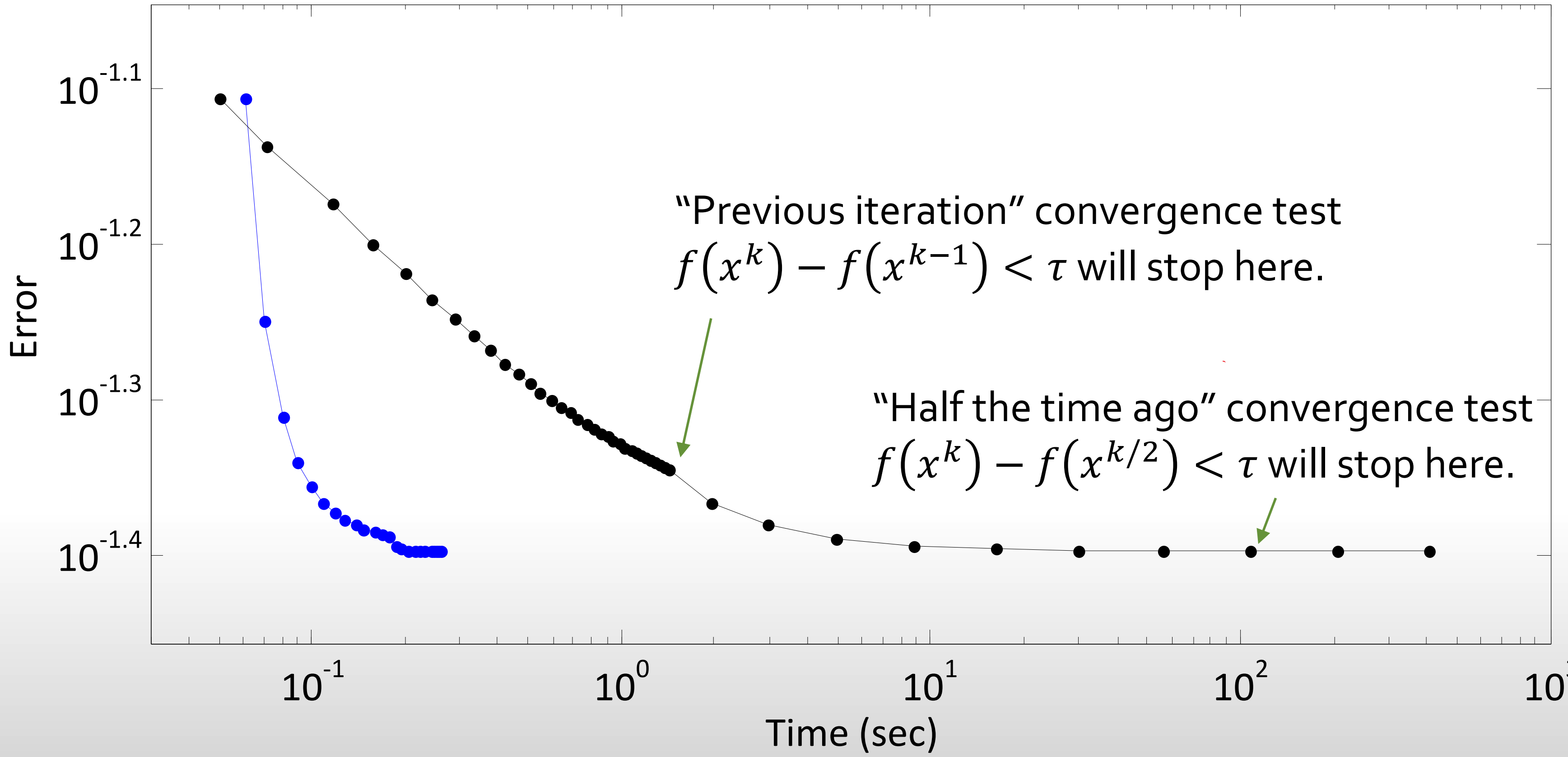
Microsoft

bad decision…


ICP, a bad 1st-order method


A second order method, slowed down 10x

CONVERGENCE RATES

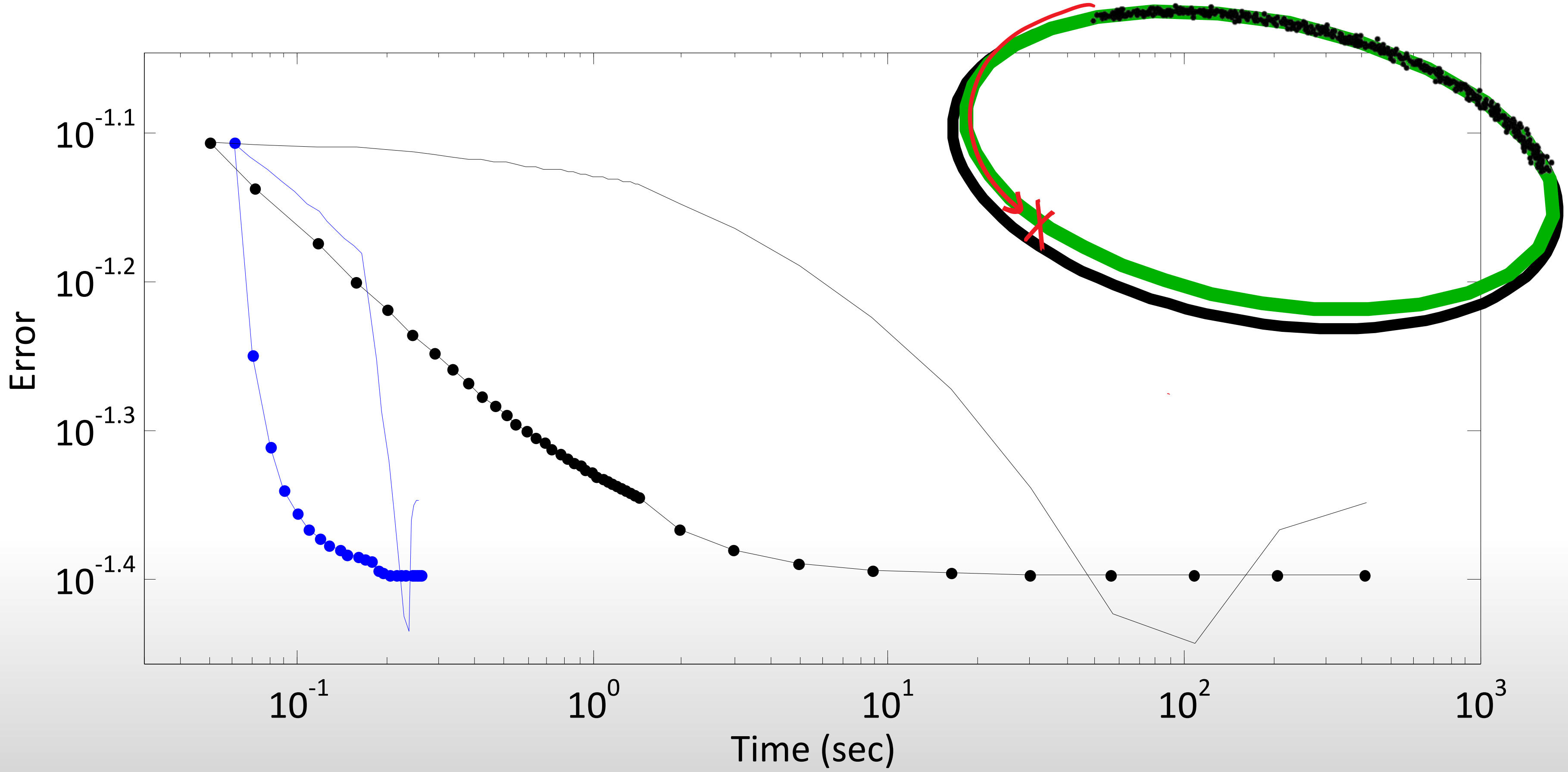


"Previous iteration" convergence test $f\left(x^k\right) - f\left(x^{k-1}\right) < \tau$ will stop here.

"Half the time ago" convergence test $f\left(x^k\right) - f\left(x^{k/2}\right) < \tau$ will stop here.

CONVERGENCE CURVES

Microsoft



AH, BUT WHAT ABOUT TEST ERROR?

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \sum_{n=1}^{N} \min_{u} f_n(u, \theta)$$

$$\begin{aligned}\hat{\theta} &= \operatorname*{argmin}_{\theta} \sum_{n=1}^{N} \min_{u} f_n(u, \theta) \\ &= \operatorname*{argmin}_{\theta} \sum_{n} \min_{u_n} f_n(u_n, \theta)\end{aligned}$$

$$\begin{aligned}\hat{\theta} &= \operatorname*{argmin}_{\theta} \sum_{n=1}^{N} \min_{t} f_n(u,\theta) \\ &= \operatorname*{argmin}_{\theta} \sum_{n} \min_{u_n} f_n(u_n,\theta) \\ &= \operatorname*{argmin}_{\theta} \min_{u_{1..N}} \sum_{n} f_n(u_n,\theta)\end{aligned}$$

[Recall that: $\min_x f(x) + \min_y g(y) = \min_{x,y} f(x) + g(y)$]

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \sum_{n=1}^{N} \min_{\boldsymbol{u}} f_n(u, \theta)$$

- Nasty objective
- M parameters

- Cost per iteration $O(N)$

Slow

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \min_{\boldsymbol{u}_{1..N}} \sum_{n} f_n(u_n, \theta)$$

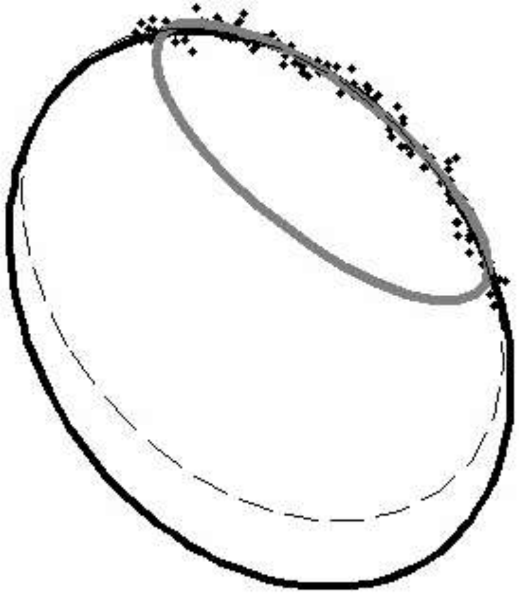
- Simple objective (no “min”)
- $M + N$ parameters

- Cost per iteration $O(NM^r)$

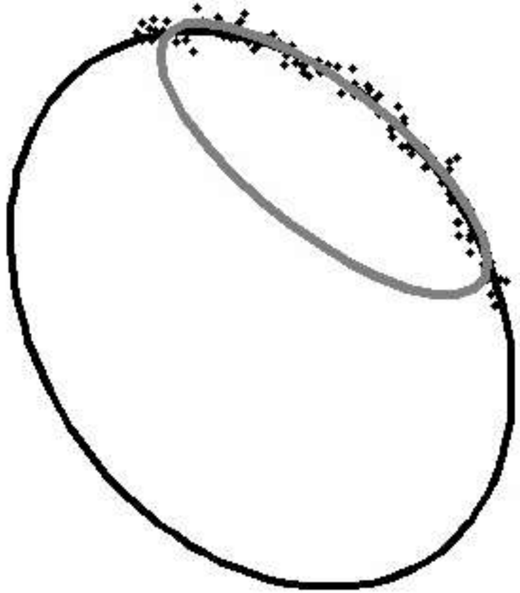
Fast

(in actual real-world wall clock time, even for very large N)

Microsoft



ICP, a bad 1st-order method



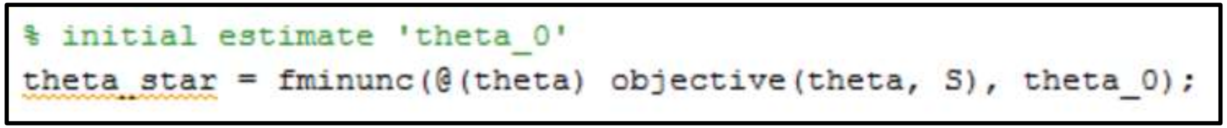
A second order method, slowed down 10x

CONVERGENCE RATES

Microsoft

SPEEDUP 3: UNDERSTAND OUR TOOLS

Microsoft



```
% initial estimate 'theta_0'
theta_star = fminunc(@(theta) objective(theta, S), theta_0);
```

Matlab's fminunc is one of many nonlinear optimizers.

Takes function $f(\boldsymbol{x}): \mathbb{R}^d \mapsto \mathbb{R}$, initial estimate $\boldsymbol{x}_0$

General "trust-region" class of strategies repeats:

- Compute update $\boldsymbol{\delta}_k$ to current guess $\boldsymbol{x}_k$
  - Using function, derivatives, "trust region radius", herbs, spices, …
- If update produces lower f value
  - "**accept**": update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{\delta}_k$

  Else
  - "**reject**": fiddle with "trust region radius"

ASIDE…

CONTINUOUS OPTIMIZATION

Andrew Fitzgibbon

Microsoft Research Cambridge

Microsoft

Given function

$$f(x): \mathbb{R}^d \mapsto \mathbb{R},$$

Devise strategies for finding x which minimizes f

- Gradient descent++: Stochastic, Block, Minibatch
- Coordinate descent++: Block
- Newton++: Gauss, Quasi, Damped, Levenberg Marquardt, dogleg, Trust region, Doublestep LM, [L-]BFGS, Nonlin CG

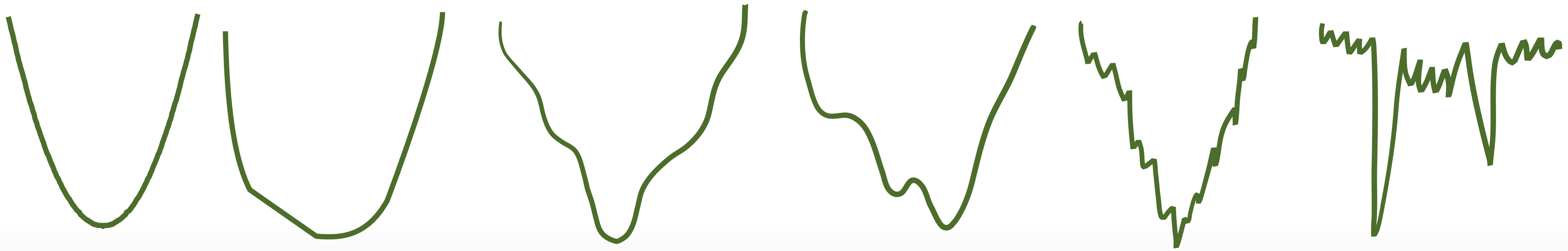
- Not covered
    - Proximal methods: Nesterov, ADMM…

GOAL

Given function

$$f(x): \mathbb{R}^d \mapsto \mathbb{R}$$

Devise strategies for finding x which minimizes f



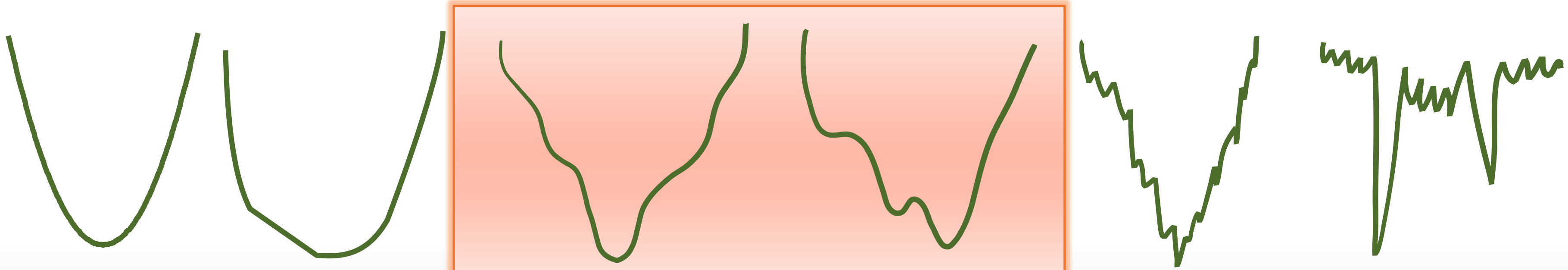
quadratic **convex** **quasiconvex** **multi-extremum** **noisy** **horrible**

CLASSES OF FUNCTIONS

Given function

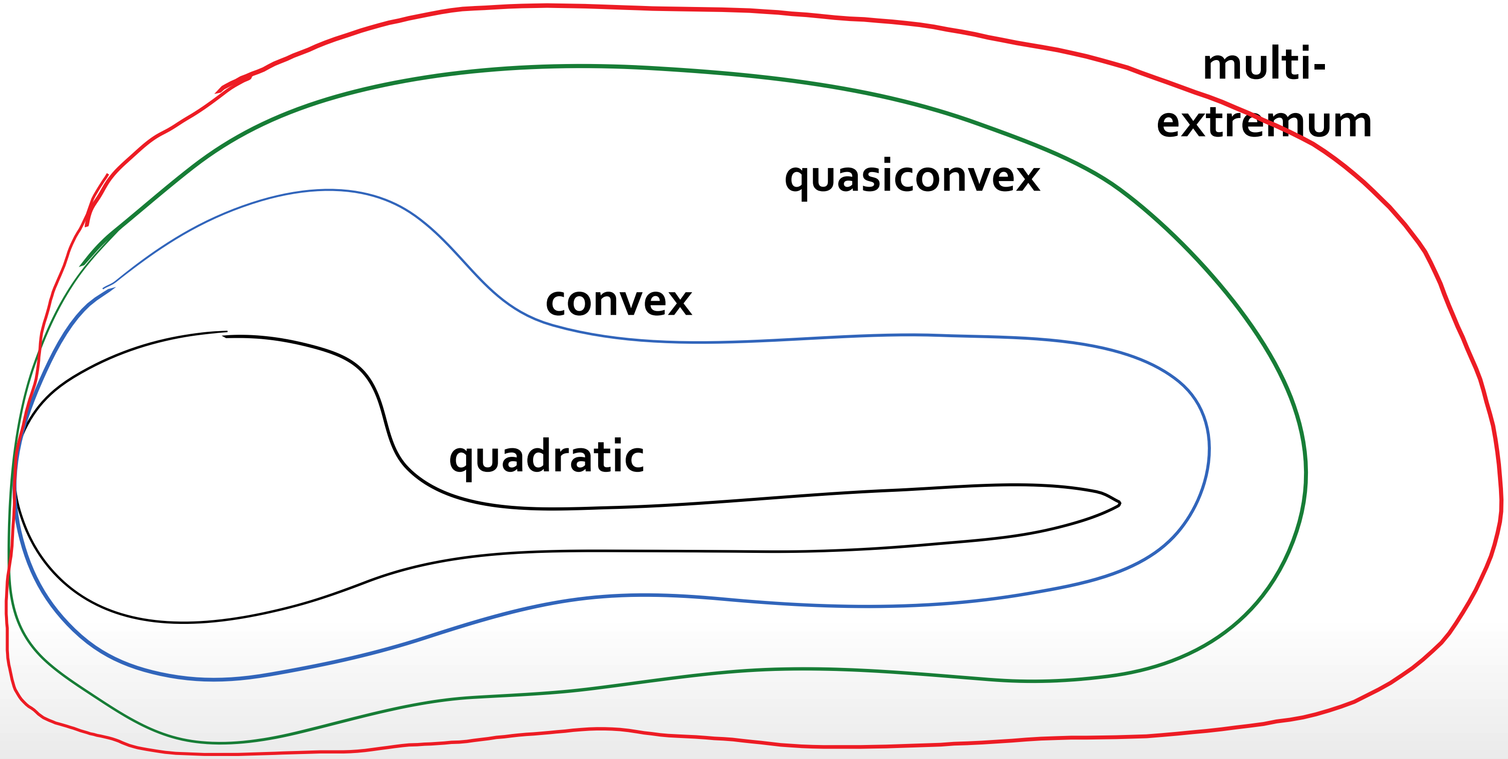
$$f(x)\colon \mathbb{R}^d \mapsto \mathbb{R}$$

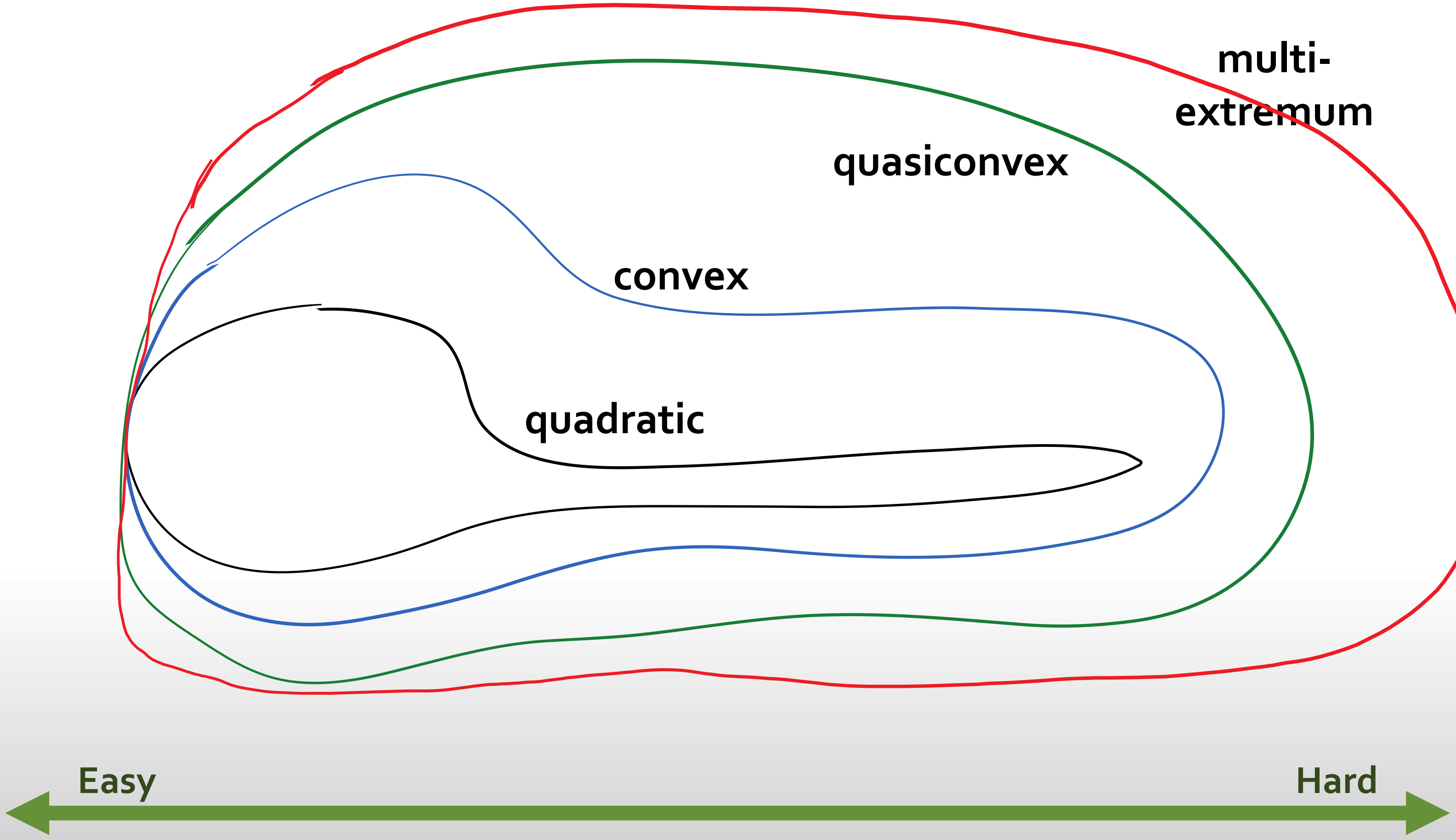
Devise strategies for finding x which minimizes f



quadratic convex quasiconvex multi-extremum noisy horrible

CLASSES OF FUNCTIONS

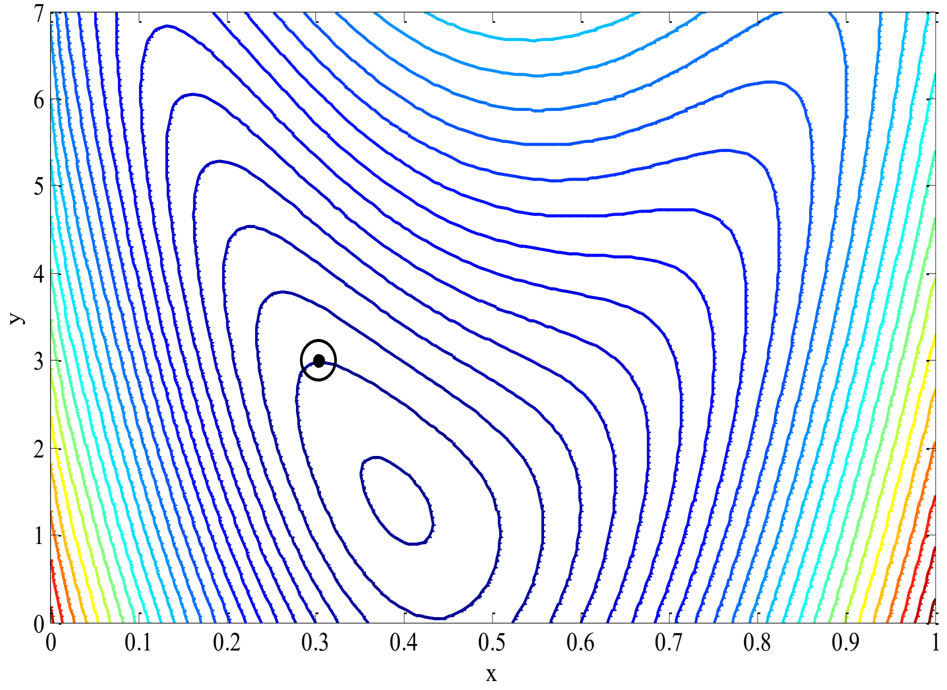

multi-
extremum
quasiconvex
convex
quadratic


multi-
extremum
quasiconvex
convex
quadratic
Easy
Hard

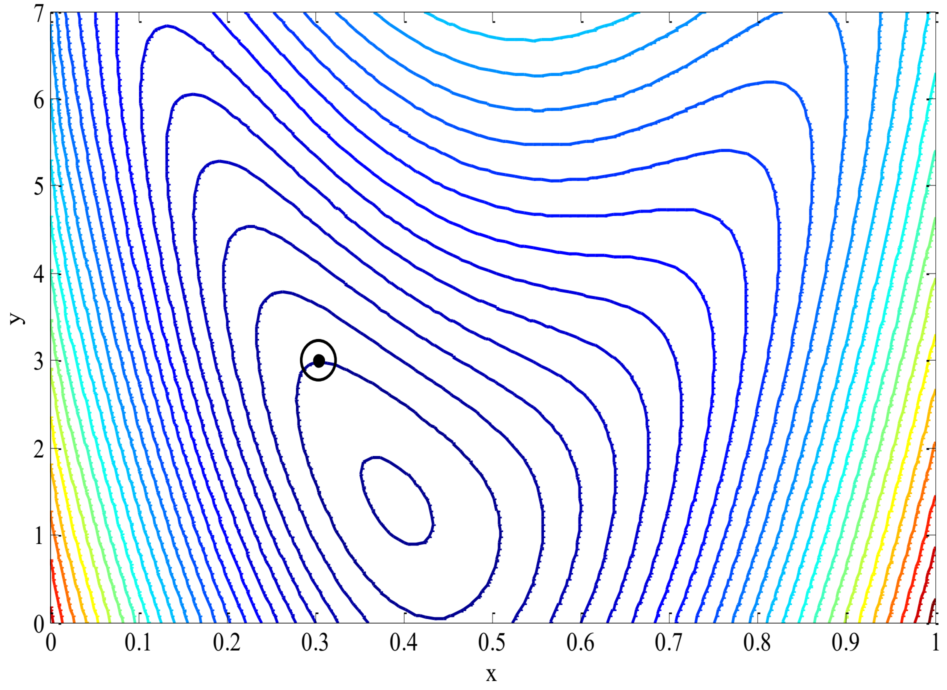
- Fast minimization depends on derivatives

