Stitching and Blending

Kari Pulli

Senior Director

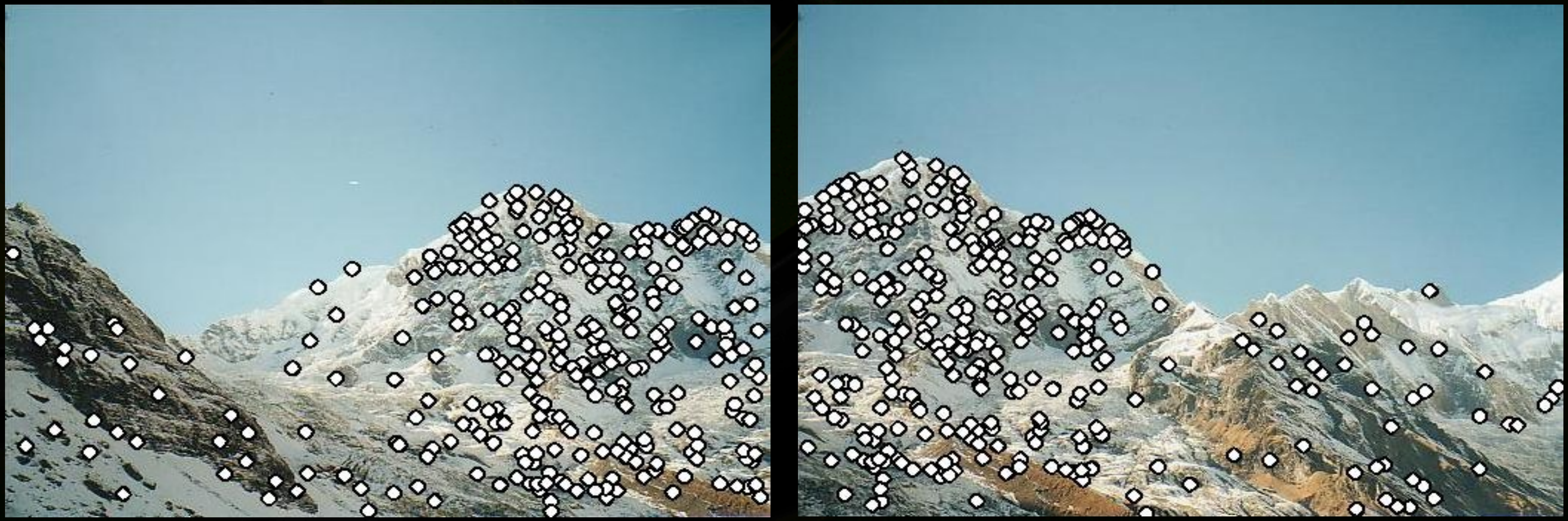
NVIDIA Research

First project

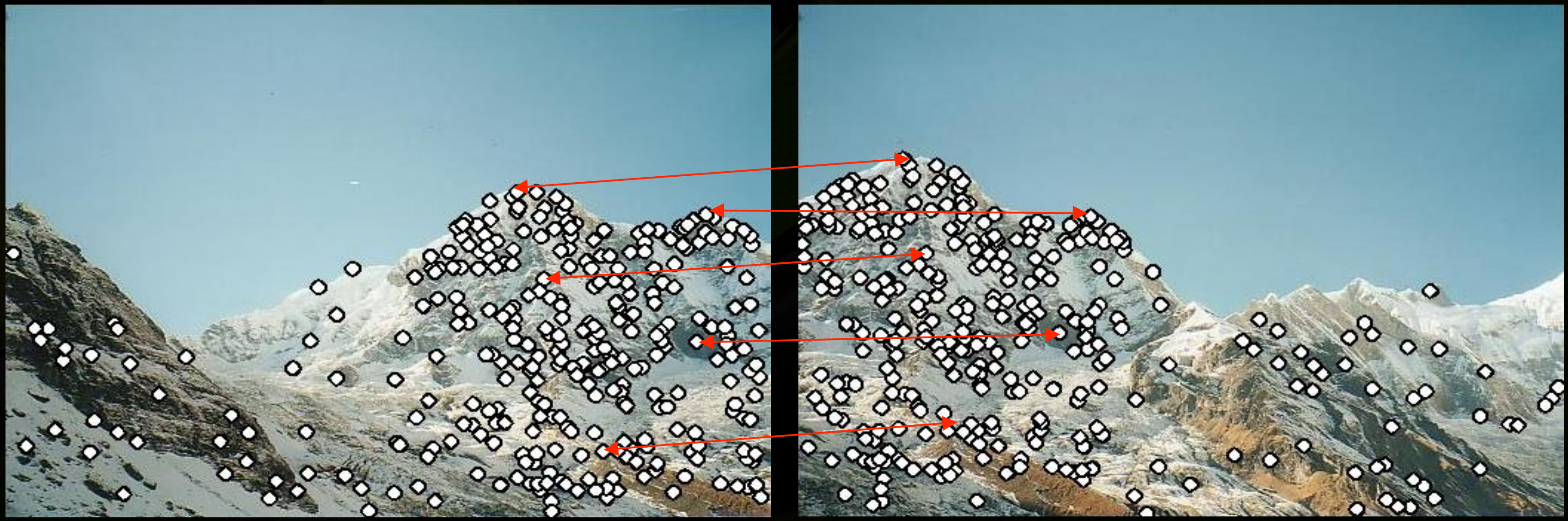
- Build your own (basic) programs
  - panorama
  - HDR (really, exposure fusion)
- The key components
  - register images so their features align
  - determine overlap
  - blend

