Dense Methods 2: Depth, Flow

CSE P576

Vitaly Ablavsky

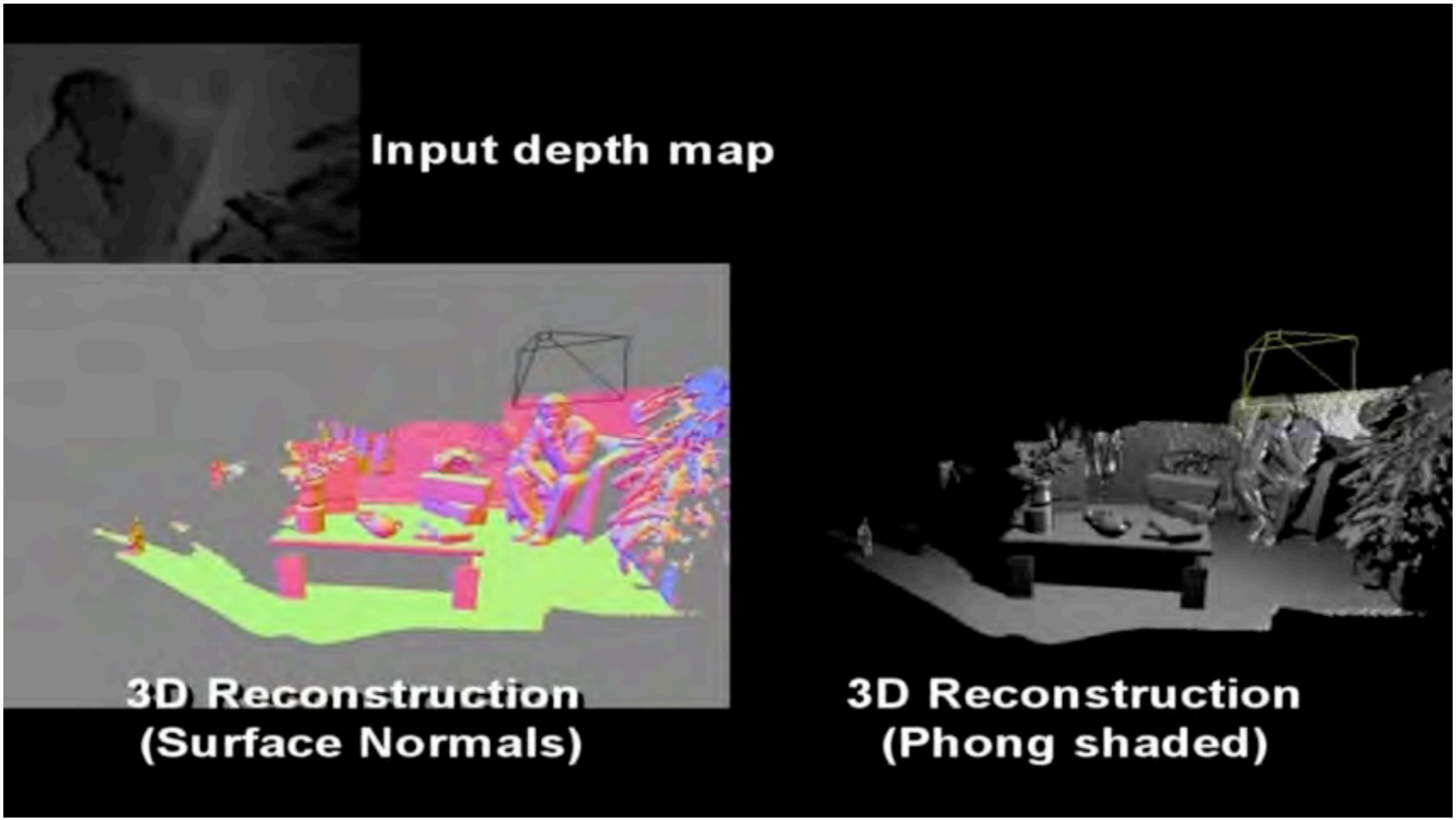
These slides were developed by Dr. Matthew Brown for CSEP576 Spring 2020 and adapted (slightly) for Fall 2021
credit → Matt
blame → Vitaly

Dense Methods 2: Depth, Flow

- Depth Imaging + Fusion, Signed Distance Functions
- Non-Rigid matching, Optical Flow, Lucas Kanade

Depth Image Fusion

- How can we combine multiple depth scans?



[KinectFusion Izadi et al]

Problem: How to Combine Depth Images into a Complete Model?

(a) Measurement (b) 2 Frames (c) 30 Frames (d) 100 Frames (e) Complete model

[Extracted from KinectFusion. Newcombe et al, 2011]

[Slides from Richard Newcombe and Steven Lovegrove]

Merging depth maps

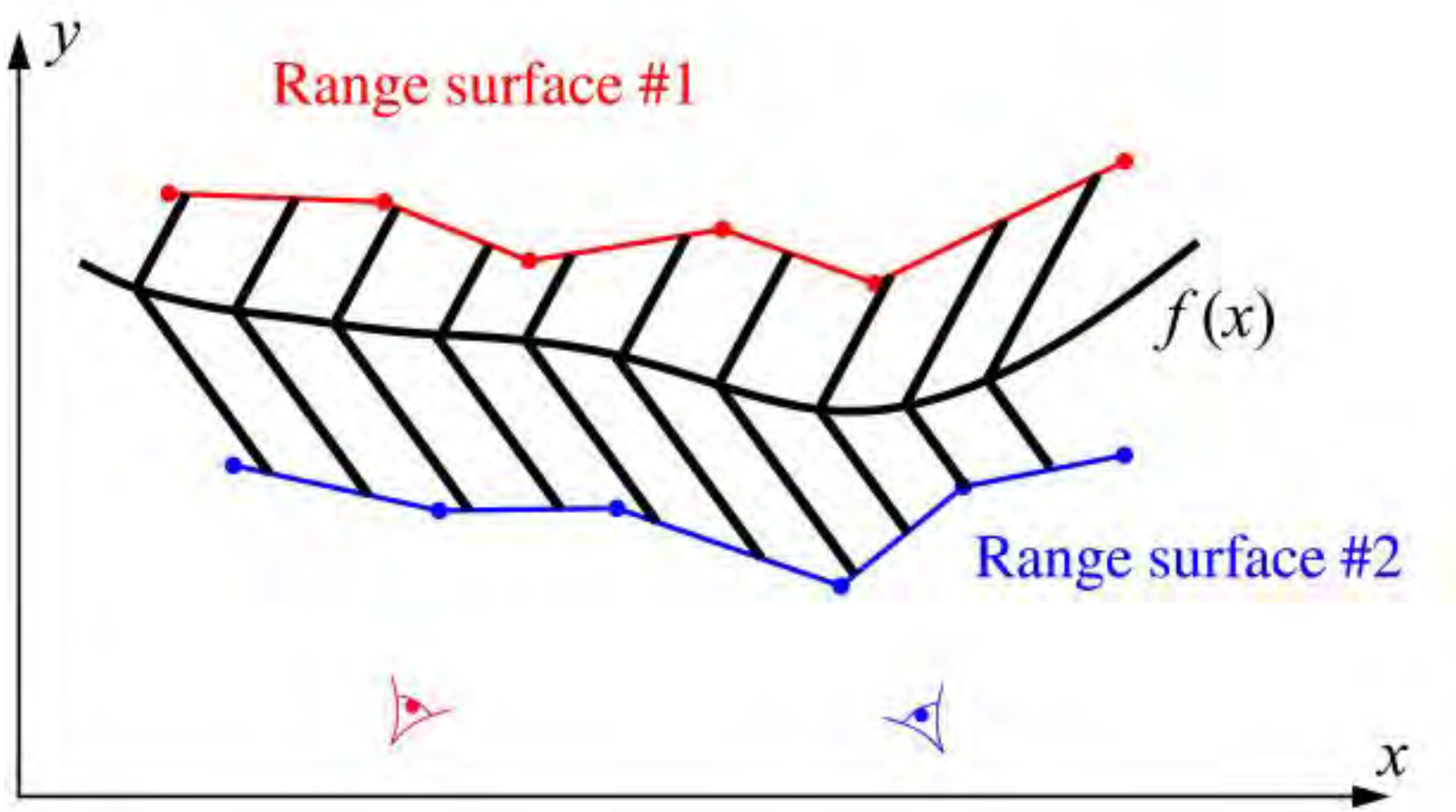
Reconstructed Surfaces

Depth map 1 Depth map 2 Combination (Union)

- Naïve combination (union) produces artifacts
- Better solution: find "average" surface
  - ➔ Surface that minimizes sum (of squared) distances to the depth maps

[From Curless & Levoy, 1996]