DERIVATIVES

Microsoft

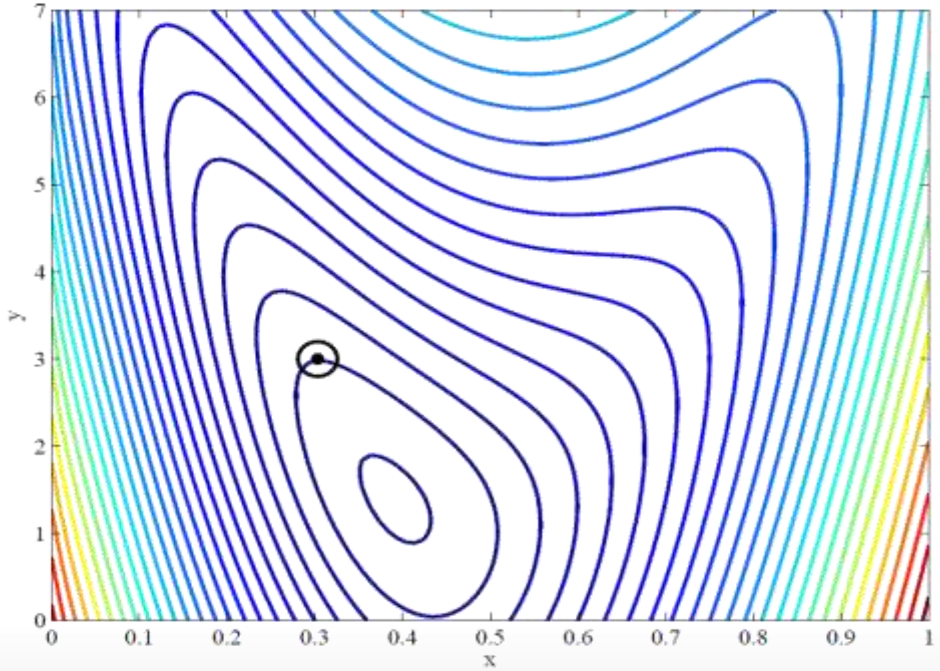


EXAMPLE



```
>> print -dmeta
```

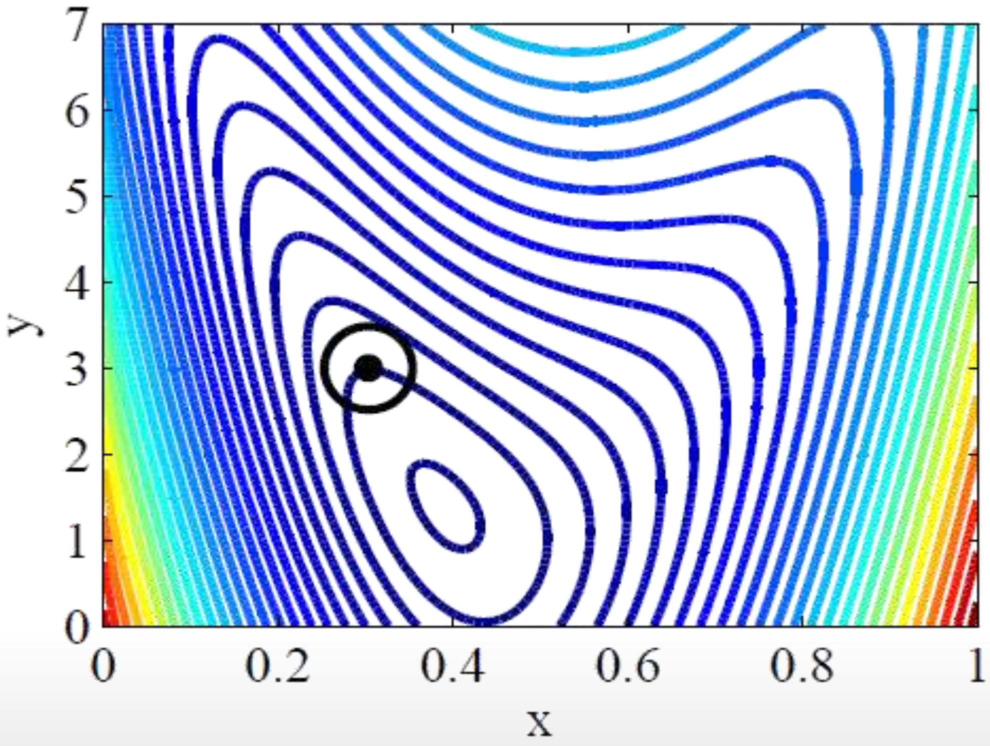
EXAMPLE



```
>> print –dpdf % then go to pdf and paste
OR
>> set(findobj(1, 'type', 'line'), 'linesmoothing', 'on') % then screengrab
```

EXAMPLE

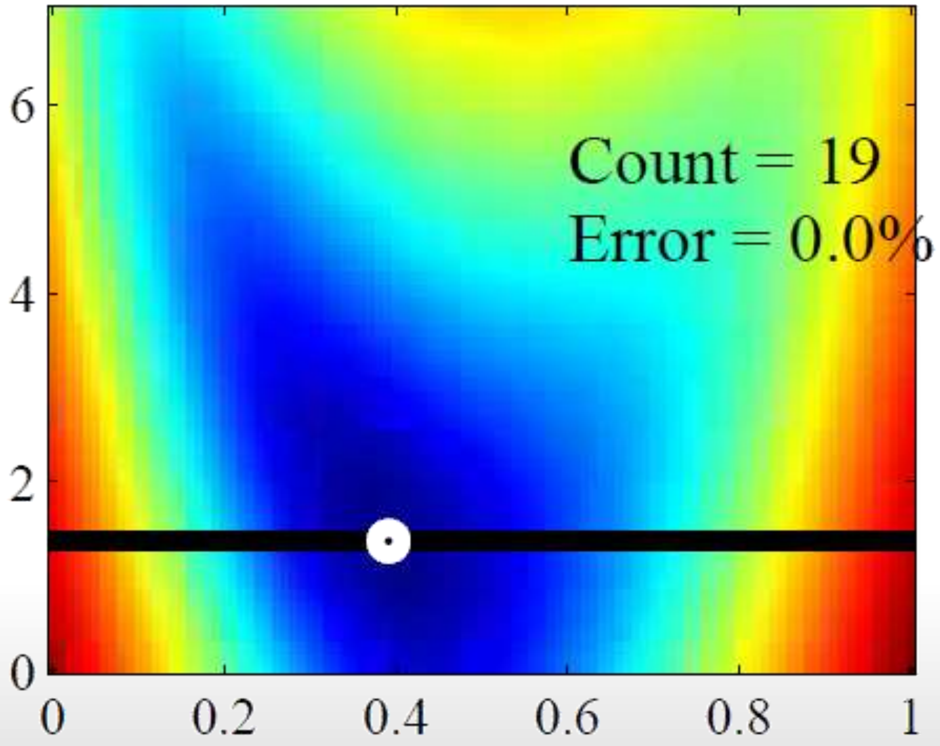
Microsoft



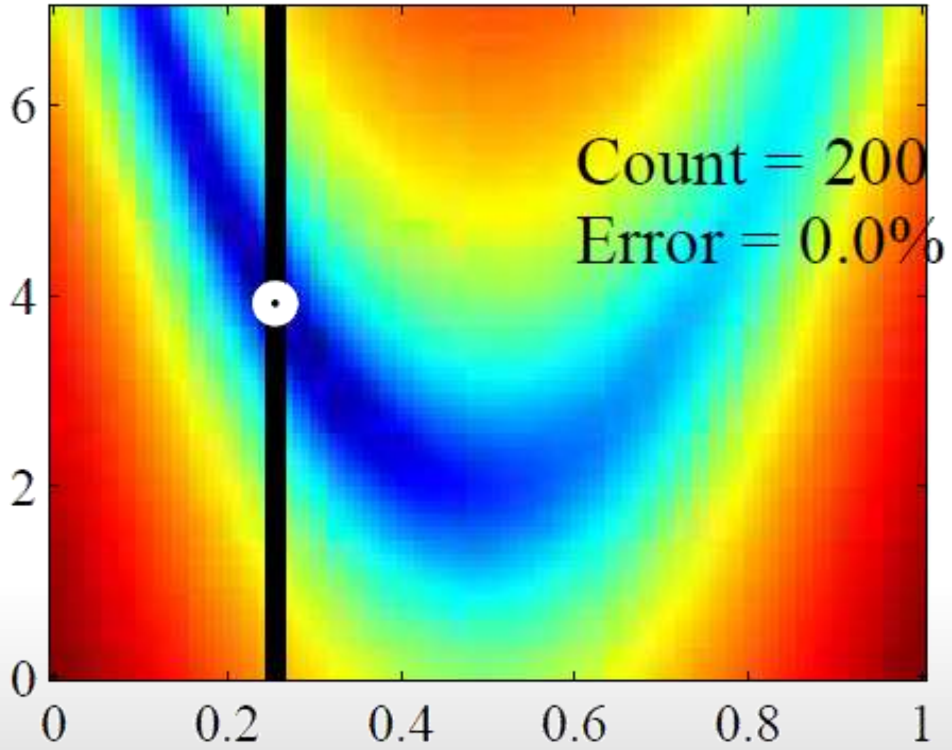
```
>> set(gcf, 'paperUnits', 'centimeters', 'paperposition', [1 1 9 6.6])
>> print –dpdf % then go to pdf and paste
```

EXAMPLE

SWITCH TO MATLAB…

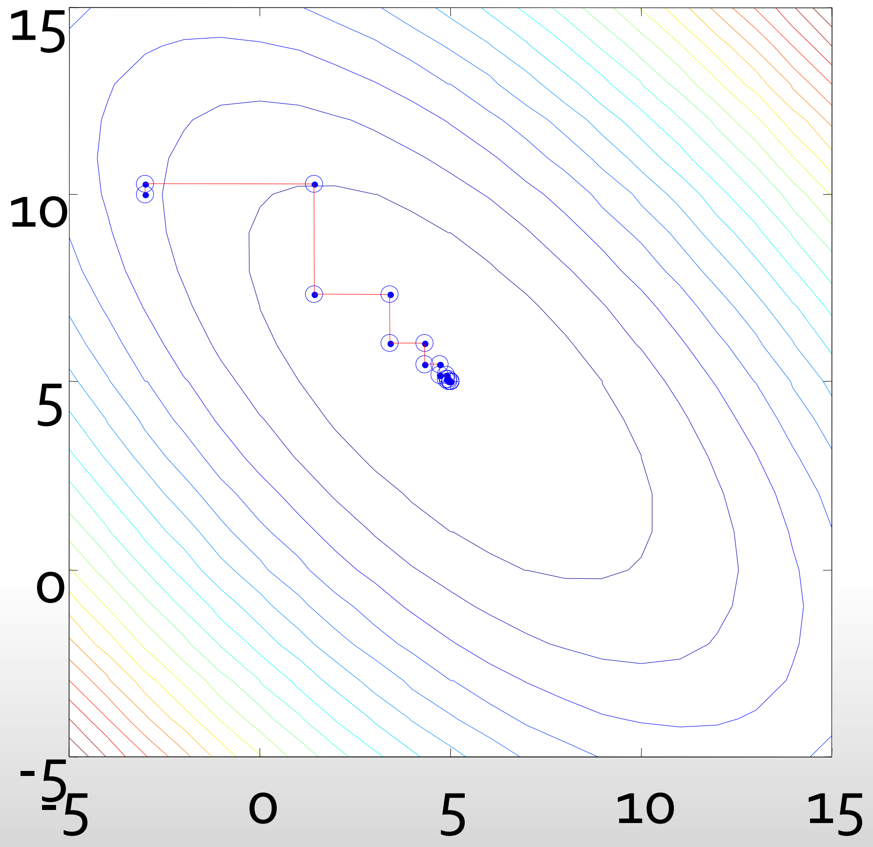


Easy



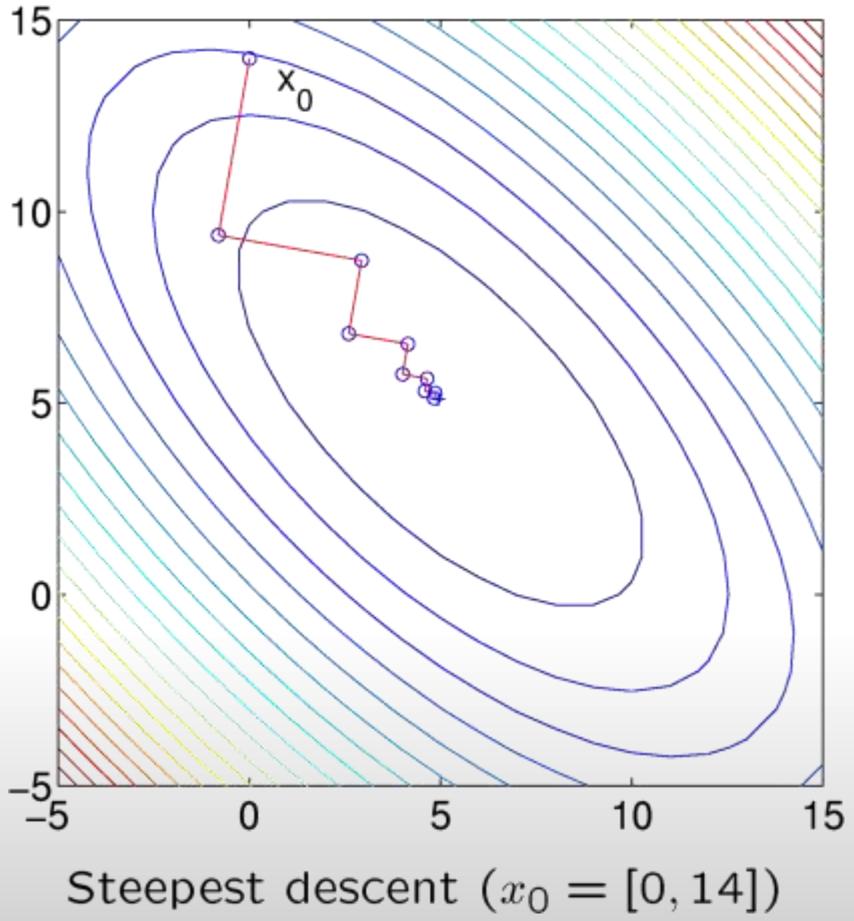
Hard

ALTERNATION



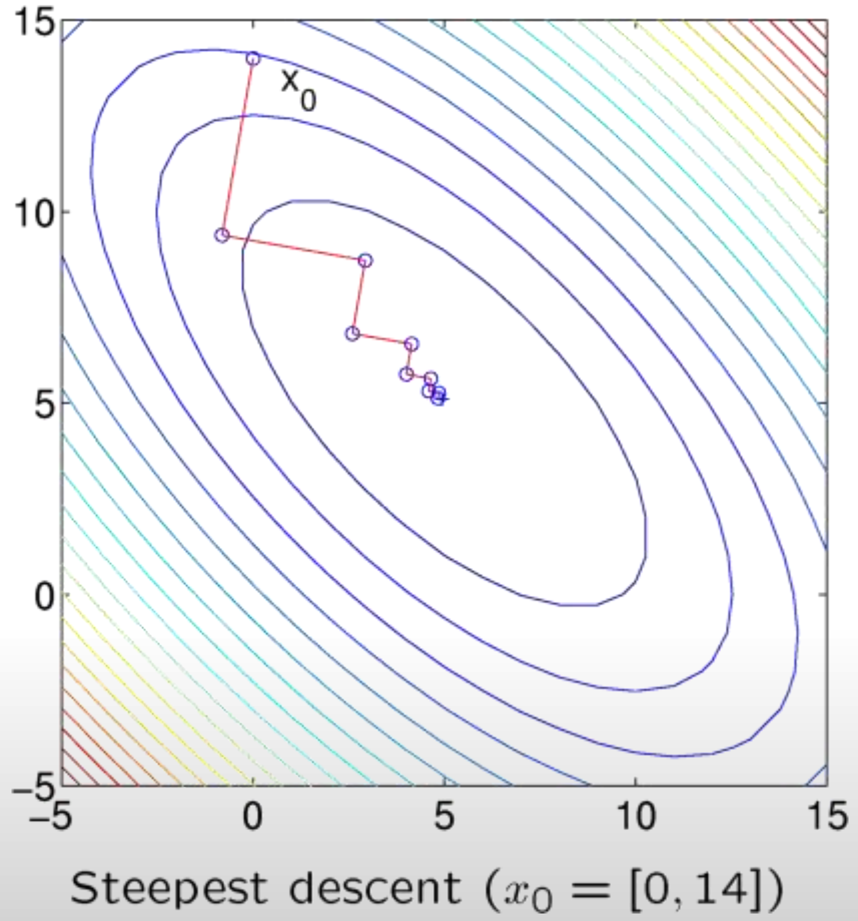
- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?

GRADIENT DESCENT



- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?

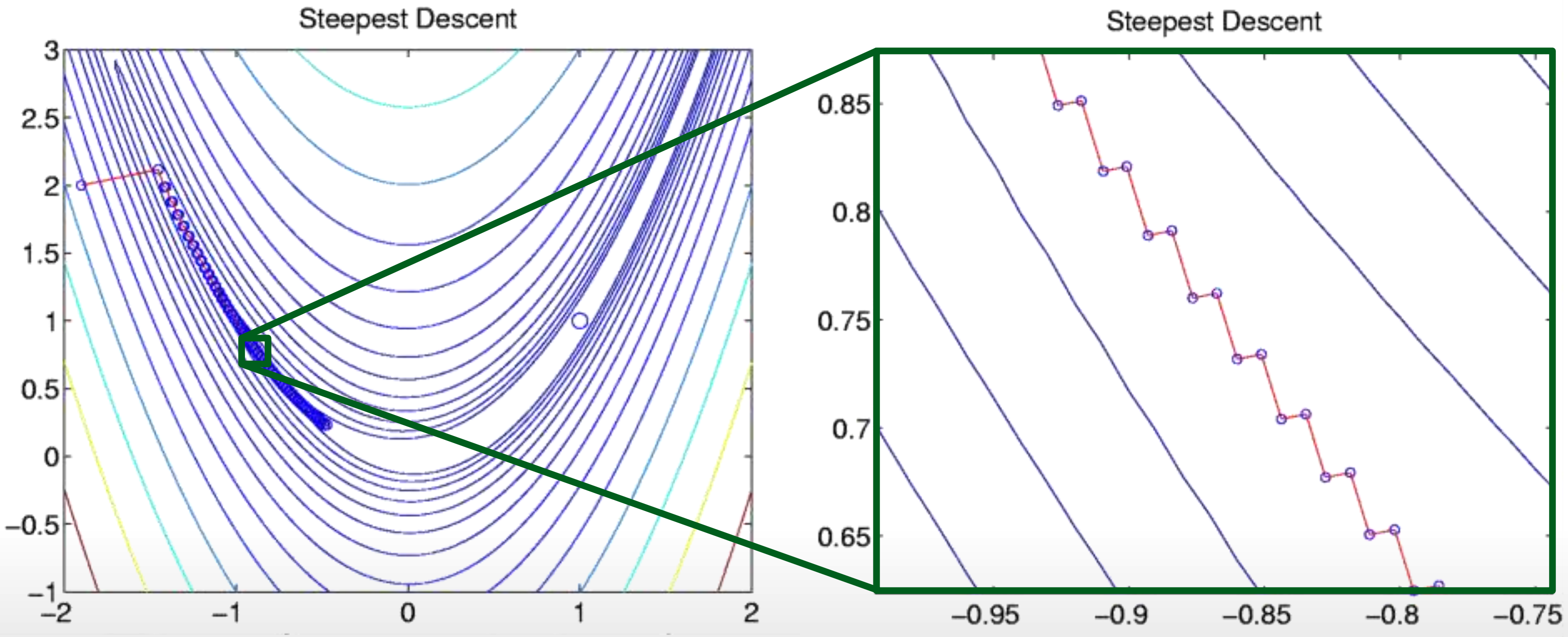
GRADIENT DESCENT



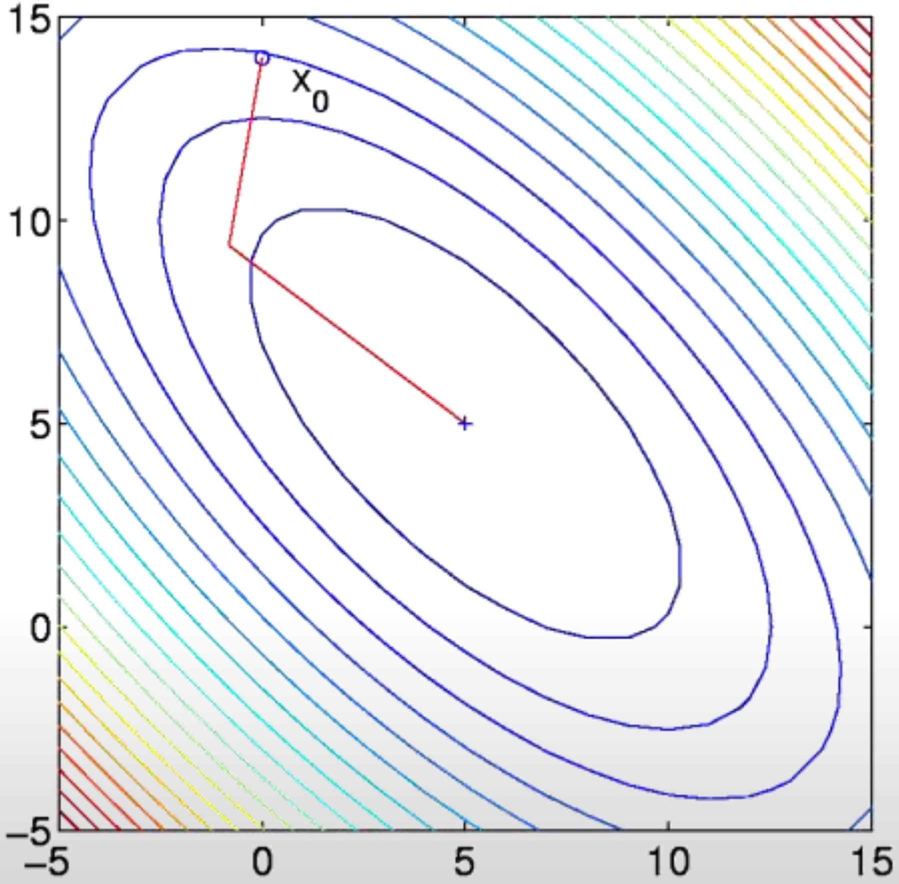
- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?
- Note that convergence proofs are available for both of the above
- But so what?

GRADIENT DESCENT

Microsoft



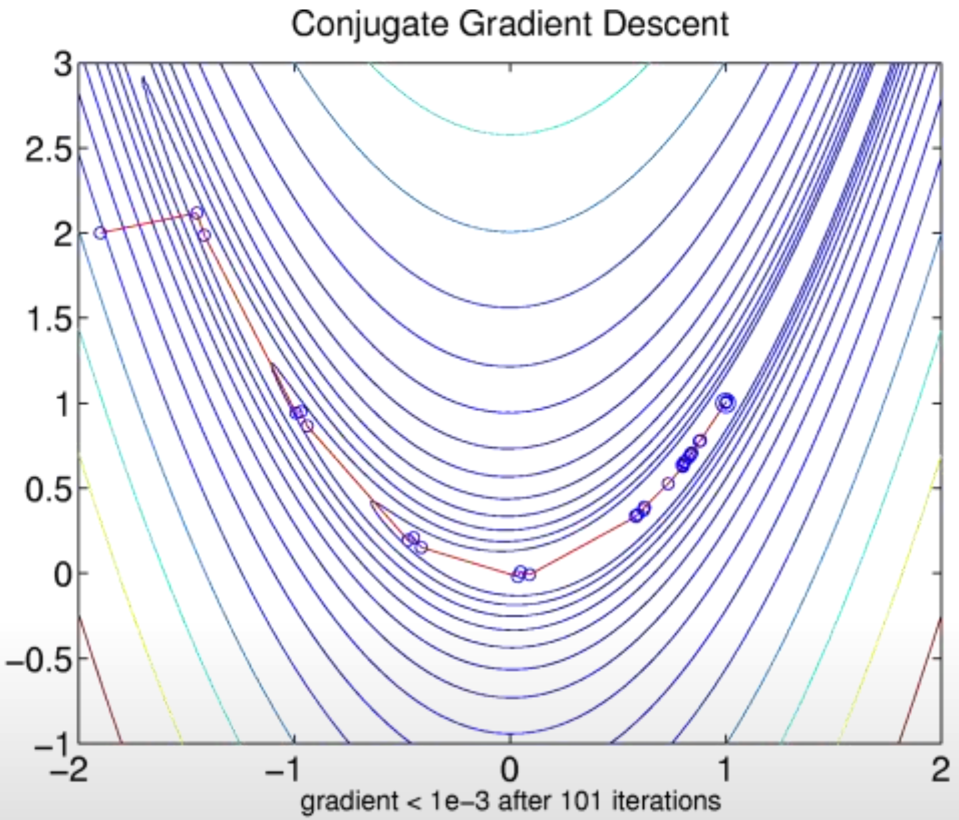
AND ON A HARD PROBLEM



- (Nonlinear) conjugate gradients
- Uses 1st derivatives only
- Avoids “undoing” previous work

USE A BETTER ALGORITHM

Microsoft



- (Nonlinear) conjugate gradients
- Uses 1st derivatives only
- And avoids "undoing" previous work
- 101 iterations on this problem

USE A BETTER ALGORITHM

Microsoft

BUT WE CAN DO BETTER…

Microsoft

- Starting with $\boldsymbol{x}$ how can I choose $\boldsymbol{\delta}$ so that $f(\boldsymbol{x}+\boldsymbol{\delta})$ is better than $f(\boldsymbol{x})$?
- So compute

$$\min_{\boldsymbol{\delta}\in\mathbb{R}^d} f(\boldsymbol{x}+\boldsymbol{\delta})$$

- But hang on, that's the same problem we were trying to solve?

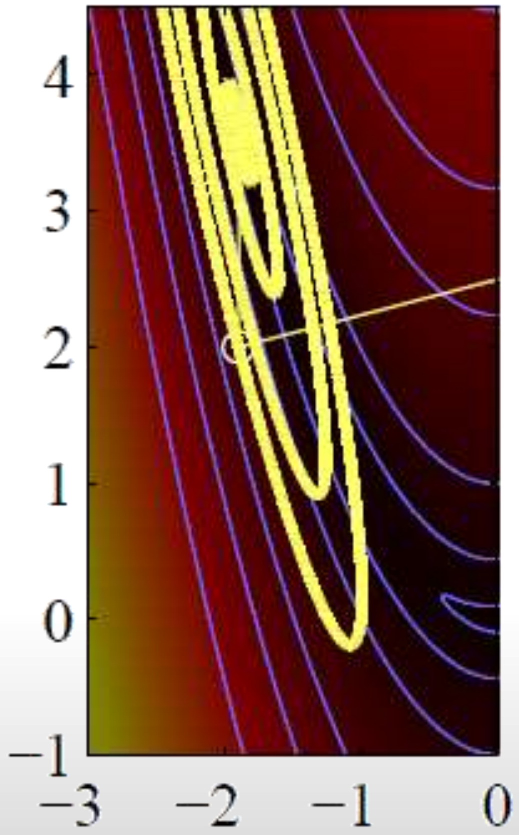
USE SECOND DERIVATIVES…

Microsoft

- Starting with x how can I choose δ so that $f(x+\delta)$ is better than $f(x)$?
- So compute

$$\min_{\delta} \; f(x+\delta)$$
$$\approx \min_{\delta} \; f(x) + \delta^\top g(x) + \tfrac{1}{2}\delta^\top H(x)\delta$$
$$g(x) = \nabla f(x)$$
$$H(x) = \nabla\nabla^\top f(x)$$

USE SECOND DERIVATIVES…



- How does it look?

$$f(x) + \delta^\top g(x) + \frac{1}{2}\delta^\top H(x)\delta$$
$$g(x) = \nabla f(x)$$
$$H(x) = \nabla\nabla^\top f(x)$$

USE SECOND DERIVATIVES…

- Choose δ so that $f(x+\delta)$ is better than $f(x)$?
- Compute

$$\min_{\boldsymbol{\delta}} f + \boldsymbol{\delta}^\top g + \frac{1}{2}\boldsymbol{\delta}^\top H\,\boldsymbol{\delta}$$

[derive]

USE SECOND DERIVATIVES…

Microsoft

- Choose δ so that $f(x+\delta)$ is better than $f(x)$?
- Compute

$$\min_{\boldsymbol{\delta}} f + \boldsymbol{\delta}^\top g + \frac{1}{2}\boldsymbol{\delta}^\top H\, \boldsymbol{\delta}$$

$$\boldsymbol{\delta} = -H^{-1} g$$

USE SECOND DERIVATIVES…

Microsoft

```
>> use demos
>> demo_taylor_2d(0, 'newton', 'rosenbrock')
>> demo_taylor_2d(0, 'newton', 'sqrt_rosenbrock')
>> demo_taylor_2d(1, 'damped newton ls', 'rosenbrock')
```

IS THAT A GOOD IDEA?

