CS 532: 3D Computer Vision
4th Set of Notes

Lecture Outline

- Binocular Stereo
  - Matching criteria
- Confidence for stereo
- Stereo beyond the Winner-Take-All algorithm

Based on slides by R. Szeliski, P. Fua, S. Seitz, M. Bleyer and R. Zabih

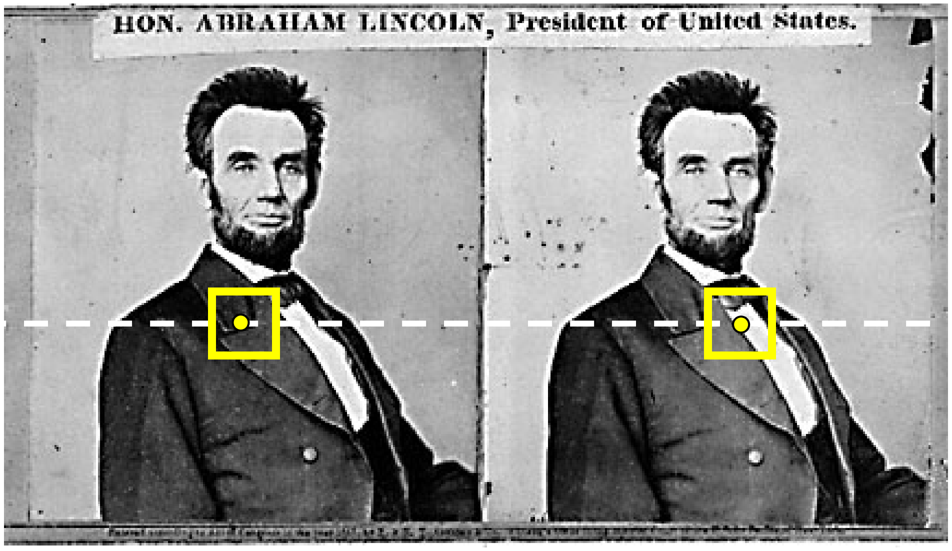
Stereo Matching

Slides by Rick Szeliski, Pascal Fua and P. Mordohai

Stereo Matching

- What are some possible algorithms?
  - match “features” and interpolate
  - match edges and interpolate
  - match all pixels with windows (coarse-fine)
  - use optimization:
    - iterative updating
    - dynamic programming
    - energy minimization (regularization, stochastic)
    - graph algorithms

Basic Stereo Algorithm



For each epipolar line

For each pixel in the left image

- compare with every pixel on same epipolar line in right image
- pick pixel with minimum match cost

Improvement: match ***windows***

Disparity

- Disparity d is the difference between the x coordinates of corresponding pixels in the left and right image

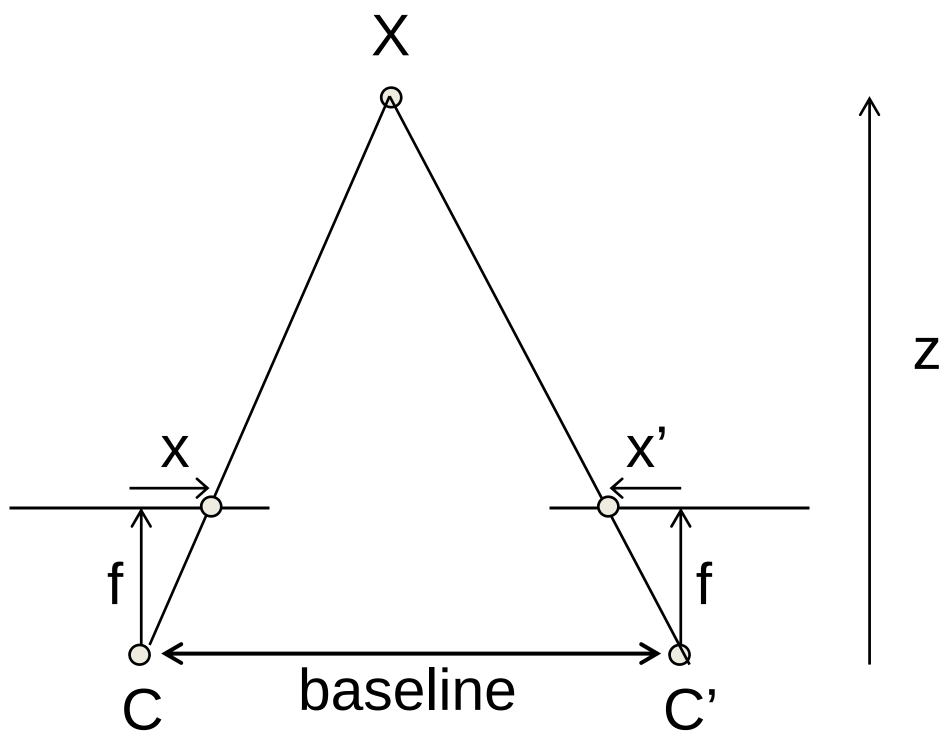
$$d = x_L - x_R$$

- Disparity is inversely proportional to depth

$$Z = \frac{bf}{d}$$

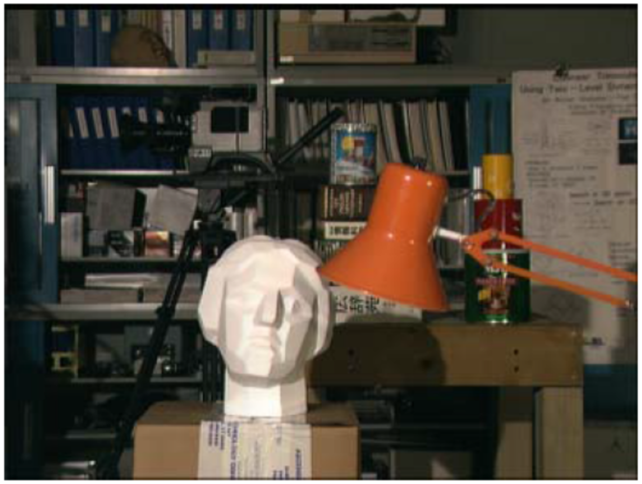
Stereo Reconstruction

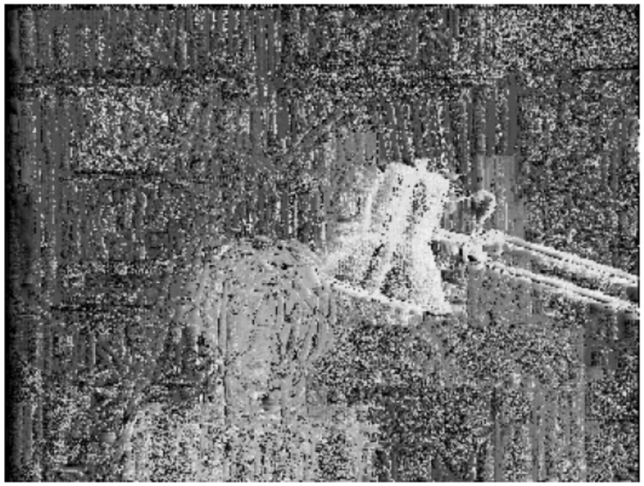
$$Z = \frac{bf}{d}$$



Naïve Stereo Algorithm

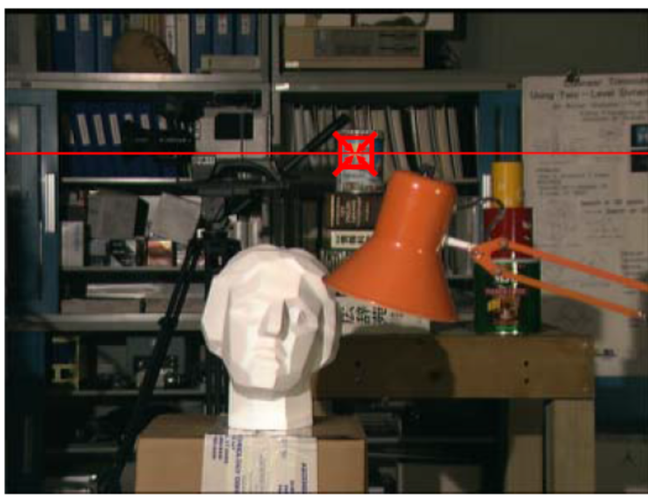
- For each pixel p of the left image:
  - Compare color of p against the color of each pixel on the same horizontal scanline in the right image
  - Select the pixel of most similar color as matching point

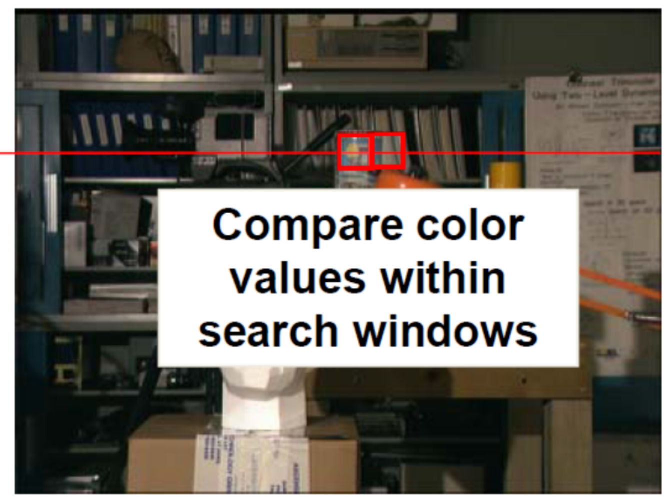




Window-Based Matching

- Instead of matching single pixels, center a small window on a pixel and match the whole window in the right image


(a) Left image



(b) Right image

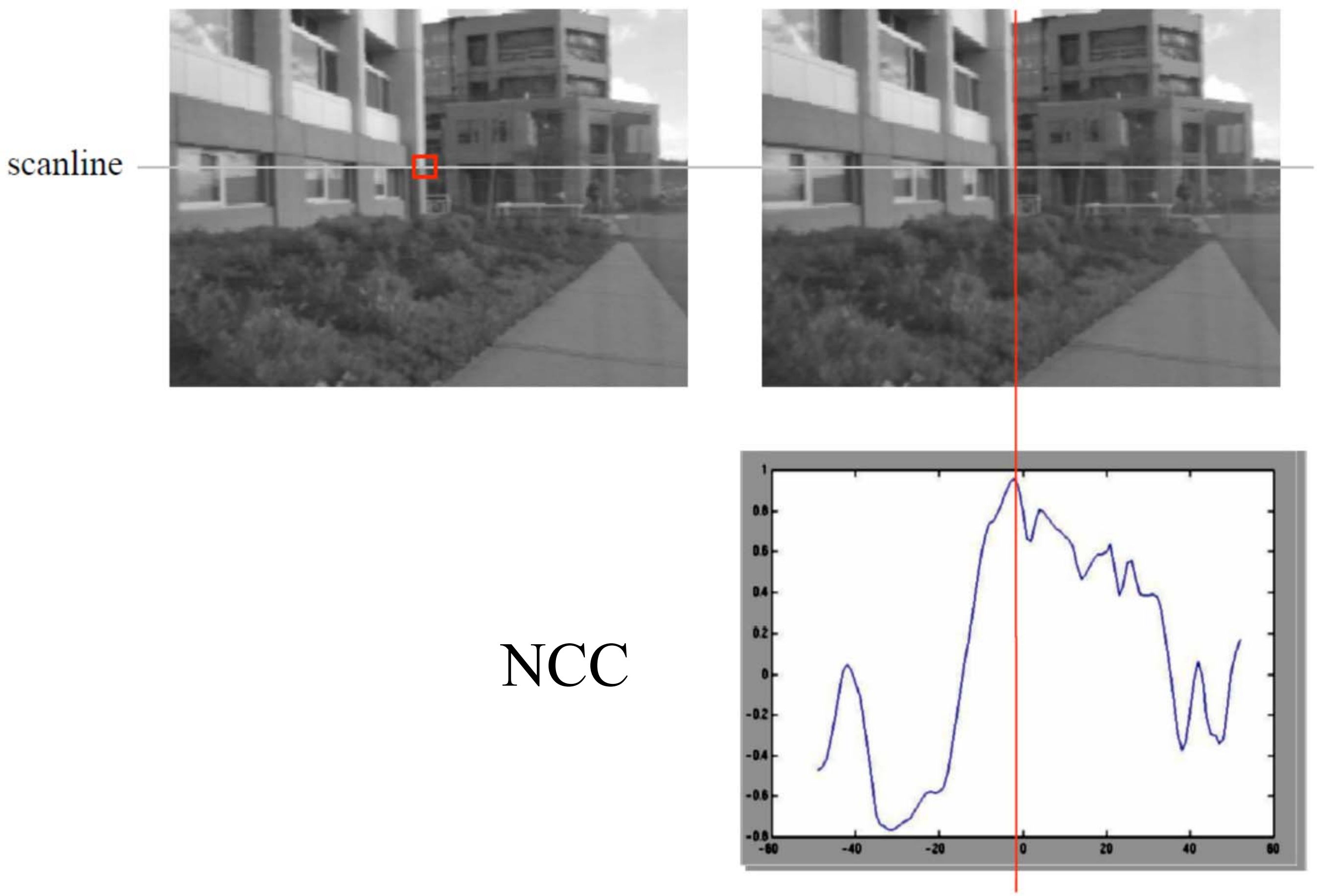
Window-Based Matching

- the disparity d_p of a pixel p in the left image is computed as

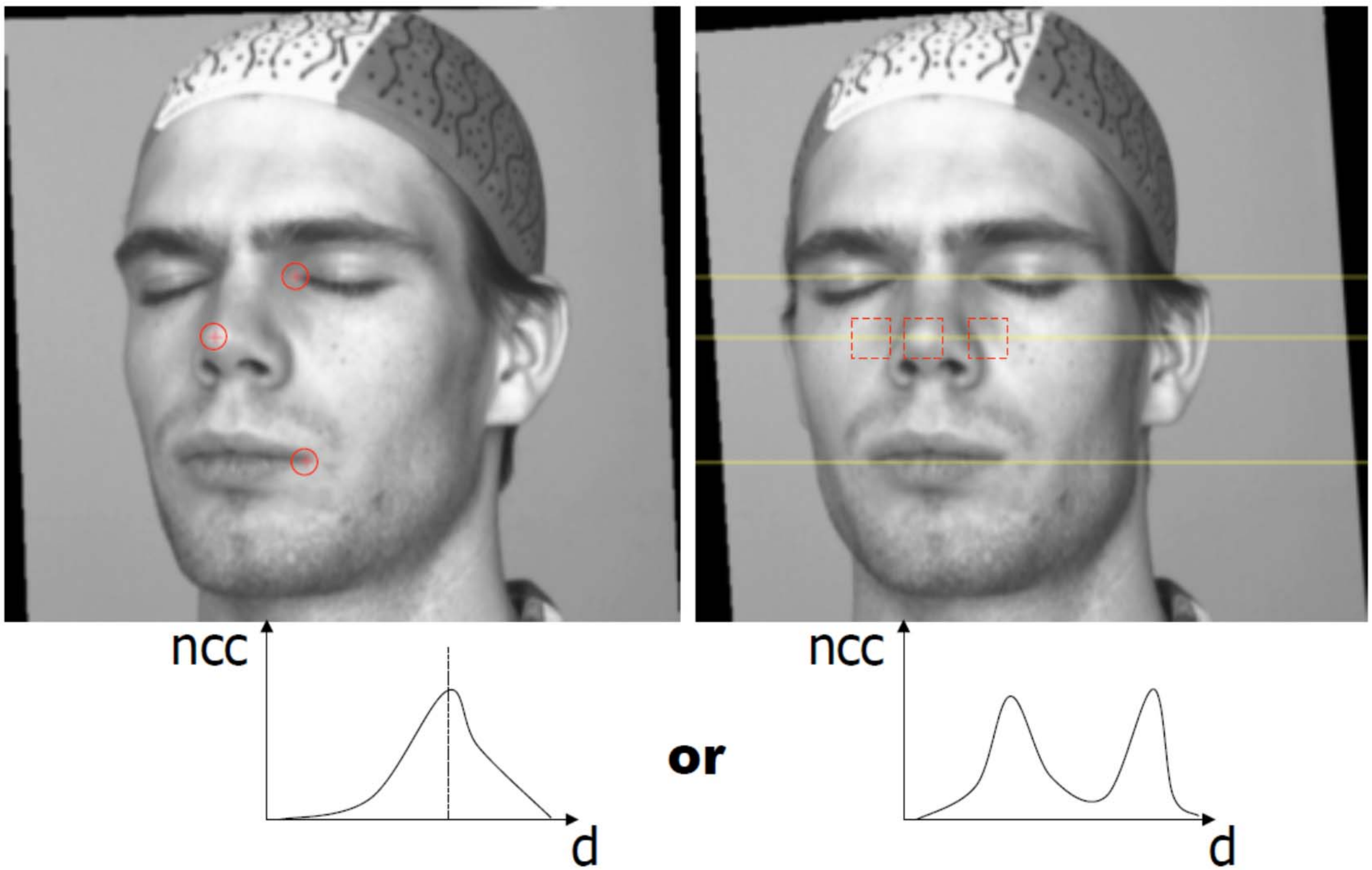
$$d_p = \arg\min_{0 \le d \le d\max} \sum_{q \in Wp} c(q, q-d)$$

- where
  - argmin returns the value at which the function takes a minimum
  - d_{max} is a parameter defining the maximum disparity (search range)
  - W_p is the set of all pixels inside the window centered on p
  - c(p,q) is a function that computes the color difference between a pixel p of the left and a pixel q of the right image