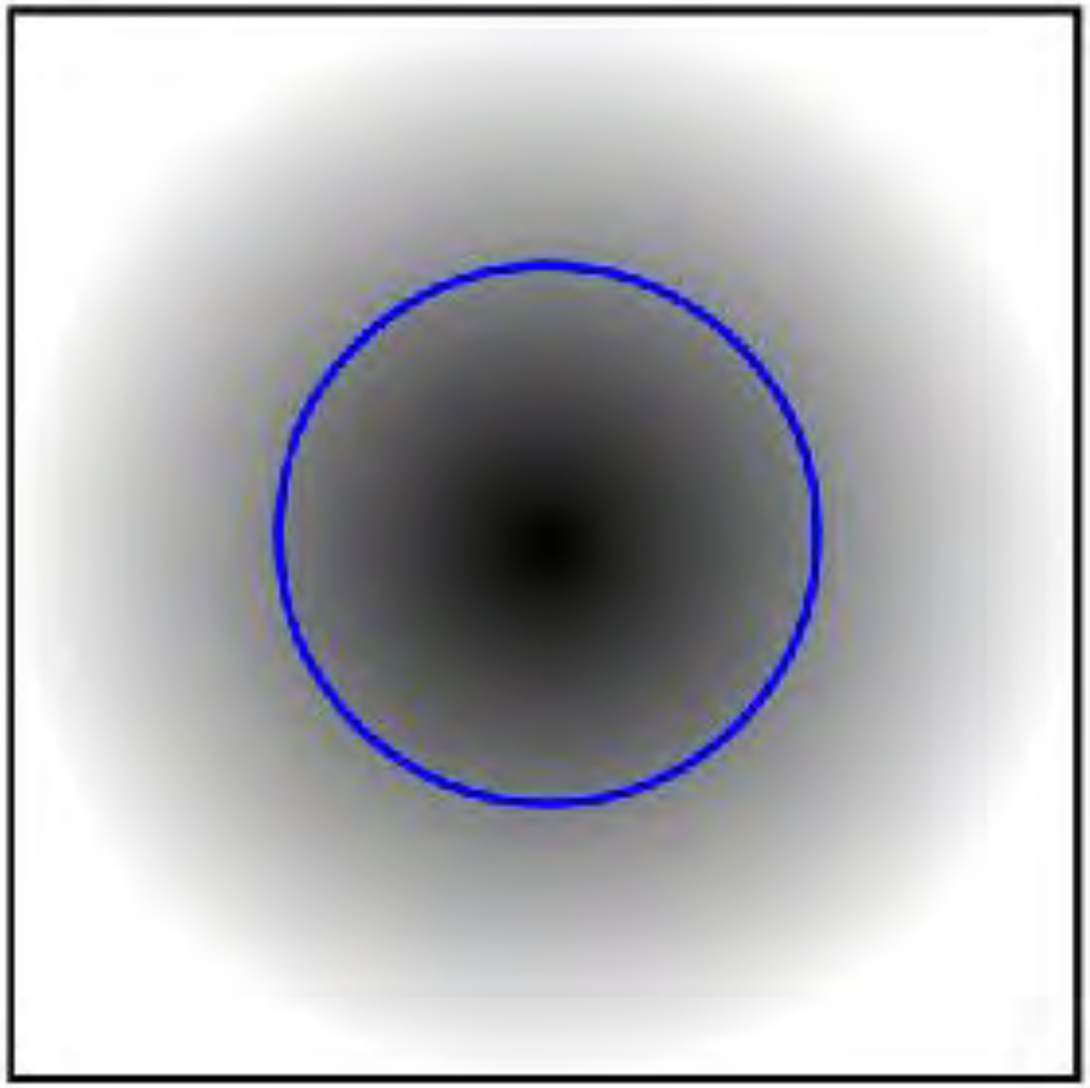
Least squares surface solution



$$E(f) = \sum_{i=1}^{N} \int d_i^2(x, f)dx$$

[Slide from Seitz, UW CSEP576]

Representing Geometry Implicitly

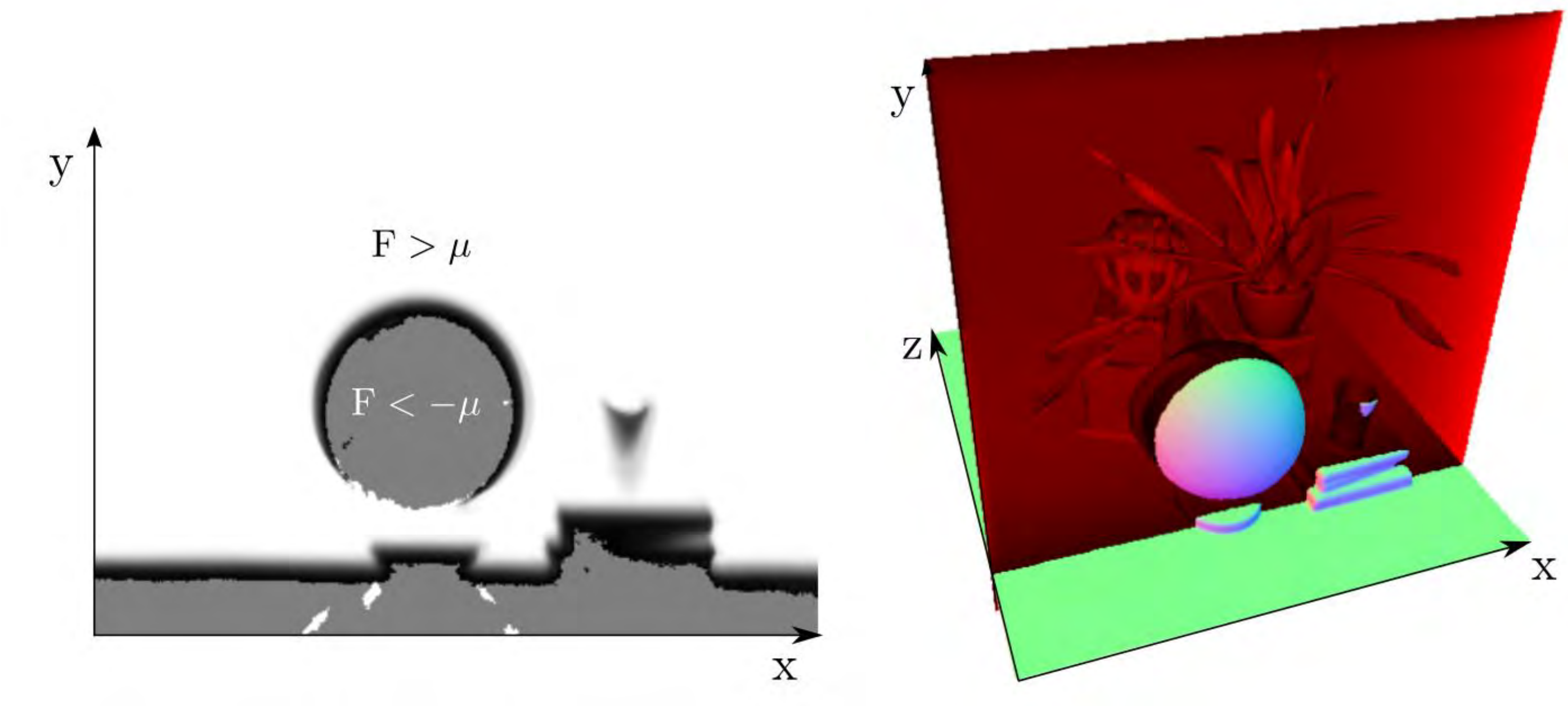


Signed Distance Functions

Example: Truncated Signed Distance Function (TSDF)