Microsoft

- Choose δ so that $f(x+\delta)$ is better than $f(x)$?
- Updates:

$$\boldsymbol{\delta}_{\text{Newton}} = -H^{-1}g$$
$$\boldsymbol{\delta}_{\text{GradientDescent}} = -\lambda g$$

USE SECOND DERIVATIVES…

Microsoft

- Updates:

$$\delta_{\text{Newton}} = -H^{-1}g$$
$$\delta_{\text{GradientDescent}} = -\lambda g$$

- So combine them:

$$\begin{aligned}\delta_{\text{DampedNewton}} &= -(H + \lambda^{-1}I_d)^{-1}g \\ &= -\lambda(\lambda H + I_d)^{-1}g\end{aligned}$$

- λ small $\Rightarrow$conservative gradient step
- λ large $\Rightarrow$Newton step

```
$\lambda = 10^{-3}; \lambda' = 3;$
while $\lambda < 10^9$
    $[f, \boldsymbol{g}, \boldsymbol{H}] = \text{error\_function}(\boldsymbol{x}_k)$        % Perhaps Gauss-Newton for H
    $\boldsymbol{\delta} = -(\boldsymbol{H} + \lambda \boldsymbol{I}) \backslash \boldsymbol{g}$        % Many ways to do this efficiently
    $\boldsymbol{x}_{new} = \boldsymbol{x}_k + \boldsymbol{\delta}$
    if $\text{error\_function}(\boldsymbol{x}_{new}) < f$:
        $\boldsymbol{x}_k = \boldsymbol{x}_{new}$        % Decreased error, accept the new $x$
        $\lambda = \lambda / \lambda'; \lambda' = 3$        % Doing well—decrease $\lambda$
    else
        $\lambda = \lambda \lambda'; \lambda' = 3\lambda'$        % Doing badly—increase $\lambda$ quick
```

UPDATING λ

Microsoft

Levenberg-Marquardt

- Just damped Newton with approximate H
- For a special form of f

$$f(x) = \sum_i f_i(x)^2$$

- where $f_i(x)$ are
  - zero-mean
  - small at the optimum

Levenberg Marquardt

- Just damped Newton with approximate H
- For a special form of f

$$f(x) = \sum_i f_i(x)^2$$

$$\nabla f(x) =$$

$$\nabla \nabla^\top f(x) =$$

Levenberg Marquardt

- Just damped Newton with approximate H
- For a special form of f

$$f(x) = \sum_i f_i(x)^2$$

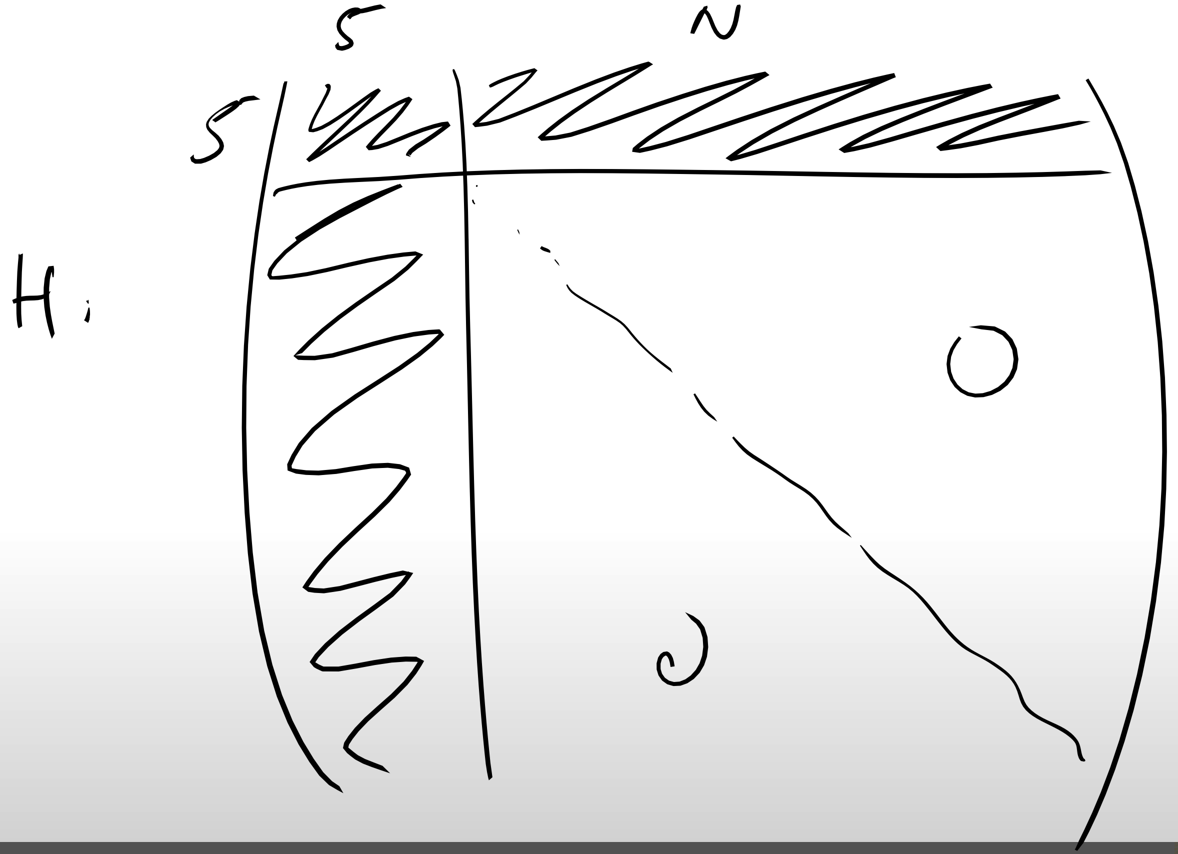
$$\nabla f(x) = \sum_i 2 f_i(x) \nabla f_i(x)$$

$$\nabla\nabla^\top f(x) = 2 \sum_i \underbrace{\left(\cancel{f_i(x)\nabla\nabla^\top f_i(x)}\right)}_{}^{\approx 0} + \overset{\mathbb{R}^d}{\boxed{\nabla f_i(x)}} \nabla^\top f_i(x)$$

- Not $O(n^3)$ if you exploit sparsity of Hessian or Jacobian

$$\mathrm{J} = \begin{bmatrix} \nabla f_1(x) \\ \vdots \\ \nabla f_n(x) \end{bmatrix}$$

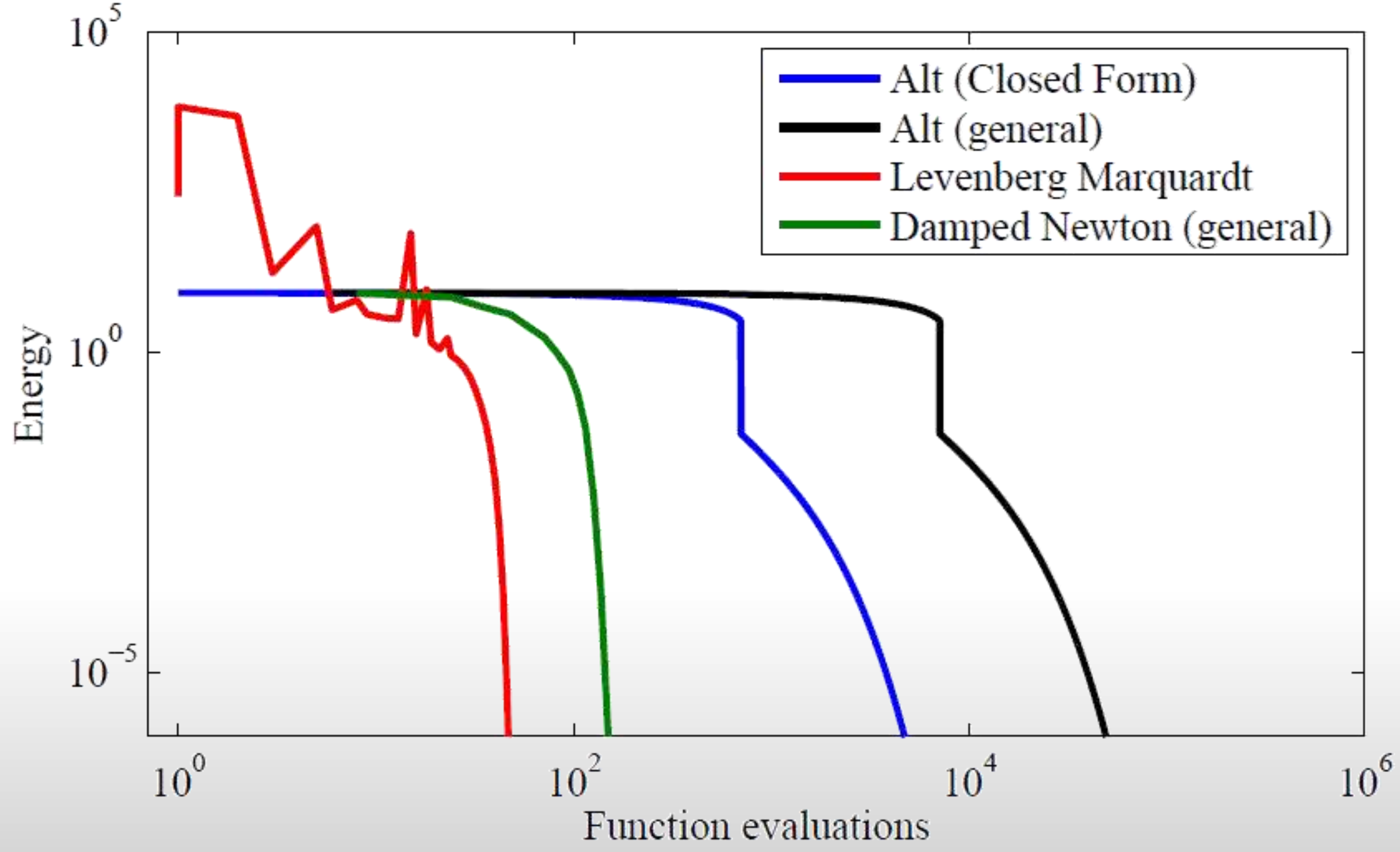
ORDER N CUBED?



$O(N^3) \to O(N)$

TYPICAL HESSIAN STRUCTURE

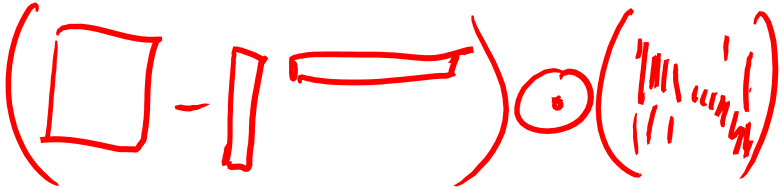
Microsoft

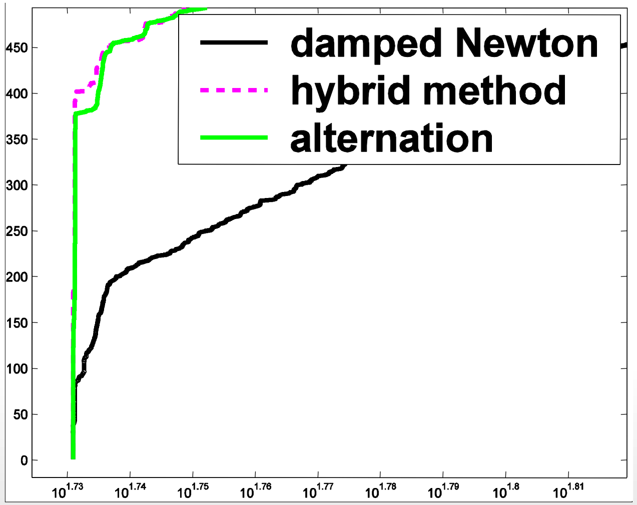


CONCLUSION: YMMV

GIRAFFE

$$\min_{A,B} \| (M - AB^T) \odot W \|^2$$





```
for k=1:500
  $x_0 = randn(n,1);$
  $x^* = minimize(f, x_0);$
  $E[k] = f(x^*)$
end
plot(sort(E));
```

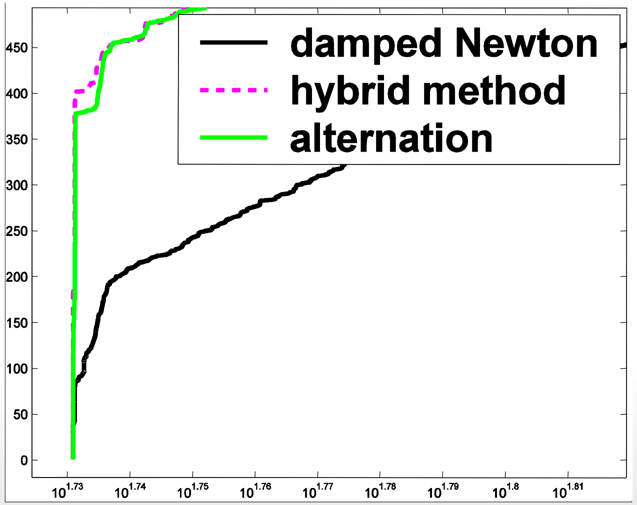
500 runs

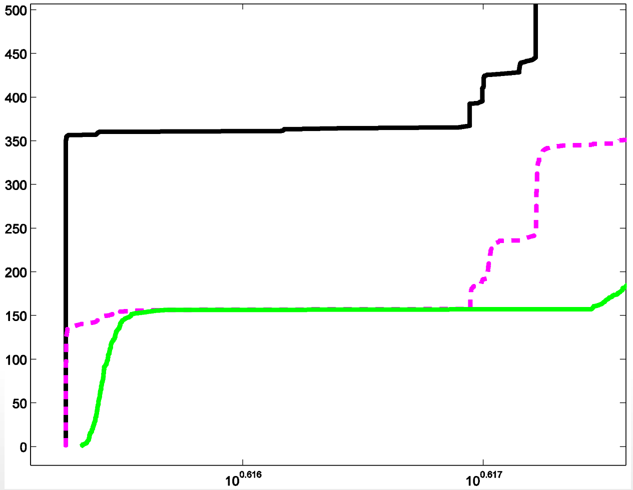
CONCLUSION: YMMV

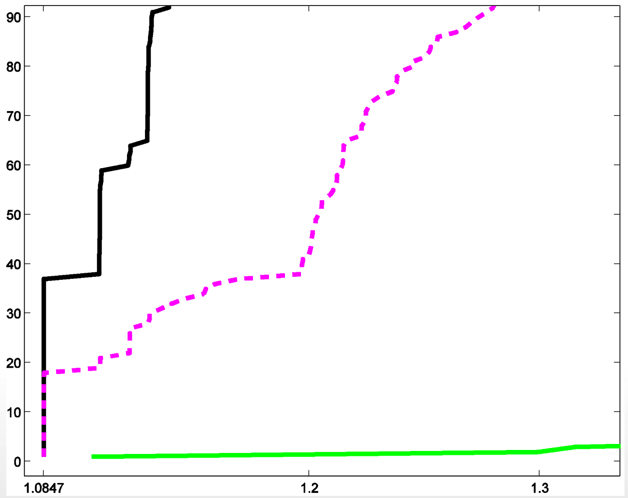
GIRAFFE

FACE

DINOSAUR







500 runs

1000 runs

1000 runs

CONCLUSION: YMMV

- On many problems, alternation is just fine
  - Indeed always start with a couple of alternation steps
- Computing 2nd derivatives is a pain
  - But you don’t need to for LM
- But just alternation is not
  - Unless you’re willing to problem-select
- Convergence guarantees are fine, but practice is what matters
- Inverting the Hessian is rarely $O(n^3)$

There is no universal optimizer

- $\nabla f = \frac{1}{\mu}\begin{bmatrix} f(x+e_1) - f(x) \\ \vdots \\ f(x+e_d) - f(x) \end{bmatrix}$
- Surprisingly accurate for e.g. $\mu = 10^{-5}$ (in double prec.)
- Incredibly slow.. Unless (see next slide)
- Useful for checking your analytic derivatives
- Incredibly slow. Try Powell or Simplex instead.
- Central differences twice as slow, somewhat more accurate

ON FINITE-DIFFERENCE DERIVATIVES

Microsoft

- Normally try e_1 to e_d sequentially
- But if we know the nonzero structure of the Jacobian, can go rather faster.

GRAPH COLOURING FOR FINITE DIFFS

- We're minimizing $f(x)$
- Many algorithms will be happier if entries of x are all "around 1".
  - E.g. don't have angle in degrees and distances in km
- Many algorithms may want f values to be "close to x or close to zero at the optimum".
  - Specifically, think about roundoff in quantities like $f(x_{k+1}) - f(x_k)$ being compared to numbers like 10^{-6}

- What about stochastic gradient descent?
  - You can do analogous 2nd order things.
- What about LBFGS?
  - I haven’t had much success with it, other folk love it…
- I tried lsqnonlin and it was really slow—why?
  - Wrong derivatives (e.g. finite-differences)
  - Didn’t use sparsity correctly
  - Didn’t set “options.Algorithm” or “options.LargeScale”.

QUESTIONS

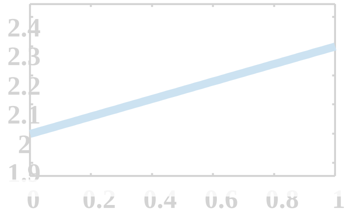
- Resources:

1. Matlab fminsearch and fminunc documentation
2. awful.codeplex.com au_optimproblem
3. Tom Minka webpage on matrix derivatives
4. Google “ceres” solver
5. UTorono “Theano” system for Python

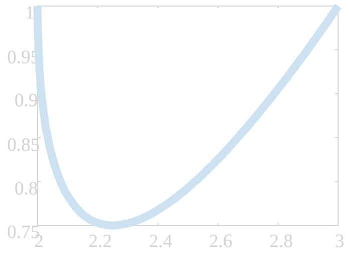
- Gotchas with lsqnonlin
  - opts.LargeScale = 'on';
  - opts.Jacobian = 'on';
- Need non-rank-def J?
- Need to implement JacobMult?

WHAT IS A SURFACE?

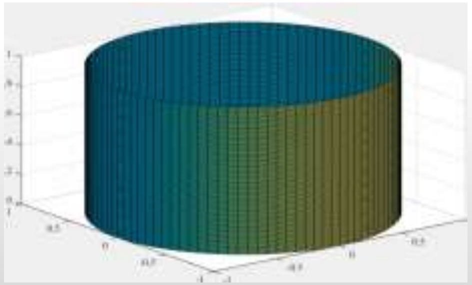
```
function y(x::Interval)::Real = .3*x + 2
```



```
function C(t::Interval)::Point2D =
      Point2D(t^2 + 2, t^2 - t + 1)
```



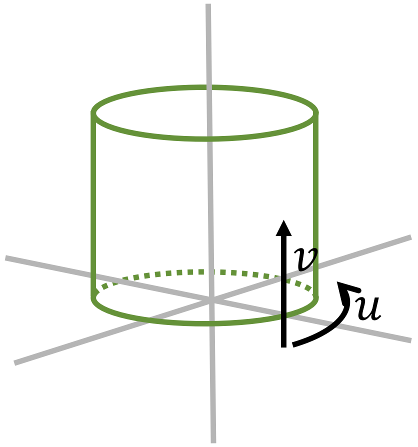
```
function S(u::Interval, v::Real)::Point3D =
      Point3D(cos(u), sin(u), v)
```



CURVES

Microsoft

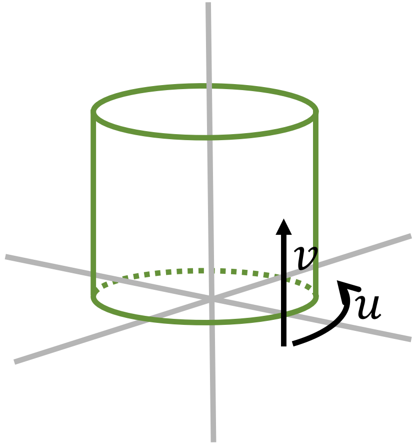
- Surface: mapping $S(\boldsymbol{u})$ from $\mathbb{R}^2 \mapsto \mathbb{R}^3$
  - E.g. cylinder $S(u, v) = (\cos u\,, \sin u\,, v)$



*the surface is actually the set $\{M(u; \Theta) | u \in \Omega\}$

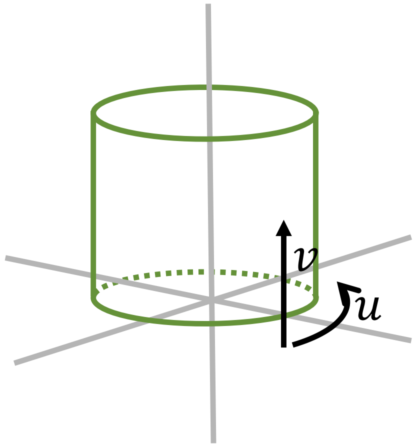
SURFACE

- Surface: mapping $S(\boldsymbol{u})$ from $\mathbb{R}^2 \mapsto \mathbb{R}^3$
  - E.g. cylinder $S(u,v) = (\cos u, \sin u, v)$
- Probably not all of $\mathbb{R}^2$, but a subset Ω
  - E.g. square $\Omega = [0,2\pi) \times [0,H]$
  - But also any union of **patch domains** $\Omega = \bigcup_p \Omega_p$



*the surface is actually the set $\{M(u;\Theta) | u \in \Omega\}$

- Surface: mapping $S(\boldsymbol{u})$ from $\mathbb{R}^2 \mapsto \mathbb{R}^3$
  - E.g. cylinder $S(u,v) = (\cos u, \sin u, v)$
- Probably not all of $\mathbb{R}^2$, but a subset Ω
  - E.g. square $\Omega = [0,2\pi) \times [0,H]$
  - But also any union of **patch domains** $\Omega = \bigcup_p \Omega_p$
- And we'll look at **parameterised** surfaces $S(\boldsymbol{u};\Theta)$
  - E.g. Cylinder $S(u,v;R,H) = (R\cos u, R\sin u, Hv)$ with $\Omega = [0,2\pi) \times [0,1]$
  - E.g. subdivision surface $S(\boldsymbol{u};X)$ where $\Theta = X \in \mathbb{R}^{3\times n}$ is matrix of **control vertices**

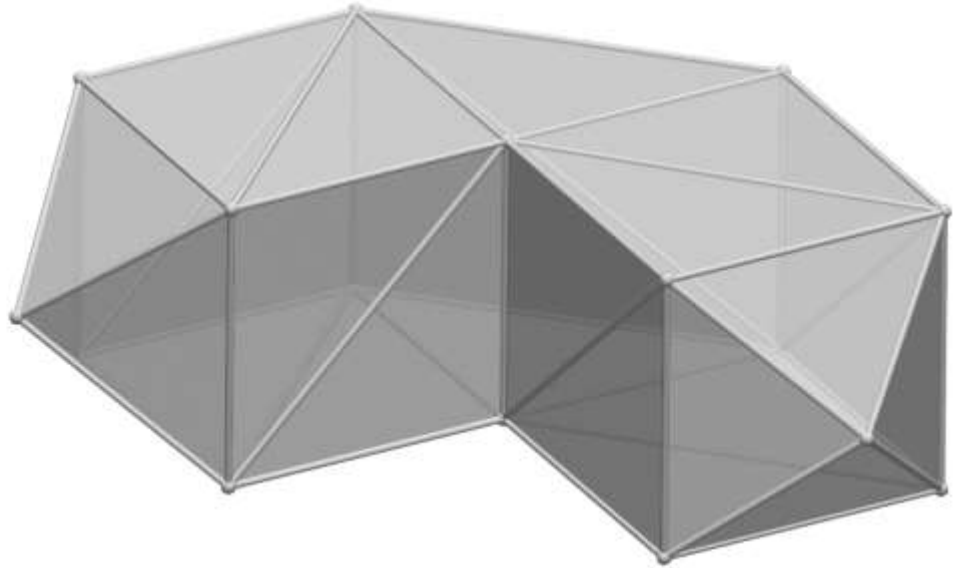


*the surface is actually the set $\{M(u;\Theta) | u \in \Omega\}$

SURFACE

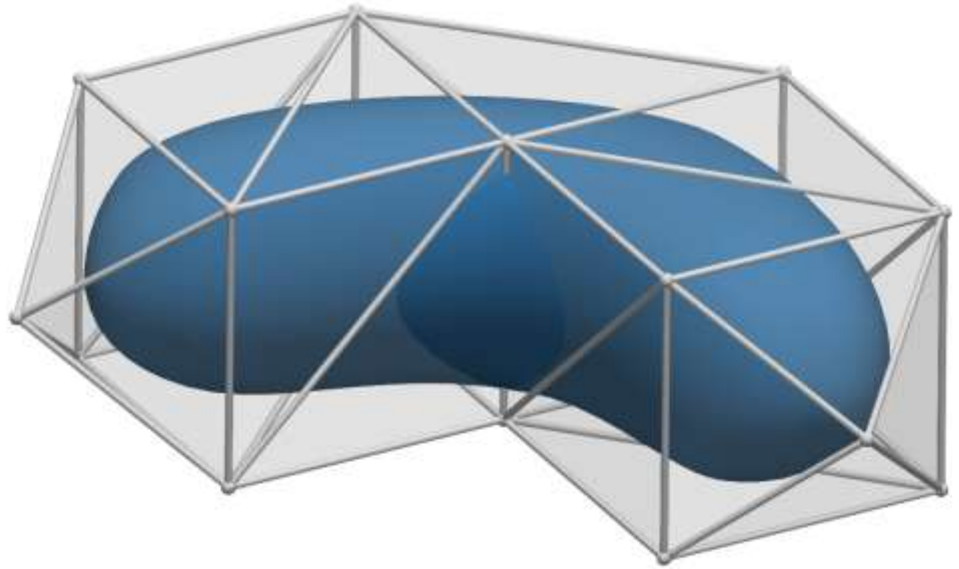
TOOL: SUBDIVISION SURFACES

Control mesh vertices $X \in \mathbb{R}^{3\times m}$
Here $m = 16$

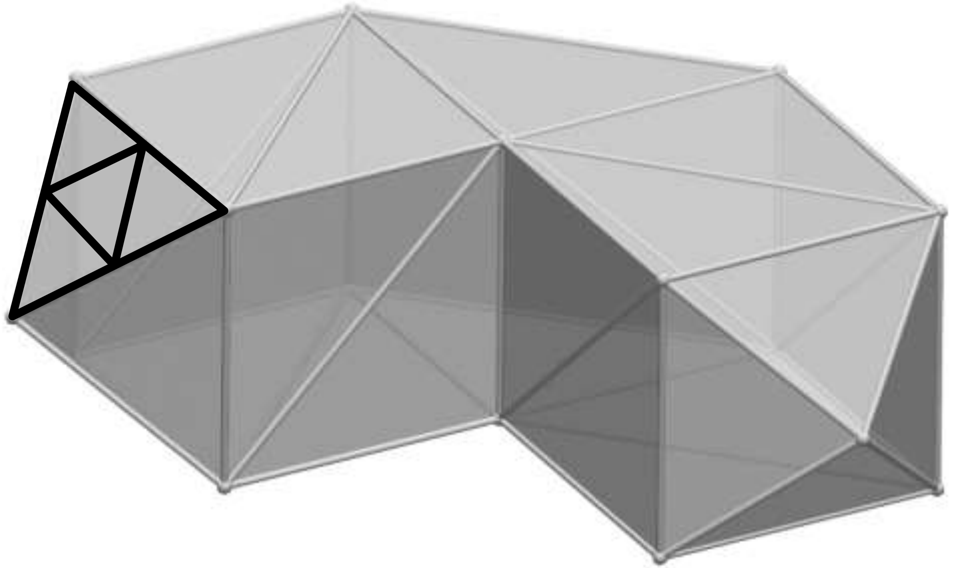


CONTROL MESH

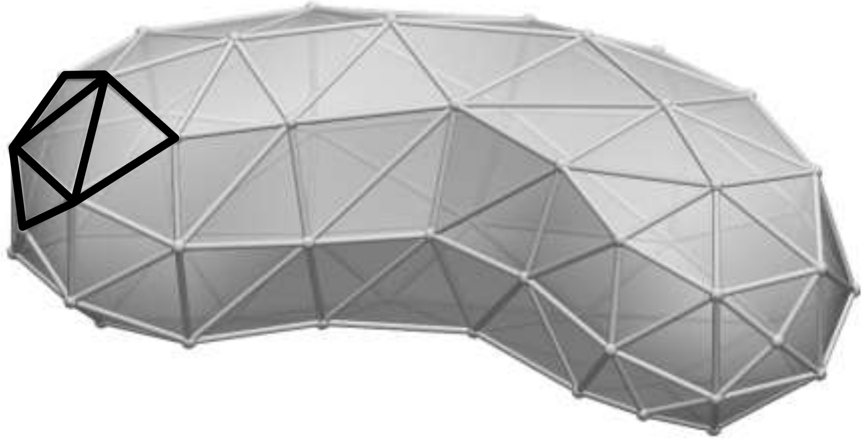
Control mesh vertices $X \in \mathbb{R}^{3\times m}$
Here $m = 16$