Cost/Score Curve

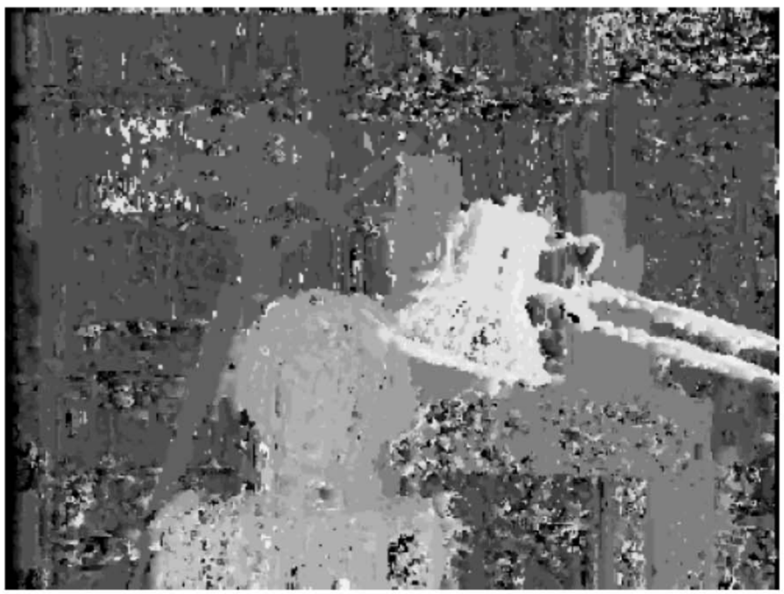


Cost/Score Curve

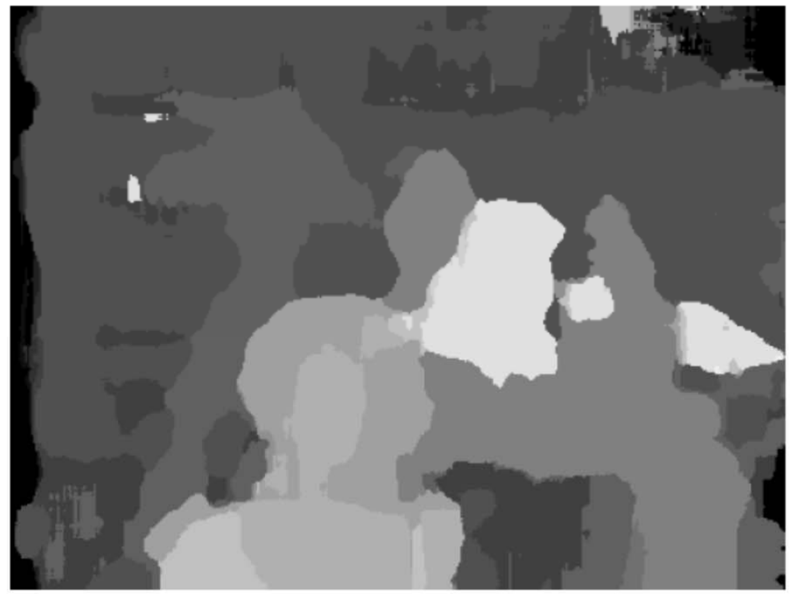


Results

- The window size is a crucial parameter



Window size = 3x3 pixels



Window size = 21x21 pixels

Challenges

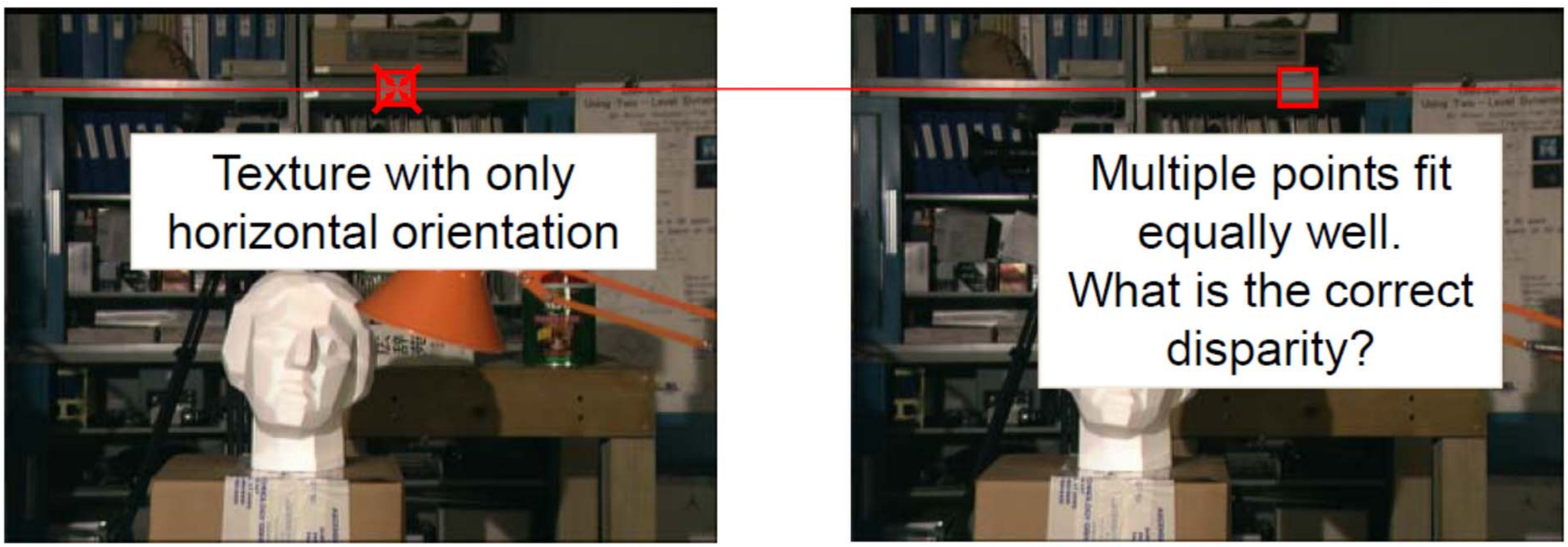
- Ill-posed inverse problem
  - Recover 3-D structure from 2-D information
- Difficulties
  - Uniform regions
  - Half-occluded pixels
  - Repeated patterns

Untextured Regions

(a) Left image (b) Right image

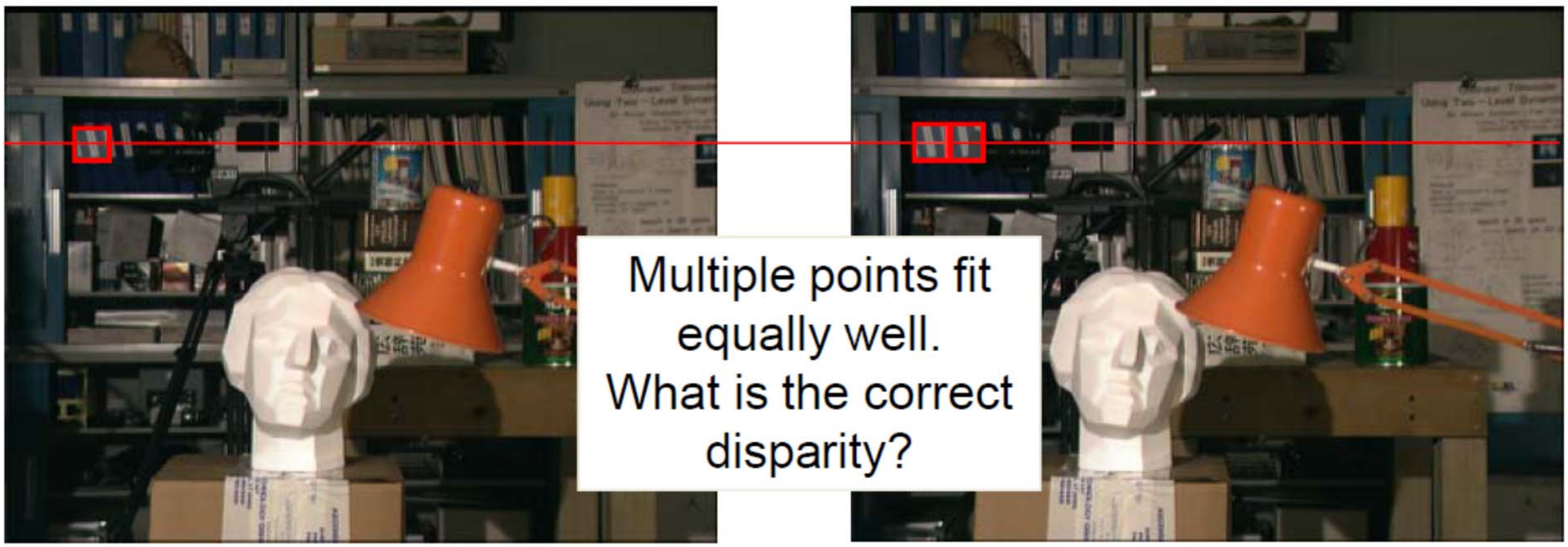
Aperture Problem

- There needs to be a certain amount of texture with vertical orientation



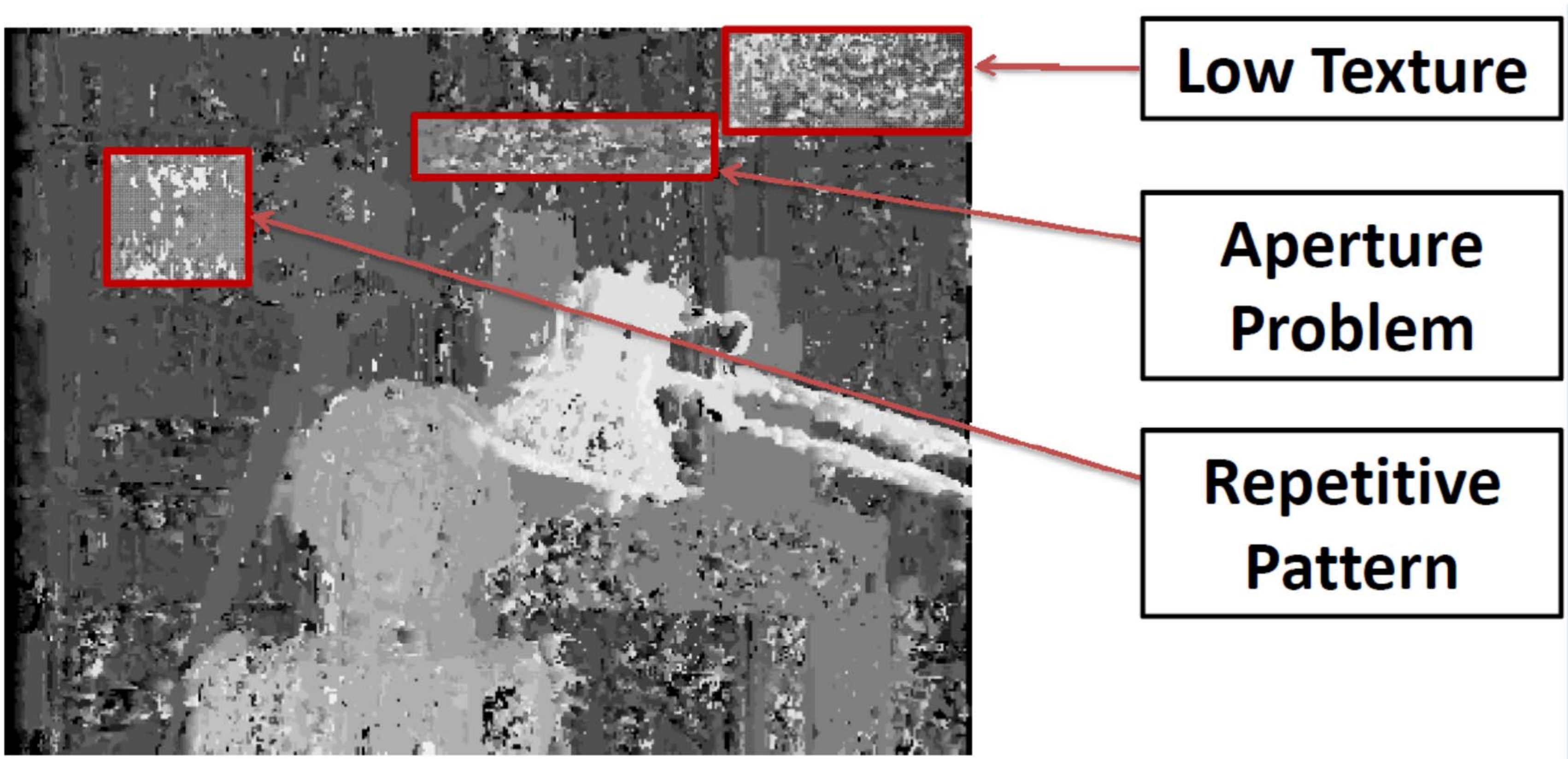
(a) Left image (b) Right image

Repetitive Patterns



(a) Left image (b) Right image

Effects of these Problems



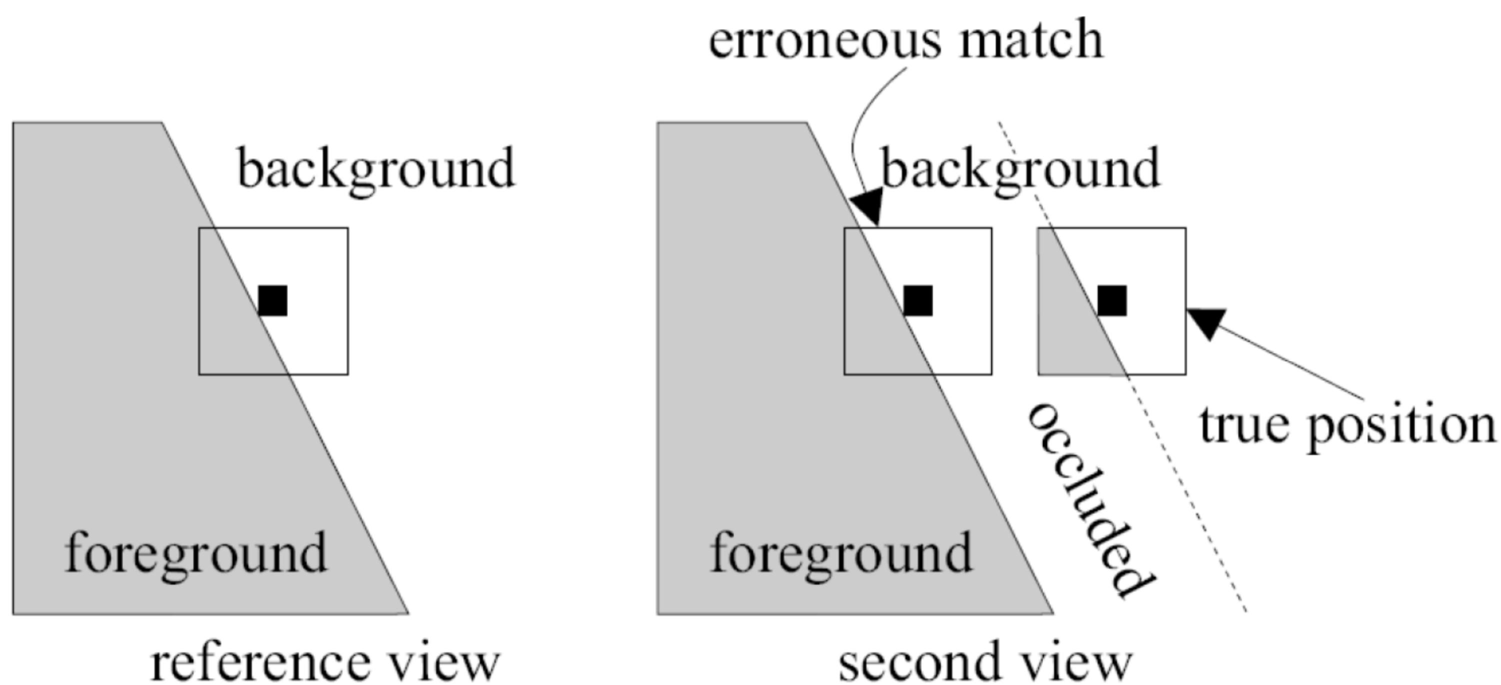
Window size = 3x3 pixels

Foreground Fattening

- By using a window as matching primitive, we have made an implicit smoothness assumption:
  - All pixels within the window are assumed to have the same disparity
- This leads to a systematic error in regions close to disparity discontinuities

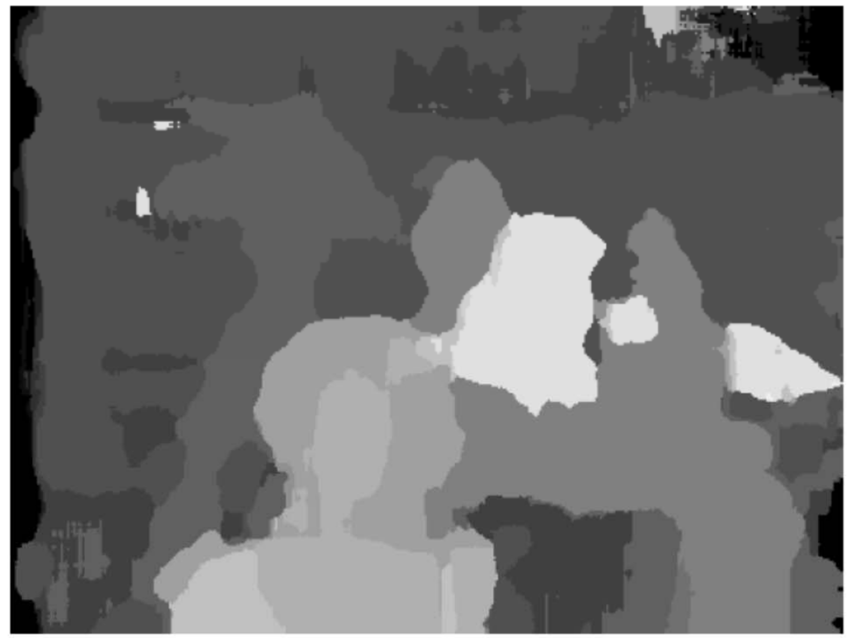
Foreground Fattening

- Background regions close to disparity discontinuities tend to be erroneously assigned to the foreground disparity



Foreground Fattening

Ground Truth Disparities



Window size = 21x21 pixels

Large vs. Small Windows

- Large windows are better for:
  - Untextured Regions
  - Aperture Problem
  - Repetitive Patterns

Why?