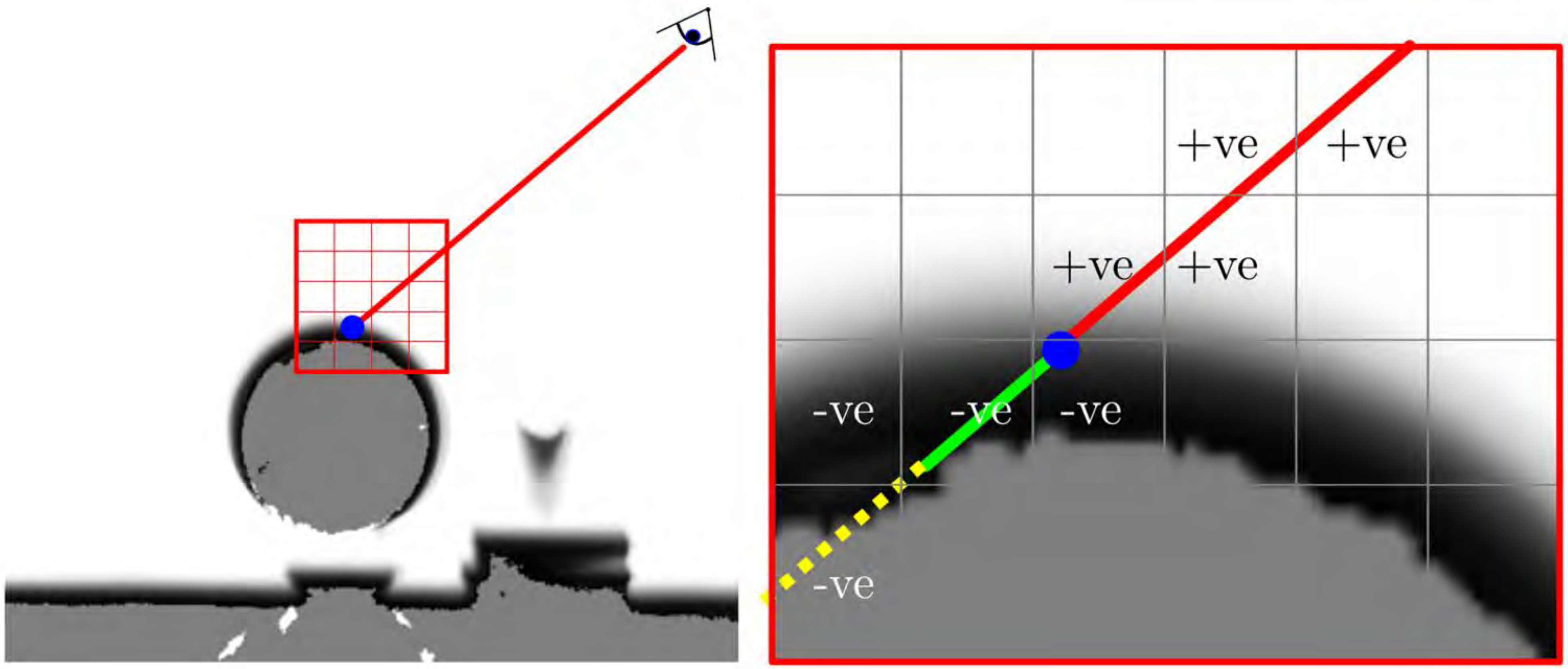
[Newcombe, 2015]

Representing Scenes with TSDF

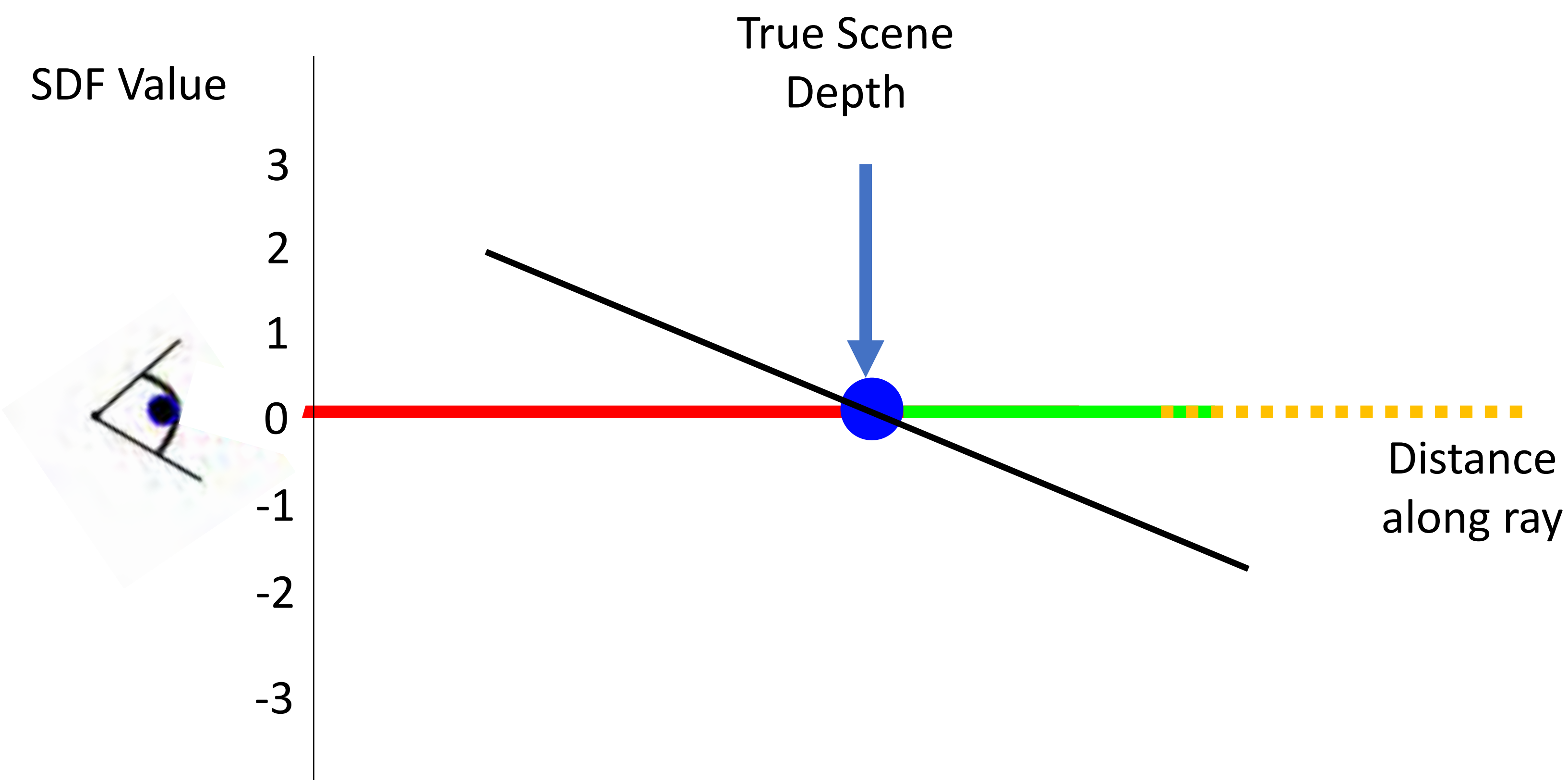


[KinectFusion, Newcombe et al, 2011]