LIMIT SURFACE

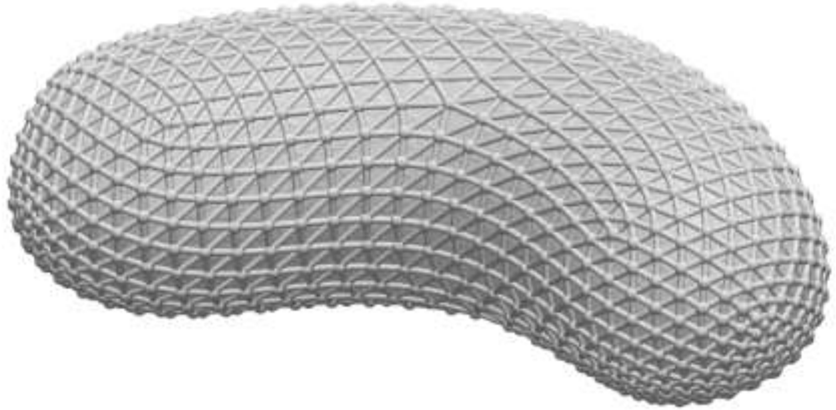


SUBDIV RULE: STEP 1. ADD NEW VERTICES



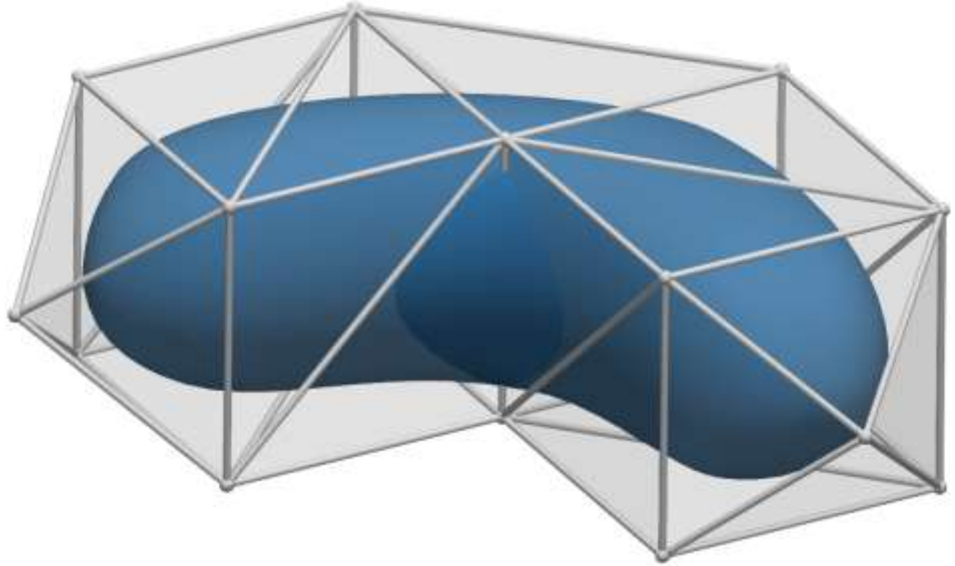
SUBDIV RULE: STEP 2. AVERAGE NEIGHBOURS

2 SUBDIVISIONS



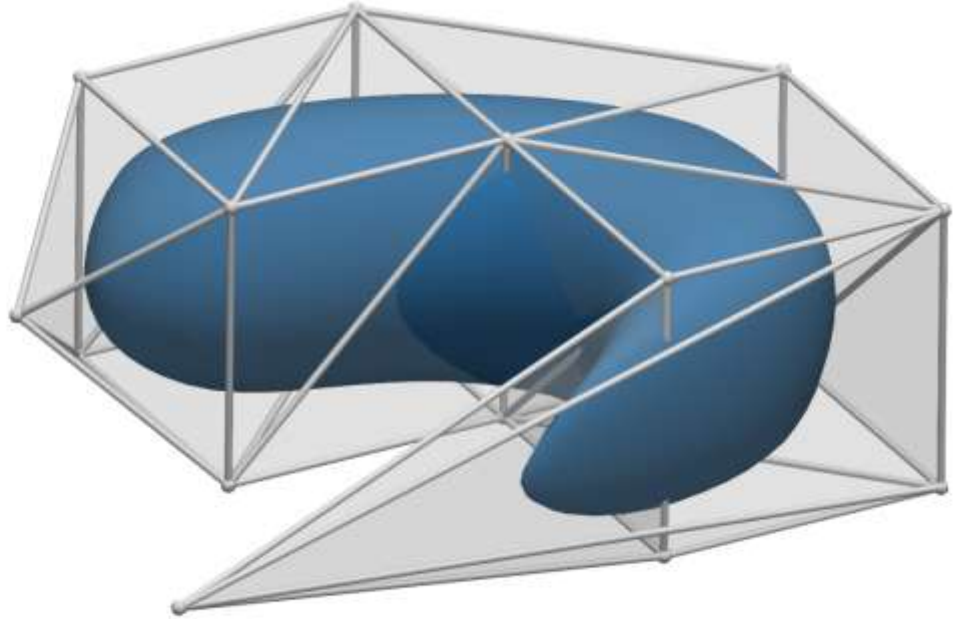
3 SUBDIVISIONS

Control mesh vertices $V \in \mathbb{R}^{3\times m}$
Here $m = 16$
Blue surface is $\{M(\boldsymbol{u}; V) \mid \boldsymbol{u} \in \Omega\}$
Ω is the grey surface



LIMIT SURFACE

Control mesh vertices $V \in \mathbb{R}^{3\times n}$
Here $n = 16$
Blue surface is $\{M(\boldsymbol{u}; V) \mid \boldsymbol{u} \in \Omega\}$
Ω is the grey surface



CONTROL VERTICES DEFINE THE SHAPE

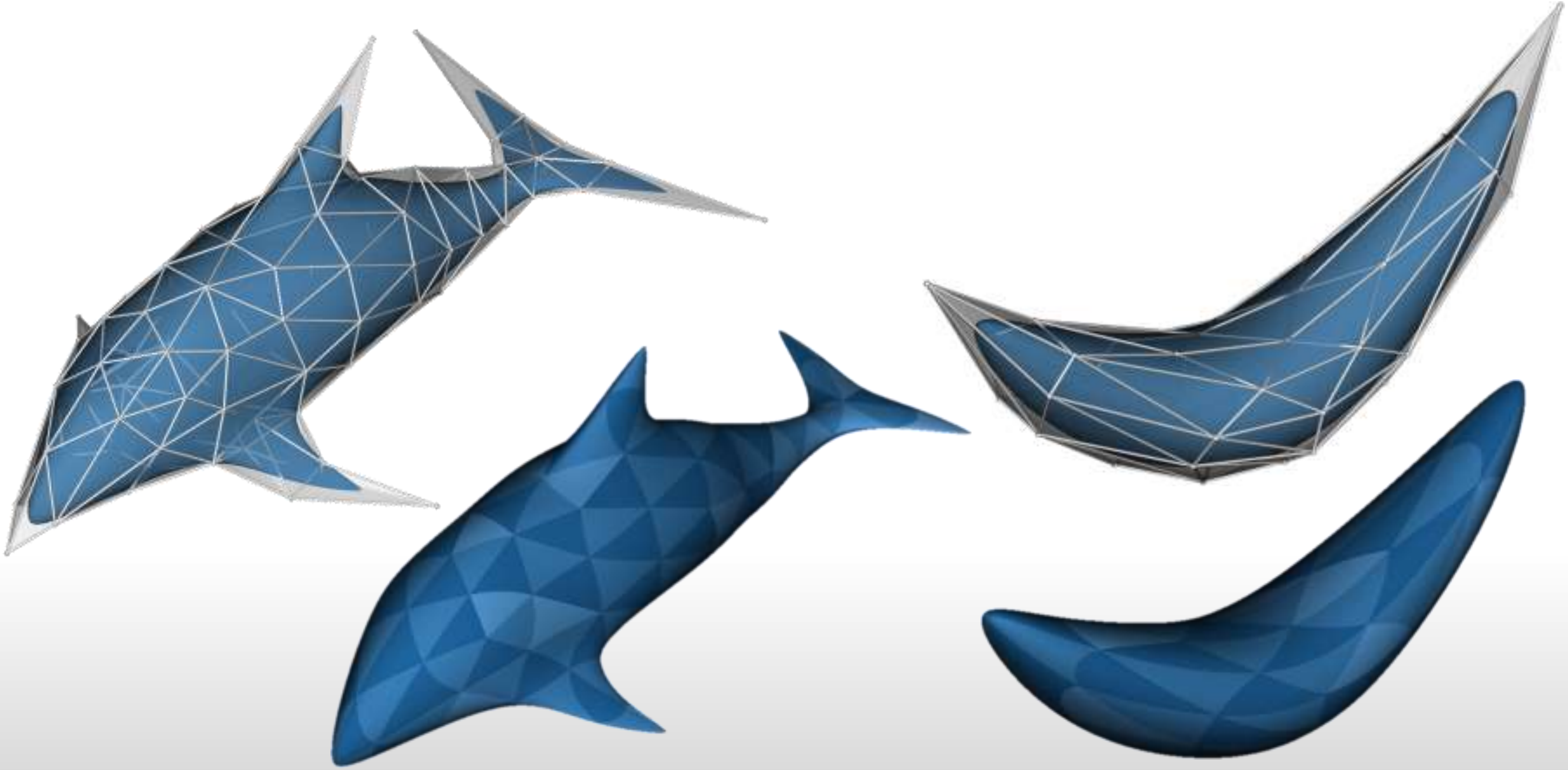
Microsoft

- Mostly, M is quite simple:

$$M(\boldsymbol{u}; X) = M(t, u, v; \boldsymbol{x}_1, \dots, \boldsymbol{x}_n) = \sum_{\substack{i+j\le 4 \\ k=1..n}} A^t_{ijk} u^i v^j \boldsymbol{x}_k$$

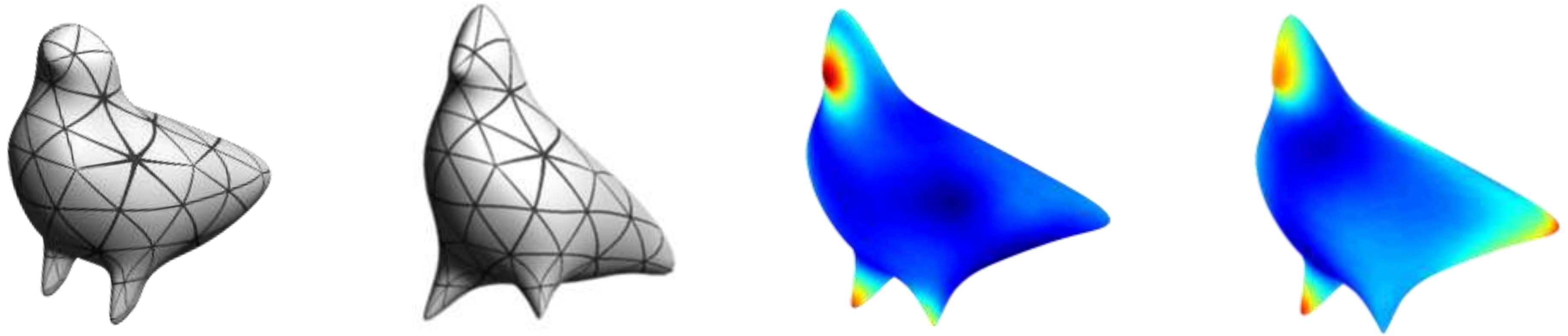
  - Integer triangle id t
  - Quartic in u, v
  - Linear in X
  - Easy derivatives
- But…
  - 2nd Derivatives unbounded although normals well defined
  - Piecewise parameter domain

SUBDIVISION SURFACE: PARAMETRIC FORM



EXAMPLES

BACK TO DOLPHINS



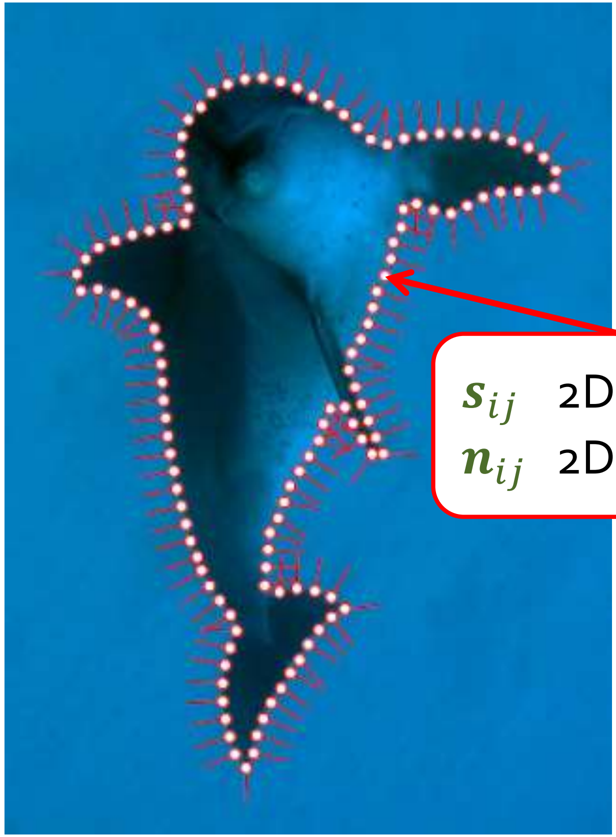
$$X_i = \mathcal{B}_0 + \alpha_{i1}\,\mathcal{B}_1 + \alpha_{i2}\,\mathcal{B}_2$$

$$X_n = \sum_{k=0}^{K} \alpha_{ik}\mathcal{B}_k$$

Linear blend shapes:
Image i represented by coefficient vector $\boldsymbol{\alpha}_i = [\alpha_{i1}, \ldots, \alpha_{iK}]$

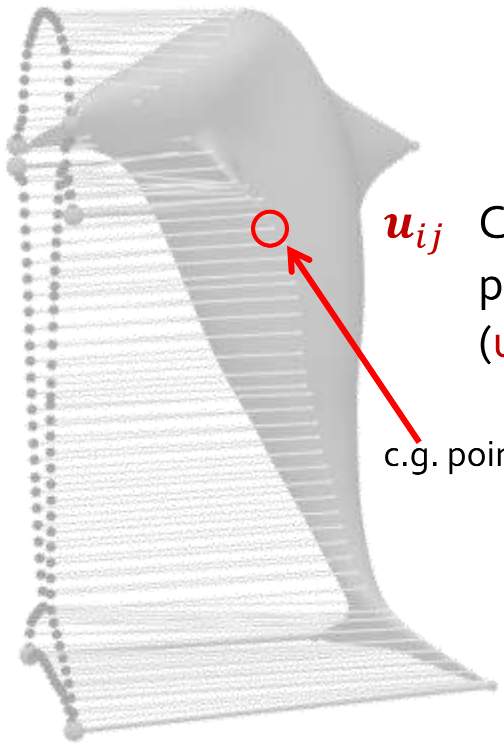
MODEL REPRESENTATION

Image i



$\boldsymbol{s}_{ij}$ 2D point

$\boldsymbol{n}_{ij}$ 2D normal

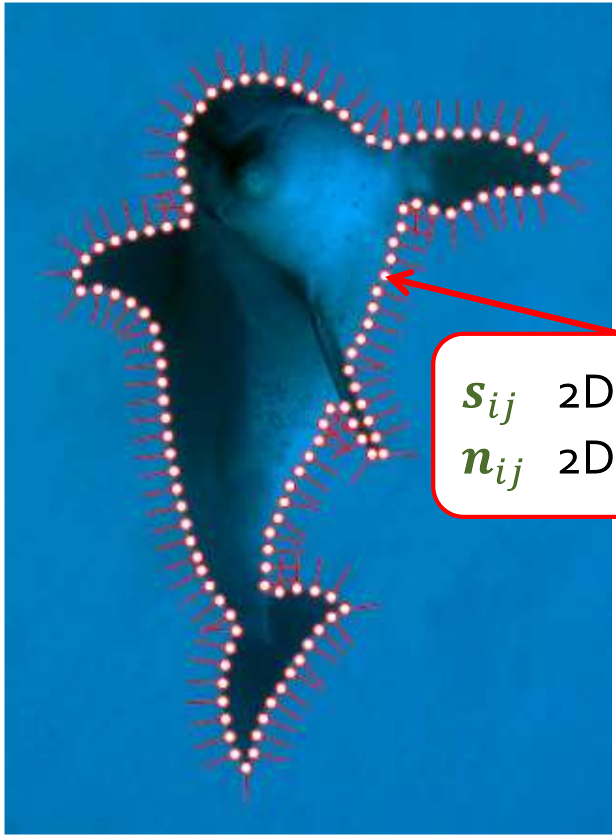


$\boldsymbol{u}_{ij}$ Contour generator preimage in $\boldsymbol{\Omega}$ (unknown)

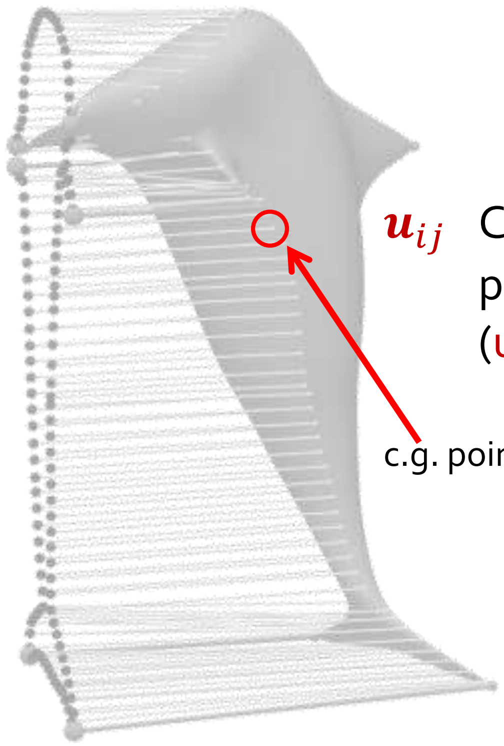
c.g. point in 3D is $M(\boldsymbol{u}_{ij}; \boldsymbol{X}_i)$

DATA TERMS

Image i



$\boldsymbol{s}_{ij}$ 2D point
$\boldsymbol{n}_{ij}$ 2D normal

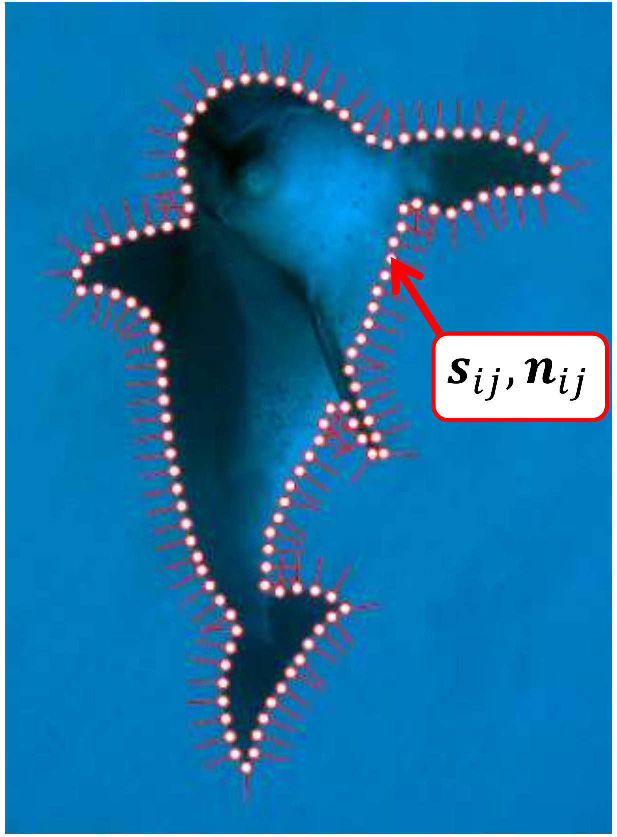


$\boldsymbol{u}_{ij}$ Contour generator preimage in $\boldsymbol{\Omega}$ (unknown)

c.g. point in 3D is $M(\boldsymbol{u}_{ij};\boldsymbol{X}_i)$

DATA TERMS

Image i



Projection
e.g. Perspective

Camera
position

Silhouette:

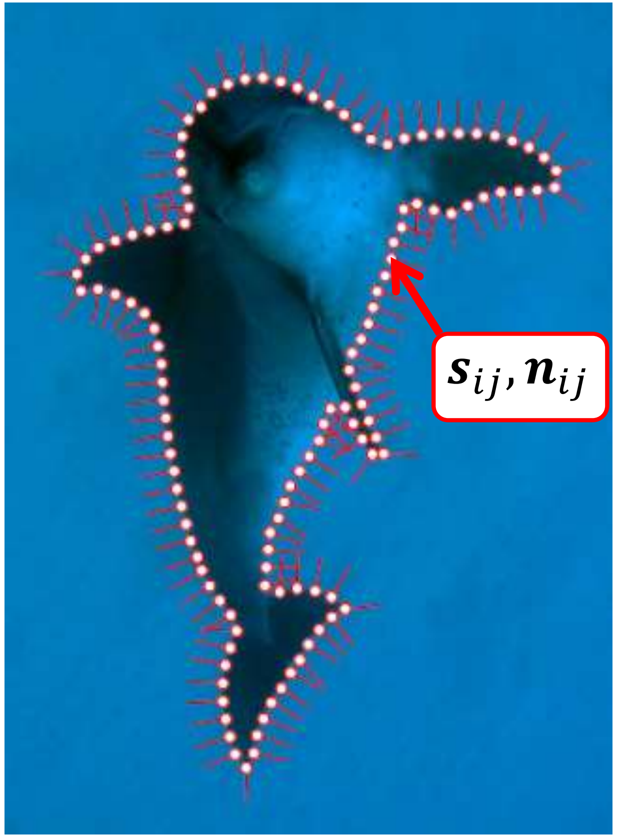
$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| s_{ij} - \pi\left(\theta_i, M\left(u_{ij}, X_i\right)\right) \right\|^2$$

Normal:

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \begin{bmatrix} n_{ij} \\ 0 \end{bmatrix} - R(\theta_i) N\left(u_{ij}, X_i\right) \right\|^2$$

DATA TERMS

Image i



Linear Blend Shapes (PCA) Model:

$$\boldsymbol{X}_i = \sum_k \alpha_{ik} \boldsymbol{B}_k$$

Silhouette:

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \boldsymbol{s}_{ij} - \pi\left(\theta_i, M\left(u_{ij}, \boldsymbol{X}_i\right)\right) \right\|^2$$

Normal:

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \begin{bmatrix} \boldsymbol{n}_{ij} \\ 0 \end{bmatrix} - R(\theta_i) N\left(u_{ij}, \boldsymbol{X}_i\right) \right\|^2$$

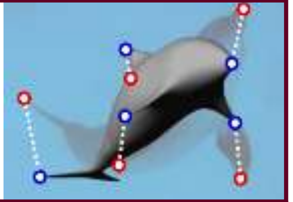
Data fidelity terms

$p(I|X_i;U)$

$$E_i^{\text{sil}} = \frac{1}{2}\sigma_{\text{sil}}^{-2}\sum_{j=1}^{S_i}\|\boldsymbol{s}_{ij} - \pi_i\left(M(\mathring{u}_{ij}|\boldsymbol{X}_i)\right)\|^2$$

$$E_i^{\text{norm}} = \frac{1}{2}\sigma_{\text{norm}}^{-2}\sum_{j=1}^{S_i}\left\|\begin{bmatrix}\boldsymbol{n}_{ij}\\0\end{bmatrix} - \nu\left(\boldsymbol{R}_i N(\mathring{u}_{ij}|\boldsymbol{X}_i)\right)\right\|^2$$

$$E_i^{\text{con}} = \frac{1}{2}\sigma_{\text{con}}^{-2}\sum_{k=1}^{K_i}\|\boldsymbol{c}_{ik} - \pi_i\left(M(\mathring{\mu}_{ik}|\boldsymbol{X}_i)\right)\|^2$$



Smooth Basis

$p(\boldsymbol{\Theta})$

$$E_m^{\text{tp}} = \frac{\bar{\lambda}^2}{2}\int_\Omega \|M_{xx}(\mathring{u}|\boldsymbol{B}_m)\|^2 + 2\|M_{xy}(\mathring{u}|\boldsymbol{B}_m)\|^2 + \|M_{yy}(\mathring{u}|\boldsymbol{B}_m)\|^2 \, \mathrm{d}\mathring{u}$$

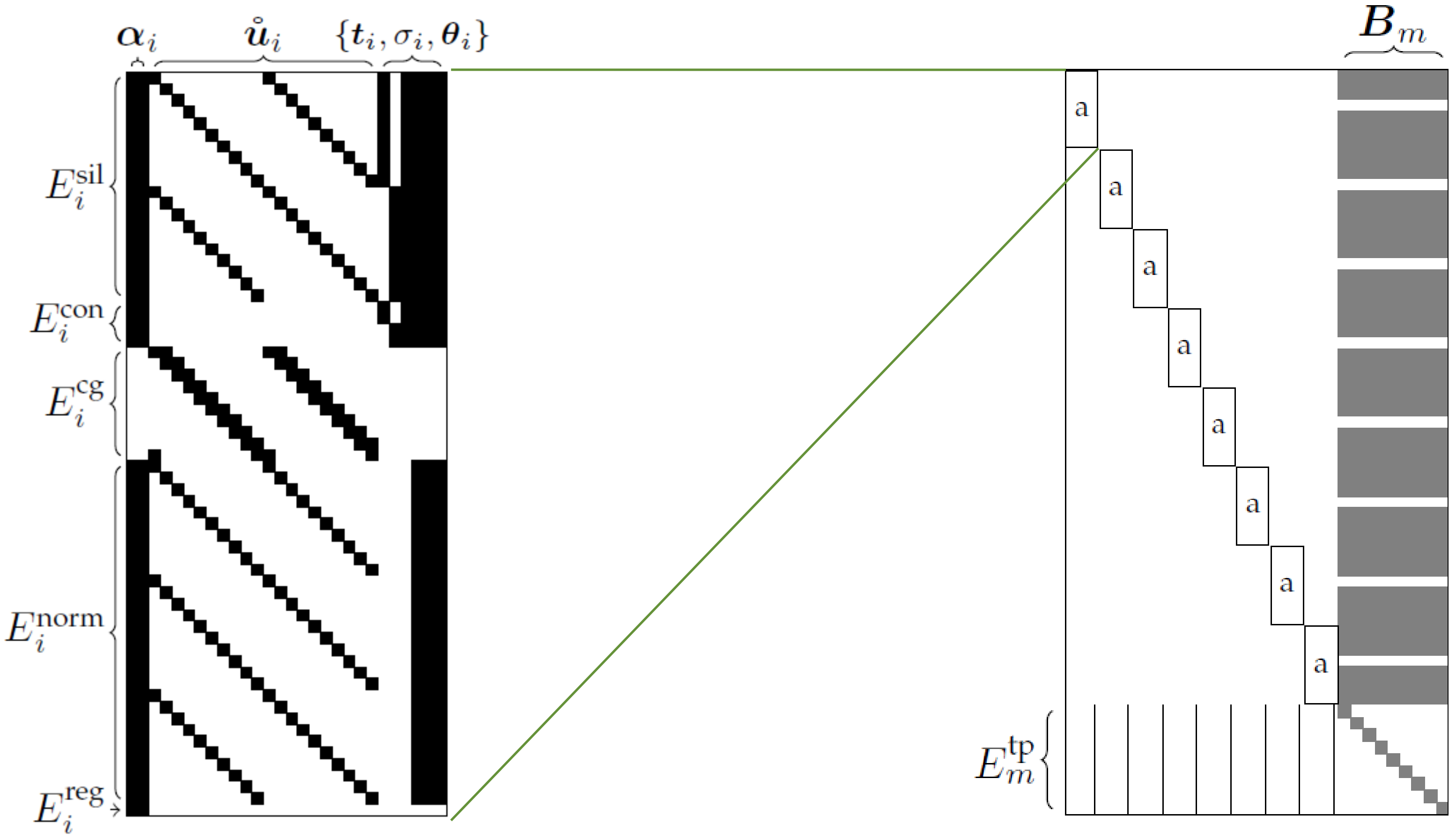
Gaussian shape weights

$$E_i^{\text{reg}} = \beta\sum_{m=1}^{D}\alpha_{im}^2$$

Smooth contour

$$E_i^{\text{cg}} = \gamma\sum_{j=1}^{S_i}\tau(d(\mathring{u}_{ij}, \mathring{u}_{i,j+1}))$$