- Small windows reduce:
  - Foreground Fattening Effect
- Problem:
  - There is no ‘optimal’ window size that can handle all these problems at once

Pixel Dissimilarity

- Sum of Squared Differences of intensities (SSD)

$$SSD(x,y;d) = \sum_{(x',y')\in \mathrm{N}(x,y)} [I_L(x',y') - I_R(x'-d,y')]^2$$

- Sum of Absolute Differences of intensities (SAD)

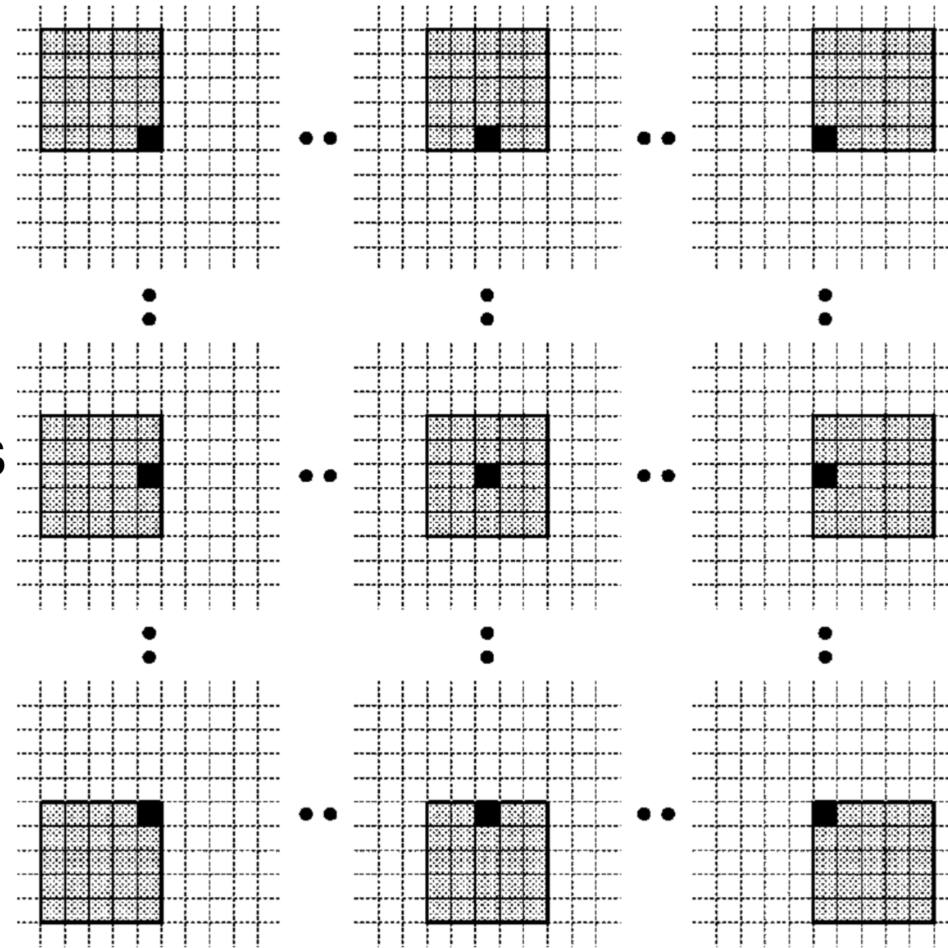
$$SAD(x,y;d) = \sum_{(x',y')\in \mathrm{N}(x,y)} |I_L(x',y') - I_R(x'-d,y')|$$

- Zero-mean Normalized Cross-correlation (NCC)

$$NCC(x,y,d) = \frac{\sum_{i\in W}(I_L(x_i,y_i) - \mu_L)(I_R(x_i - d, y_i) - \mu_R)}{\sigma_L \sigma_R}$$

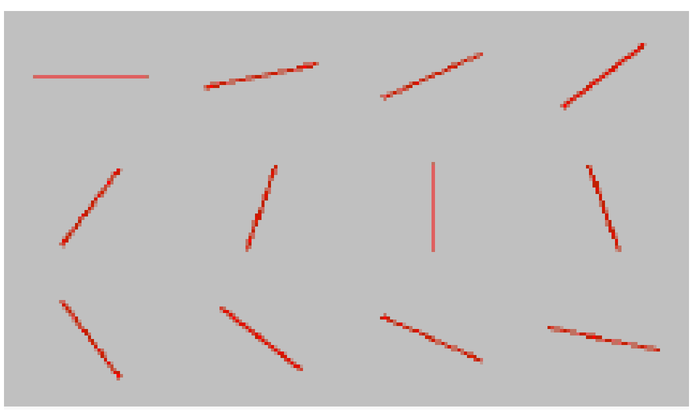
Shiftable Windows

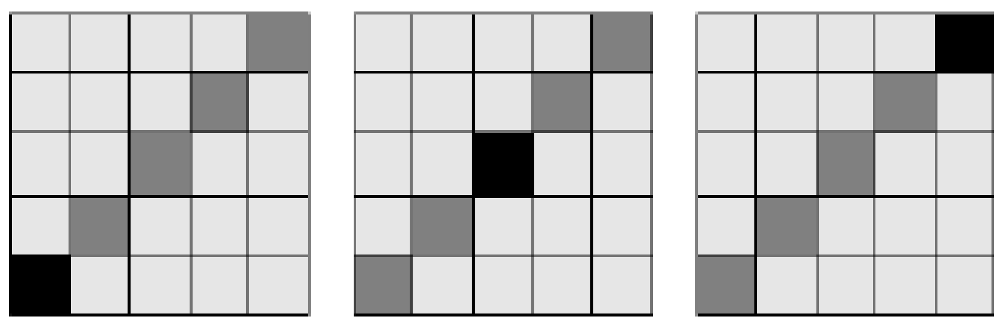
- Avoid having using matching windows that straddle two surfaces
  - Disparity will not be constant for all pixels
- Shift the window around the reference pixel
  - Keep the one with min cost (max NCC)



Rod-shaped Filters

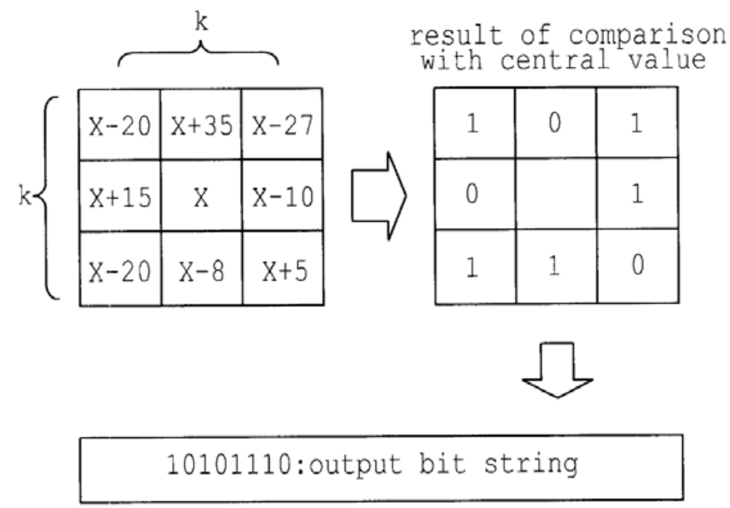
- Instead of square windows aggregate cost in rod-shaped shiftable windows
- Search for one that minimizes the cost (assume that it is an iso-disparity curve)





Alternative Dissimilarity Measures

- Rank and Census transforms
- Rank transform:
  - Define window containing R pixels around each pixel
  - Count the number of pixels with lower intensities than center pixel in the window
  - Replace intensity with rank (0..R-1)
  - Compute SAD on rank-transformed images
- Census transform:
  - Use bit string, defined by neighbors, instead of scalar rank
- Robust against illumination changes



Locally Adaptive Support

Apply weights to contributions of neighboring pixels according to similarity and proximity

(a) left support window (b) right support window (c) color difference between (a) and (b)

Locally Adaptive Support

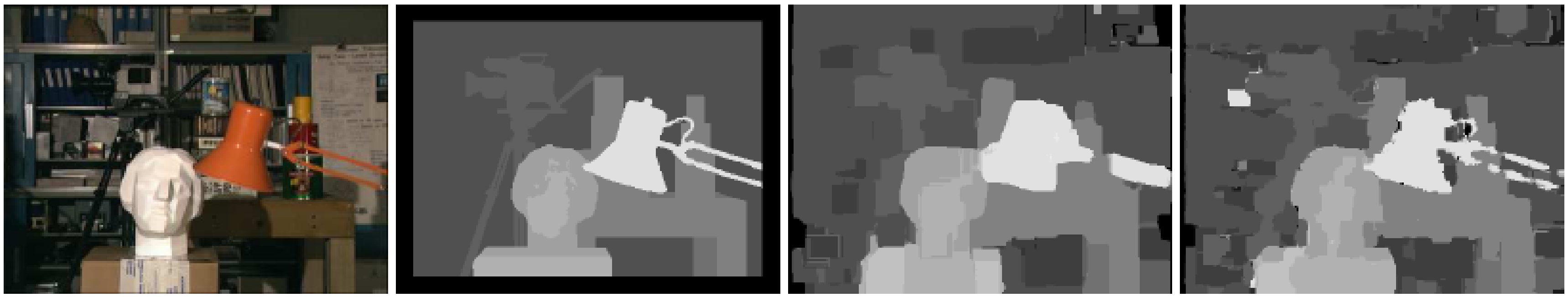
- Similarity in CIE Lab color space:

$$\Delta c_{pq} = \sqrt{(L_p - L_q)^2 + (a_p - a_q)^2 + (b_p - b_q)^2}$$

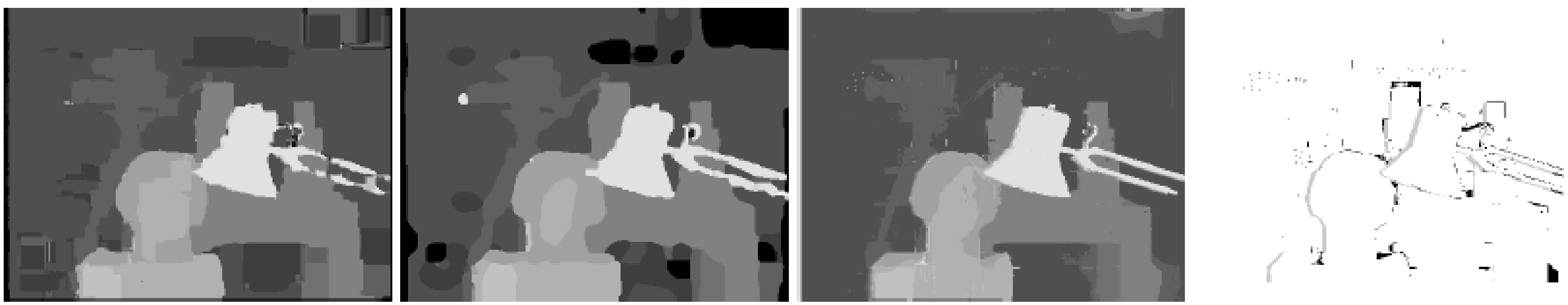
- Proximity: Euclidean distance

- Weights: $w(p,q) = k \cdot \exp\left(-(\frac{\Delta c_{pq}}{\gamma_c} + \frac{\Delta g_{pq}}{\gamma_p})\right)$

Locally Adaptive Support: Results



(a) left image (b) ground truth (c) shiftable win. [7] (d) compact win. [3]

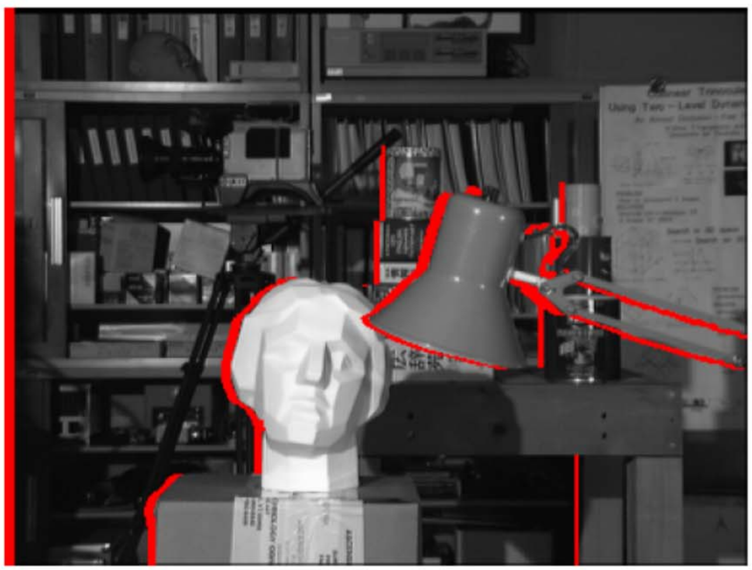


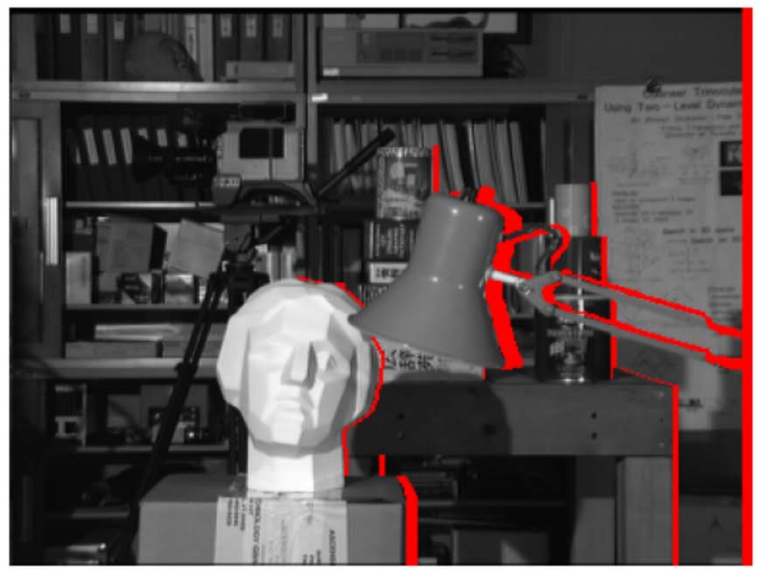
(e) variable win. [4] (f) Bay. diff. [19] (g) our result (h) bad pixels (error > 1)

Implement SAD

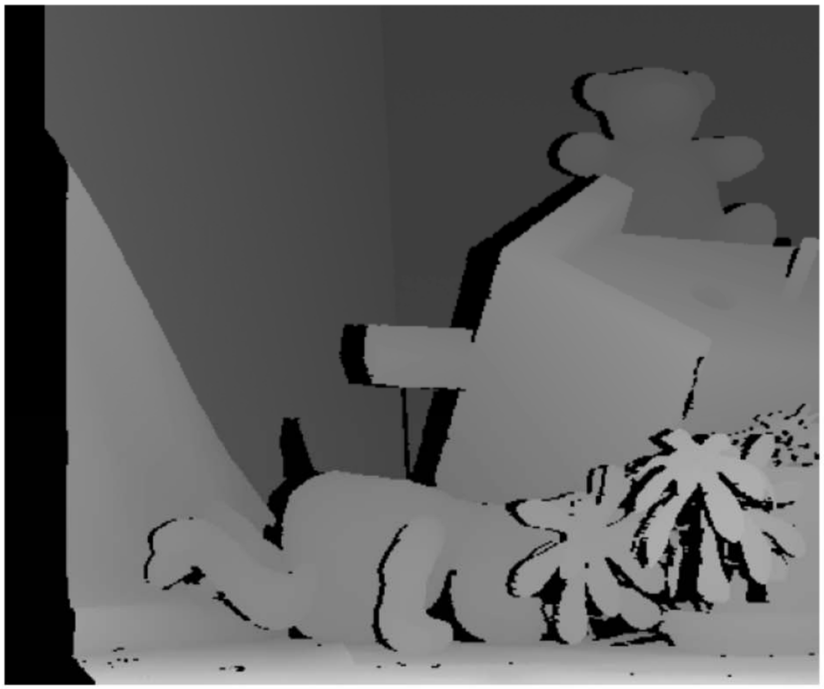
Occlusion

- There are pixels that are only visible in one of the two views (red pixels in the images)
- For occluded, pixels there exists no correspondence => We cannot estimate disparity

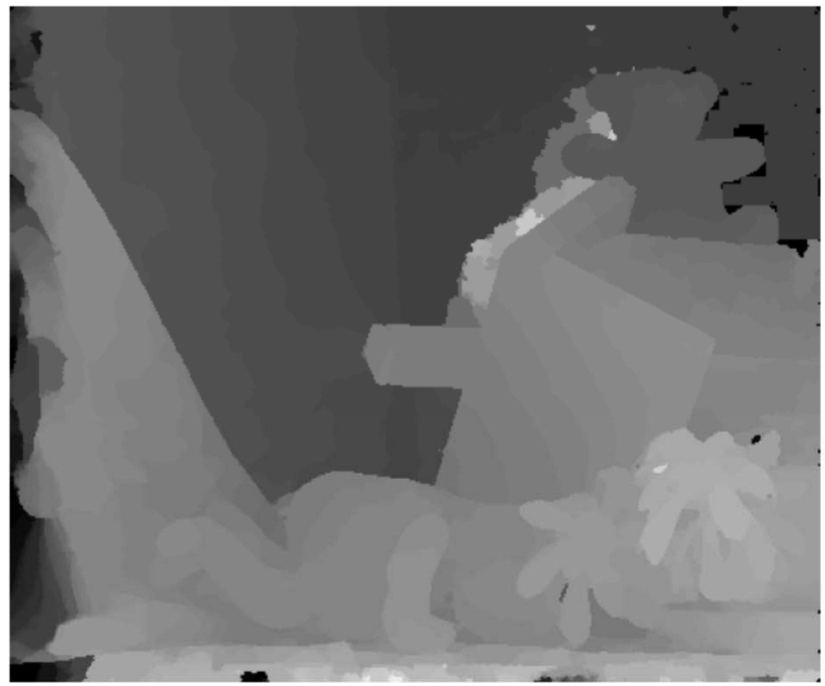




Effects of Occlusion



Ground Truth with Occlusions in Black



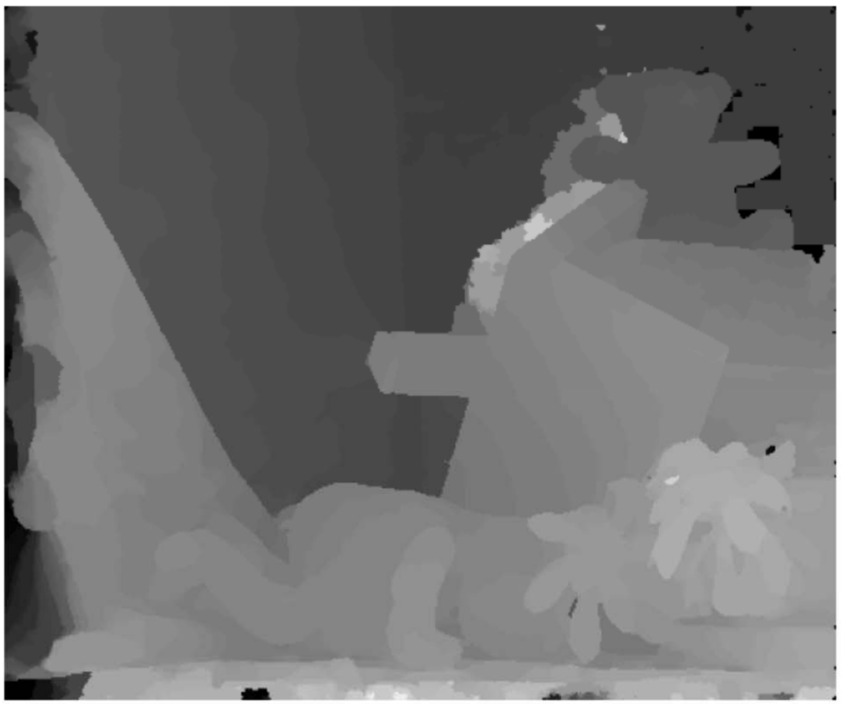
Core Algorithm of [Hosni,ICIP09]

Approximate adaptive support weight implementation

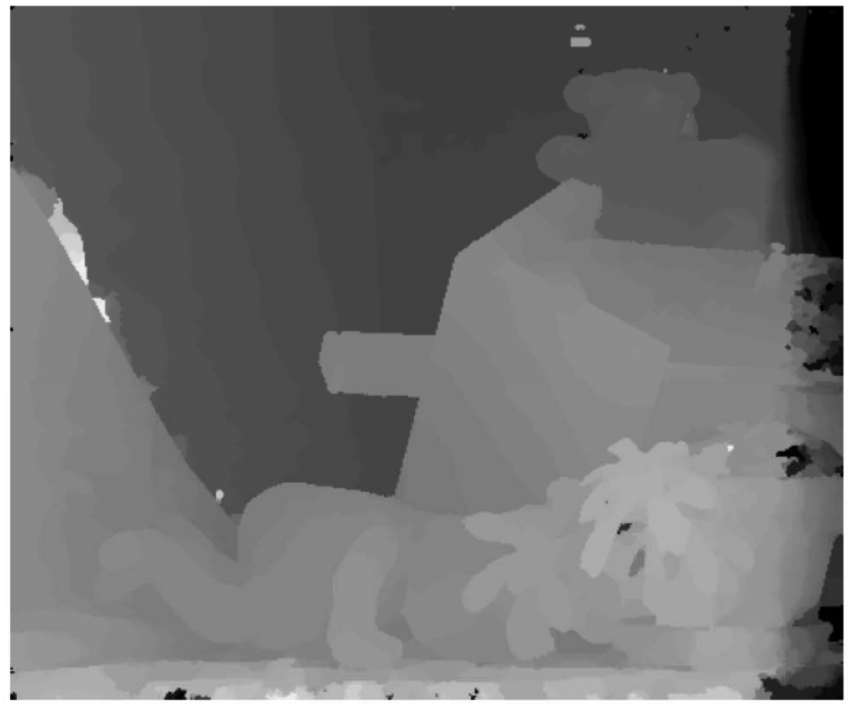
Left-Right Consistency

- Compute 2 disparity maps
  - Using the left image as reference frame
  - Using the right image as reference frame
- Left-right consistency check:
  - For each pixel p_l of the left view:
  - Lookup p_l's matching point m_r in the right view using the left disparity map
  - For the pixel m_r, lookup its matching point q_l in the left view using the right disparity map
  - If $p_l = q_l$ the disparity is assumed to be correct
  - Otherwise, the disparity is invalidated
- Check typically fails for
  - Occluded pixels
  - Mismatched pixels