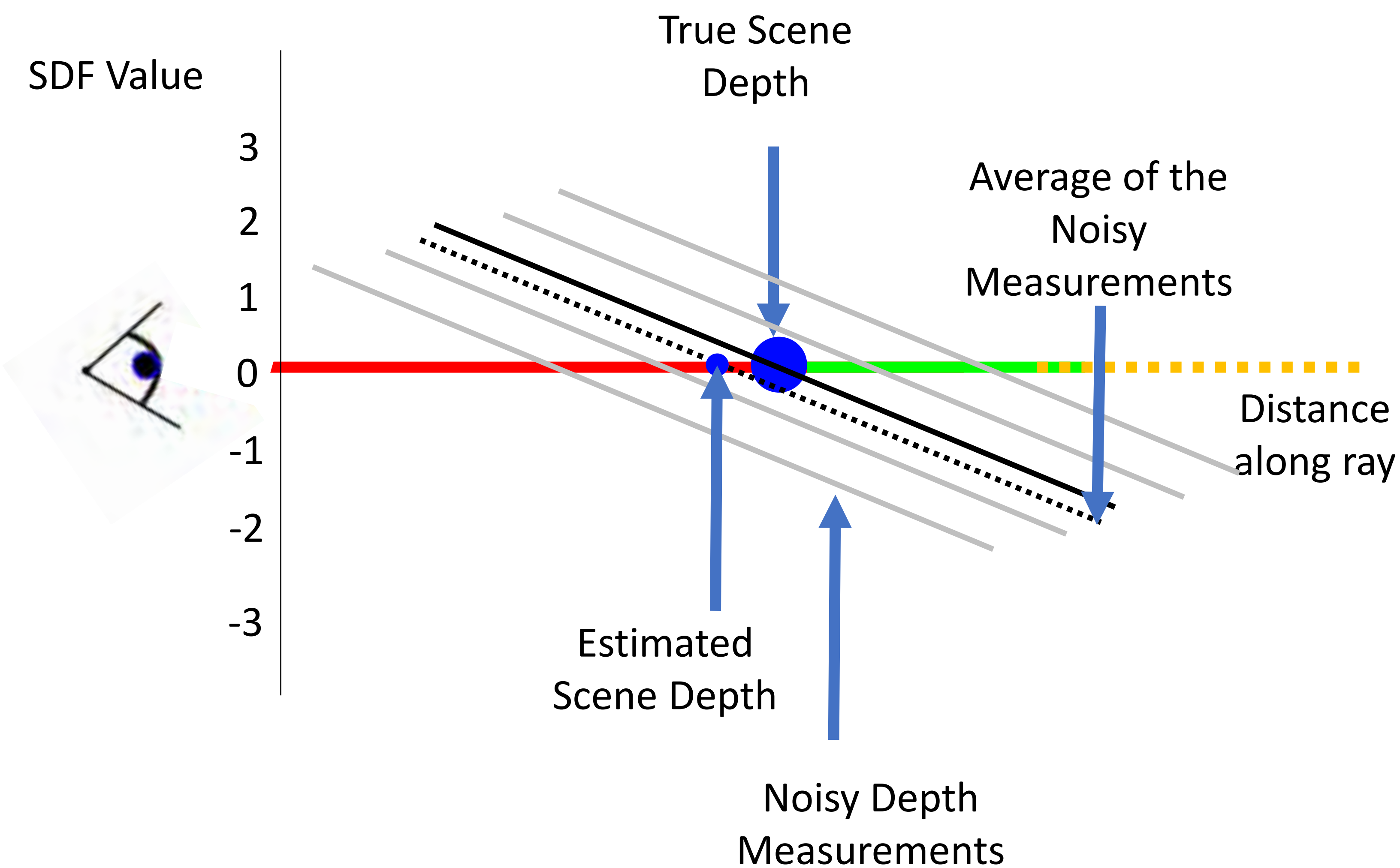
A Single Ray Observation in TSDF



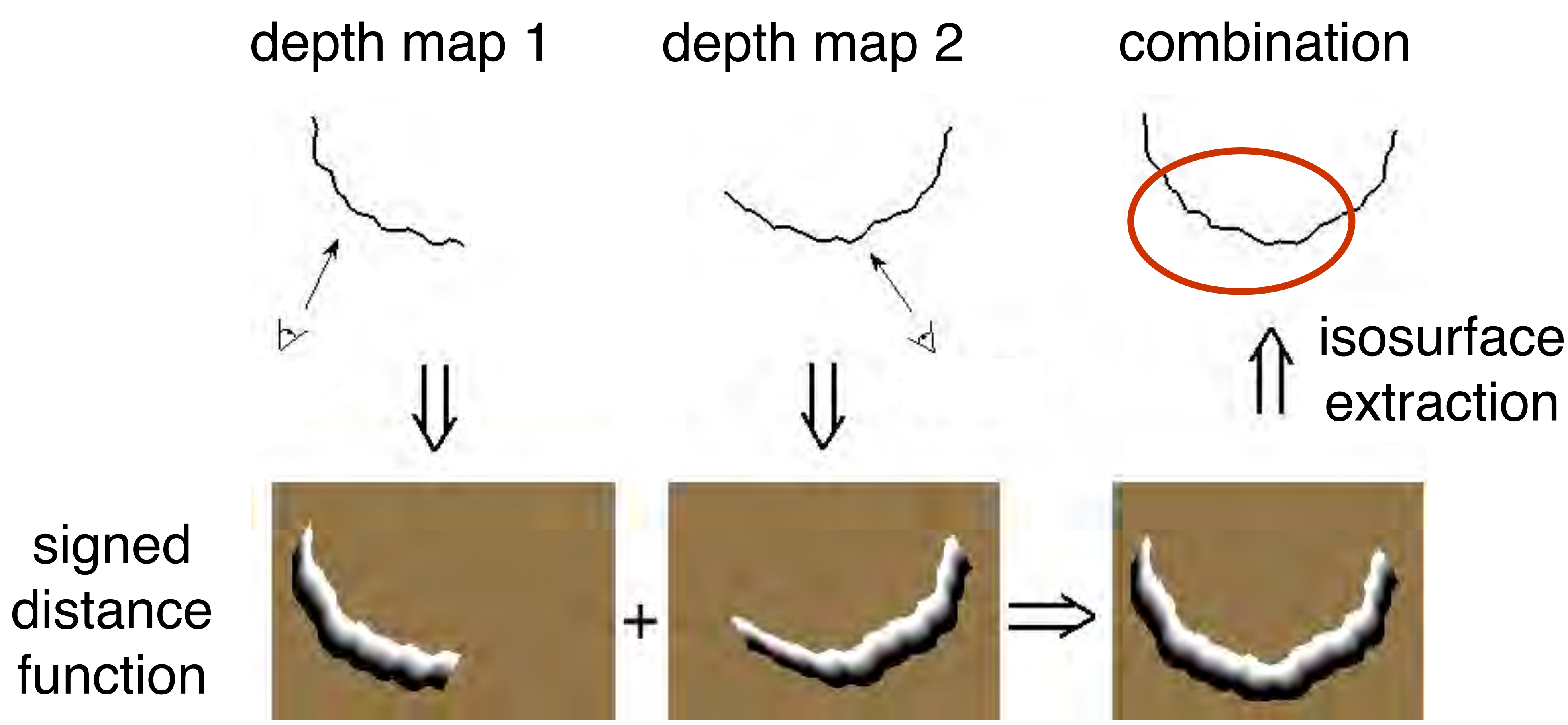
Ray Observations in TSDF



Fusing Noisy Ray Observations in TSDF



VRIP [Curless & Levoy 1996]



Merging Depth Maps: Temple Model

input image

317 images
(hemisphere)

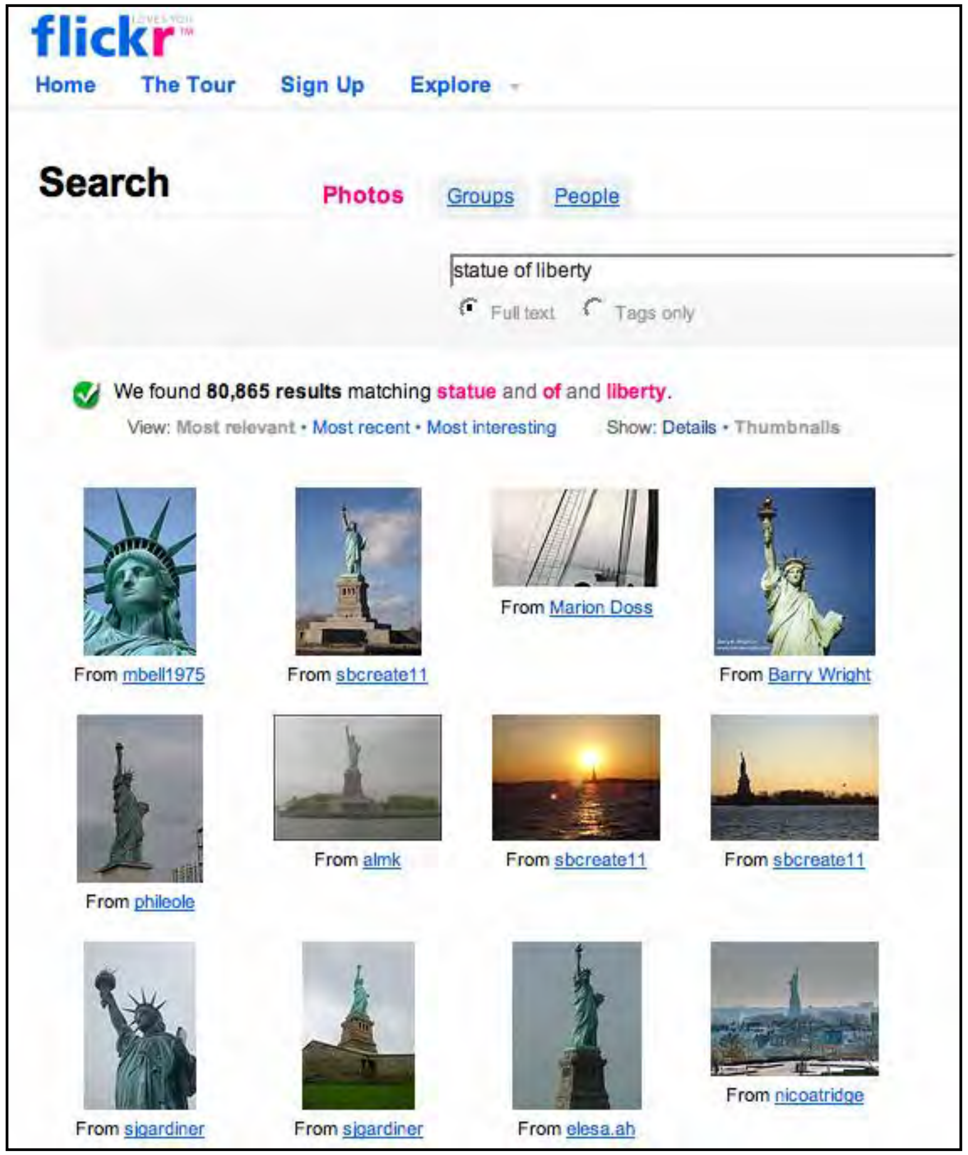
ground truth model

Goesele, Curless, Seitz, 2006

Michael Goesele

Application: Multi-view stereo from Internet Collections

[Goesele, Snavely, Curless, Hoppe, Seitz, ICCV 2007]