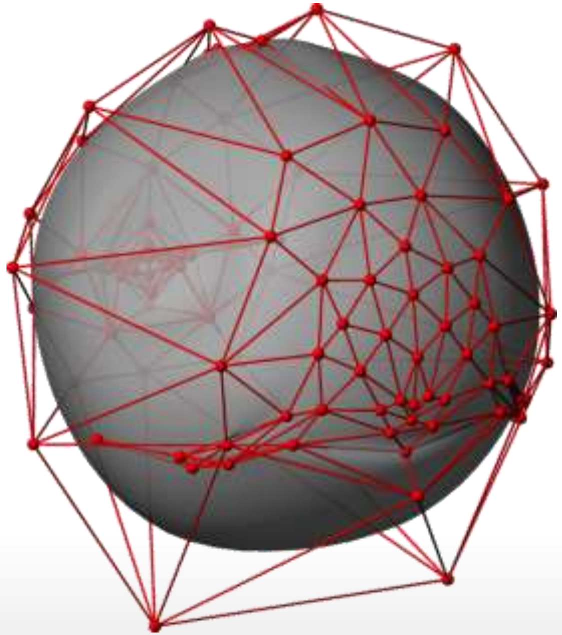
$$\boldsymbol{X}_i = \sum_{m=0}^{D}\alpha_{im}\boldsymbol{B}_m$$


$\boldsymbol{\alpha}_i$
$\mathring{\boldsymbol{u}}_i$
$\{\boldsymbol{t}_i, \sigma_i, \boldsymbol{\theta}_i\}$
E_i^{sil}
E_i^{con}
E_i^{cg}
E_i^{norm}
E_i^{reg}
$\boldsymbol{B}_m$
a
a
a
a
a
a
a
a
E_m^{tp}

$$E_i^{\text{sil}} = \frac{1}{2}\sigma_{\text{sil}}^{-2} \sum_{j=1}^{S_i} \| s_{ij} - \pi_i \left(M(\mathring{u}_{ij} | X_i) \right) \|^2$$

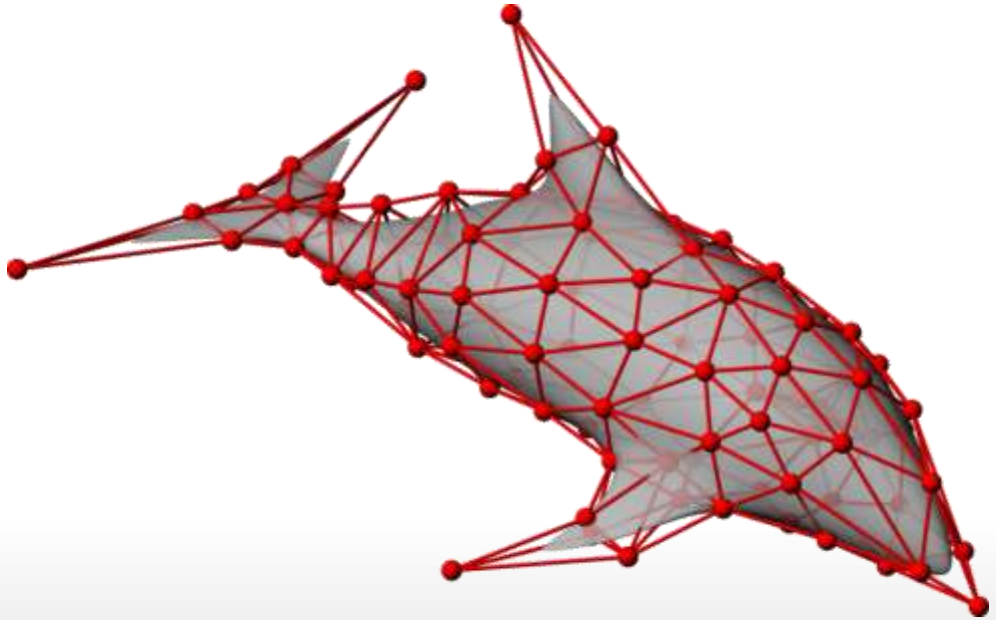
$\sum \alpha_{ik} B_k$

- Can focus on this term to understand entire optimization.
  - Total number of residuals n = number of silhouette points. Say $300N$ ($N =$ number of images) $\approx 10{,}000$
  - Total number of unknowns $2n + KN + m$ where $m \approx 3K \times$ number of vertices $\approx 3{,}000$



This is true, but misleading

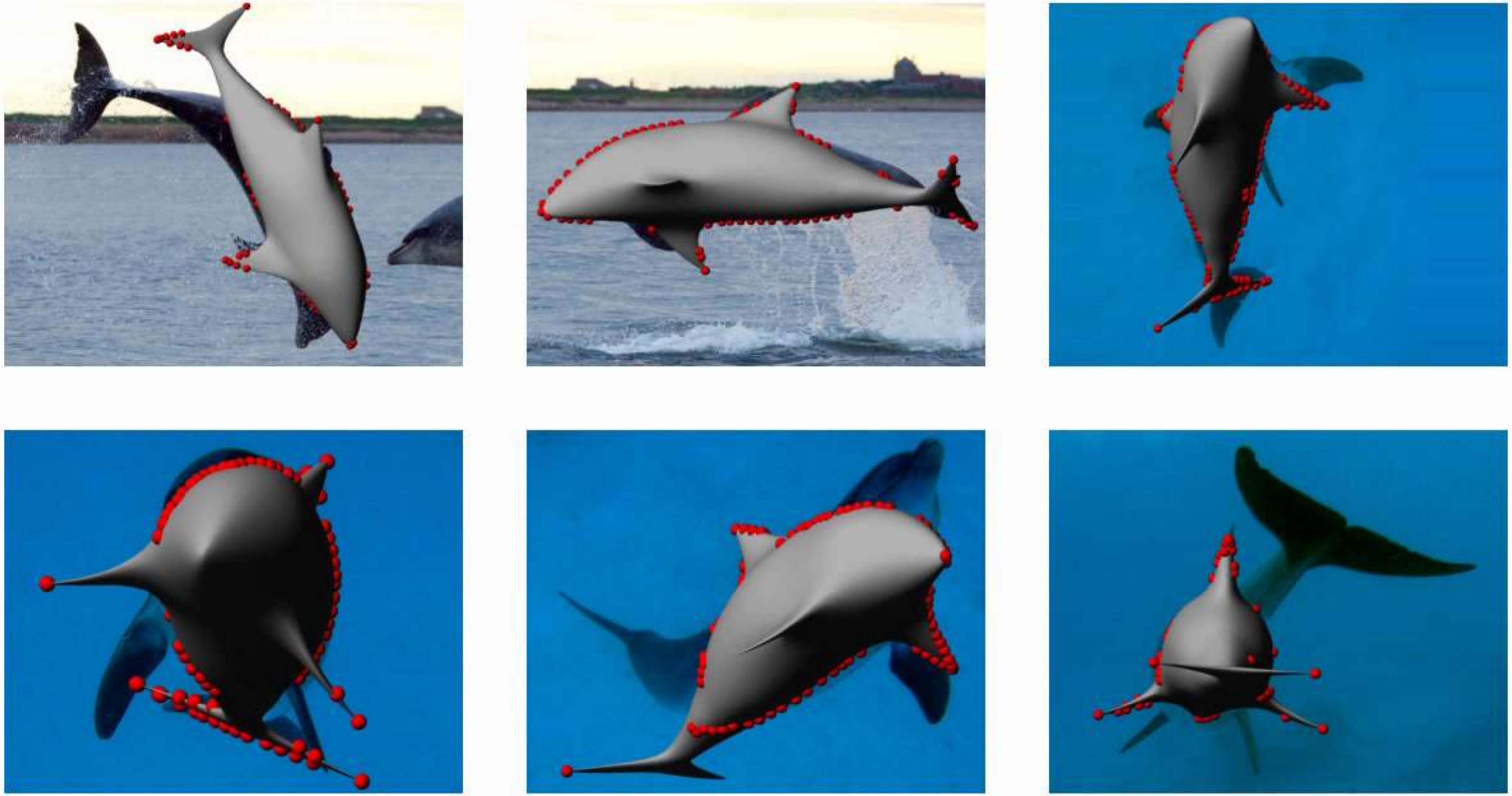
INITIAL ESTIMATE FOR MEAN SHAPE



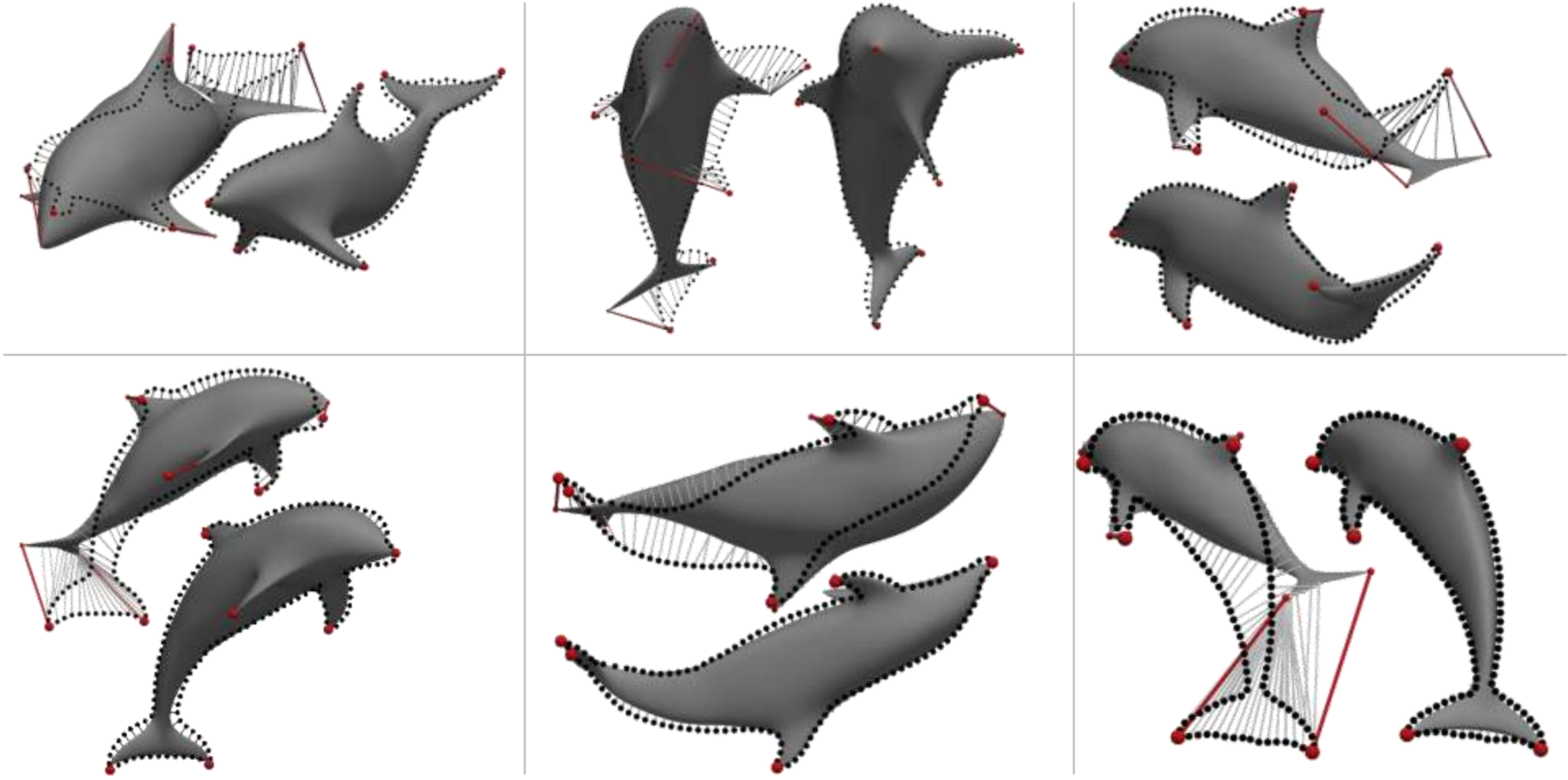
True initial estimate: only the *topology* is really important.
But the easiest way to get the topology is to build a rough template.

INITIAL ESTIMATE FOR MEAN SHAPE

Microsoft

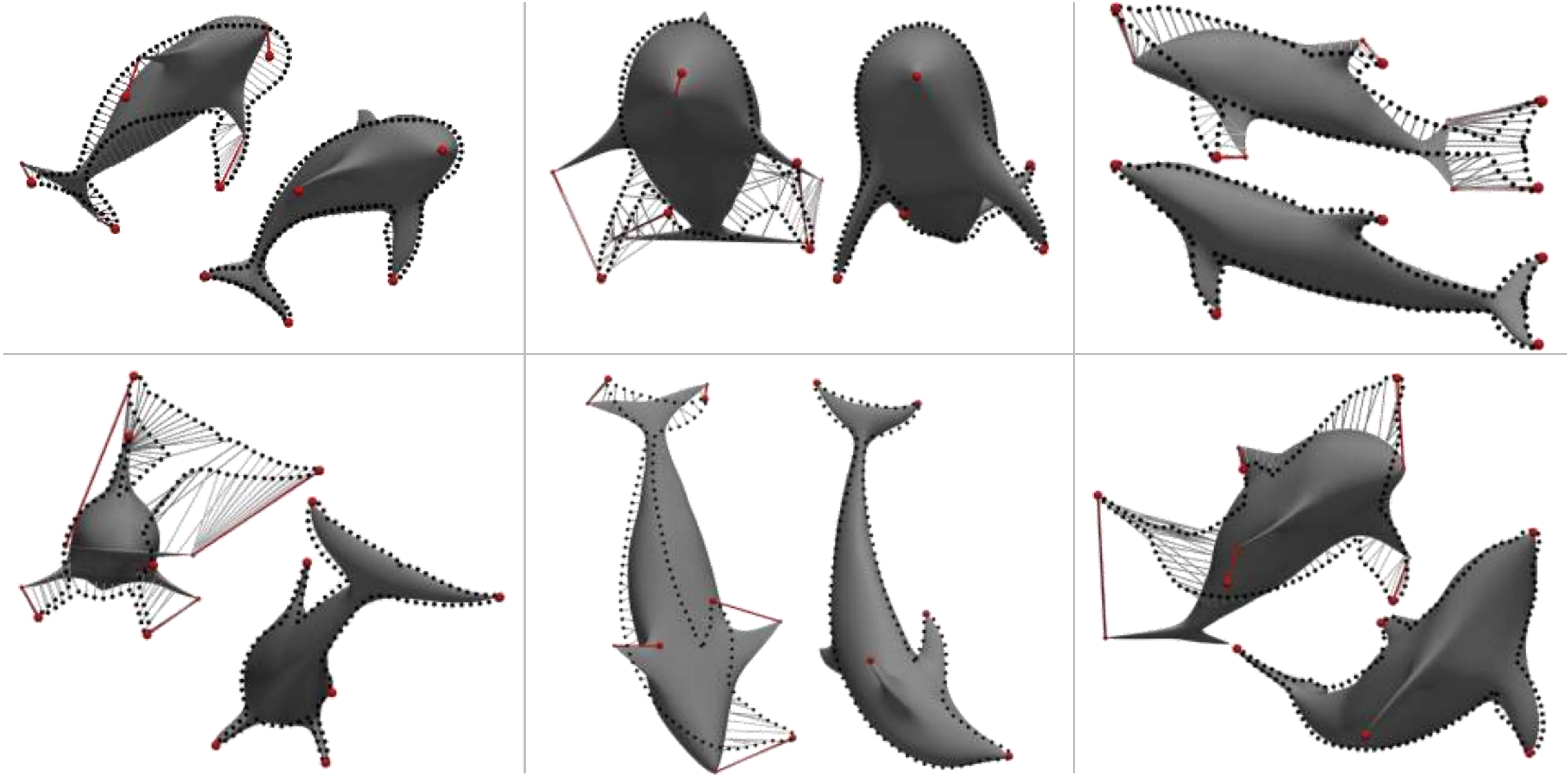


Morphable model parameters: I



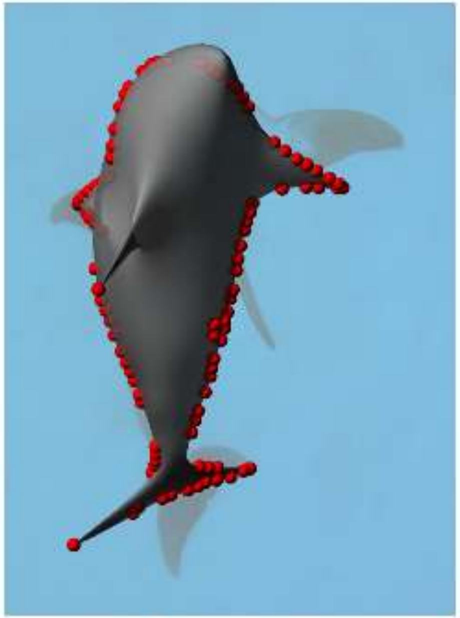
EXAMPLE RESULTS

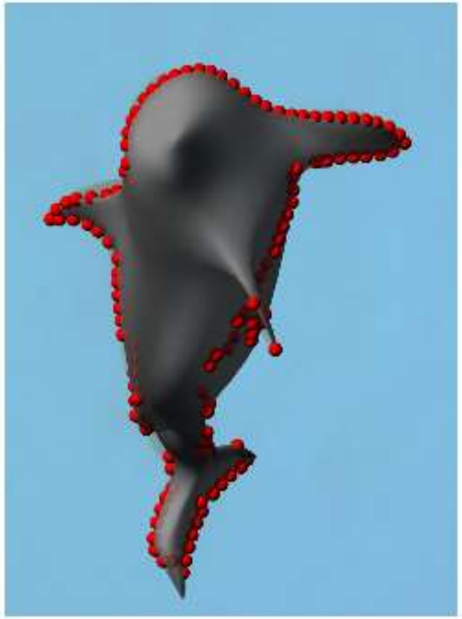
Microsoft

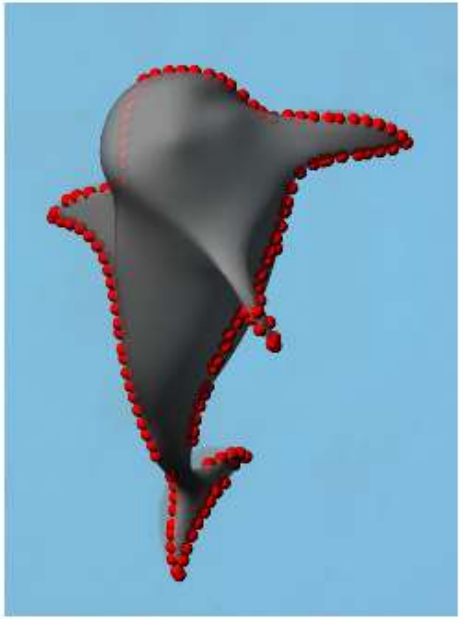


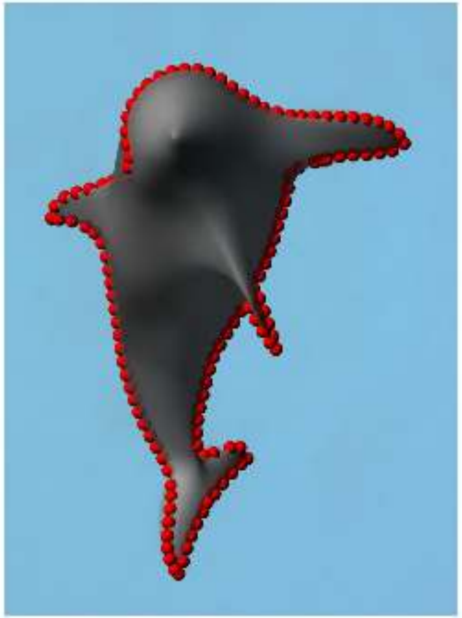
EXAMPLE RESULTS

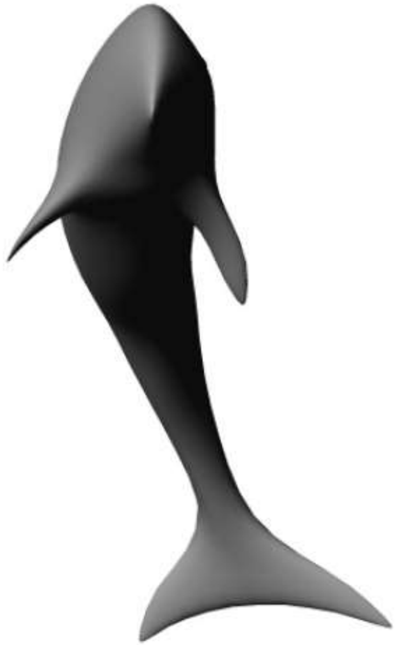
Microsoft

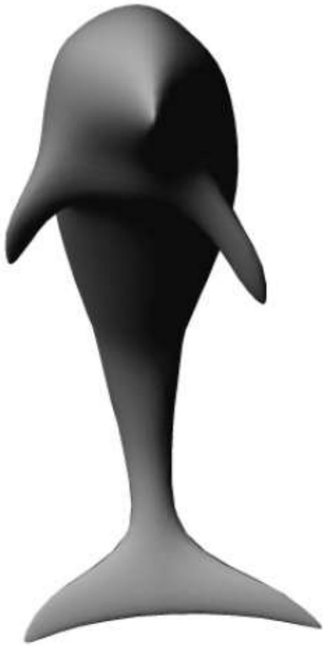

(a) Initial estimate.

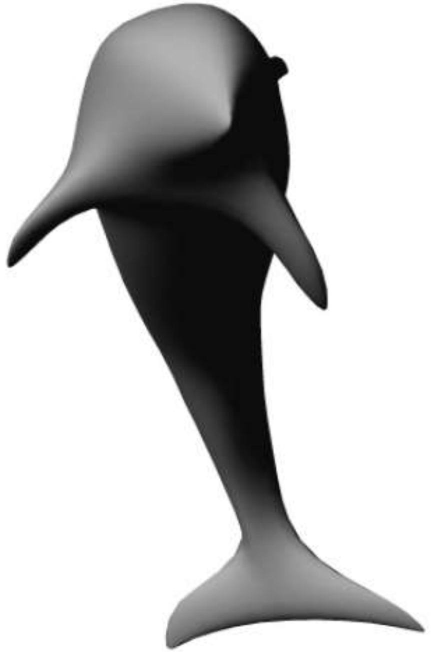

(b) Only continuous local optimization, as described in Sec. 4.1.


(c) As (b), but including iterations of our global search (Sec. 4.2).


(d) As (c), but with reparametrization around extraordinary vertices.