Left-Right Consistency

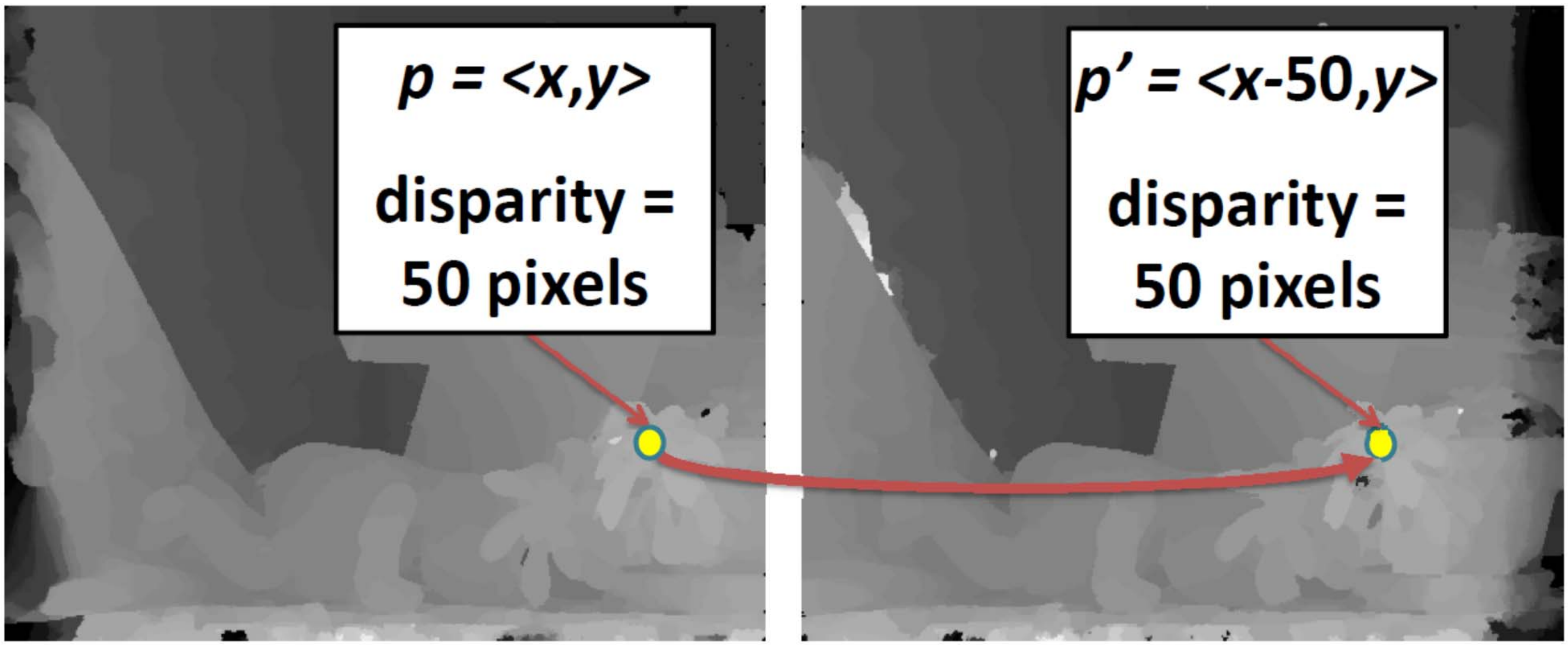


Disparity Map (Left Reference)



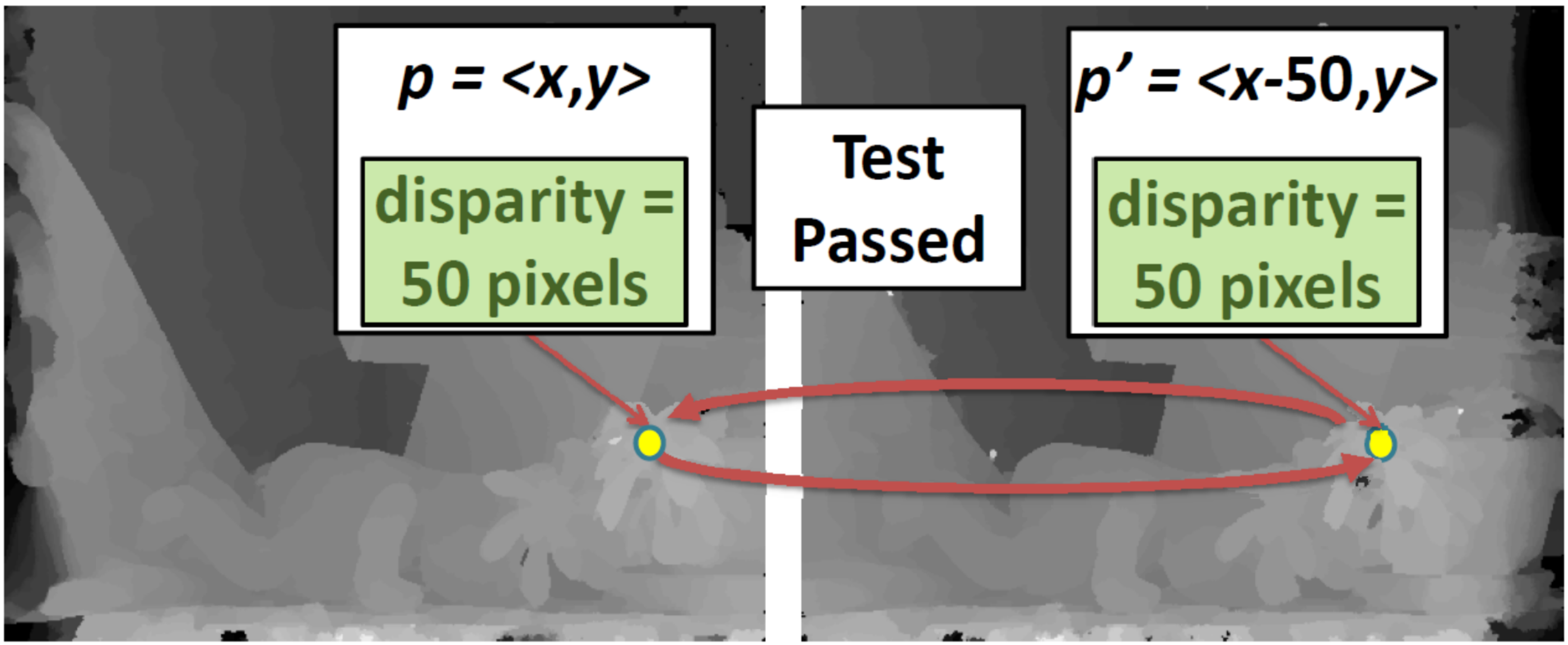
Disparity Map (Right Reference)

Left-Right Consistency



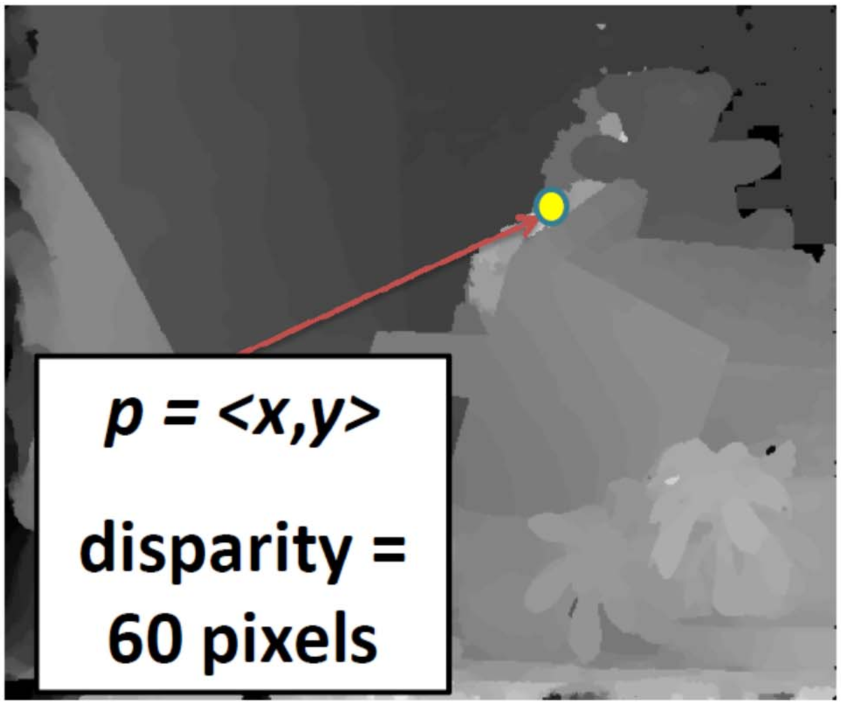
Disparity Map (Left Reference) Disparity Map (Right Reference)

Left-Right Consistency

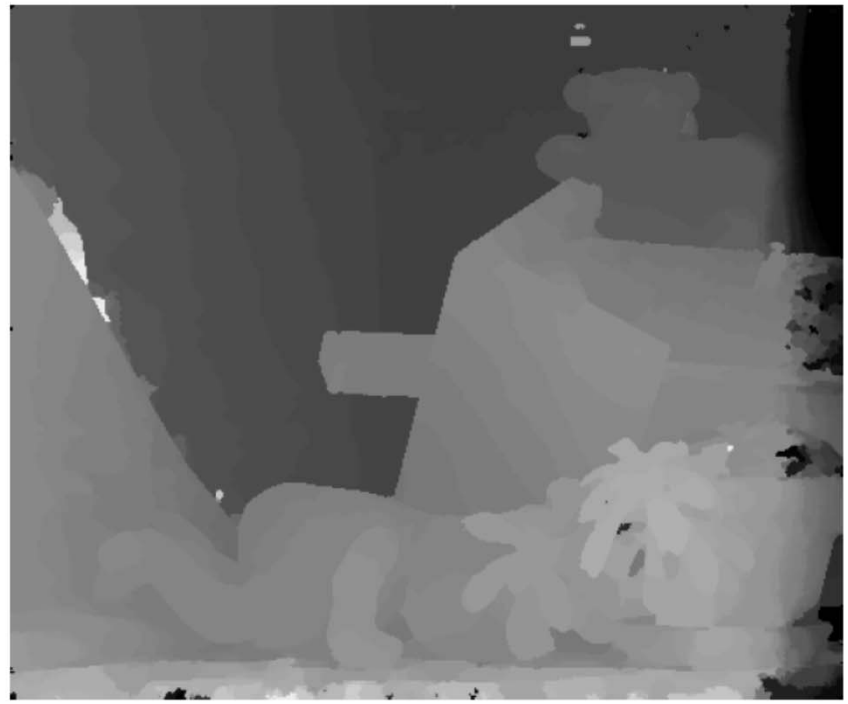


Disparity Map (Left Reference)   Disparity Map (Right Reference)

Left-Right Consistency

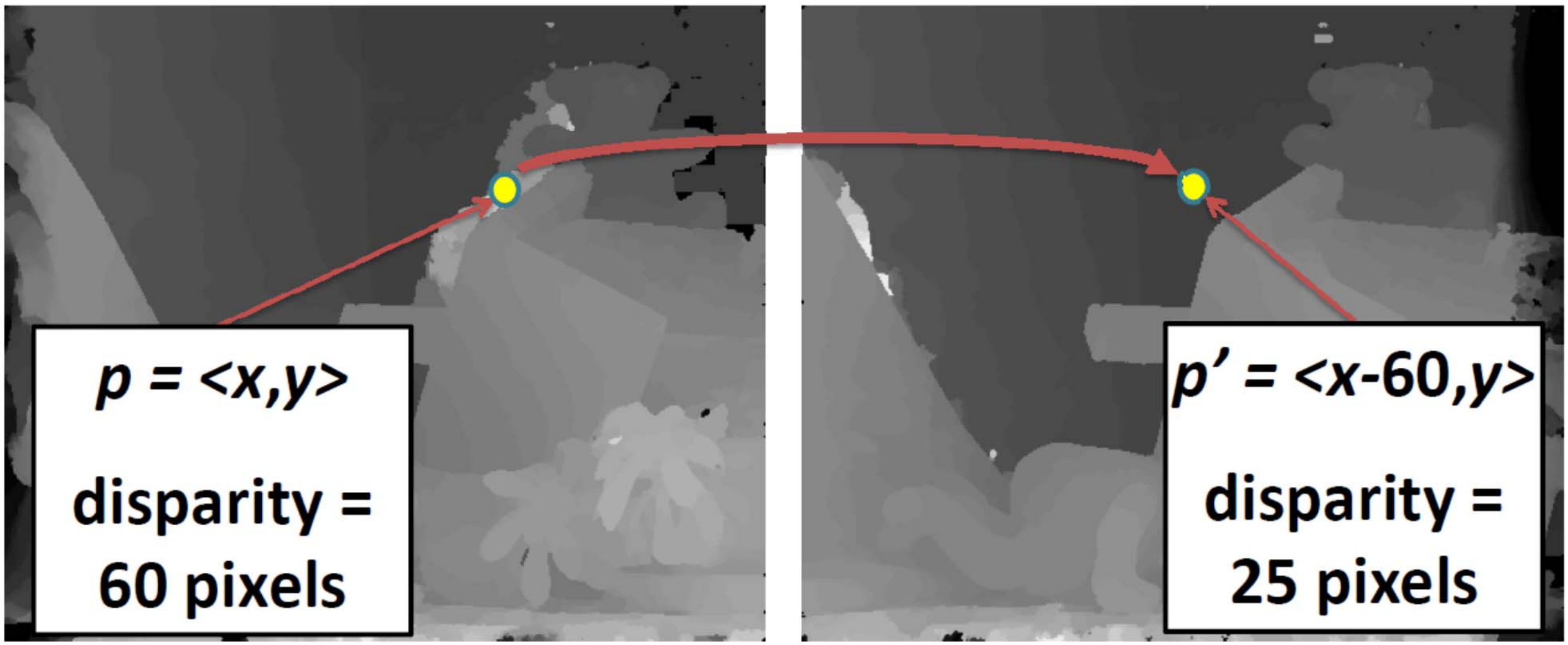


Disparity Map (Left Reference)



Disparity Map (Right Reference)

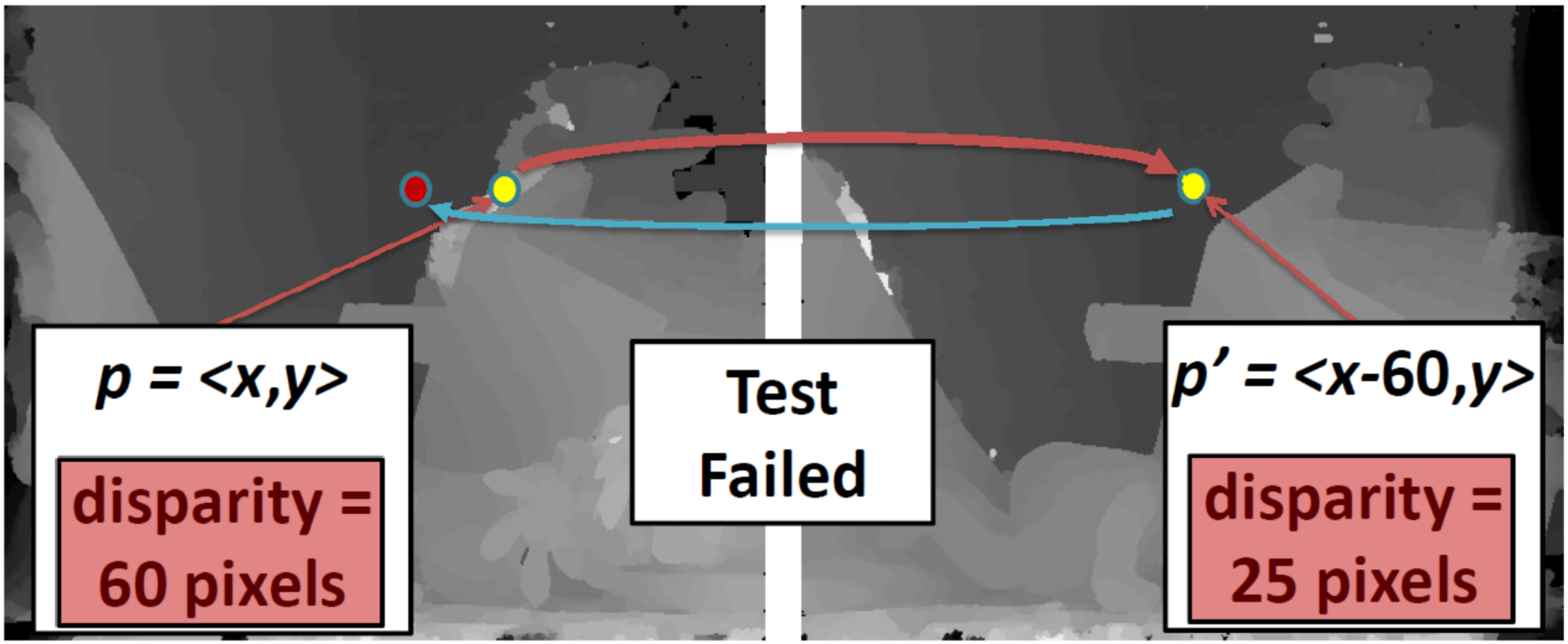
Left-Right Consistency



Disparity Map (Left Reference)

Disparity Map (Right Reference)

Left-Right Consistency



Disparity Map (Left Reference) Disparity Map (Right Reference)