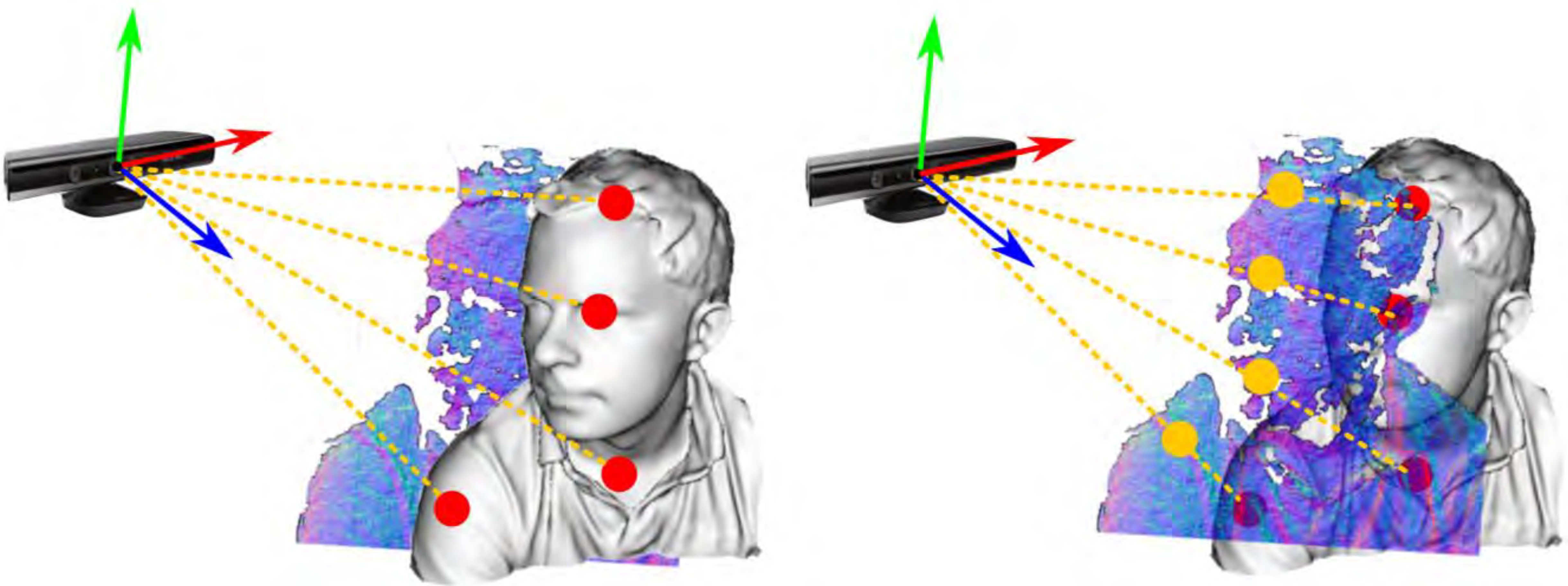
KinectFusion: Dense Surface Tracking and Mapping in Real-Time

- Uses an RGB-D Sensor
- First Dense SLAM System
- Interleaves:
  1. TSDF Fusion (Map)
  2. Projective ICP (Track)
- Efficient to implement on GPU Compute Architecture
- Memory for Scene is O(N^3)

Newcombe, Izadi et al

Iterated Closest Point

- Estimate camera pose from unmatched point clouds



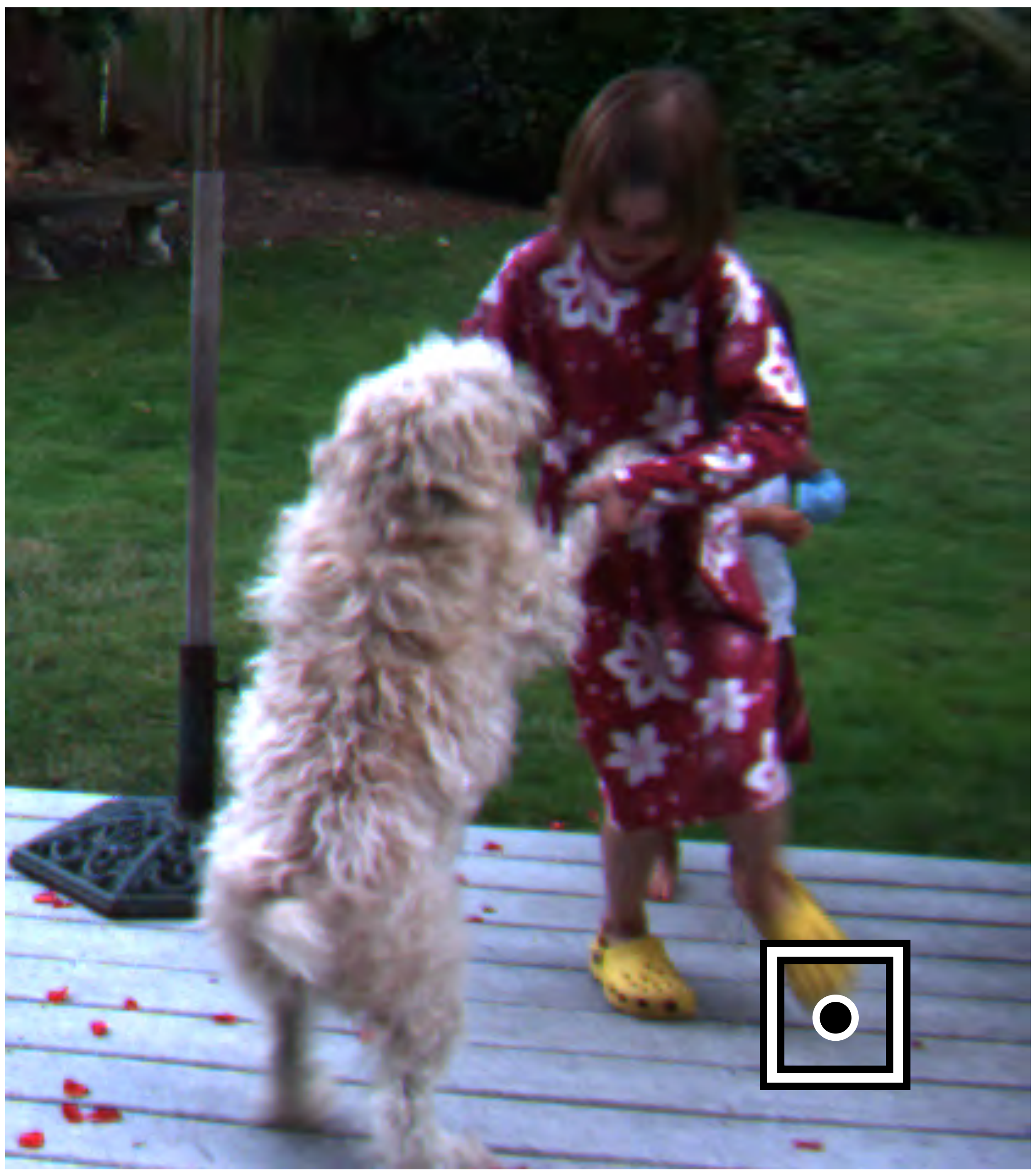
- Assign points in the scan yellow to closest model point red
- Compute pose (R,t) of the scanner using correspondences
- Re-assign closest points and iterate until converged