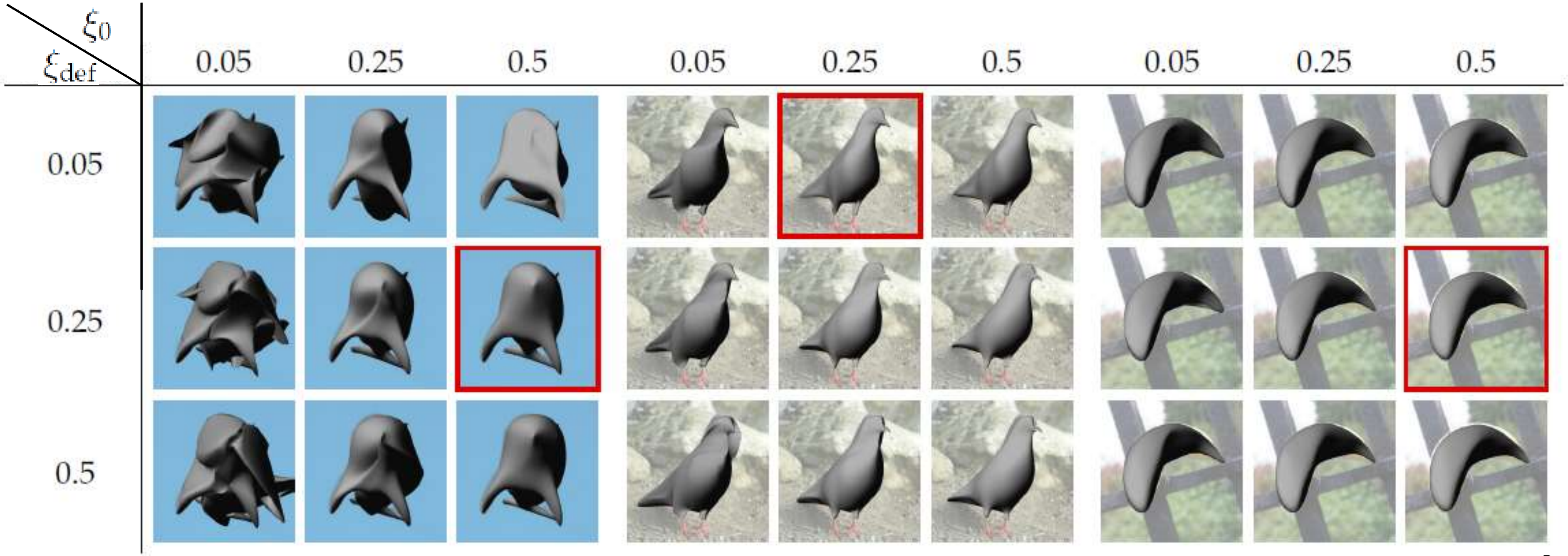

8

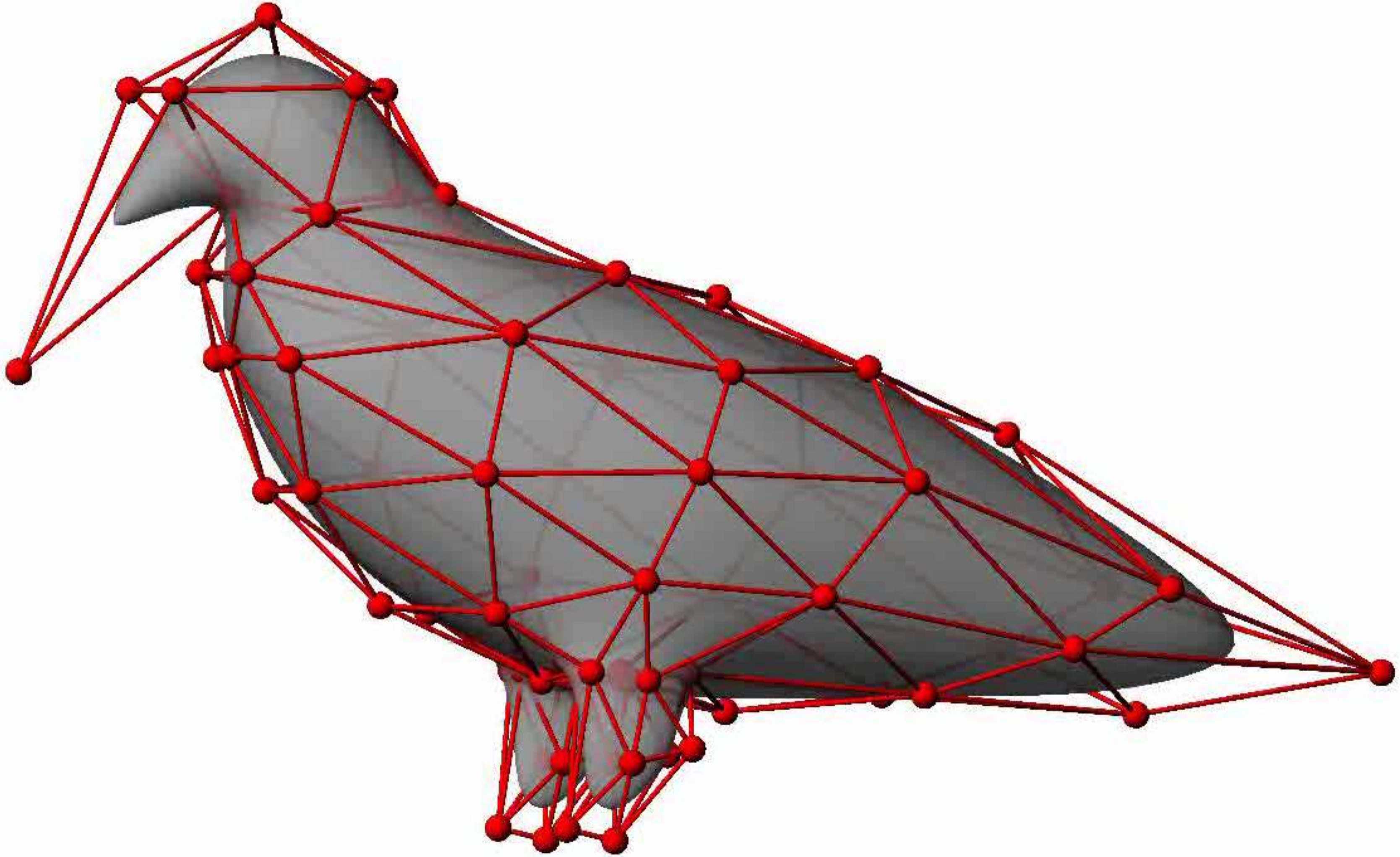

16

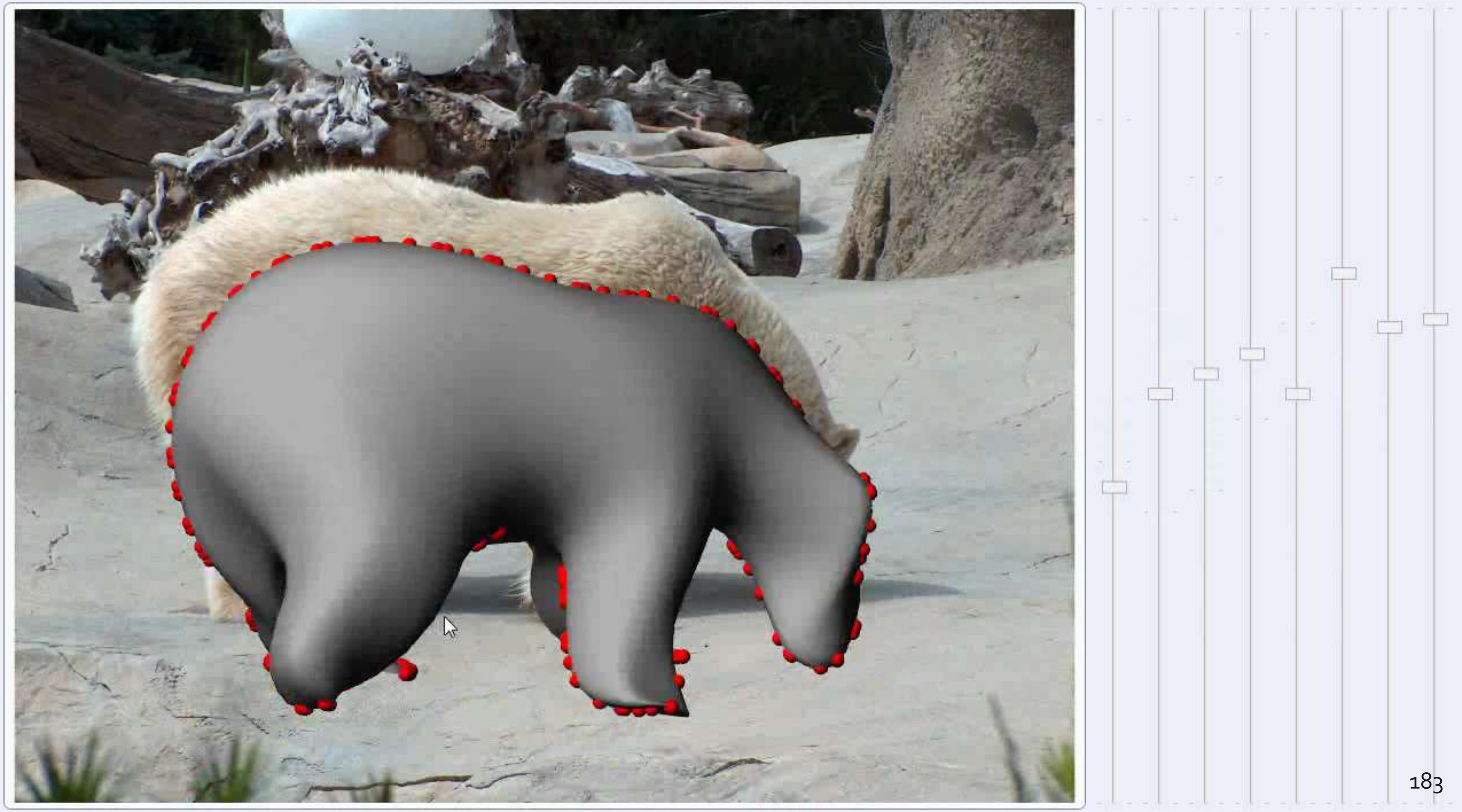

32

NUMBER OF IMAGES

$$E = \overset{\text{“Pixel” terms: noise level params}}{\boxed{\sum_{i=1}^{n}\left(E_i^{\mathrm{sil}} + E_i^{\mathrm{norm}} + E_i^{\mathrm{con}}\right)}} + \overset{\text{“Dimensionless” terms}}{\boxed{\sum_{i=1}^{n}\left(E_i^{\mathrm{cg}} + E_i^{\mathrm{reg}}\right)}} + \overset{\text{“Smoothness” terms}}{\boxed{\boldsymbol{\xi_0^2} E_0^{\mathrm{tp}} + \boldsymbol{\xi_{\mathrm{def}}^2} \sum_{i=1}^{n} E_m^{\mathrm{tp}}}}$$





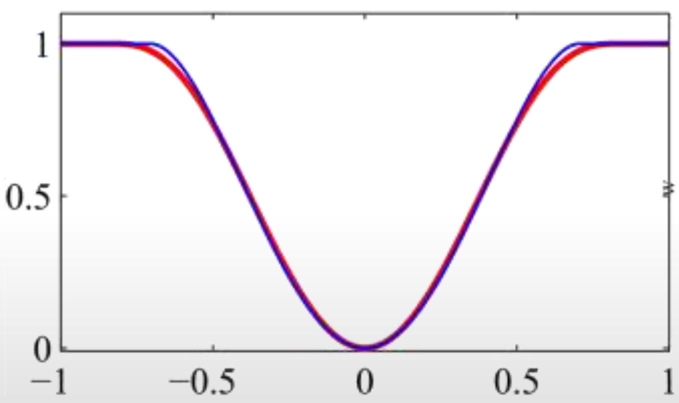


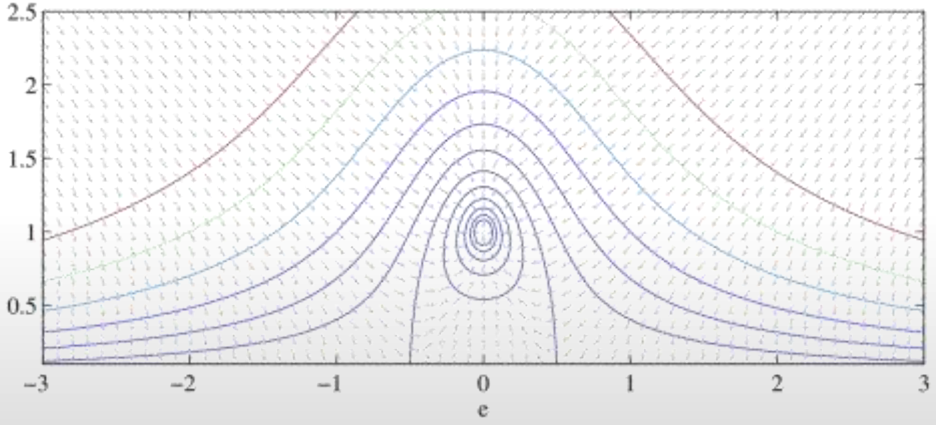
DELL

$$\psi(x) = \min_w w^2x^2 + (1-w^2)^2 = f(x) = \begin{cases} \frac{r^2}{2}\left(2 - \frac{r^2}{2}\right), & x < 0 \\ 1, & x \geq 0 \end{cases}$$

$$\psi(x) = \min_w \phi(x, w)$$ [Zöllhofer et al '14]

$$\phi(x,w) = w^2x^2 + (1-w^2)^2$$ [Li, Sumner, Pauly '08]





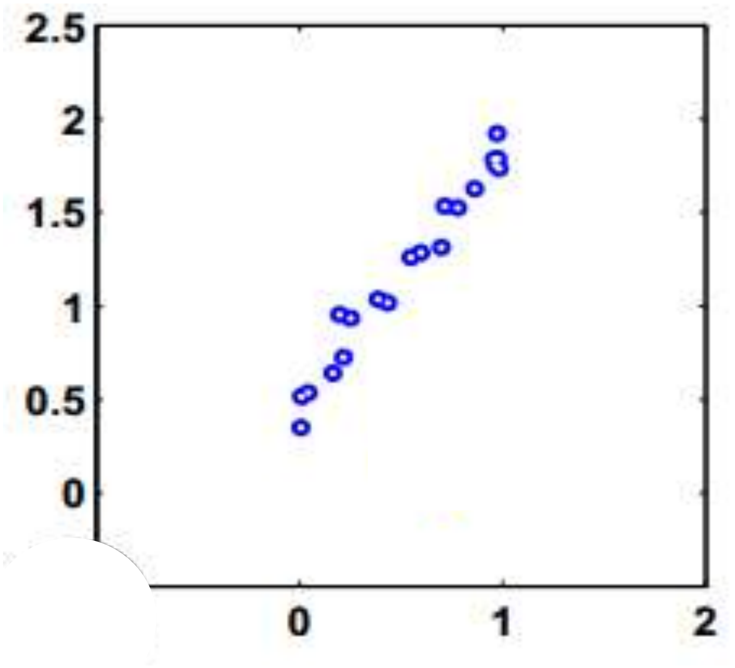
Red: Tukey's biweight

Blue: "Lifted" kernel ψ

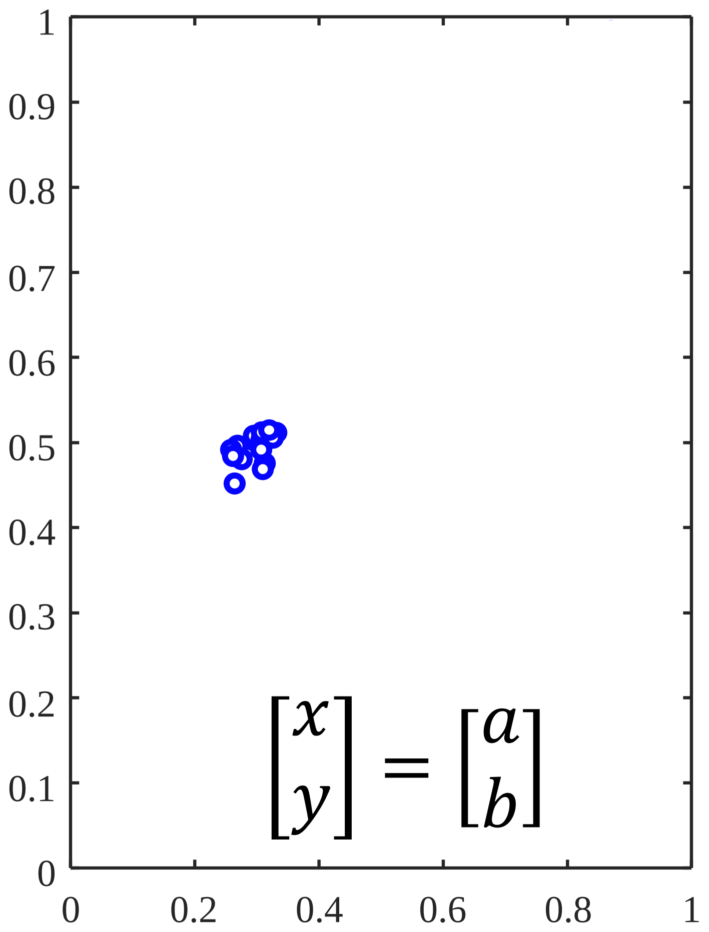
CONVERGES FASTER, FROM FARTHER AWAY

Robust estimation

[BLACK AND RANGARANJAN, CVPR 91] – NEARLY
[LI, PAULY, SUMNER, SIGGRAPH 08] – NEARLY
[ZOLLHÖFER, SIGGRAPH 14] — BASICALLY
[ZACH, ECCV 14] — DEFINITELY

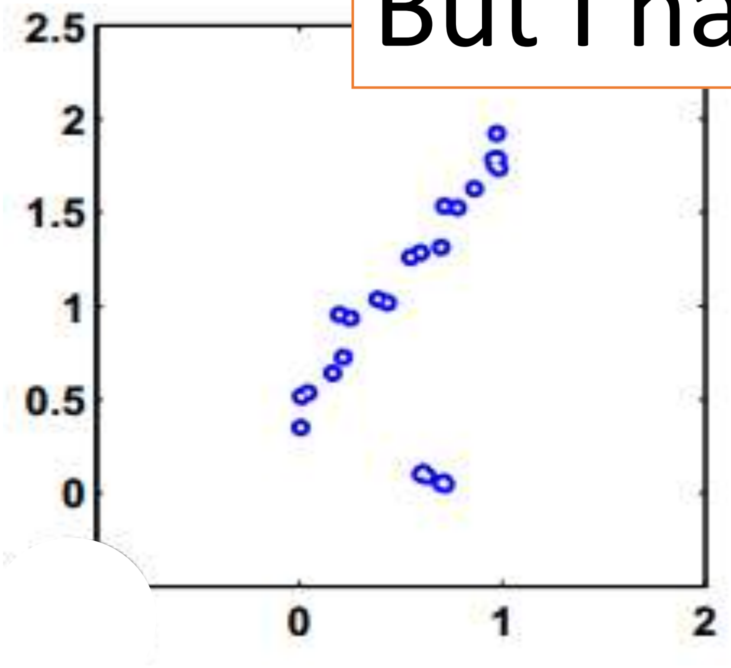


$$y = ax + b$$

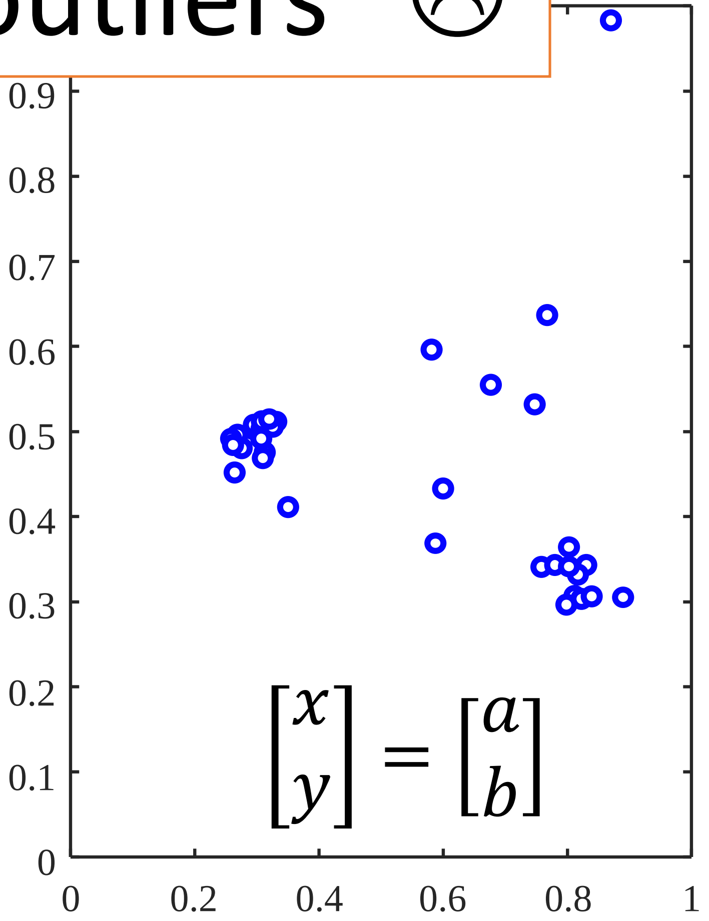


$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

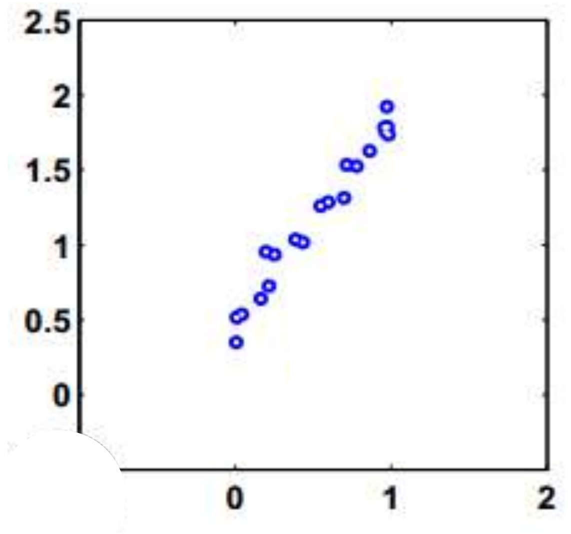
But I have “outliers” ☹



$$y = ax + b$$



$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

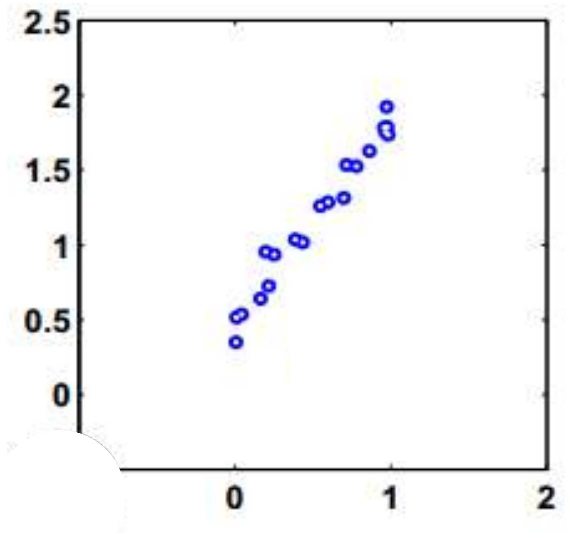


How do I fit a line to data samples $\boldsymbol{s}_i = (x_i, y_i)$?

For this example, let us suppose true inlier model is $y = a_1 x + a_2 + \mathcal{N}(0, \sigma)$

Alg. 1: $\boldsymbol{a} = [\boldsymbol{x}\ \mathrm{ones}(\boldsymbol{x})] \backslash \boldsymbol{y}$

Alg. 2: $\boldsymbol{a} = \underset{\boldsymbol{a}}{\operatorname{argmin}} \sum_i (y_i - a_1 x_i - a_2)^2$



How do I fit a line to data samples $\boldsymbol{s}_i = (x_i, y_i)$?

For this example, let us suppose true inlier model is $y = a_1 x + a_2 + \mathcal{N}(0, \sigma)$

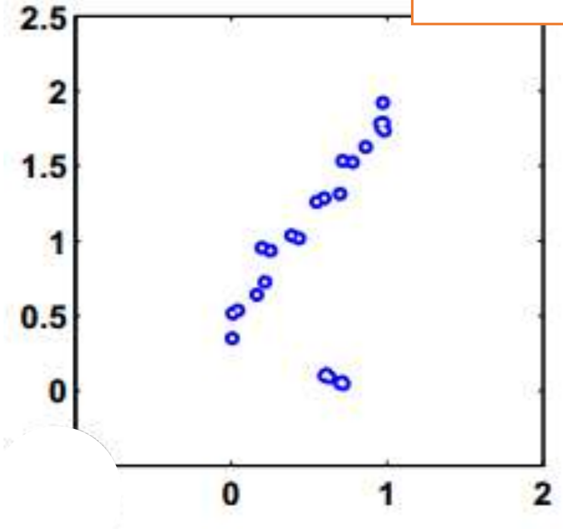
Alg. 1: $\boldsymbol{a} = [\boldsymbol{x}\ \text{ones}(\boldsymbol{x})] \backslash \boldsymbol{y}$

Alg. 2: $\boldsymbol{a} = \underset{\boldsymbol{a}}{\operatorname{argmin}} \sum_i (y_i - a_1 x_i - a_2)^2$

```
>> a = lsqnonlin(@(a) y - a(1)*x - a(2), [1 1]);
```

Works really well because objective is sum-of-squares

But I have “outliers” ☹

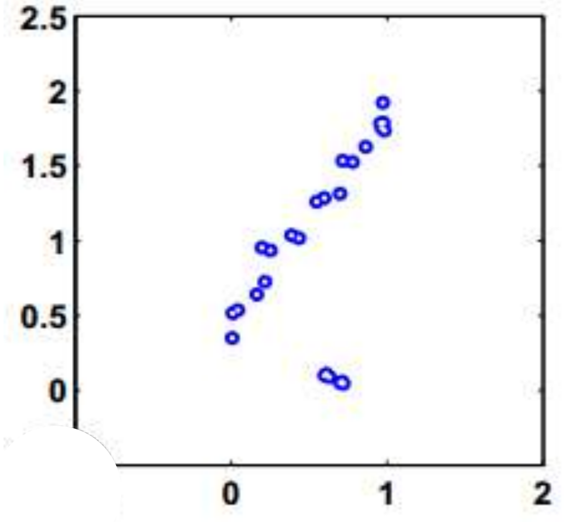


How do I fit a line to data samples $\boldsymbol{s}_i = (x_i, y_i)$?

For this example, let us suppose true inlier model is $y = ax + b + \mathcal{N}(0, \sigma)$

Alg. 1: ~~$\boldsymbol{a} = [\boldsymbol{x}\ \mathrm{ones}(\boldsymbol{x})] \backslash \boldsymbol{y}$~~

Alg. 2: $\boldsymbol{a} = ?$



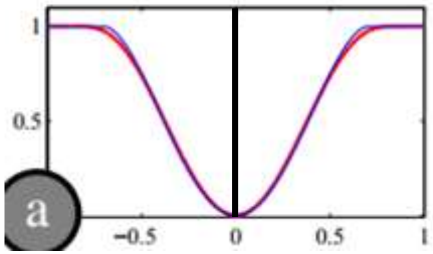
How do I fit a line to data samples $\boldsymbol{s}_i = (x_i, y_i)$?

For this example, let us suppose true inlier model is $y = ax + b + \mathcal{N}(0, \sigma)$

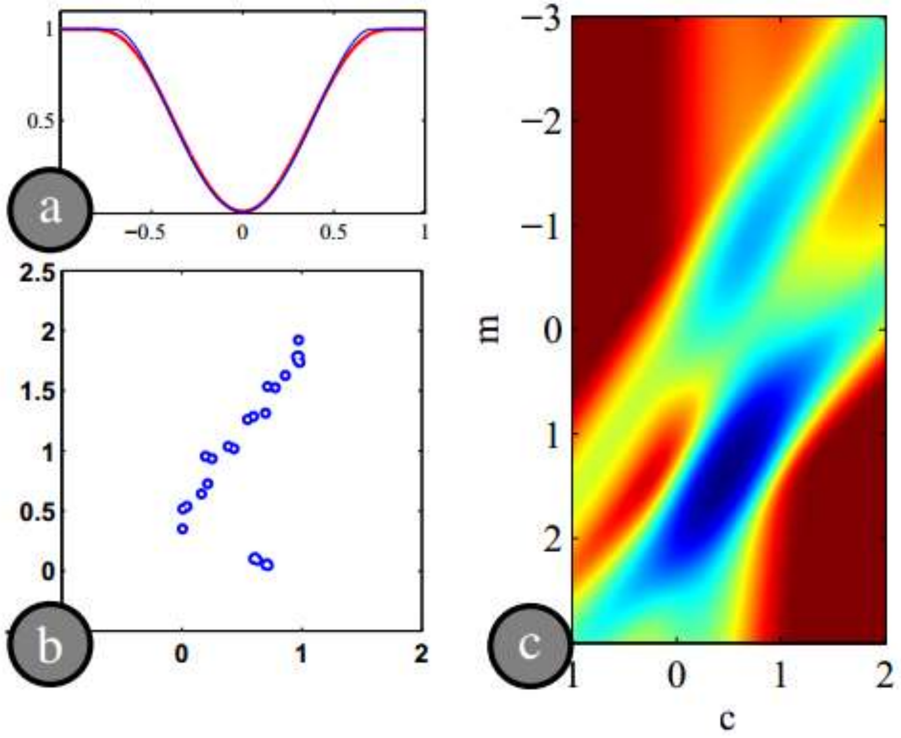
Alg. 1: $\boldsymbol{a} = [\boldsymbol{x}\ \mathrm{ones}(\boldsymbol{x})] \backslash \boldsymbol{y}$

Alg. 2: $\boldsymbol{a} = \underset{\boldsymbol{a}}{\operatorname{argmin}} \sum_i \psi(y_i - a_1 x_i - a_2)$

```
>> a = fminunc(@(a) sum(psi(y - a(1)*x - a(2))), [1
1]);
```



$\psi(x)$



$$\min_{\boldsymbol{a}} \sum_i \psi(y_i - a_1 x_i - a_2)$$

Global minimum in a good place

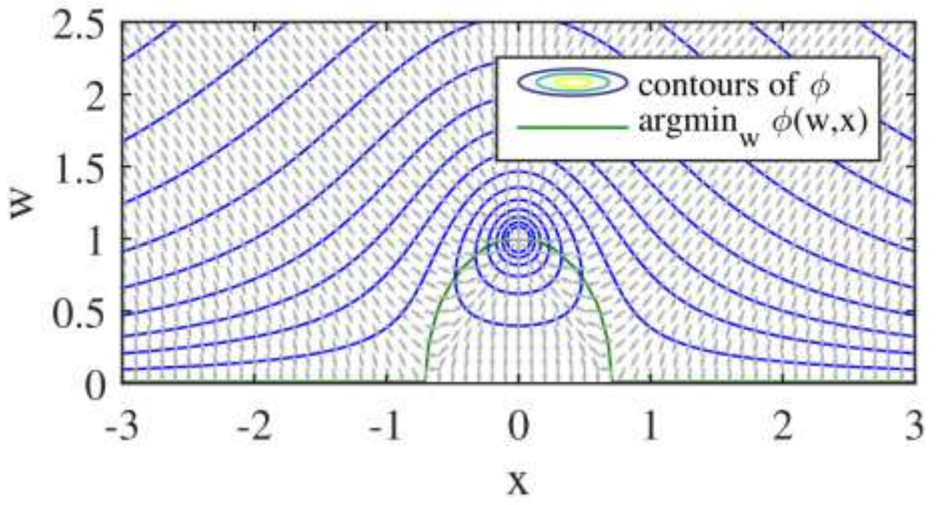
But hard to optimize:

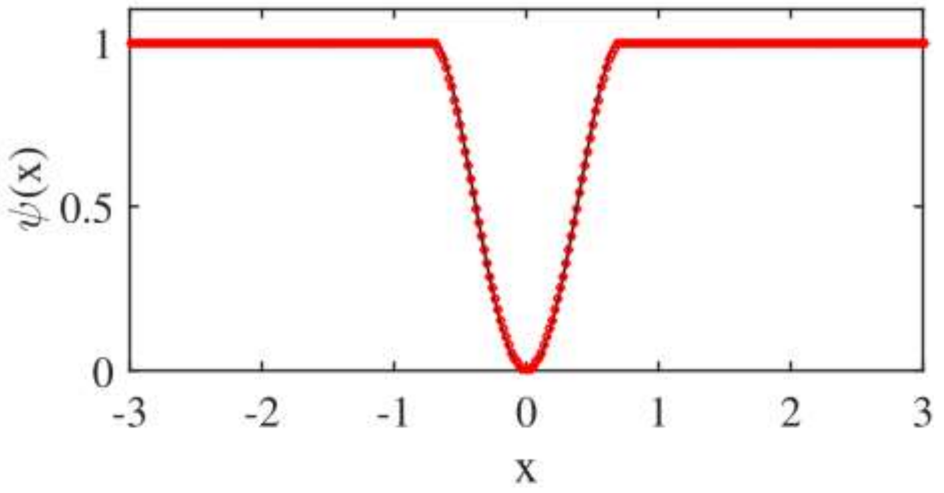
- Multiple optima
- Huge flat spots

Robust kernels can be expressed as minimization over “outlier process” variables [e.g. Geman & Reynolds ‘92, Black & Rangarajan ‘95]

$$\phi(x, w) = w^2 x^2 + (1 - w^2)^2$$

$$\psi(x) = \min_w \phi(x, w)$$





Data residual for i^{th} data point: $f_i(\boldsymbol{a}) = y_i - a_1 x_i - a_2$

"Lifted" robust kernel: $\phi(x, w) = w^2 x^2 + (1 - w^2)^2$

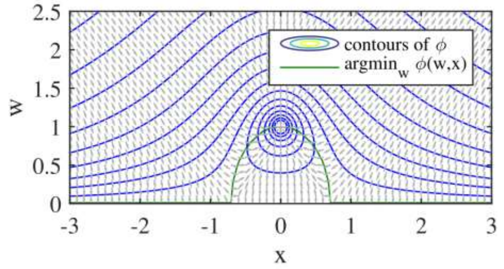
Gives kernel: $\psi(x) = \min_w \phi(x, w)$