Confidence Measures for Stereo Matching

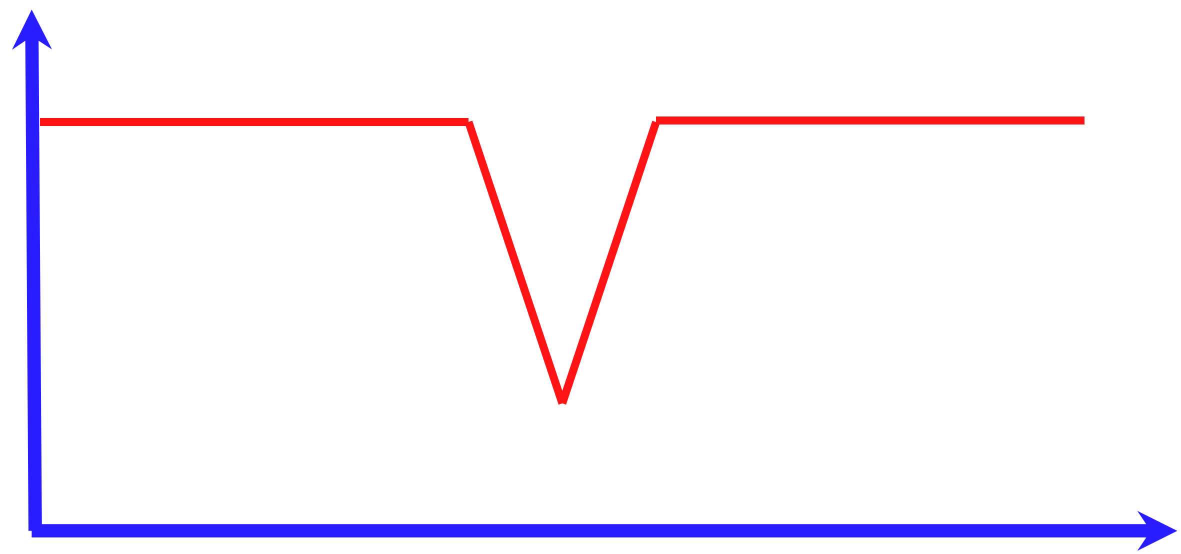
Cost Functions

$$SAD(x,y,d) = \sum_{i \in W} |I_L(x_i,y_i) - I_R(x_i - d, y_i)|$$

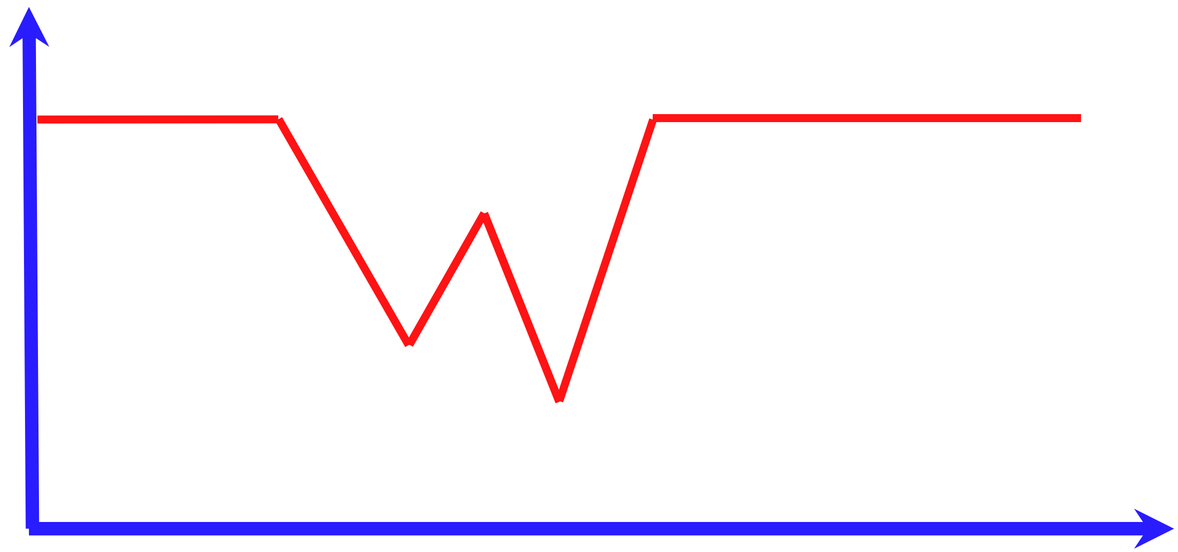
$$NCC(x,y,d) = \frac{\sum_{i \in W}(I_L(x_i,y_i) - \mu_L)(I_R(x_i - d, y_i) - \mu_R)}{\sigma_L \sigma_R}$$

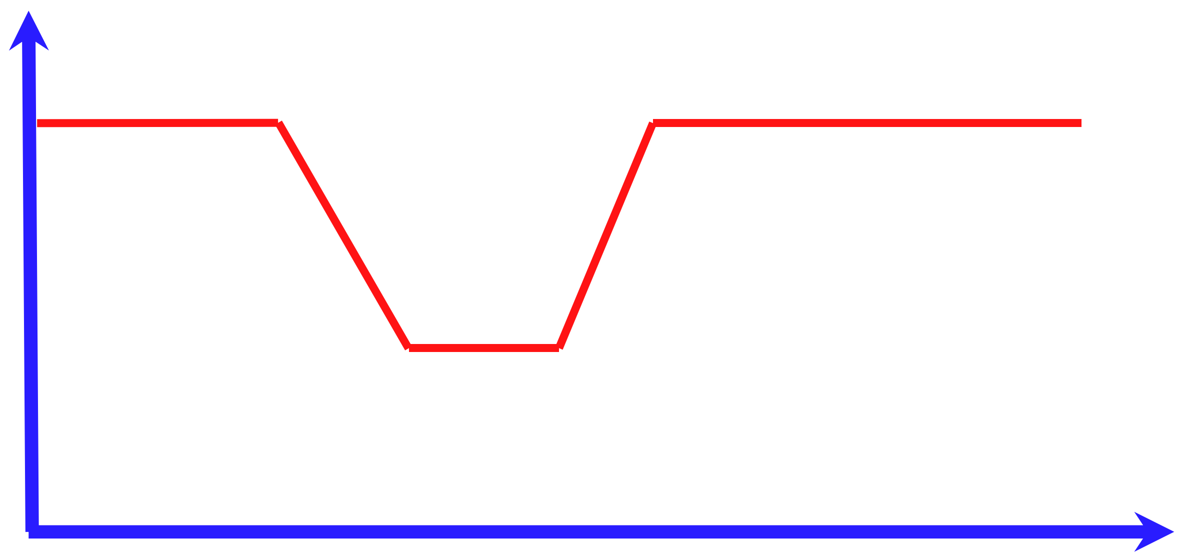
- Functions of disparity d
  - $d=x_L-x_R$

Ideal Cost Curve



Non-Ideal Cost Curves

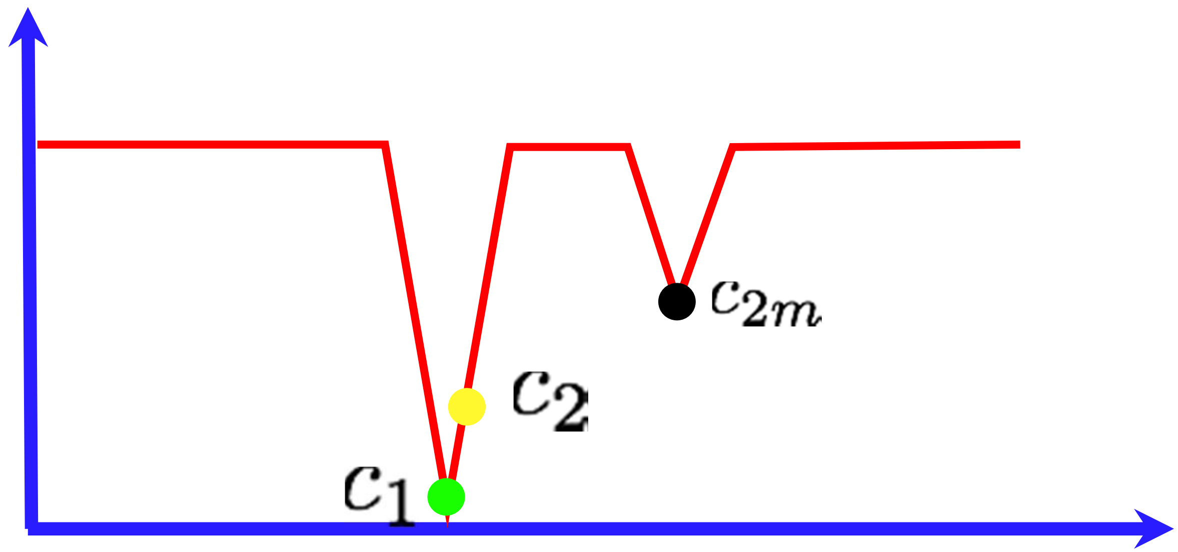




Correspondence Uncertainty Measures

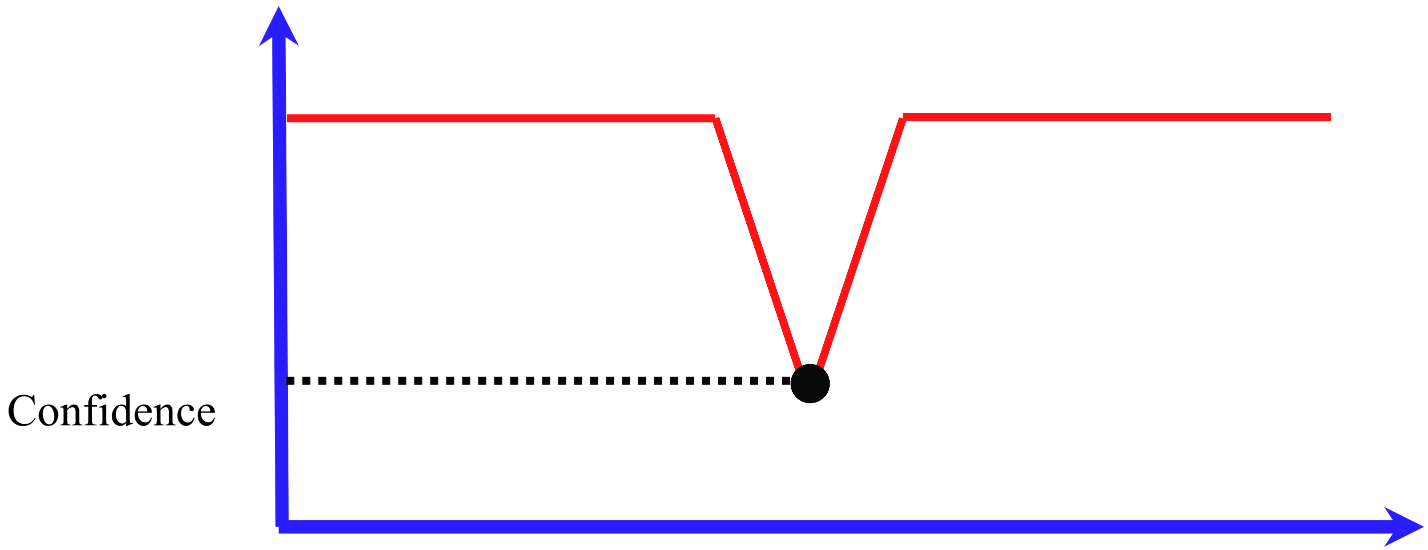
1. Matching Cost
2. Local Properties of the Cost Curve
3. Local Minima of the Cost Curve
4. The Entire Cost Curve
6. Consistency Between the Left and Right Disparity Maps
7. More…

Notation



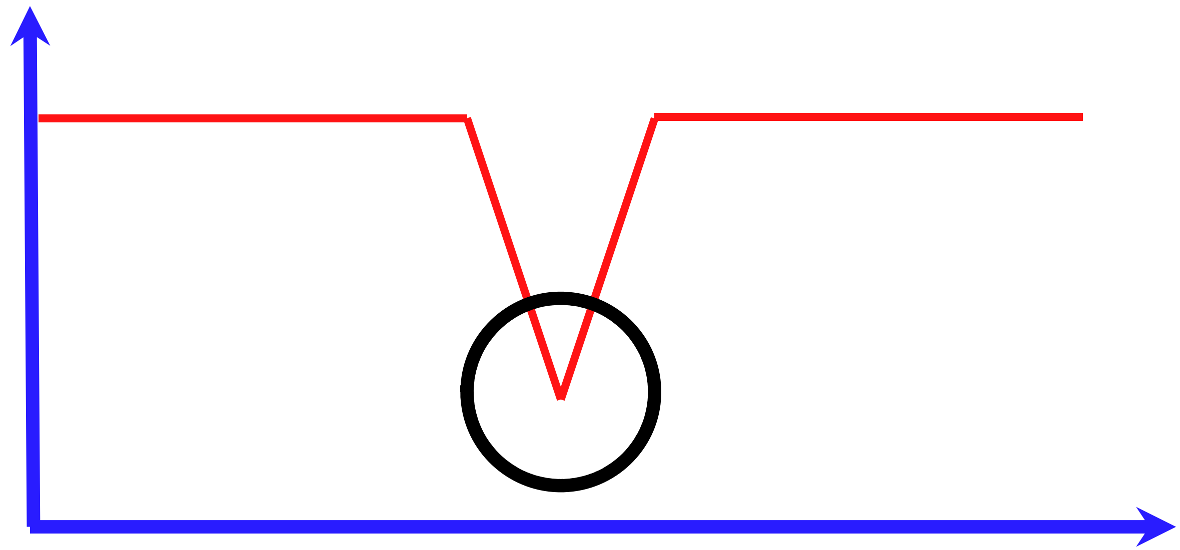
c_1 global minimum of the cost curve
c_2 second smallest value of the cost curve
c_{2m} second smallest value of the cost curve that is also a local minimum

Matching Cost



Simply using matching cost:
Low cost values correspond to high confidence
high cost values correspond to low confidence