2-view Rigid Matching

- **1D search**, points constrained to lie along epipolar lines

2-view Non-Rigid Matching

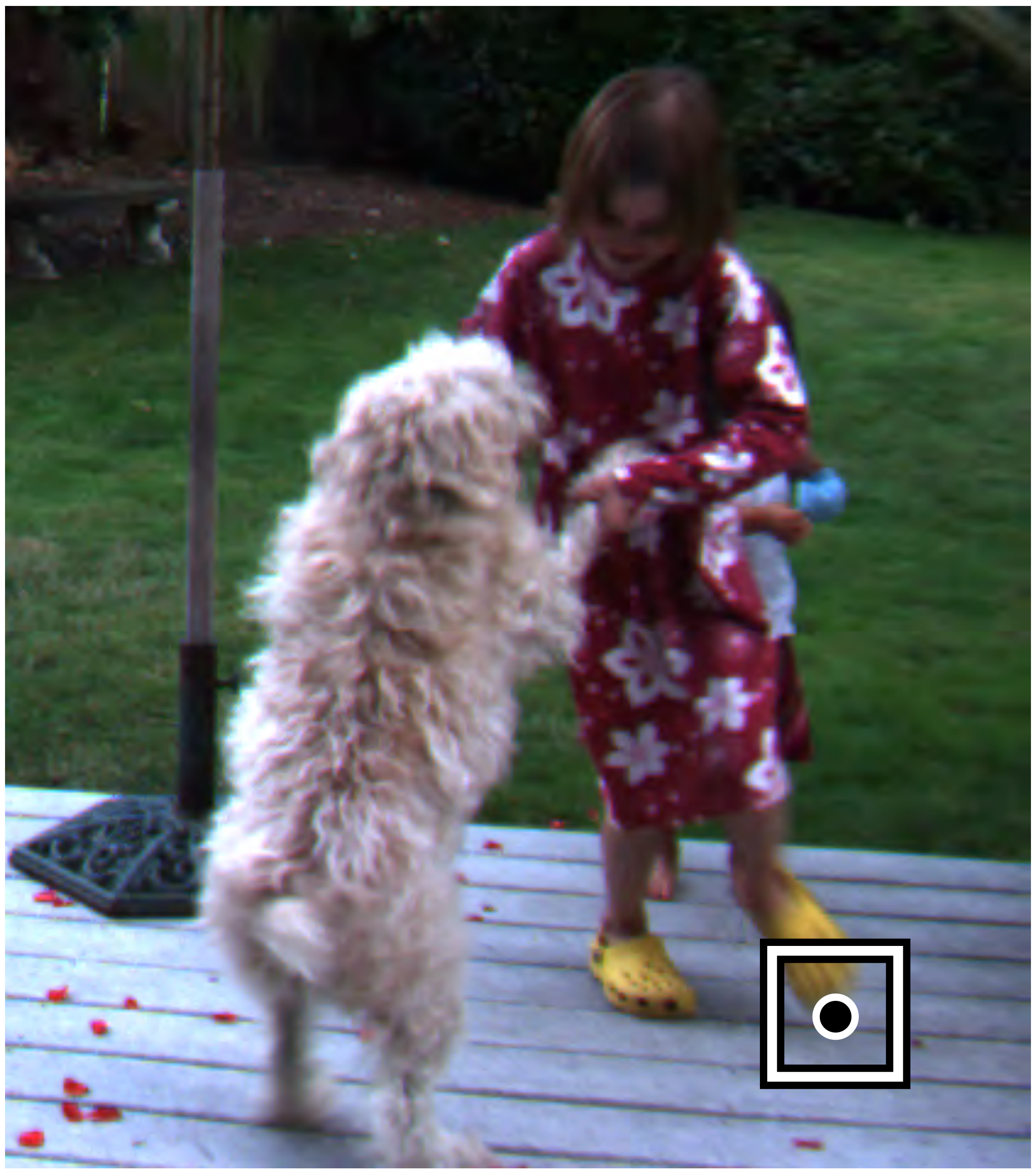
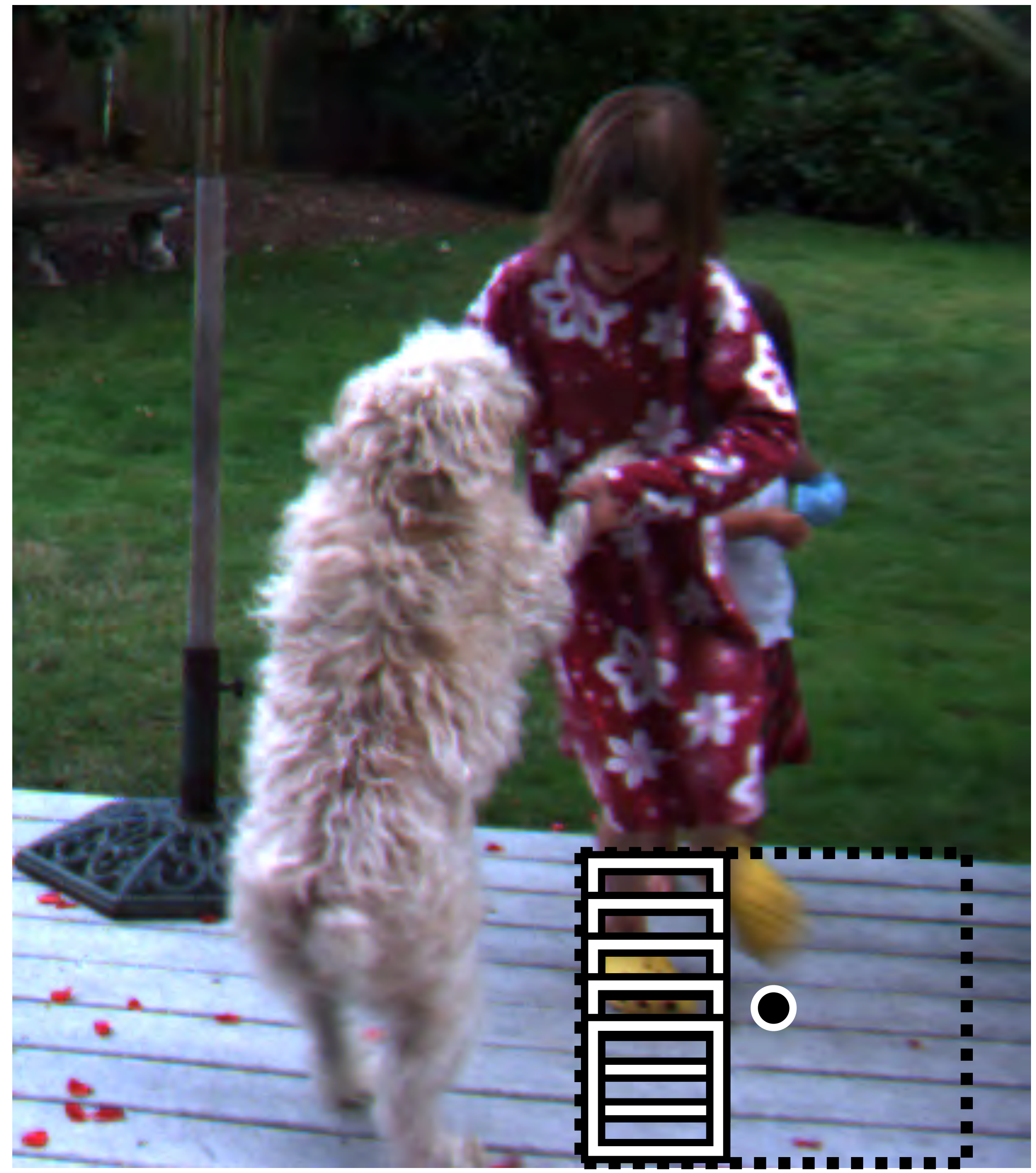
- **2D search**, points can move anywhere in the image




[vision.middlebury.edu/flow]

2-view Non-Rigid Matching

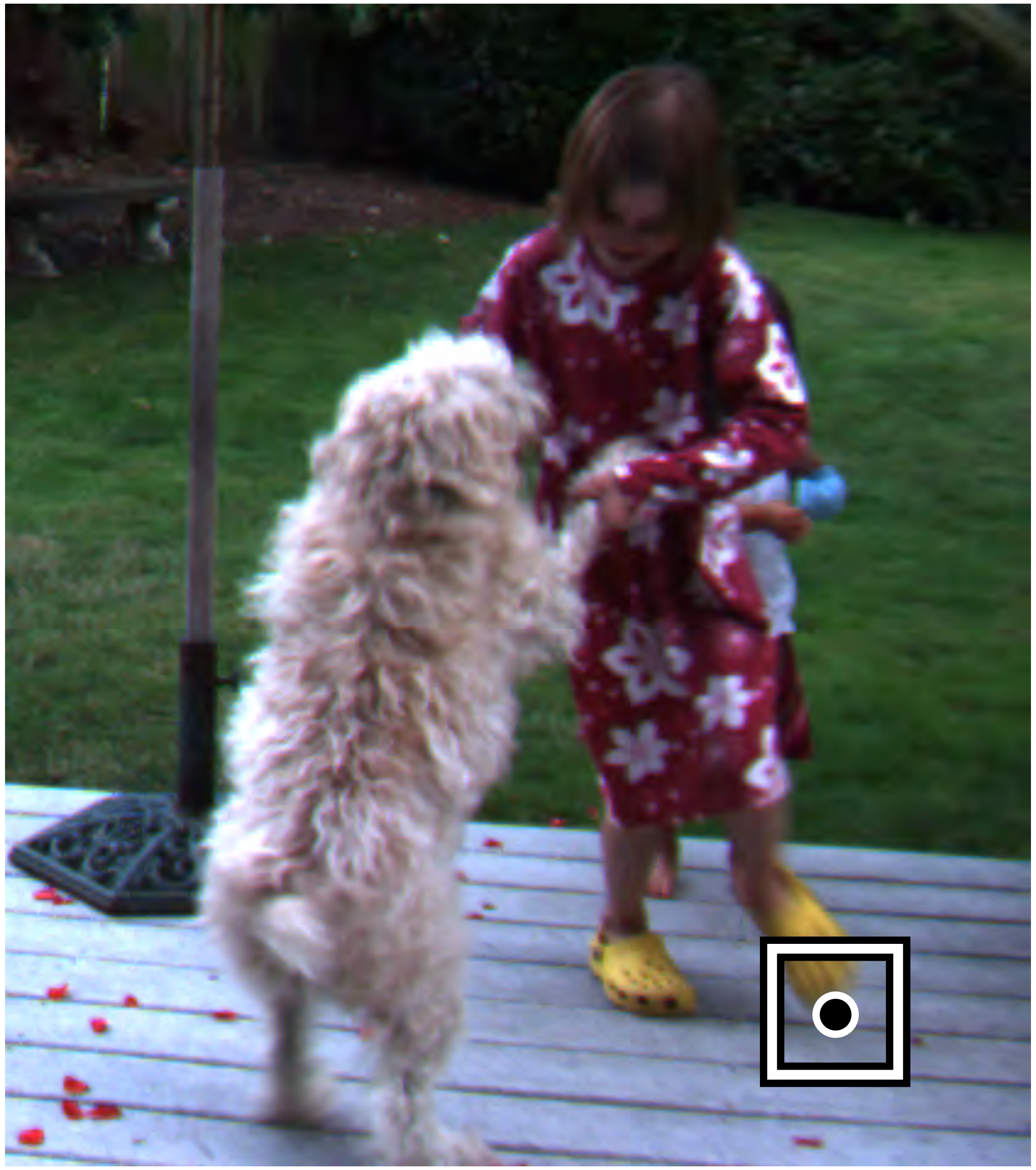
- **2D search**, points can move anywhere in the image