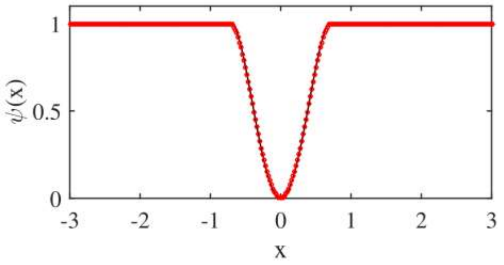
And original nasty problem: $\min_{\boldsymbol{a}} \sum_i \psi(f_i(\boldsymbol{a}))$

Becomes: $\min_{\boldsymbol{a}} \sum_i \min_w w^2 f_i^2(\boldsymbol{a}) + (1 - w^2)^2$

$$\min_{\boldsymbol{a}} \sum_i \min_{w_i} w_i^2 f_i^2(\boldsymbol{a}) + \left(1 - w_i^2\right)^2$$

$$\min_{\boldsymbol{a}} \min_{w_i} \sum_i w_i^2 f_i^2(\boldsymbol{a}) + \left(1 - w_i^2\right)^2$$



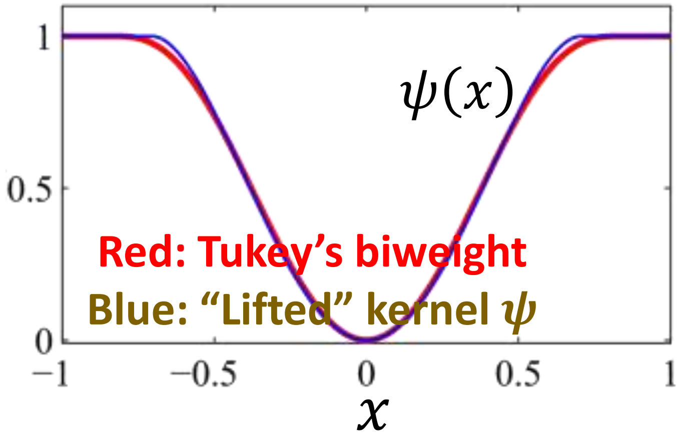


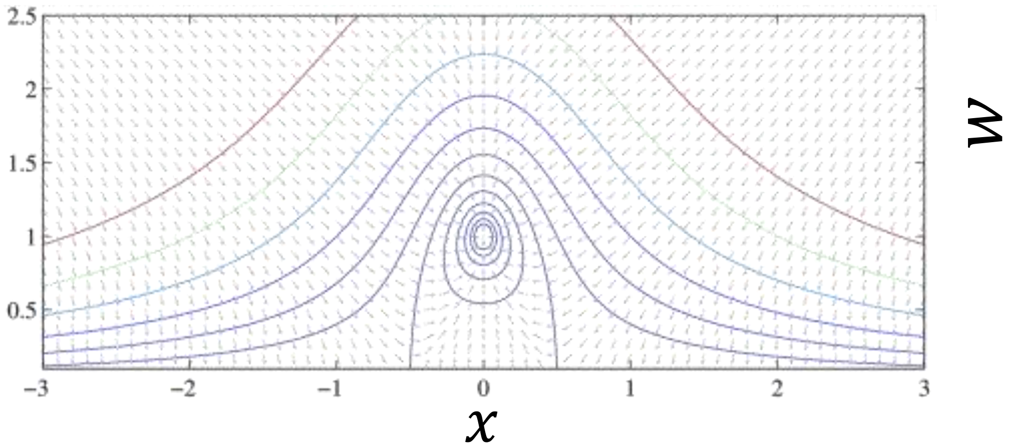
Which is in the Gauss-Newton form…

$$\psi(x) = \min_{w} w^2x^2 + (1-w^2)^2 = f(x) = \begin{cases} \frac{r^2}{2}\left(2 - \frac{r^2}{2}\right), & x < 0 \\ 1, & x \geq 0 \end{cases}$$

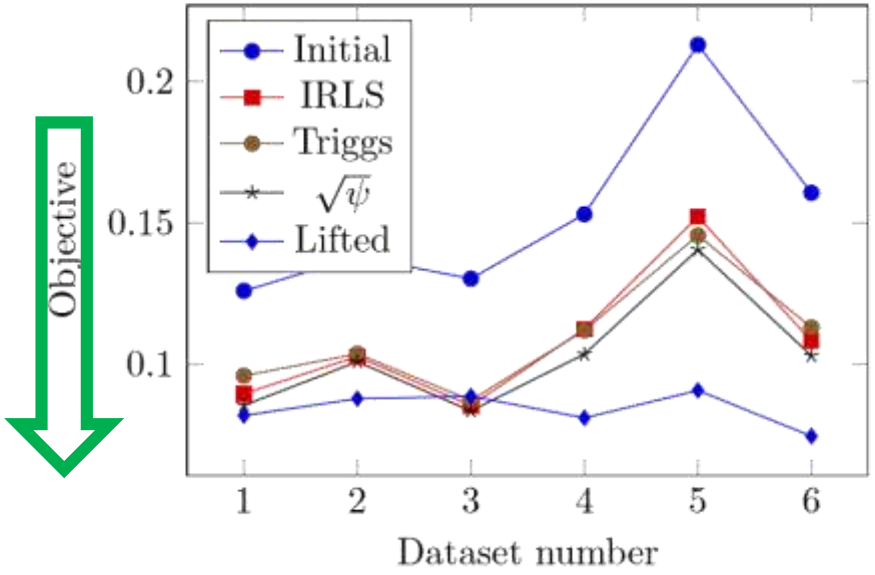
$$\psi(x) = \min_{w} \phi(x, w)$$ [Zöllhofer et al ’14]

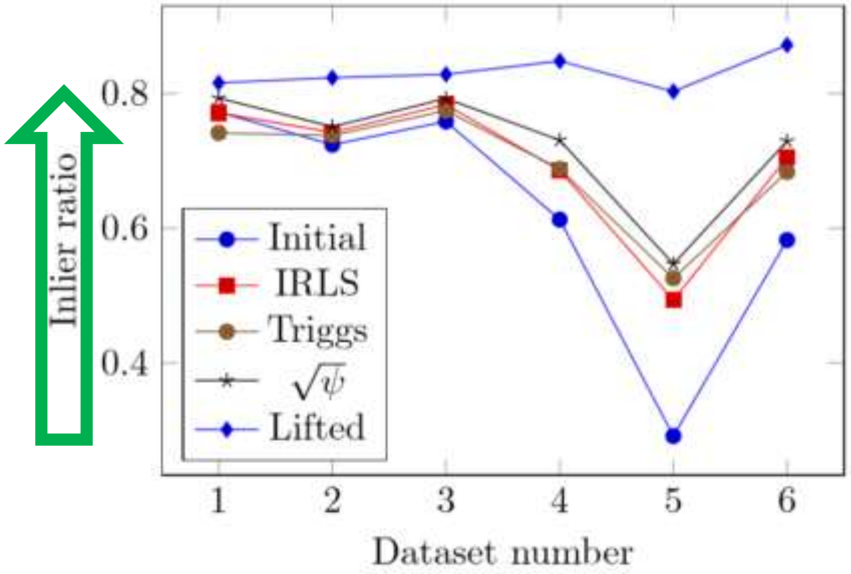
$$\phi(x, w) = w^2x^2 + (1-w^2)^2$$ [Li, Sumner, Pauly ’08]





3D reconstruction datasets: up to 10^6 parameters, 10^6 measurements





Before [Zach '14], no-one used the Gauss-Newton structure, so never beat IRLS (iterated reweighted least squares), with its ICP-like convergence.

Robust kernels can be expressed as minimization over "outlier process" variables [e.g. Geman & Reynolds '92, Black & Rangarajan '95]

Residual r_i passes through robust kernel $\psi(r)$, e.g.

$$\psi(r) = \frac{r^2}{1+r^2} = \min_s (s^2 r^2 + (1-s)^2)$$

And

$$\min_\theta \sum_{i=1}^{n} \psi\big(r_i(\theta)\big) \quad \rightarrow \quad \min_{\theta, s_1, \ldots, s_n} \sum_{i=1}^{n} \phi(r_i(\theta), s_i)$$

But until [Zach '14], no-one used Gauss-Newton structure of RHS, so never beat IRLS (iterated reweighted least squares), with its ICP-like convergence.

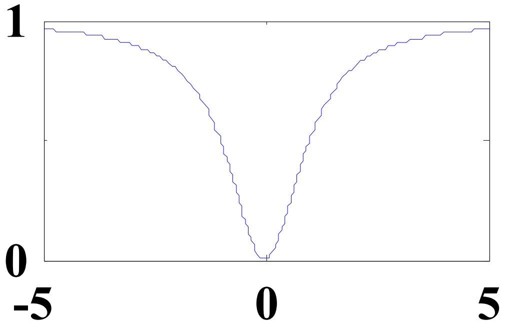
BUNDLE ADJUSTMENT WITH ROBUST KERNELS

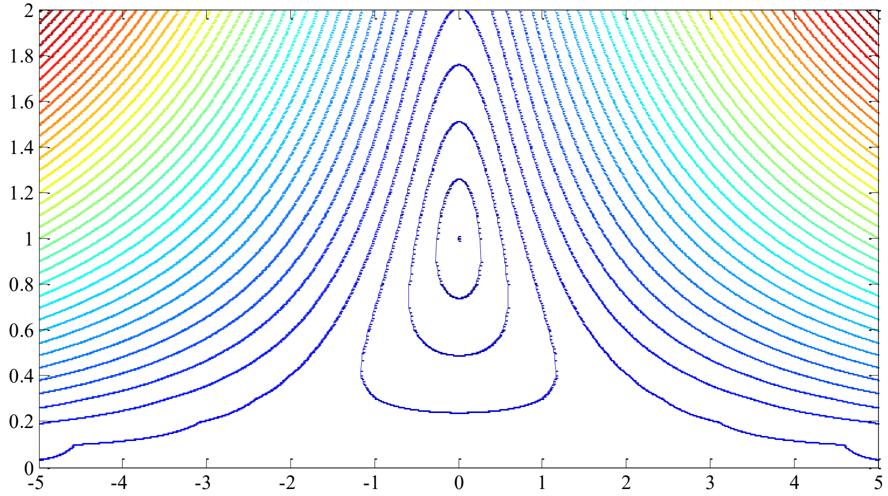
Microsoft

Robust kernels can be expressed as minimization over “outlier process” variables [e.g. Geman & Reynolds ‘92, Black & Rangarajan ‘95]

Residual r_i passes through robust kernel $\psi(r)$, e.g.

$$\psi(r) = \frac{r^2}{1+r^2} = \min_s(s^2 r^2 + (1-s)^2)$$





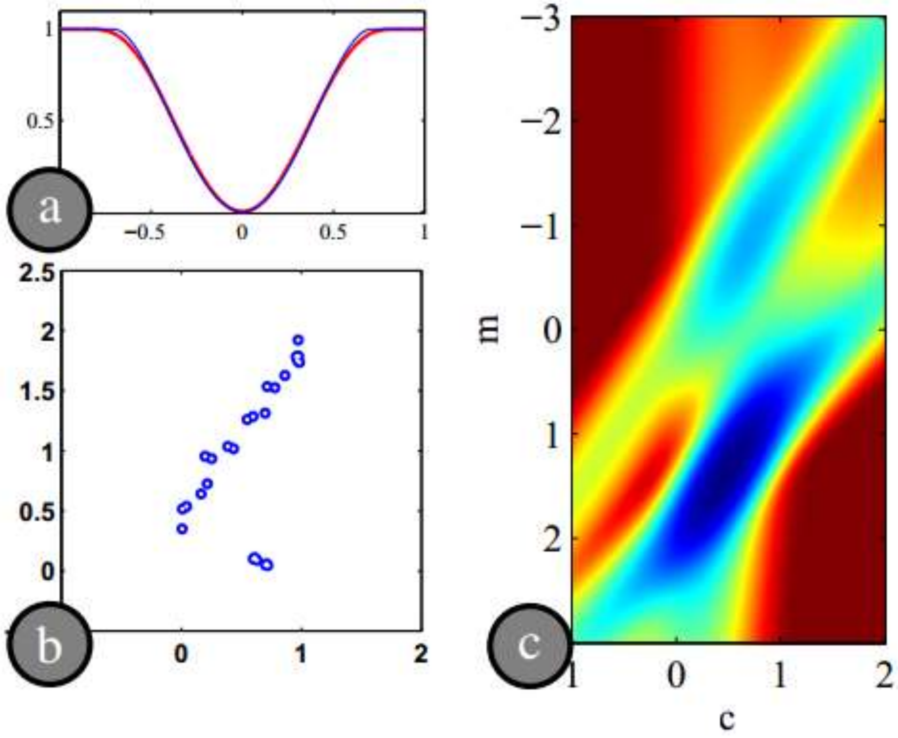
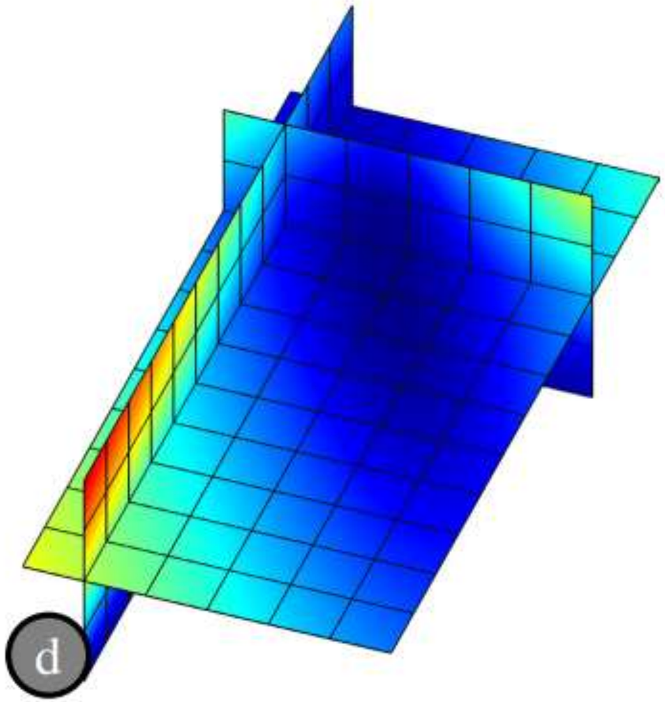
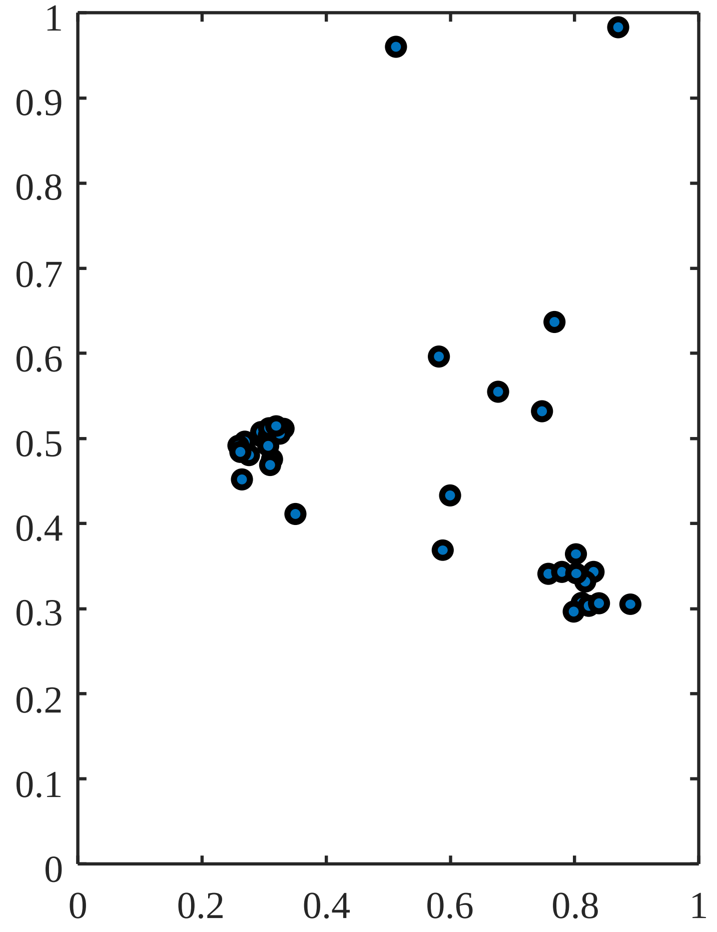


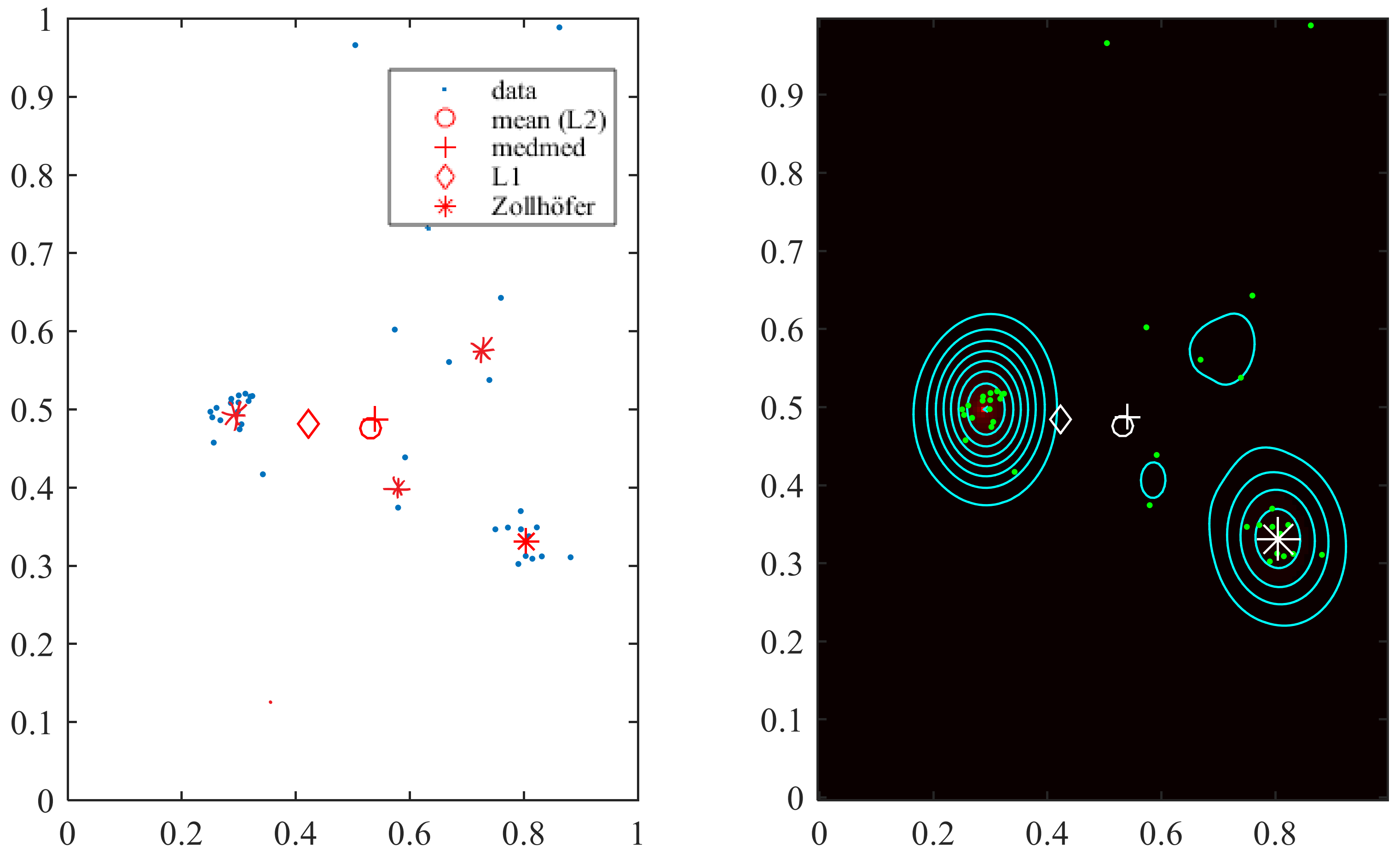
Figure 4: *Robust kernel (Sec. 5.1.2).* **(a)** *Our kernel $\psi(e)$ (blue) has similar shape to the standard Tukey's biweight kernel (red).* **(b)** *A 2D line fitting problem with two minima. Data points $y_i \approx mx_i + c$.* **(c)** *Energy landscape of $f(m, c) = \sum_i \psi(y_i - mx_i - c)$ is complicated.*



(d) *3D slice through $(2+n)$ dimensional landscape of lifted function $F(m, c, w_1, ..., w_n) = \sum_i w_i^2(y_i - mx_i - c)^2 + (1 - w_i^2)^2$ is simpler. Minimization of lifted F found the global optimum on 82.4% of runs, in contrast to 43.0% on two-parameter f, which also had 20.1% outright failures vs. 0% on lifted.*



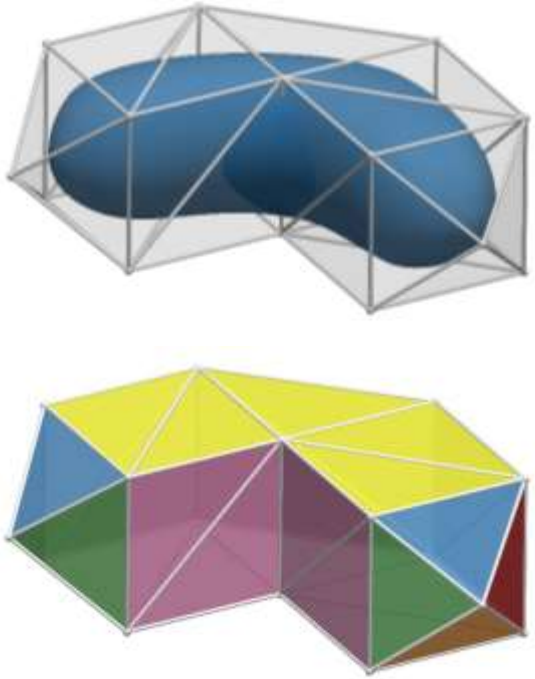
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

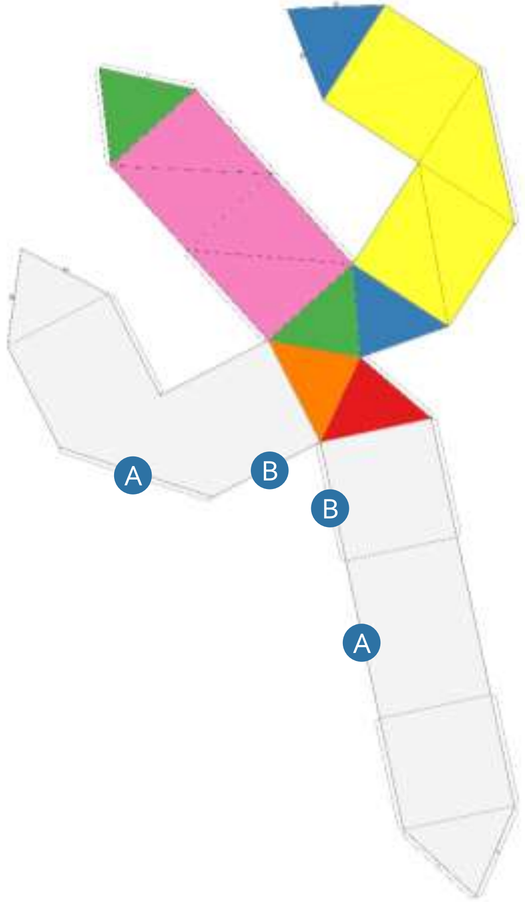


data
mean (L2)
medmed
L1
Zollhöfer
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

SUBDIV PECULIARITIES 1: PIECEWISE DOMAIN

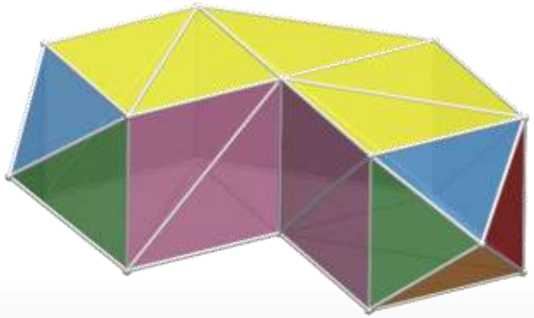
- Parameter domain Ω is in pieces
  - Typically not unwrappable to a plane

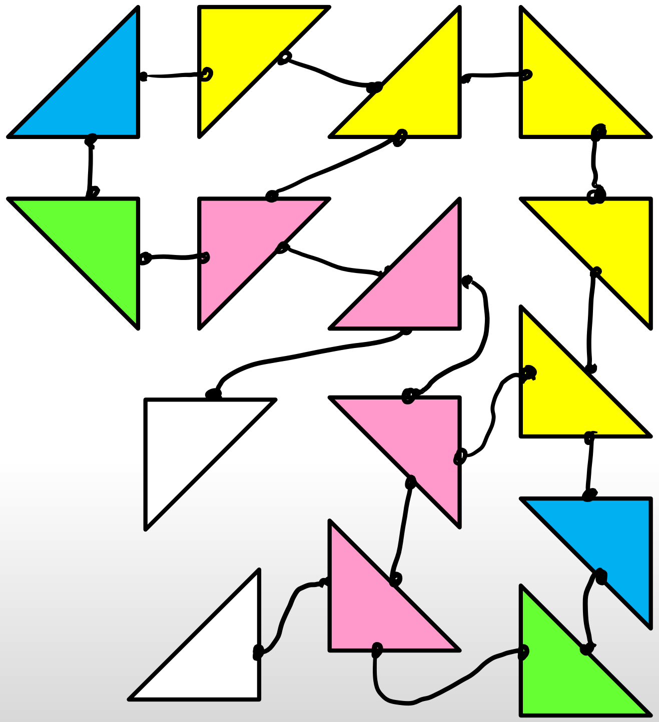




PIECEWISE DOMAIN

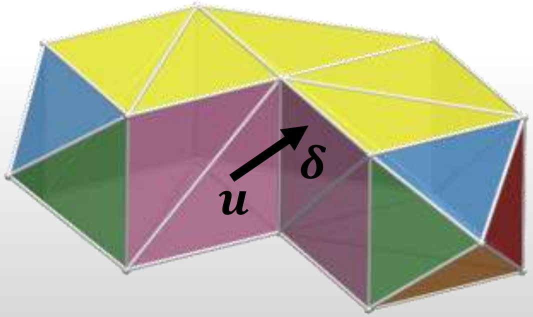
- Parameter domain Ω: pieces with connectivity graph

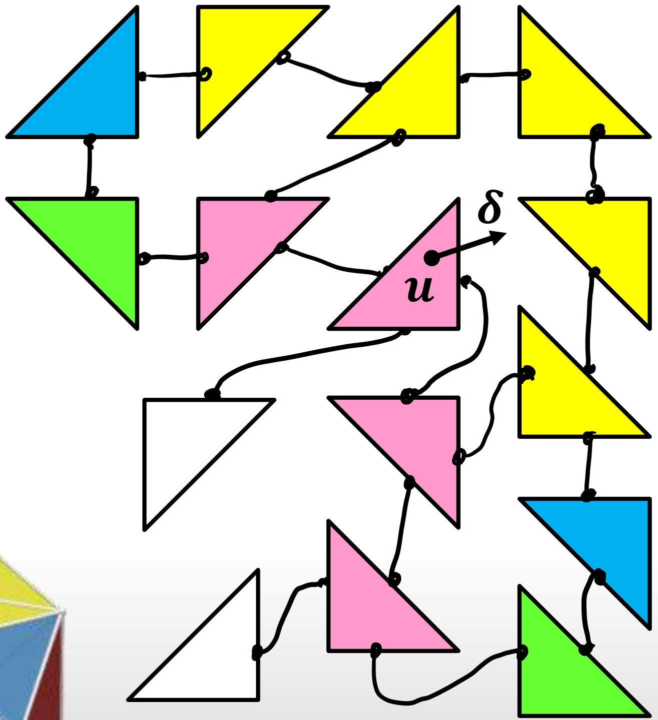




PIECEWISE DOMAIN

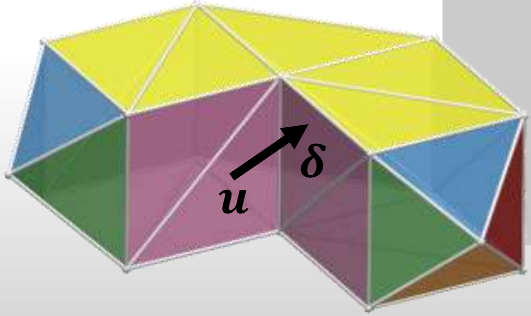
- At point $\boldsymbol{u} = (p, u, v)$
- Easy to get direction $\boldsymbol{\delta}$ from $M_{\boldsymbol{u}}$ etc.
- But need $\boldsymbol{u} + \lambda\boldsymbol{\delta}$
  - **`Override ceres::Evaluator::Plus`**
- Easy *inside* patch
- Need *outside* too