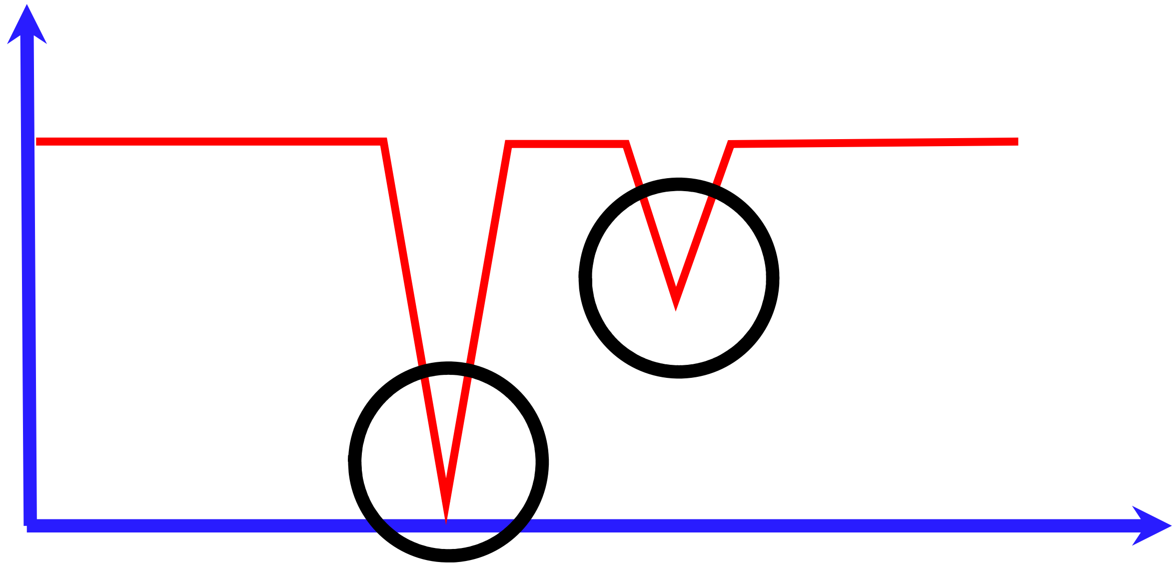
Local Properties of the Cost Curve



$$C_{CUR} = -2c(d_1) + c(d_1 - 1) + c(d_1 + 1)$$

Low curvature indicates flat region around minimum cost

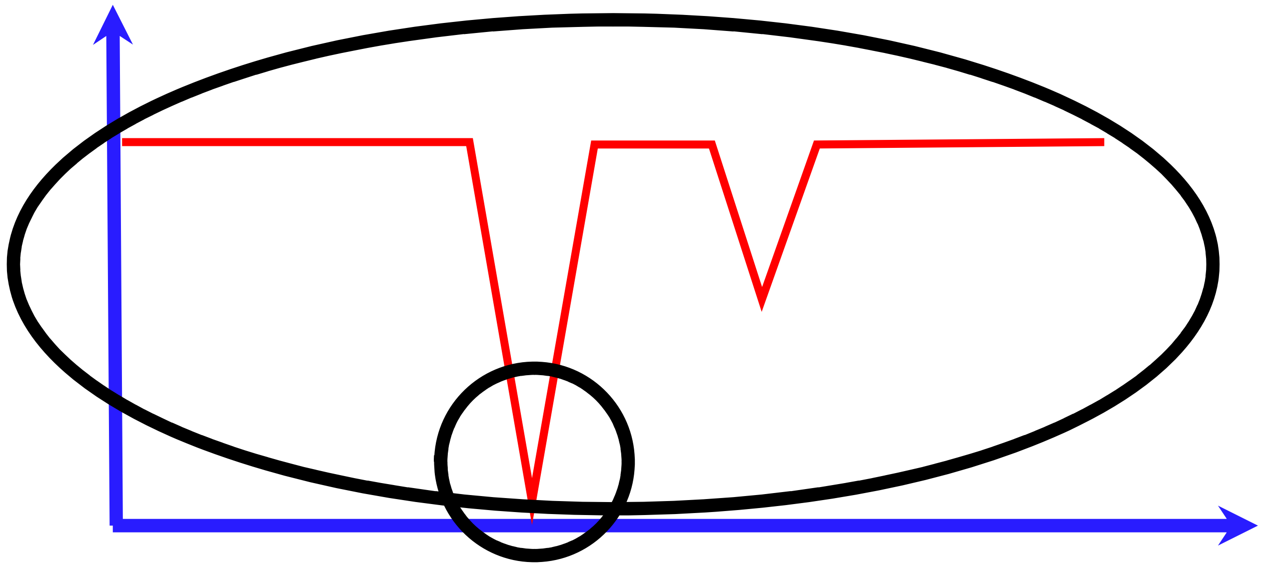
Local Minima of the Cost Curve



$$C_{PKR} = \frac{c_{2m}}{c_1} \qquad C_{\text{PKRN}} = \frac{c_2}{c_1}$$

Match is ambiguous if multiple strong candidates exist

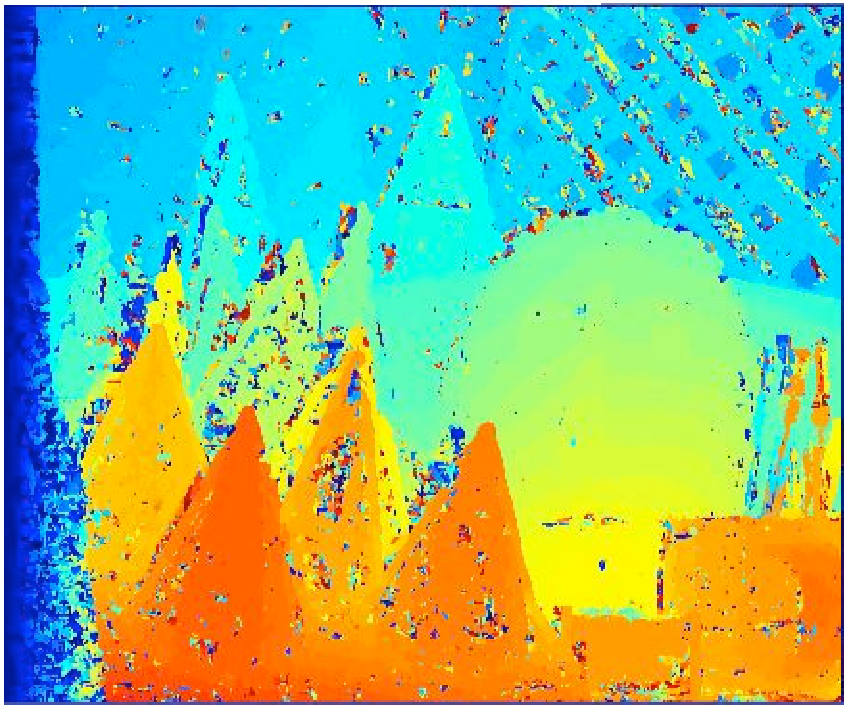
The Entire Cost Curve

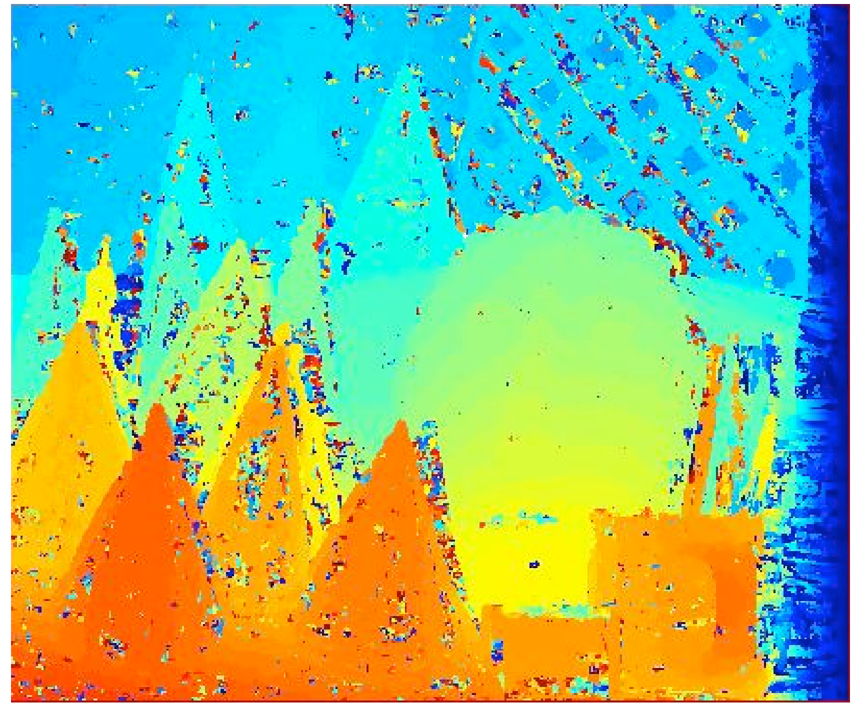


$$C_{AML} = \frac{e^{-\frac{(c_1 - c_1)^2}{2\sigma^2_{AML}}}}{\sum_d e^{-\frac{(c(d) - c_1)^2}{2\sigma^2_{AML}}}}$$

Tests for both flat regions and multiple strong candidates by converting cost curve to probability function and measuring the probability of the best match

Left-Right Consistency





$$C_{LRC}(x, y) = -|d_1 - D_R(x - d_1, y)|$$

LRC is not binary as before, but equal to difference of corresponding disparities

Distinctiveness-based Confidence Measures

- Distinctiveness: Perceptual distance to the most similar other point in the search window in the reference image

Stereo
Beyond Winner-Take-All

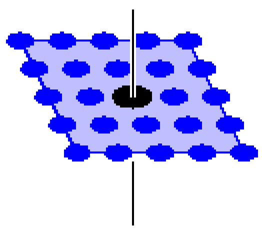
Stereo with Non-Linear Diffusion

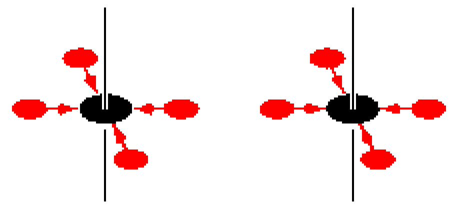
- Problem with traditional approach:
  - gets confused near discontinuities
- Non-Linear Diffusion:
  - use iterative (non-linear) aggregation to obtain better estimate

Diffusion

- Average energy with neighbors + starting value

$$E(x,y,d) \leftarrow (1-4\lambda)E(x,y,d) + \lambda \sum_{(k,l)\in\mathcal{N}_4} E(x+k,y+l,d)$$
$$+\beta(E_0(x,y,d) - E(x,y,d))$$



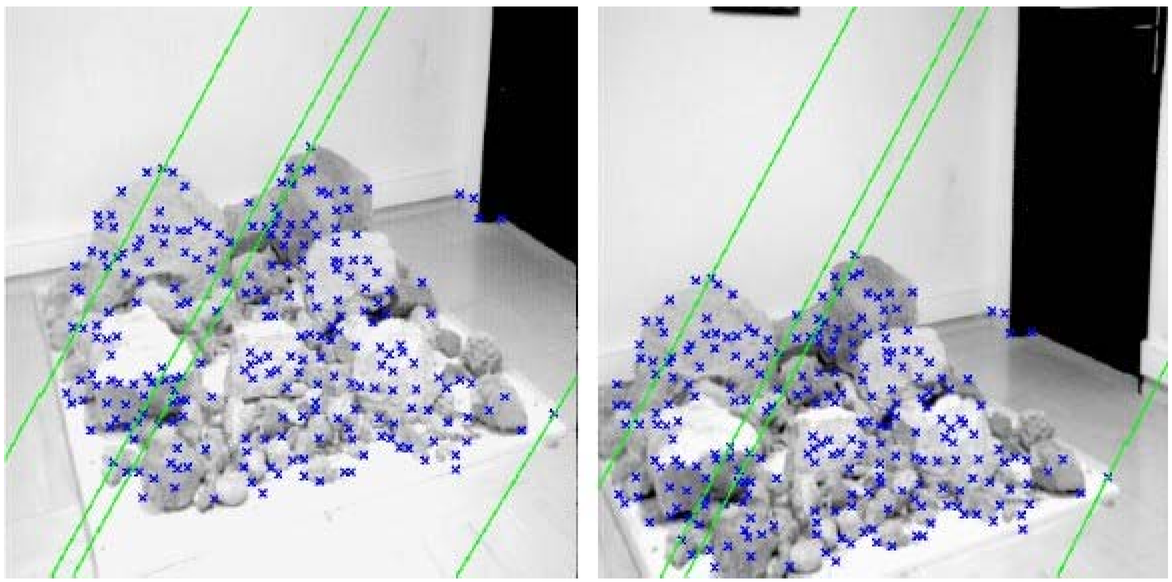


- window diffusion

Requires appropriate stopping criteria to prevent excessive smoothing

Feature-based Stereo

- Match “corner” (interest) points

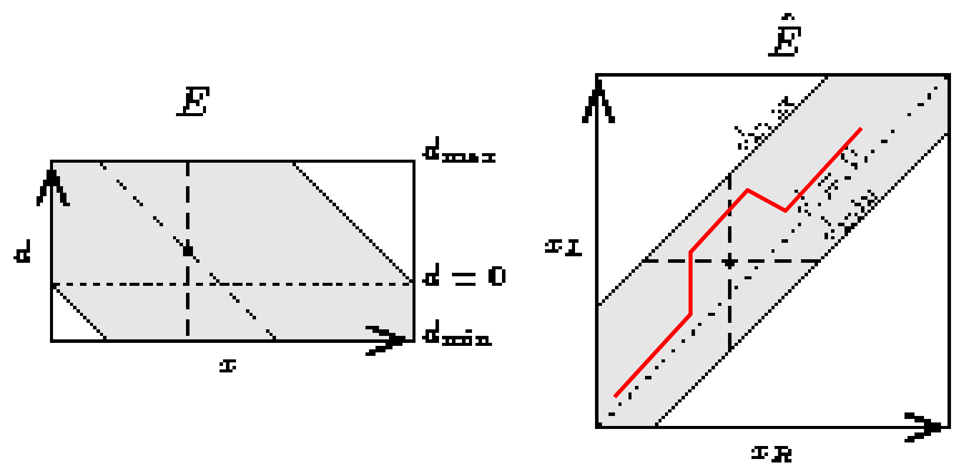


- Interpolate complete solution

Dynamic Programming

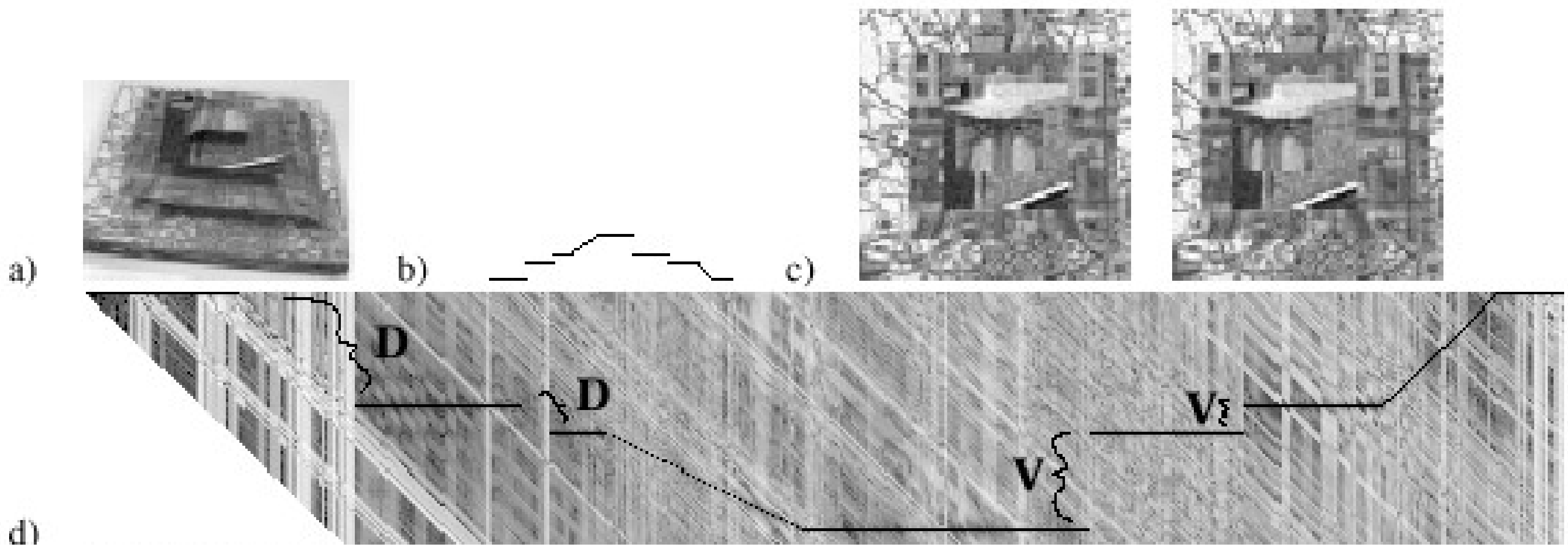
- 1-D cost function

$$
\begin{aligned}
E(\mathbf{d}) &= \sum_{x,y} \rho_P(d_{x+1,y} - d_{x,y}) + \sum_{x,y} E_0(x, y; d) \\
\tilde{E}(x, y, d) &= E_0(x, y; d) + \\
&\quad \min_{d'} \left(\tilde{E}(x-1, y, d') + \rho_P(d_{x,y} - d'_{x-1,y}) \right)
\end{aligned}
$$



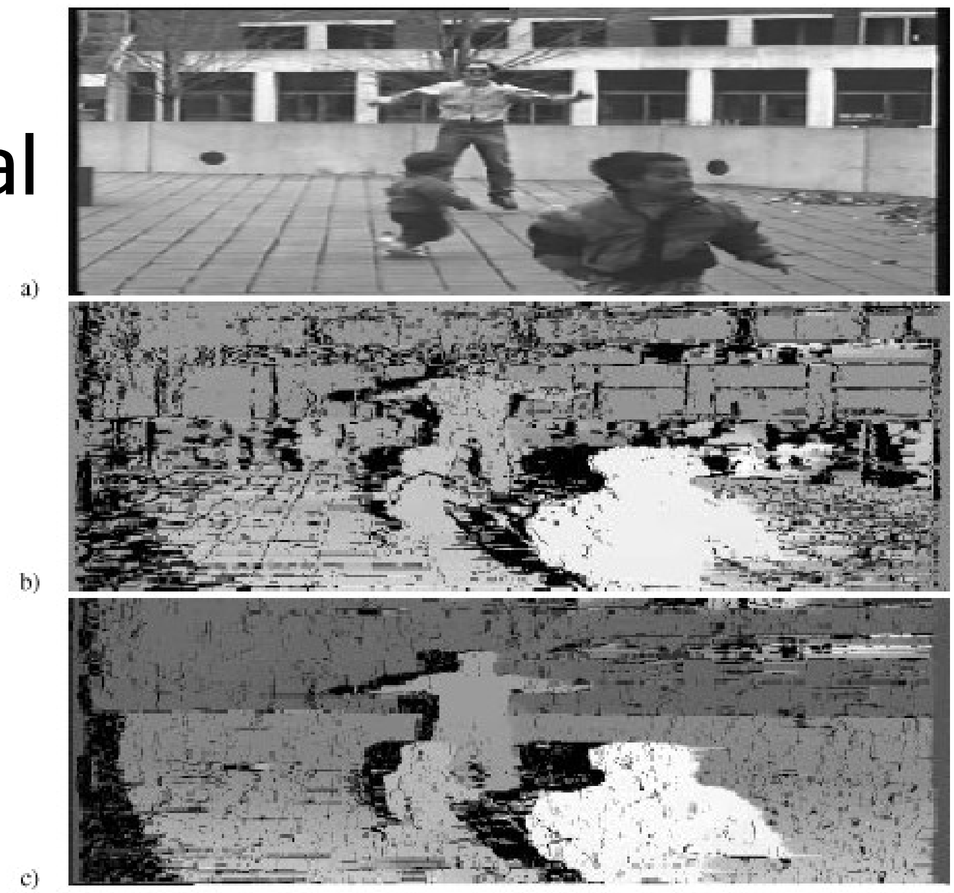
Dynamic Programming

- Disparity space image and min. cost path



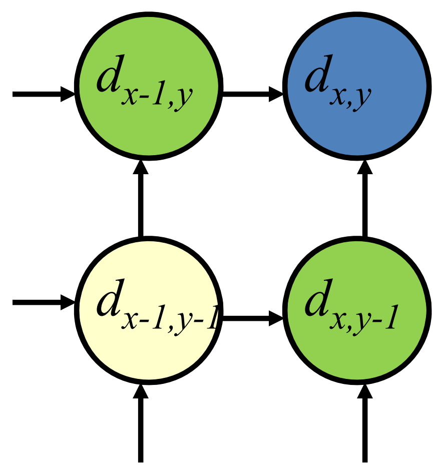
Dynamic Programming

- Sample result (note horizontal streaks)



Dynamic Programming

- Can we apply this trick in 2D as well?

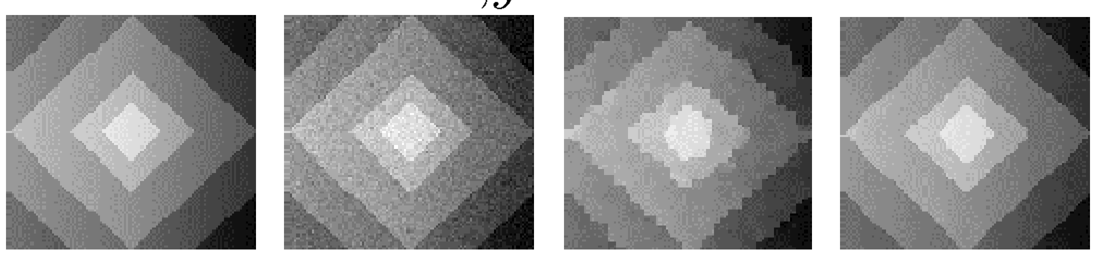


No: $d_{x,y-1}$ and $d_{x-1,y}$ may depend on different values of $d_{x-1,y-1}$

Graph Cuts

- Solution technique for general 2D problem

$$
\begin{aligned}
E_{\text{total}}(\mathbf{d}) &= E_{\text{data}}(\mathbf{d}) + \lambda E_{\text{smoothness}}(\mathbf{d}) \\
E_{\text{data}}(\mathbf{d}) &= \sum_{x,y} f_{x,y}(d_{x,y}) \\
E_{\text{smoothness}}(\mathbf{d}) &= \sum_{x,y} \rho(d_{x,y} - d_{x-1,y}) \\
&+ \sum_{x,y} \rho(d_{x,y} - d_{x,y-1})
\end{aligned}
$$



(a) original image (b) observed image (c) local min w.r.t. standard moves (d) local min w.r.t. α-expansion moves

a-expansion moves

In each *a*-expansion a given label "*a*" grabs space from other labels

initial solution

-expansion

-expansion

-expansion

-expansion

-expansion

-expansion

-expansion

For each move choose the expansion that gives the largest decrease in the energy:
binary optimization problem

Feature Extraction

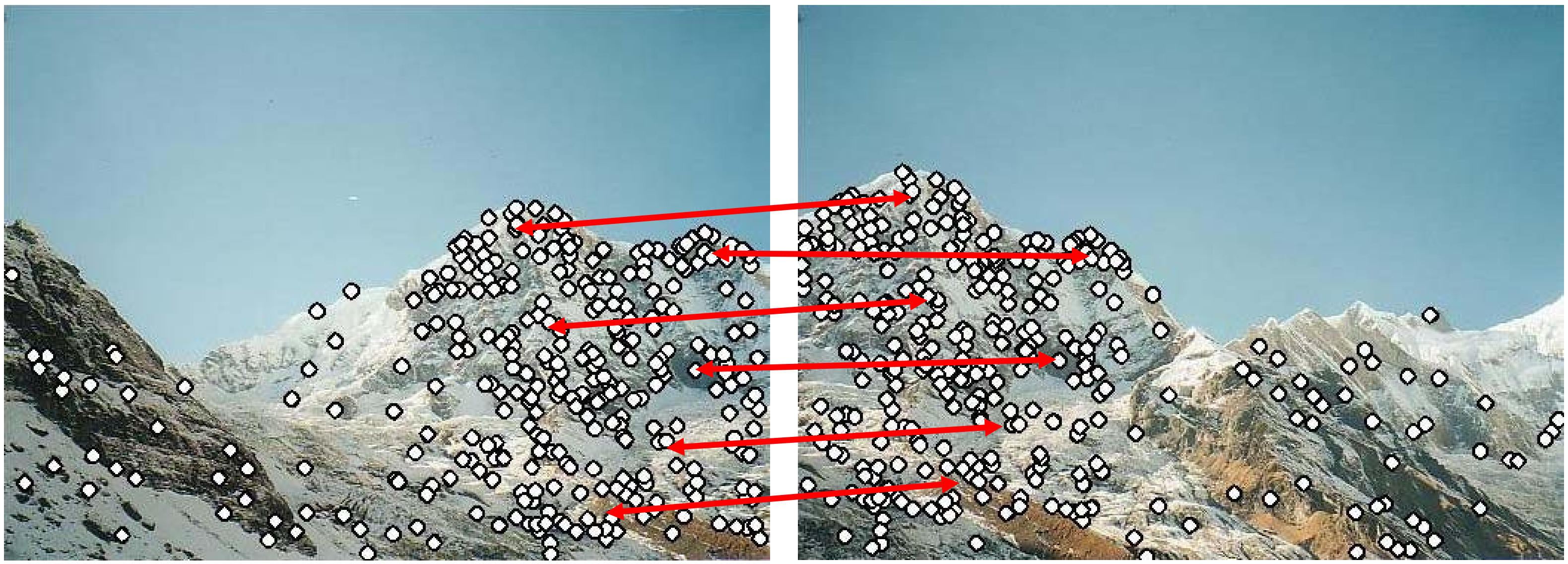
Feature extraction: Corners

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?


Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?



Step 1: extract features

Step 2: match features

Why extract features?

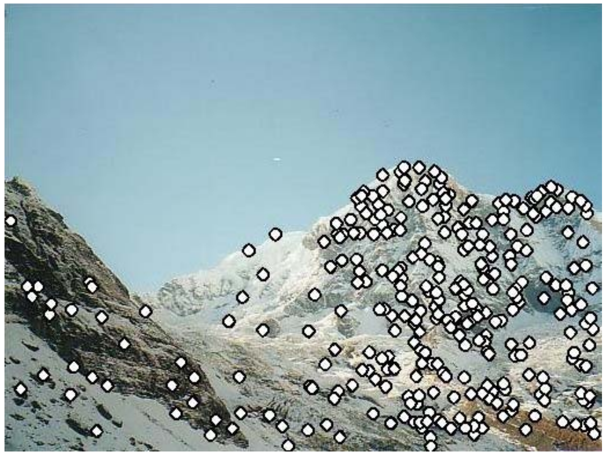
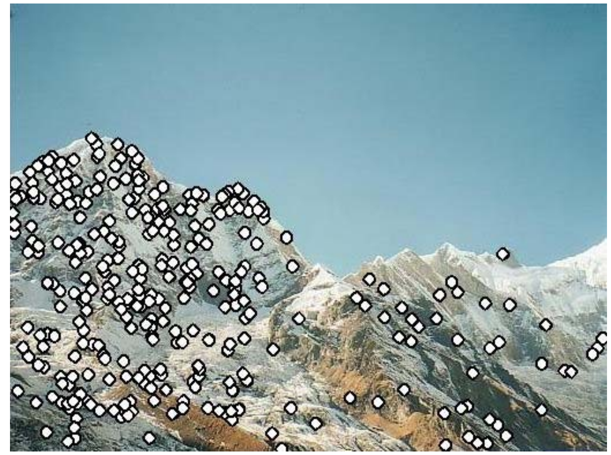
- Motivation: panorama stitching
  - We have two images – how do we combine them?

Step 1: extract features

Step 2: match features

Step 3: align images

Characteristics of good features

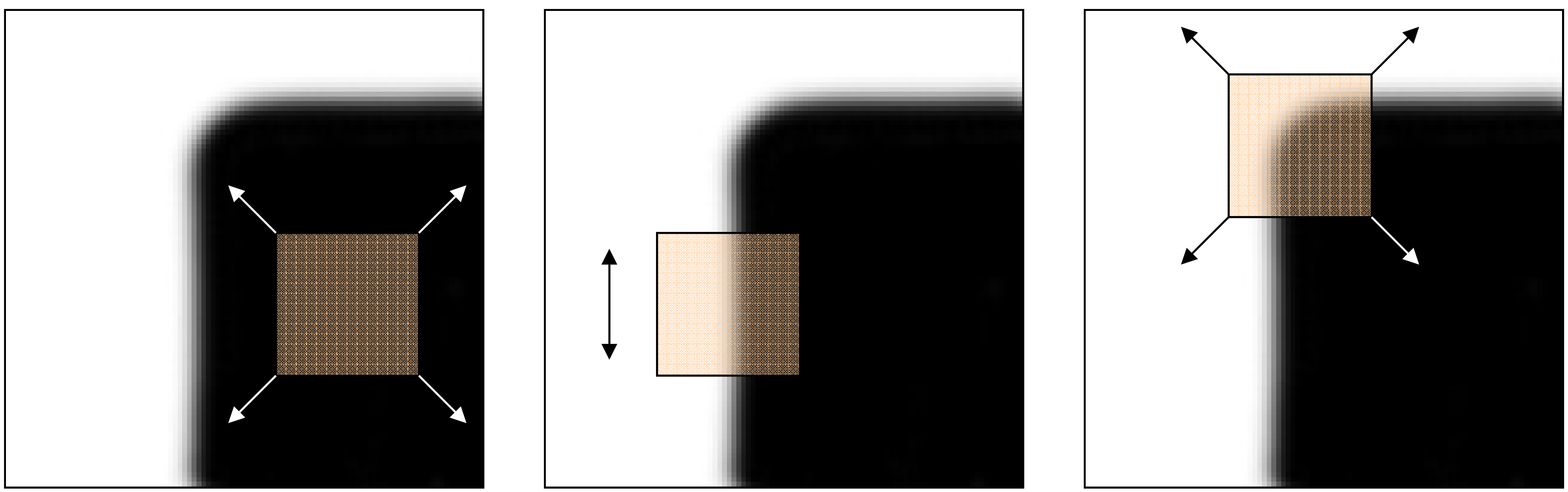
- Repeatability
  - The same feature can be found in several images despite geometric and photometric transformations
- Saliency
  - Each feature is distinctive
- Compactness and efficiency
  - Many fewer features than image pixels
- Locality
  - A feature occupies a relatively small area of the image; robust to clutter and occlusion

Applications

- Feature points are used for:
  - Image alignment
  - 3D reconstruction
  - Motion tracking
  - Robot navigation
  - Indexing and database retrieval
  - Object recognition

Corner Detection: Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region: no change in all directions

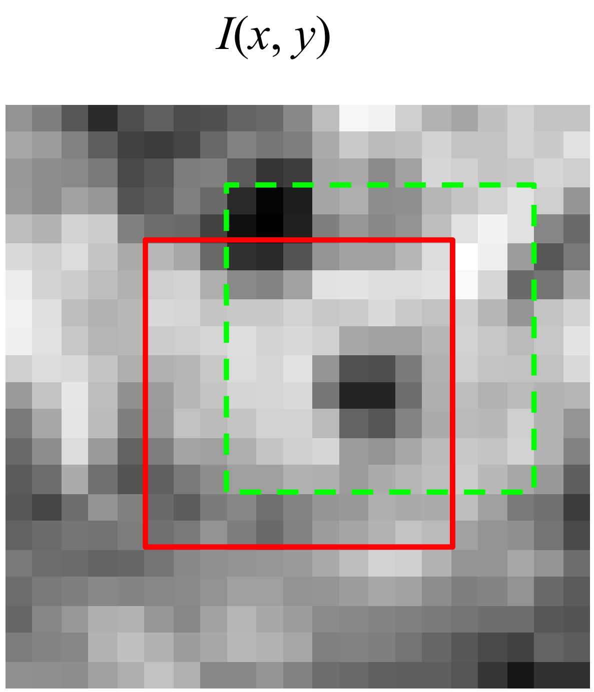
"edge": no change along the edge direction

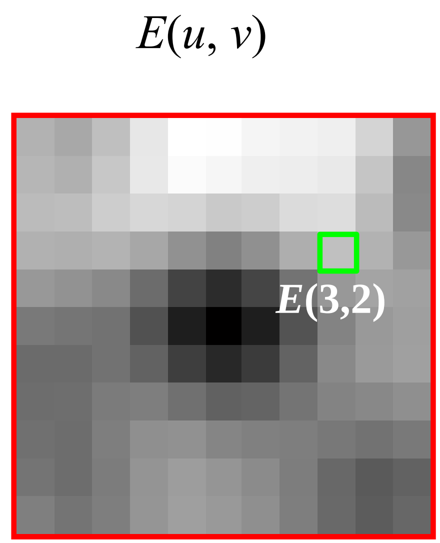
"corner": significant change in all directions

Corner Detection: Mathematics

Change in appearance of window W for a shift $[u,v]$:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

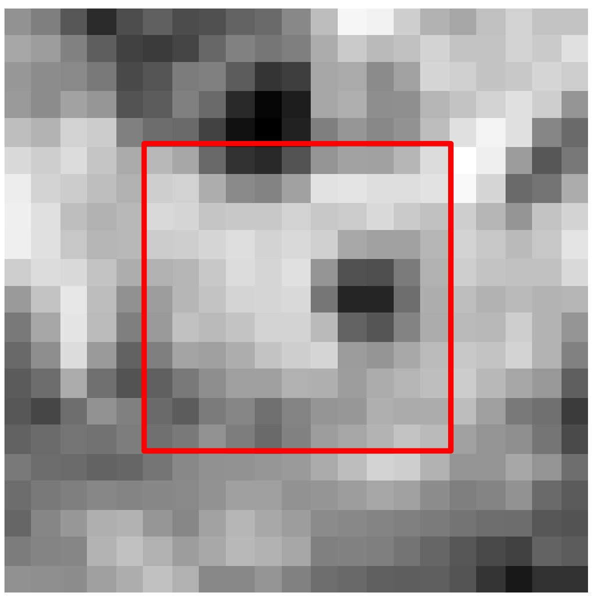


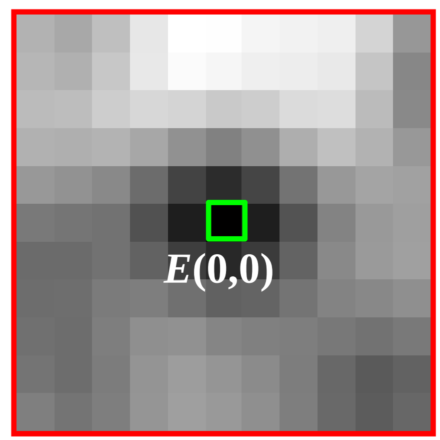


Corner Detection: Mathematics

Change in appearance of window W for a shift $[u,v]$:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$





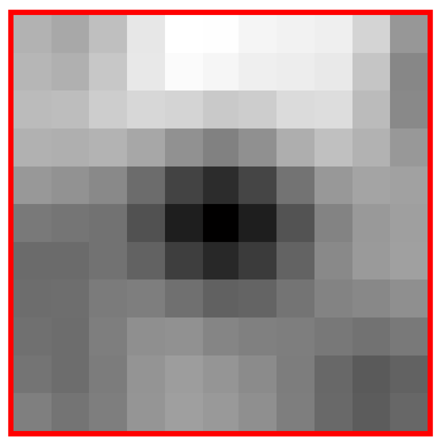
Corner Detection: Mathematics

Change in appearance of window W for a shift $[u,v]$:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

We want to find out how this function behaves for small shifts

$E(u, v)$



Corner Detection: Mathematics

- First-order Taylor approximation for small motions [*u*, *v*]:

$$I(x+u, y+v) \approx I(x,y) + I_x u + I_y v$$

- Let's plug this into $E(u,v)$:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

$$\approx \sum_{(x,y)\in W} [I(x,y) + I_x u + I_y v - I(x,y)]^2$$

$$= \sum_{(x,y)\in W} [I_x u + I_y v]^2 = \sum_{(x,y)\in W} I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2$$

Corner Detection: Mathematics

The quadratic approximation can be written as

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a *second moment matrix* computed from image derivatives:

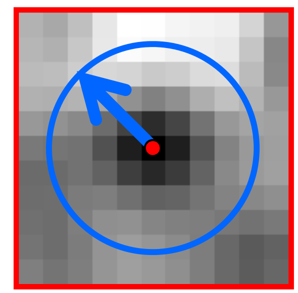
$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix}$$

(the sums are over all the pixels in the window W)

Interpreting the second moment matrix

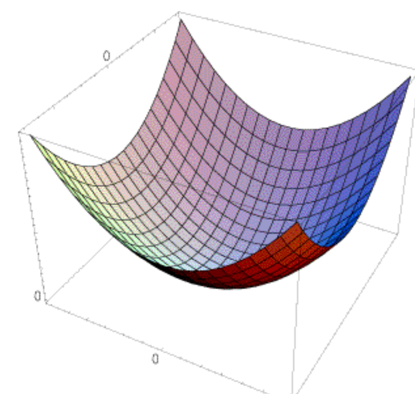
- The surface $E(u,v)$ is locally approximated by a quadratic form. Let's try to understand its shape.
  - Specifically, in which directions does it have the smallest/greatest change?

$E(u, v)$



$$E(u,v) \approx [u \ \ v] \ M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix}$$



Interpreting the second moment matrix

First, consider the axis-aligned case (gradients are either horizontal or vertical)

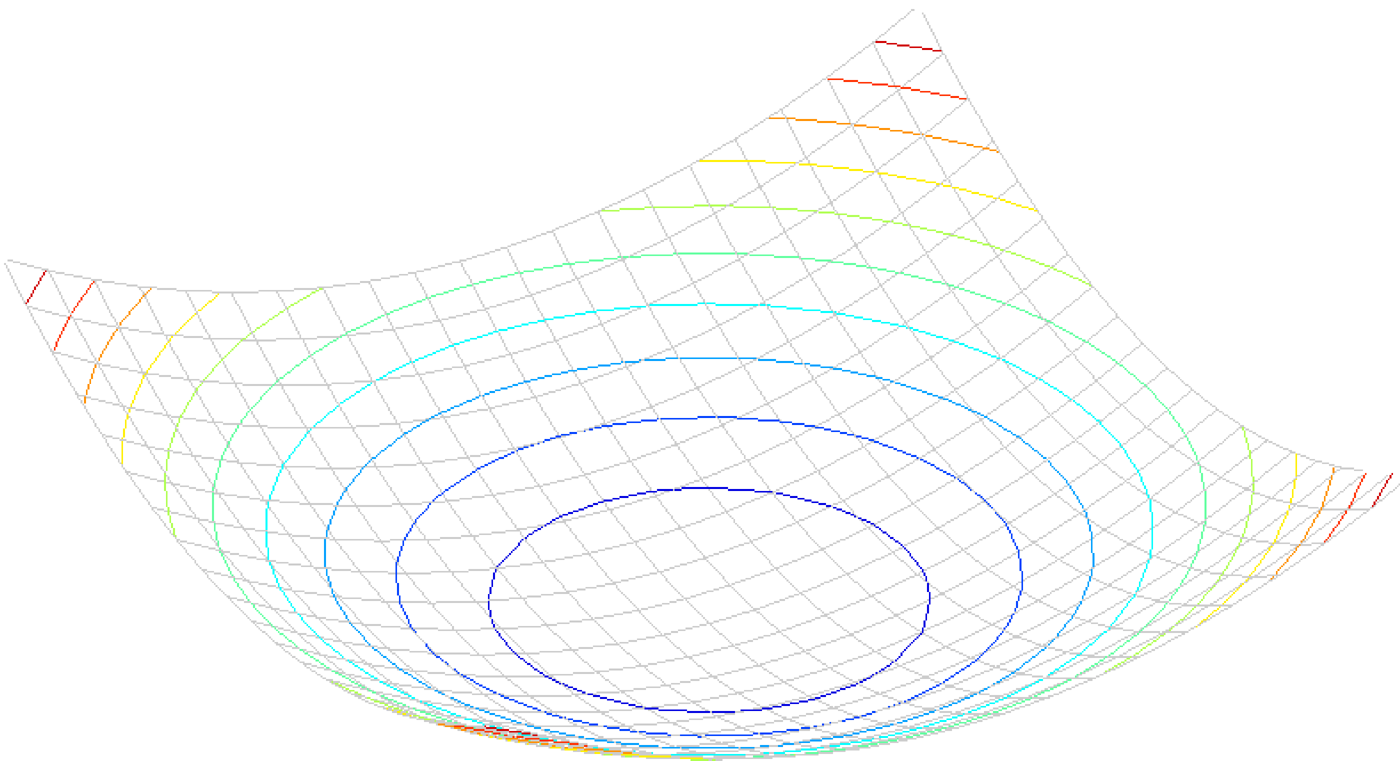
$$M = \begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

If either a or b is close to 0, then this is **not** a corner, so look for locations where both are large.

Interpreting the second moment matrix

Consider a horizontal "slice" of $E(u, v)$: $$[u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$

This is the equation of an ellipse.



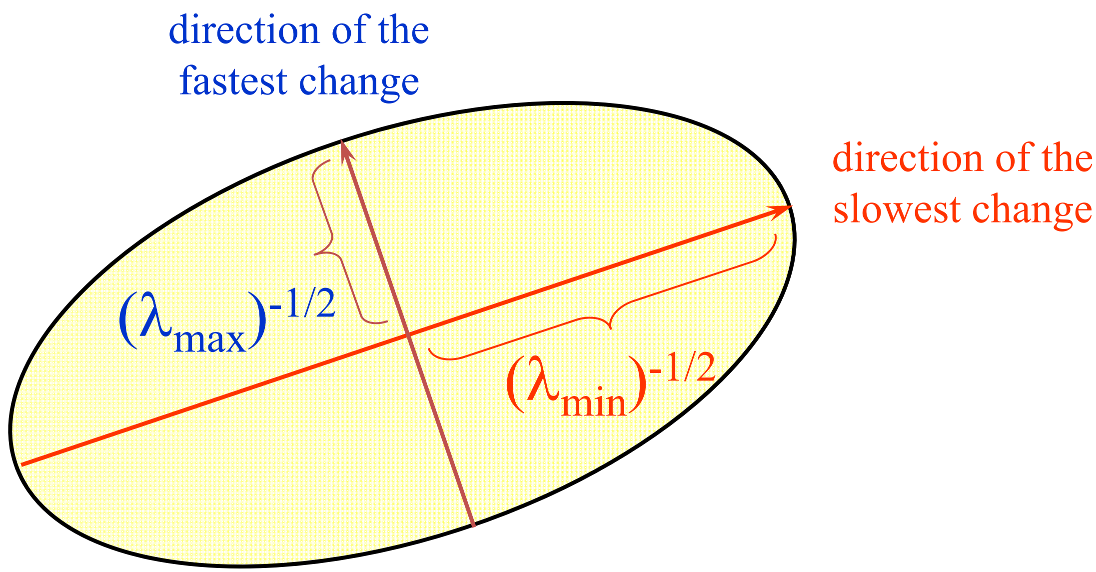
Interpreting the second moment matrix

Consider a horizontal "slice" of $E(u, v)$: $$[u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$

This is the equation of an ellipse.