[vision.middlebury.edu/flow]

2-view Non-Rigid Matching

- **2D search**, points can move anywhere in the image



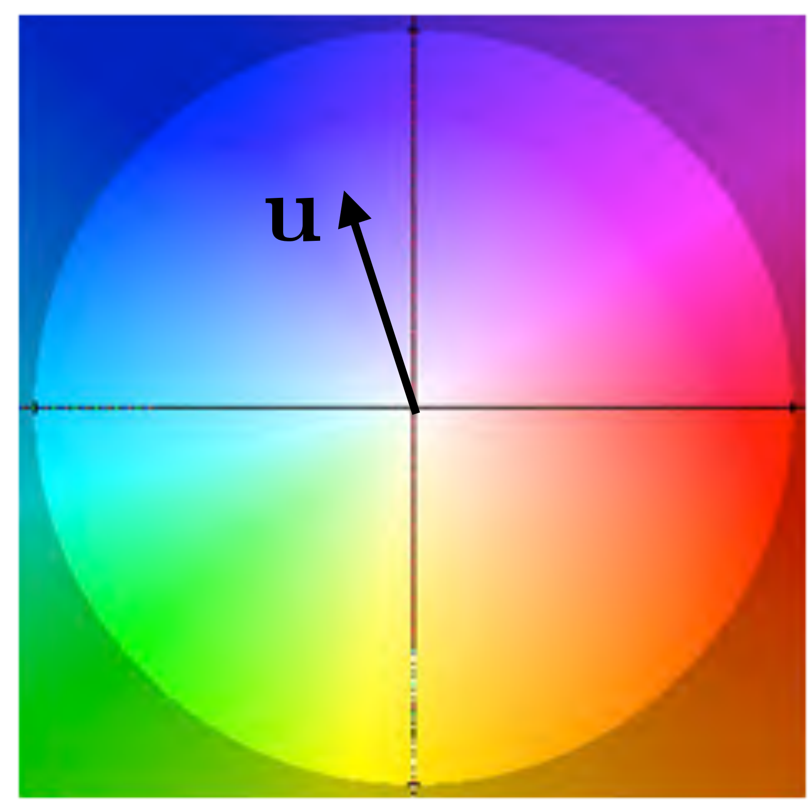
[vision.middlebury.edu/flow]

2-view Non-Rigid Matching

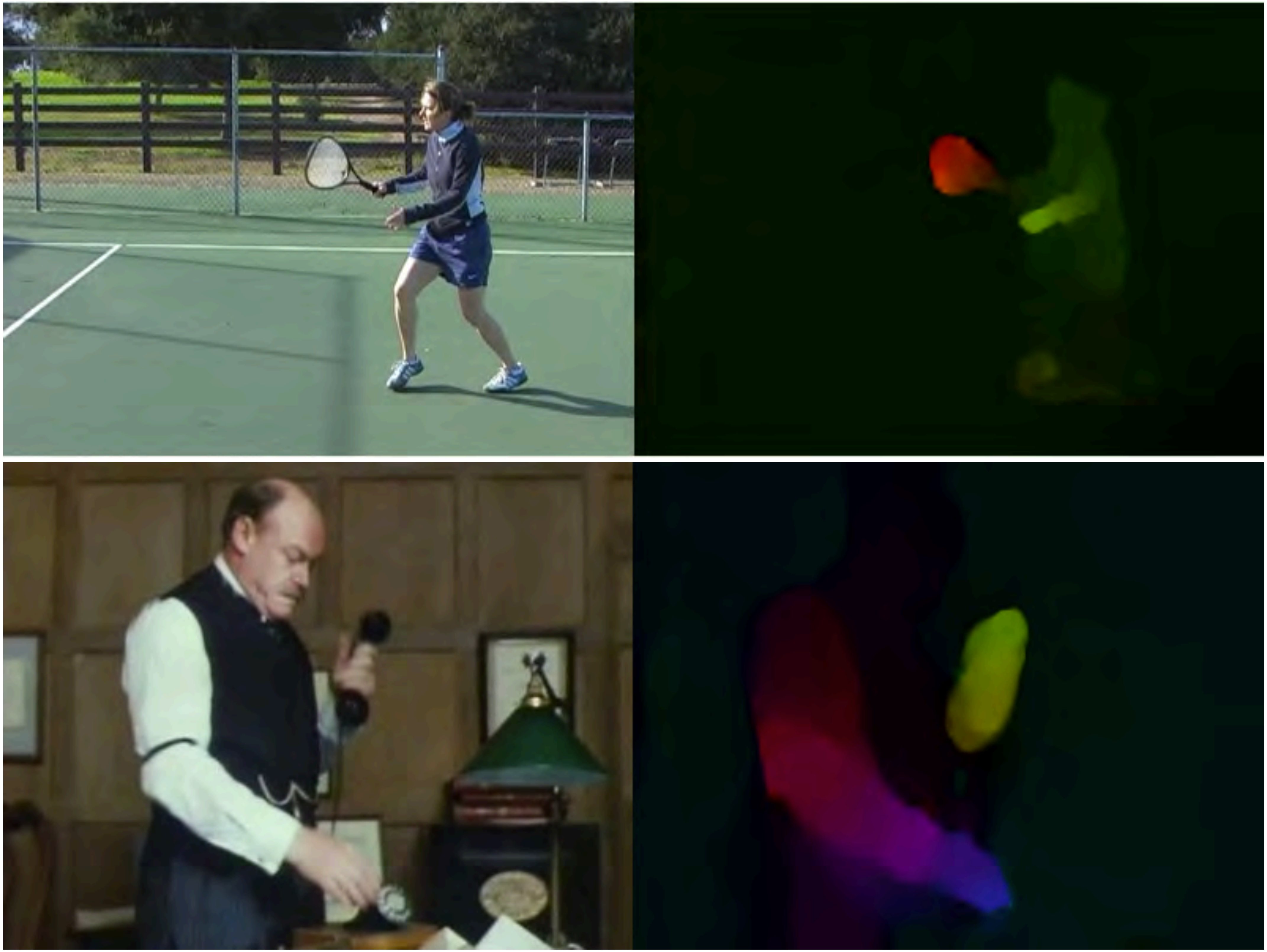
- **2D search**, points can move anywhere in the image

[vision.middlebury.edu/flow]

Optical Flow: Example 1



Optical Flow: Example 2



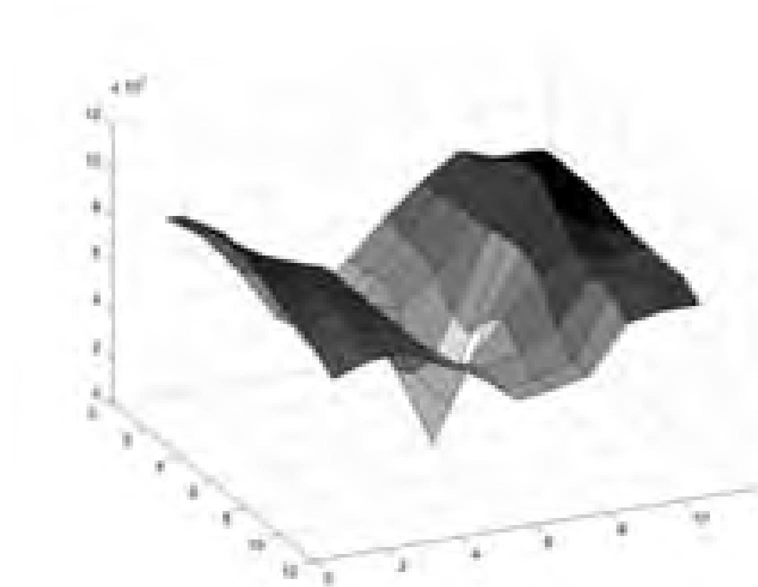
[Brox Malik 2011]