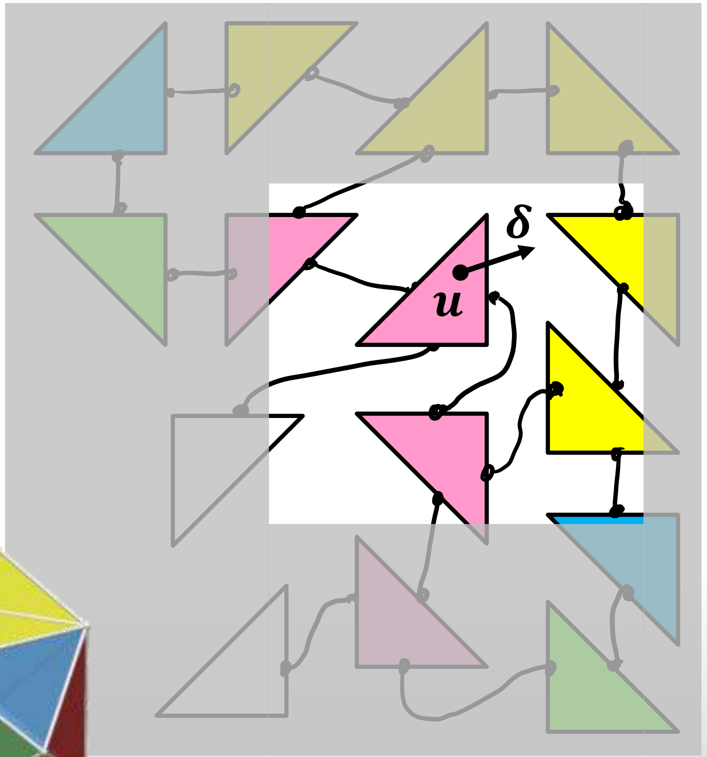




PIECEWISE DOMAIN

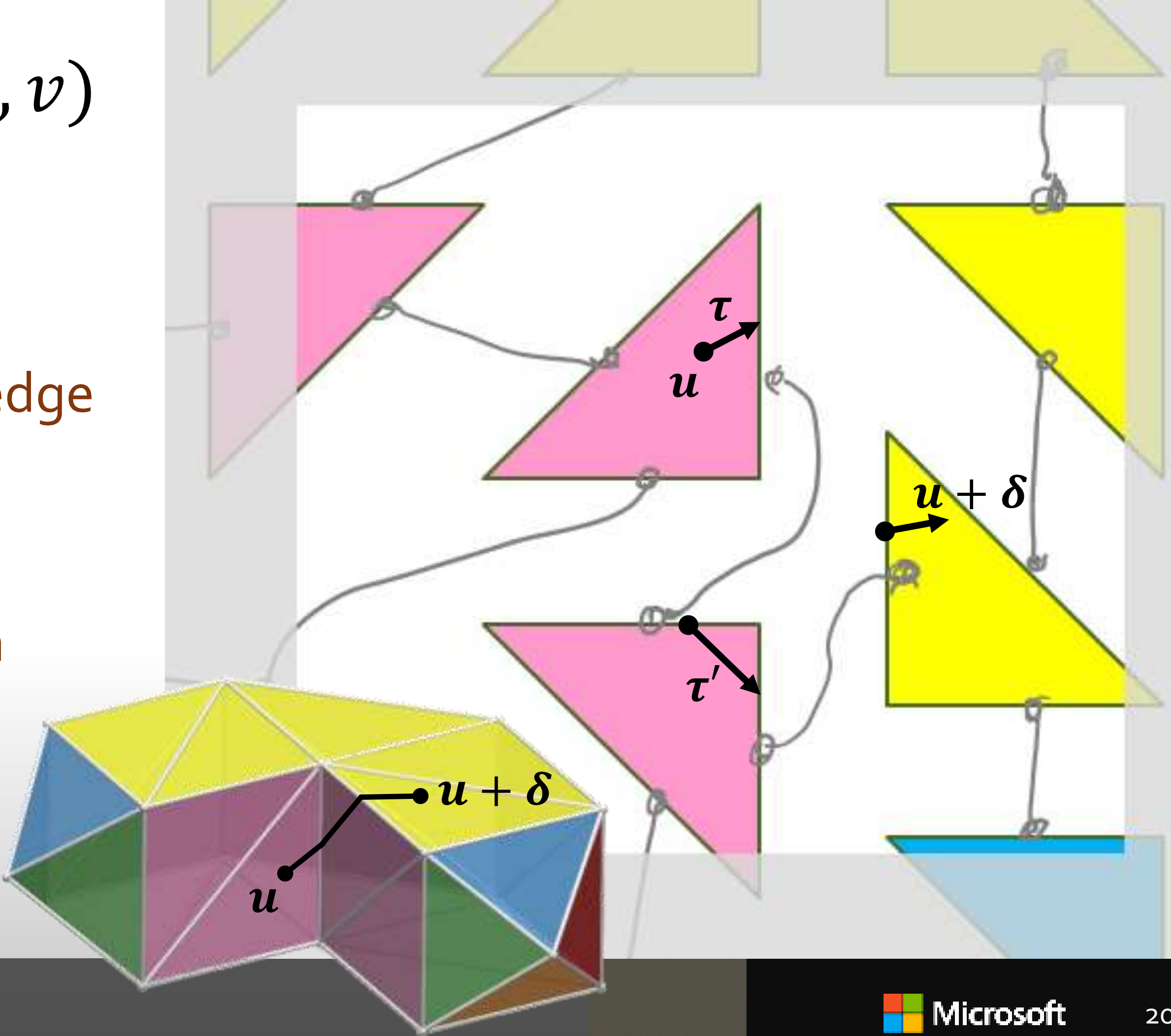
- At point $\boldsymbol{u} = (p, u, v)$
- Need $\boldsymbol{u} + \lambda\boldsymbol{\delta}$
- *Outside* patch:
  - Move distance τ to edge
  - Change direction
  - Move $\delta - \tau$
  - Repeat in next patch





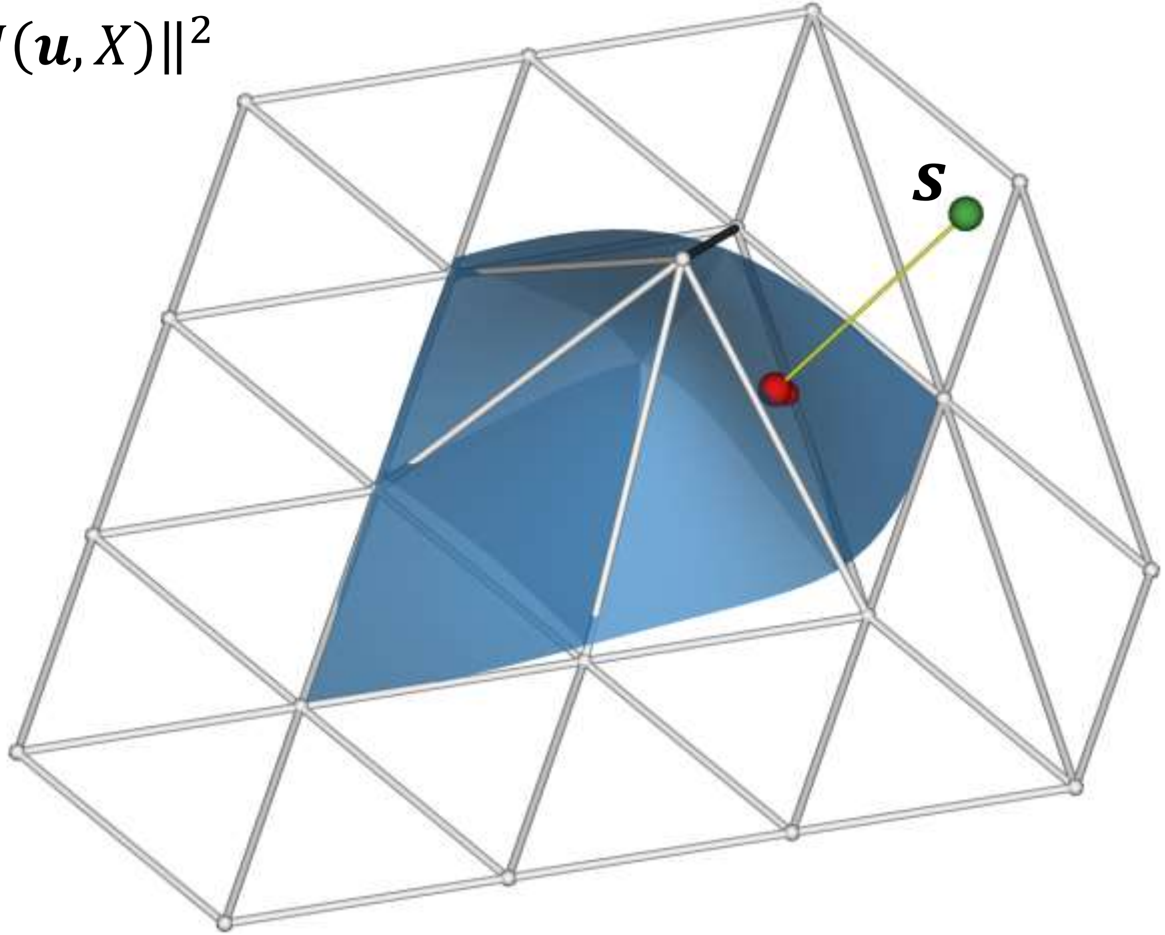
PIECEWISE DOMAIN

- At point $\boldsymbol{u} = (p, u, v)$
- Need $\boldsymbol{u} + \lambda\boldsymbol{\delta}$
- *Outside* patch:
  - Move distance τ to edge
  - Change direction
  - Move $\delta - \tau$
  - Repeat in next patch


$\boldsymbol{\tau}$
$\boldsymbol{u}$
$\boldsymbol{u} + \boldsymbol{\delta}$
$\boldsymbol{\tau}'$
$\boldsymbol{u} + \boldsymbol{\delta}$
$\boldsymbol{u}$

PIECEWISE DOMAIN

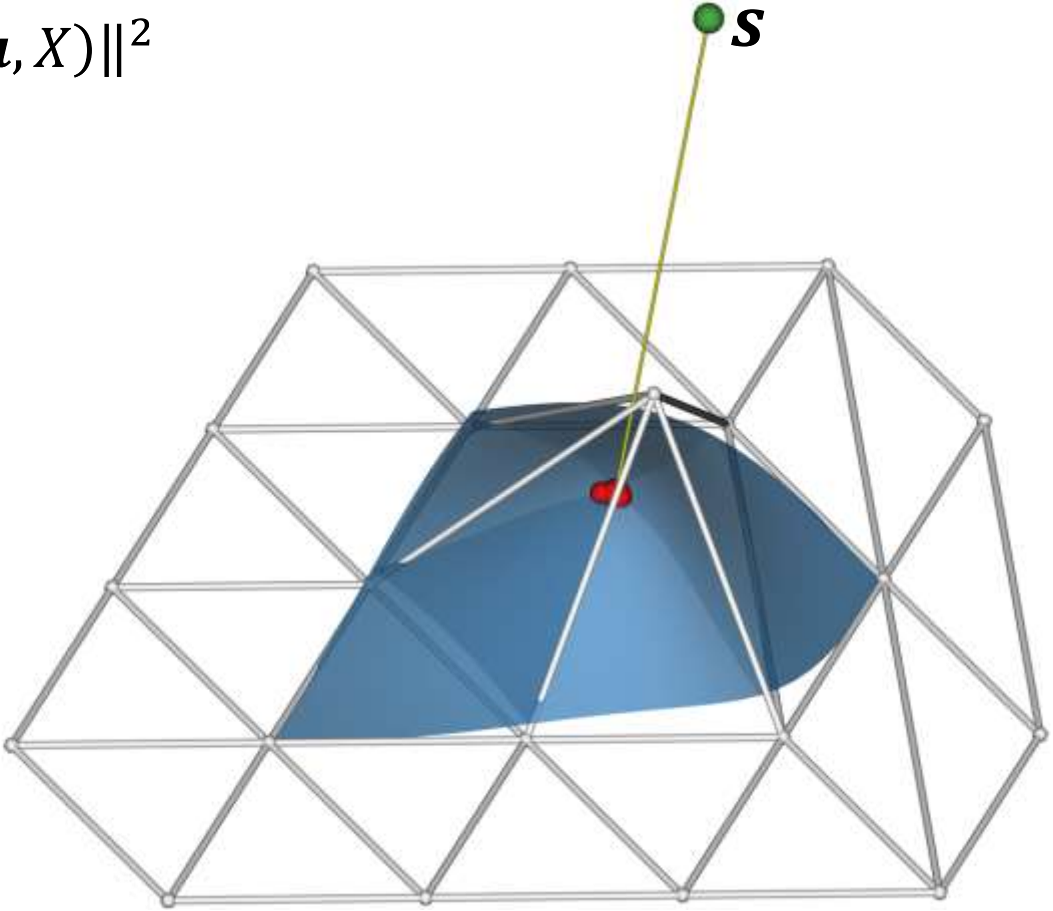
$$E(\boldsymbol{u}) = \|\boldsymbol{s} - M(\boldsymbol{u}, X)\|^2$$



EXAMPLE: SINGLE CLOSEST POINT PROBLEM

Microsoft

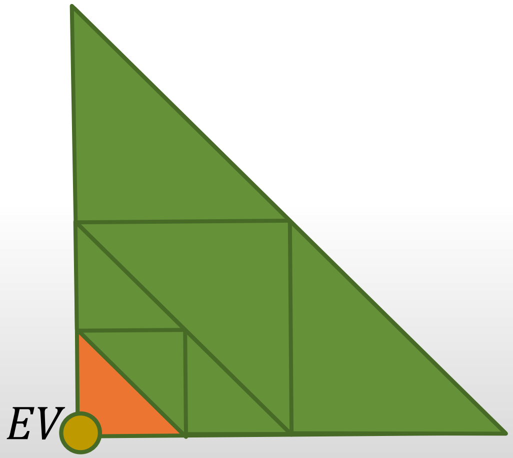
$$E(\boldsymbol{u}) = \|\boldsymbol{s} - M(\boldsymbol{u}, X)\|^2$$



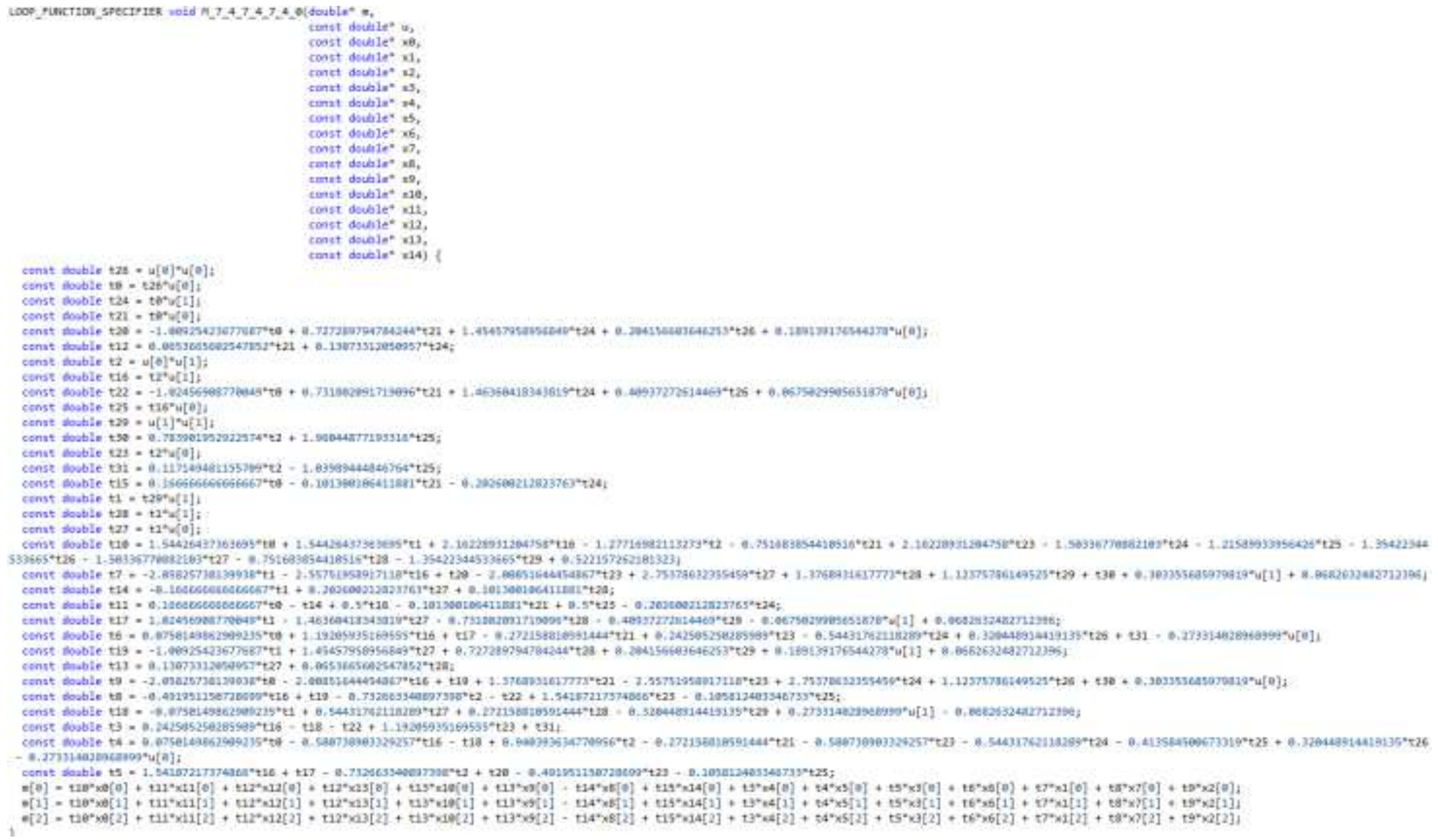
EXAMPLE: SINGLE CLOSEST POINT PROBLEM

SUBDIV PECULIARITIES 2: EXTRAORDINARY VERTICES

- Any vertex of valency $\neq 6$ is an "extraordinary vertex"
  - Call a triangle with an EV an "irregular triangle"
- Normals and surface at EVs well defined and well behaved
  - But spline evaluation rule is not…
- Solution: virtually subdivide irregular triangles
  - Each green element is still linear in X, quartic in u, v
  - Need to generate different A_{ijk} for $\sum A_{ijk} u^i v^j \boldsymbol{X}_k$
  - All autogenerated C code using Sympy
    - Go to depth 5, and then handle "vestigial patch"
    - Initially just use spline coeffs from neighbour



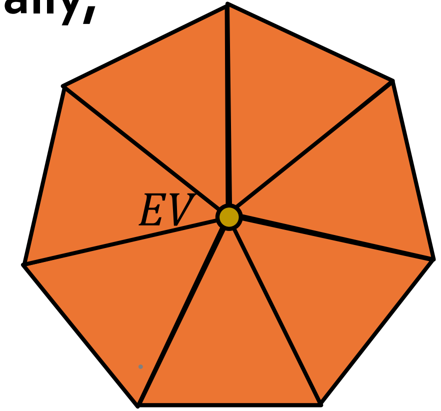
EXTRAORDINARY VERTICES



```
LOOP_FUNCTION_SPECIFIER void M_7_4_7_4_7_4_0(double* m,
                                               const double* u,
                                               const double* x0,
                                               const double* x1,
                                               const double* x2,
                                               const double* x3,
                                               const double* x4,
                                               const double* x5,
                                               const double* x6,
                                               const double* x7,
                                               const double* x8,
                                               const double* x9,
                                               const double* x10,
                                               const double* x11,
                                               const double* x12,
                                               const double* x13,
                                               const double* x14) {
  const double t26 = u[0]*u[0];
  const double t0 = t26*u[0];
  const double t24 = t0*u[1];
  const double t21 = t0*u[0];
  const double t20 = -1.00925423677687*t0 + 0.727289794784244*t21 + 1.45457958956849*t24 + 0.204156603646253*t26 + 0.189139176544278*u[0];
  const double t12 = 0.0653665602547852*t21 + 0.13073312050957*t24;
  const double t2 = u[0]*u[1];
  const double t16 = t2*u[1];
  const double t22 = -1.02456908770049*t0 + 0.731002091719096*t21 + 1.46360418343819*t24 + 0.40937272614469*t26 + 0.0675029905651878*u[0];
  const double t25 = t16*u[0];
  const double t29 = u[1]*u[1];
  const double t30 = 0.783901952922574*t2 + 1.90044877193318*t25;
  const double t23 = t2*u[0];
  const double t31 = 0.117149481155709*t2 - 1.03989444846764*t25;
  const double t15 = 0.166666666666667*t0 - 0.101300106411881*t21 - 0.202600212823763*t24;
  const double t1 = t29*u[1];
  const double t28 = t1*u[1];
  const double t27 = t1*u[0];
  const double t10 = 1.54426437363695*t0 + 1.54426437363695*t1 + 2.16228931204758*t16 - 1.27716982113273*t2 - 0.751683854418516*t21 + 2.16228931204758*t23 - 1.50336770882103*t24 - 1.21589933956426*t25 - 1.35422344
533665*t26 - 1.50336770882103*t27 - 0.751683854418516*t28 - 1.35422344533665*t29 + 0.522157262181323;
  const double t7 = -2.05825730139938*t1 - 2.55751958917118*t16 + t20 - 2.00851644454867*t23 + 2.75378632355459*t27 + 1.37689316177773*t28 + 1.12375786149525*t29 + t30 + 0.303355685979819*u[1] + 0.0682632482712396;
  const double t14 = -0.166666666666667*t1 + 0.202600212823763*t27 + 0.101300106411881*t28;
  const double t11 = 0.166666666666667*t0 - t14 + 0.5*t16 - 0.101300106411881*t21 + 0.5*t25 - 0.202600212823763*t24;
  const double t17 = 1.02456908770049*t1 - 1.46360418343819*t27 - 0.731002091719096*t28 - 0.40937272614469*t29 - 0.0675029905651878*u[1] + 0.0682632482712396;
  const double t6 = 0.0750149862909235*t0 + 1.19205935169555*t16 + t17 - 0.272158810591444*t21 + 0.242505250285989*t23 - 0.544317621182889*t24 + 0.320448914419135*t26 + t31 - 0.273314028968999*u[0];
  const double t19 = -1.00925423677687*t1 + 1.45457958956849*t27 + 0.727289794784244*t28 + 0.204156603646253*t29 + 0.189139176544278*u[1] + 0.0682632482712396;
  const double t13 = 0.13073312050957*t27 + 0.0653665602547852*t28;
  const double t9 = -2.05825730139938*t0 - 2.00851644454867*t16 + t19 + 1.37689316177773*t21 - 2.55751958917118*t23 + 2.75378632355459*t24 + 1.12375786149525*t26 + t30 + 0.303355685979819*u[0];
  const double t8 = -0.491951150728699*t16 + t19 - 0.732663340897398*t2 - t22 + 1.54187217374866*t23 - 0.105812403346733*t25;
  const double t18 = -0.0750149862909235*t1 + 0.544317621182889*t27 + 0.272158810591444*t28 - 0.320448914419135*t29 + 0.273314028968999*u[1] - 0.0682632482712396;
  const double t3 = 0.242505250285989*t16 - t18 - t22 + 1.19205935169555*t23 + t31;
  const double t4 = 0.0750149862909235*t0 - 0.580730903329257*t16 - t18 + 0.900393634770956*t2 - 0.272158810591444*t21 - 0.580730903329257*t23 - 0.544317621182889*t24 - 0.413584500673319*t25 + 0.320448914419135*t26
 - 0.273314028968999*u[0];
  const double t5 = 1.54187217374868*t16 + t17 - 0.732663340897398*t2 + t20 - 0.491951150728699*t23 - 0.105812403346733*t25;
  m[0] = t10*x0[0] + t11*x11[0] + t12*x12[0] + t12*x13[0] + t13*x10[0] + t13*x9[0] - t14*x8[0] + t15*x14[0] + t3*x4[0] + t4*x5[0] + t5*x3[0] + t6*x6[0] + t7*x1[0] + t8*x7[0] + t9*x2[0];
  m[1] = t10*x0[1] + t11*x11[1] + t12*x12[1] + t12*x13[1] + t13*x10[1] + t13*x9[1] - t14*x8[1] + t15*x14[1] + t3*x4[1] + t4*x5[1] + t5*x3[1] + t6*x6[1] + t7*x1[1] + t8*x7[1] + t9*x2[1];
  m[2] = t10*x0[2] + t11*x11[2] + t12*x12[2] + t12*x13[2] + t13*x10[2] + t13*x9[2] - t14*x8[2] + t15*x14[2] + t3*x4[2] + t4*x5[2] + t5*x3[2] + t6*x6[2] + t7*x1[2] + t8*x7[2] + t9*x2[2];
}
```

SUBDIV PECULIARITIES 2: VANISHING DERIVATIVES

“Neighbour extrapolation” for vestigial patch looks OK visually, but EVs have other issues:



- Vanishing first derivatives: $\lim_{\boldsymbol{u} \to EV} M_{\boldsymbol{u}}(\boldsymbol{u}, X) = \mathbf{0}$
  - Saddle point for gradient-based optimization.
- Unbounded second derivatives
  - Infinite thin-plate energy (inconvenience).
  - Derivatives with respect to normal, although well defined, are unstable using chain-rule (inconvenience).
- Solutions
  - **Reparameterise** the function near the extraordinary vertex.
  - **Replace** the function near the extraordinary vertex.

Example bad parameterization:

$$\boldsymbol{m}(s) = (x, y) = (\sqrt{s}, \sin(\sqrt{s})) \qquad s \in \mathbb{R}^+$$

$$\mathbf{m}'(\mathrm{s}) = \frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}s}(s) = \left(\frac{1}{2\sqrt{s}}, \frac{\cos(\sqrt{s})}{2\sqrt{s}}\right)$$

$$\Rightarrow \lim_{s \to 0} \boldsymbol{m}'(s) \to (\infty, \infty)$$

Reparameterise $s = t^2$

$$\boldsymbol{m}(t) = (x, y) = (t, \sin(t))$$

$$\boldsymbol{m}'(t) = \frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}t}(t) = (1, \cos(t))$$

$$\Rightarrow \lim_{t \to 0} \boldsymbol{m}_t(t) \to (1,1)$$

REPARAMETERISING TO FIX DERIVATIVES

- Using subdivs is easy
  - The messy stuff is encapsulated in Eval_M*(), and Plus()
  - Google's "Ceres" solver does all the Levenberg-Marquardt
- Continuous optimization often doesn't need a very good initial estimate
- Using subdivs allows correspondences $\boldsymbol{u}_i$ to update *during* the optimization
  - If ICP takes a long time, this may not…
  - But you **must** exploit sparsity
- Future work:
  - Dogs, hinted ARAP, skeleton, even more speed, …

CONCLUSIONS ETC

Microsoft

- Seen a few students nastily bitten by collapsing meshes

- So what's changed? How do I get bitten by the bug, not the hornet?
  1. Sum over data, not model
  2. Use modern (2006) regularizers
  3. Vary everything
  4. Define clean interpolants

- CLOSED FORM" SOLUTIONS OFTEN SOLVE A NEARBY CONVEX PROBLEM.
- SO DOES ANY 2ND ORDER OPTIMIZER.
- IF YOU HAVE FOUND A QUADRATIC SUBPROBLEM, SO WILL LEVENBERG-MARQ.
- YOU CAN DIFFERENTIATE THROUGH PRETTY MUCH ANYTHING.

- SCALING IS IMPORTANT. MEASURE IN NATURAL UNITS.

- Finite diffs fine, just expensive
- Myths: you don't need to find the optimum
- Parameter tuning
- Constrained optimization

CONCLUSIONS

Microsoft