Diagonalization of M: $$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

The axis lengths of the ellipse are determined by the eigenvalues and the orientation is determined by R



Quick Eigenvalue/Eigenvector Review

The **eigenvectors** of a matrix A are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

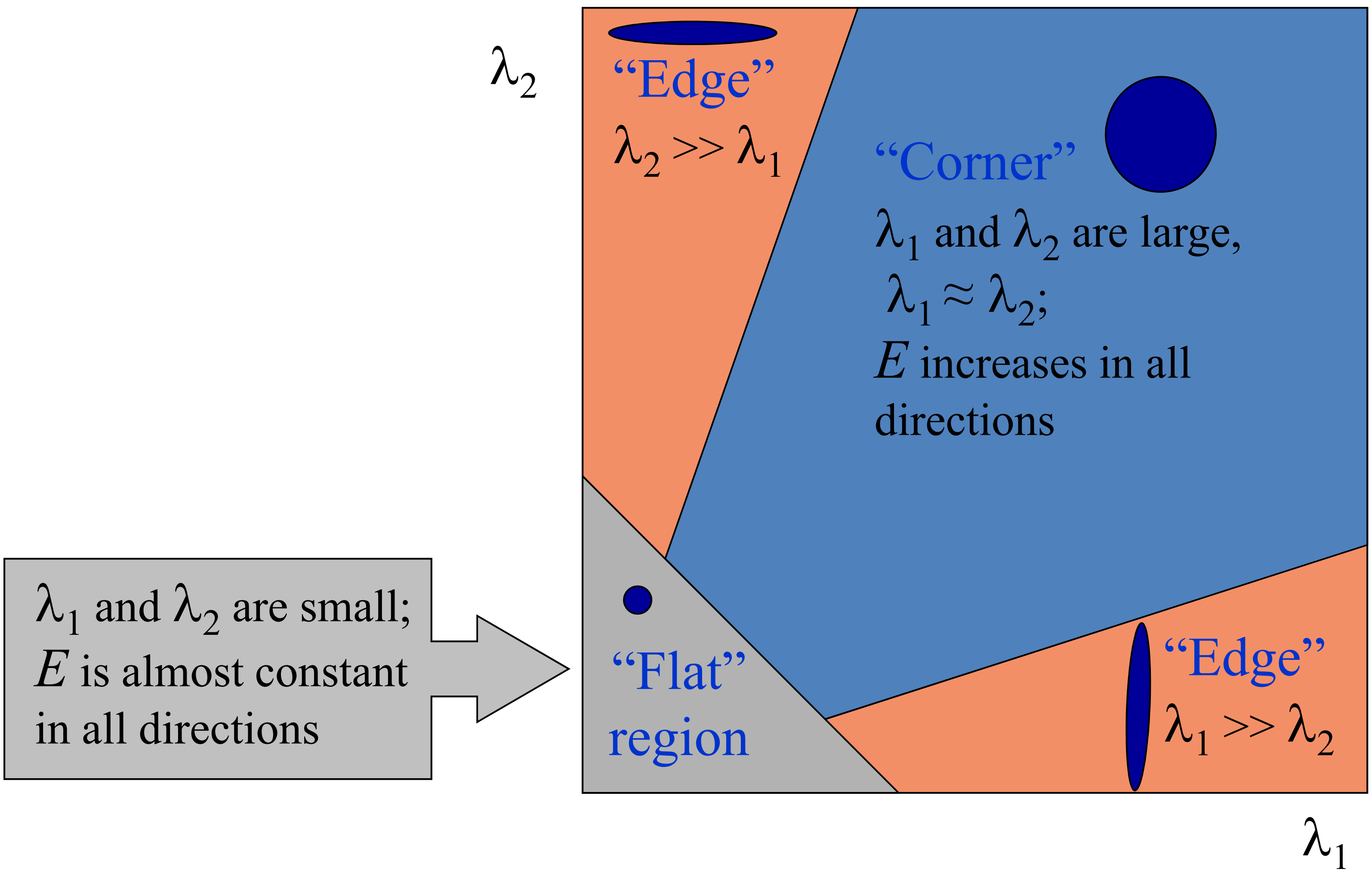
- The eigenvalues are found by solving: $$det(A - \lambda I) = 0$$
- In our case, $\boldsymbol{A = H}$ is a 2x2 matrix, so we have $$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$
- The solution: $$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Interpreting the eigenvalues

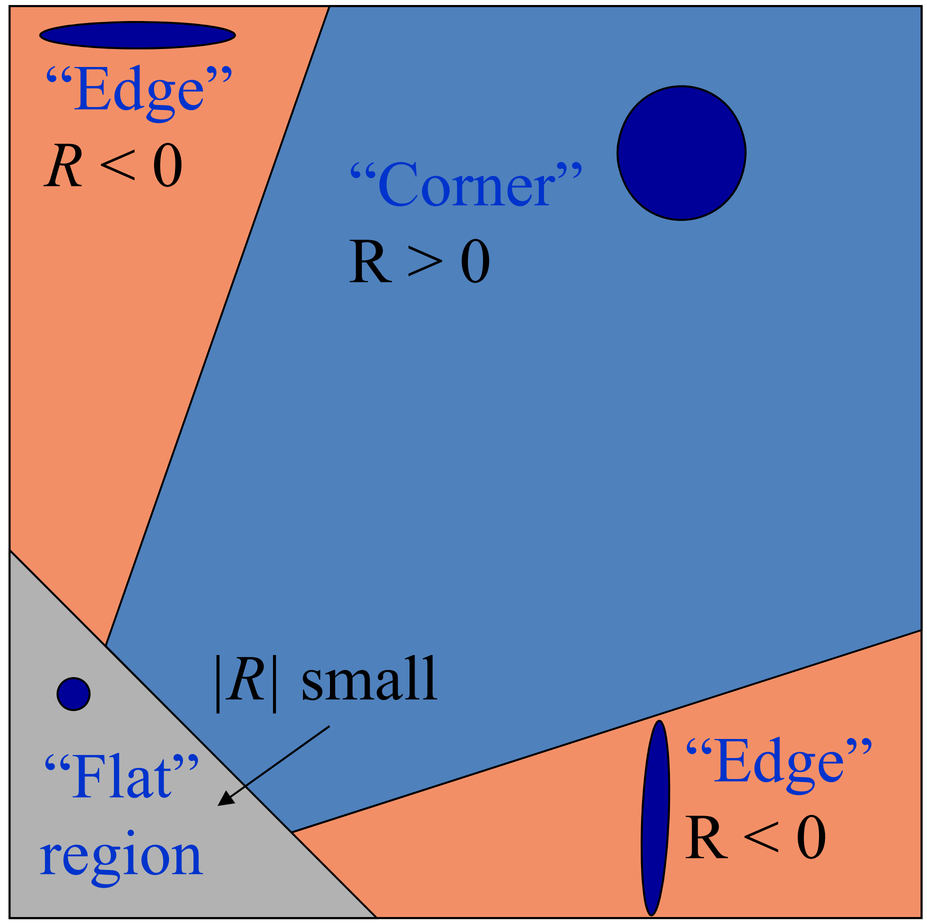
Classification of image points using eigenvalues of *M*:



Corner response function

$$R = \det(M) - \alpha \, \mathrm{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

α: constant (0.04 to 0.06)



The Harris corner detector

1. Compute partial derivatives at each pixel
2. Compute second moment matrix *M* in a Gaussian window around each pixel:

$$M = \begin{bmatrix} \sum_{x,y} w(x,y) I_x^2 & \sum_{x,y} w(x,y) I_x I_y \\ \sum_{x,y} w(x,y) I_x I_y & \sum_{x,y} w(x,y) I_y^2 \end{bmatrix}$$

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

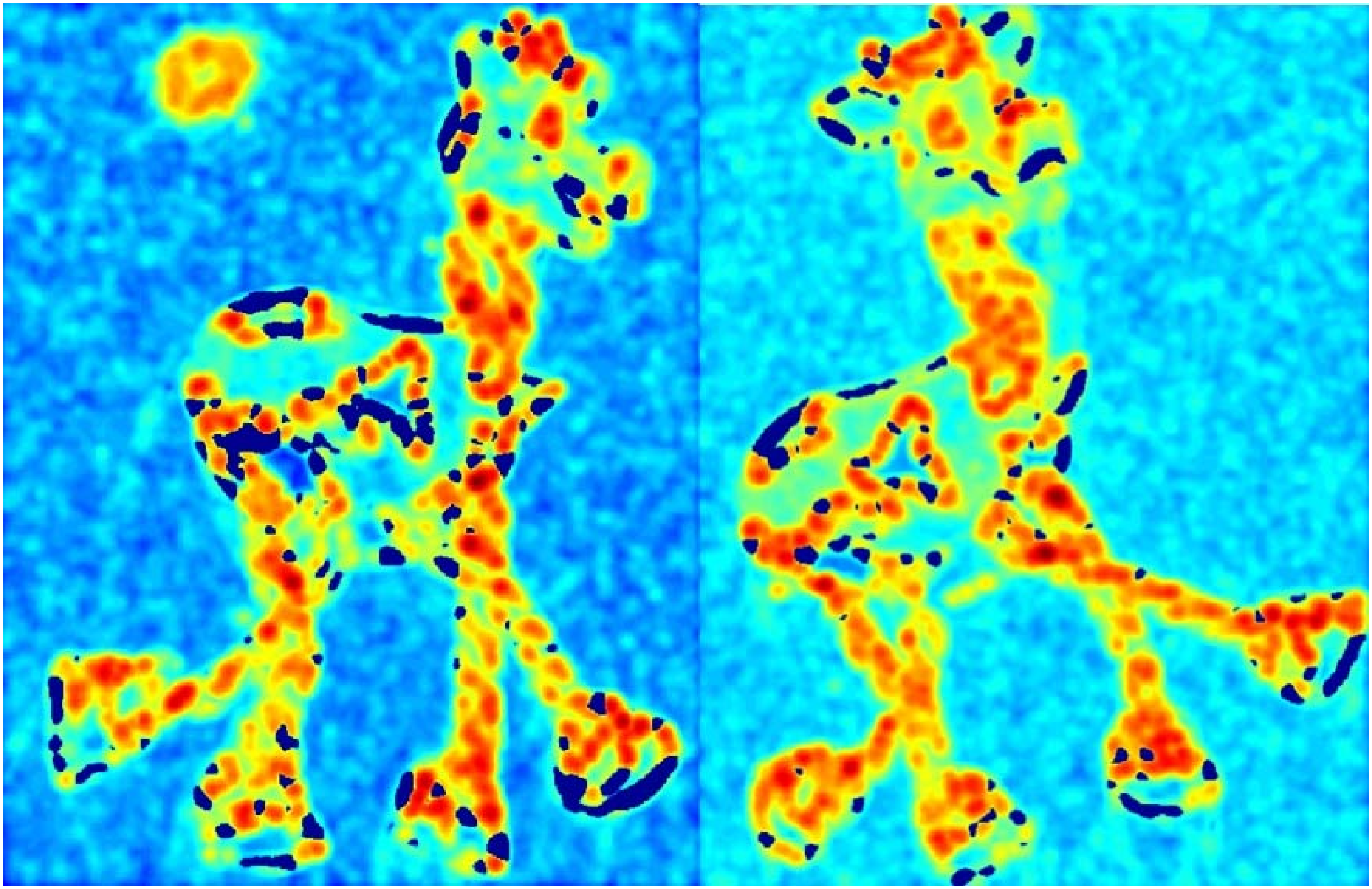
The Harris corner detector

1. Compute partial derivatives at each pixel
2. Compute second moment matrix M in a Gaussian window around each pixel
3. Compute corner response function R

Harris Detector: Steps

Harris Detector: Steps

Compute corner response R

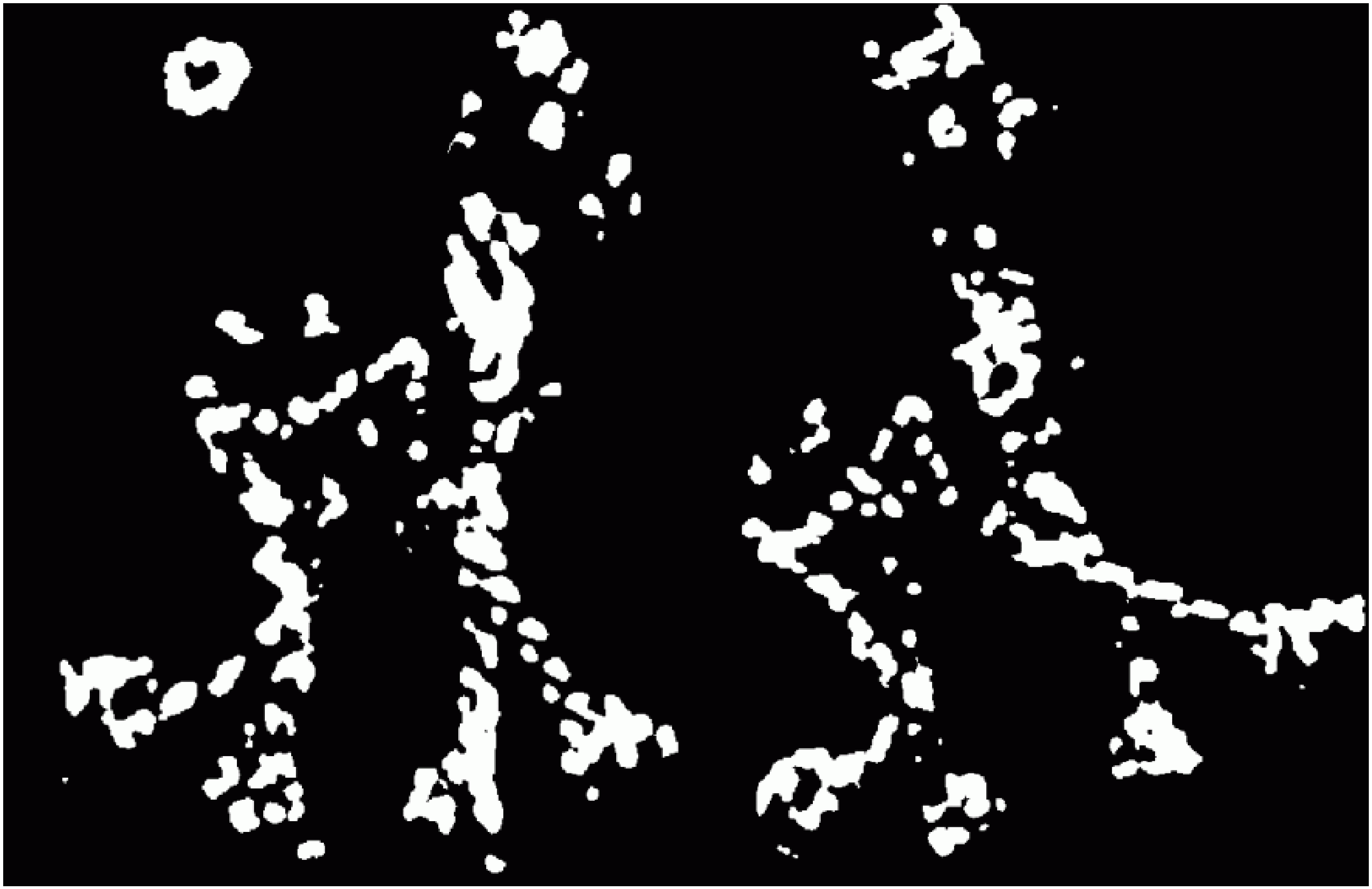


The Harris corner detector

1. Compute partial derivatives at each pixel
2. Compute second moment matrix M in a Gaussian window around each pixel
3. Compute corner response function R
4. Threshold R
5. Find local maxima of response function (non-maximum suppression)

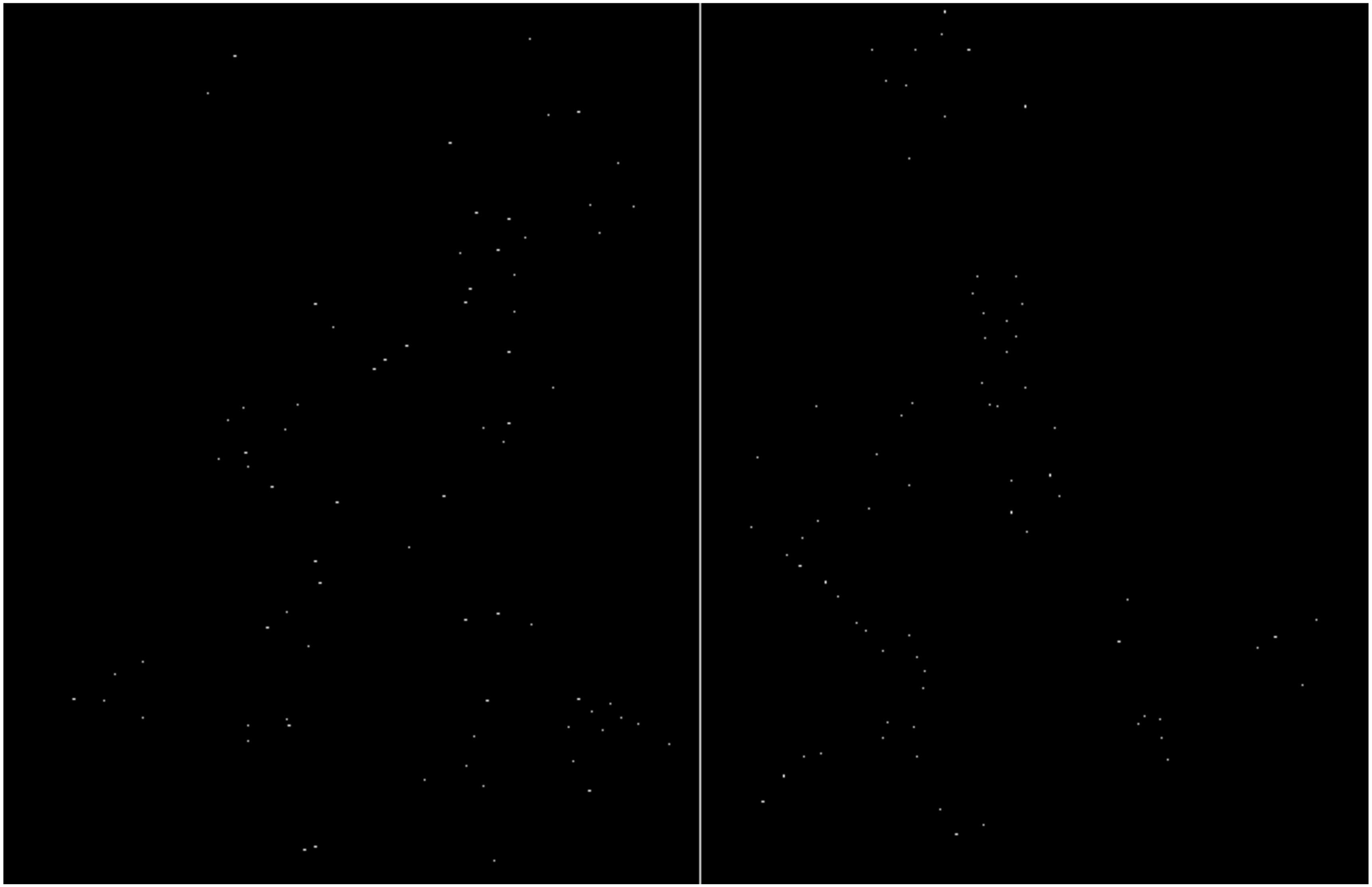
Harris Detector: Steps

Find points with large corner response: $R >$ threshold



Harris Detector: Steps

Take only the points of local maxima of *R*



Harris Detector: Steps