Lucas Kanade

- The previous algorithm performed a discrete search over displacements/flow vectors **u**
- We can do better by looking at the structure of the error surface:

$I_0(\mathbf{x})$

$I_1(\mathbf{x})$



$$e = |\mathbf{I}_1(\mathbf{x}+\mathbf{u}) - \mathbf{I}_0(\mathbf{x})|^2$$

Lucas Kanade

- This is the Lucas-Kanade algorithm for 2D image flow

Try out `LucasKanade.ipynb` from the course webpage

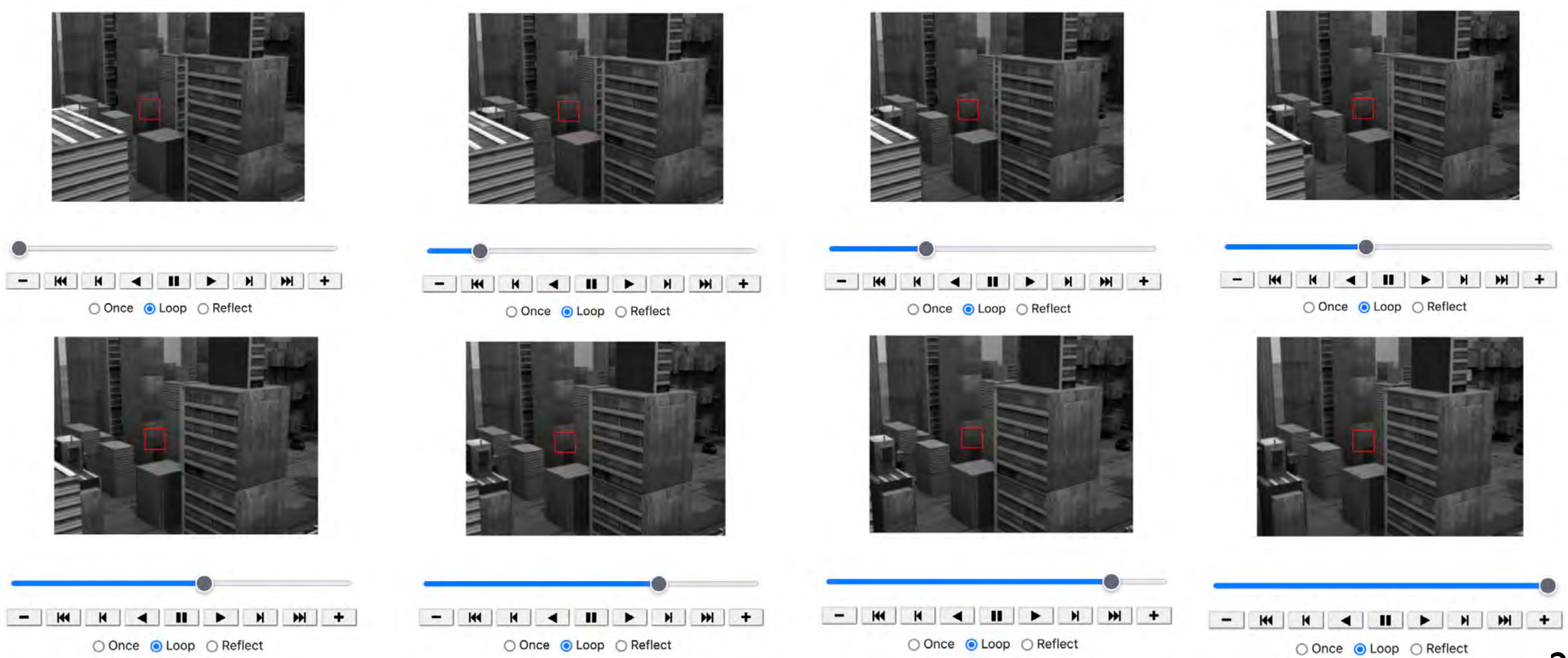
Lucas-Kanade Jupyter Notebook

Putting it together: Track the Sequence

[11]:
```python
# Run patch tracking on whole sequence

# Starting location
p = offset0
track = [p]

# Run Lukas Kanade tracking on each frame
for i in range(1, len(images)):
    # Coarse to fine (for efficiency, you would normally downsample in a pyramid. Here we just blur)
    p,_,_,_ = LucasKanade2d(images[0], images[i], offset0, guess_p1 = p, its=20, blur_sigma=2.0)
    p,_,_,_ = LucasKanade2d(images[0], images[i], offset0, guess_p1 = p, its=10, blur_sigma=0.0)
    track.append(tuple(p))
```



Flow at a pixel

- Look at previous equation at a single pixel:

$$\frac{\partial I_1}{\partial \mathbf{x}}^T \Delta \mathbf{u} = I_0(\mathbf{x}) - I_1(\mathbf{x})$$

Flow Ambiguity

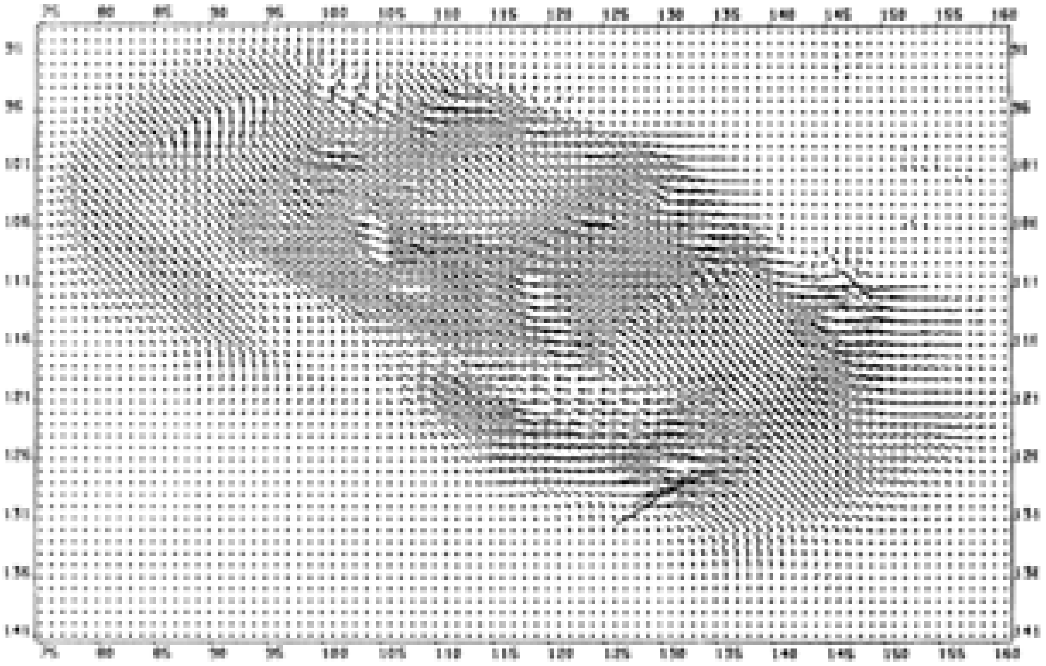
- Optical Flow Constraint:

$$\frac{\partial I}{\partial t} + \nabla I^T \mathbf{v} = 0$$

- The stripes can be interpreted as moving vertically, horizontally (rotation), or somewhere in between!
- The component of velocity parallel to the edge is unknown

Horn-Schunk

- The optical flow constraint gives 1 equation per pixel to solve for the velocity field (2 parameters per pixel)



We can use other considerations, such as smoothness, to find a plausible velocity field, e.g.,

$$e_{HS} = \sum \left(\frac{\partial I}{\partial t} + \nabla I^T \mathbf{v} \right)^2 + \alpha |\Delta \mathbf{v}|^2$$

[Horn Schunck 1981, Szeliski Ch. 9.1.3]

Brightness Constancy

- All the methods presented in this lecture have relied on the assumption that

$$I_1(\mathbf{x} + \mathbf{u}) \approx I_0(\mathbf{x})$$

- This is called the **brightness constancy** assumption

- Taylor expansion for small motion at a single pixel = optical flow constraint
- Horn-Schunk = optical flow constraint + smoothing over **u**
- Lucas-Kanade = brightness constancy over patches with gradient based search for **u**

Next Lecture

- Visual Recognition, Linear Classification