We need to match (align) images

Detect feature points in both images



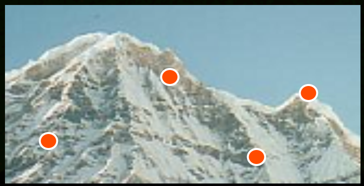
Find corresponding pairs



Use these pairs to align images

Matching with Features

- **Problem 1:**
  - **Detect the *same* point *independently* in both images**

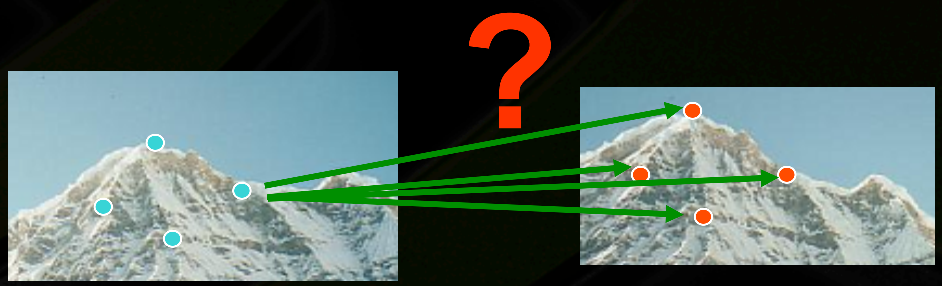


no chance to match!

We need a repeatable detector

Matching with Features

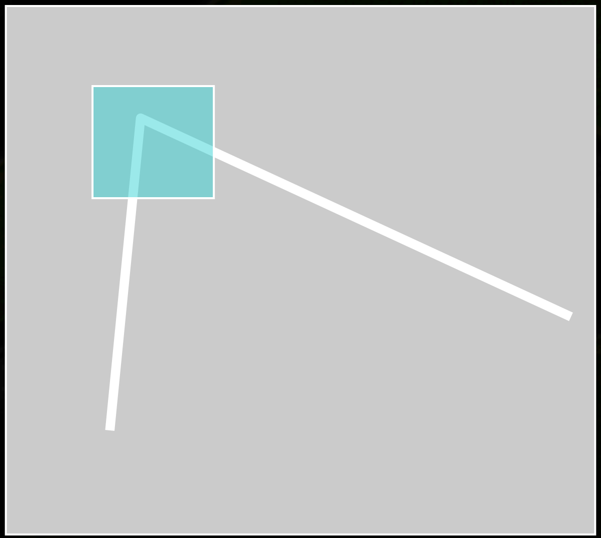
- **Problem 2:**
  - **For each point correctly recognize the corresponding one**



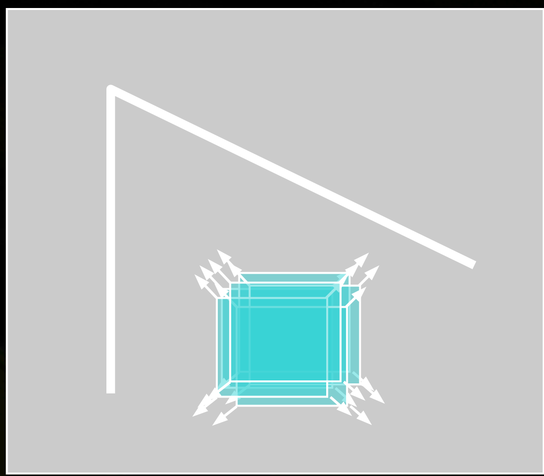
We need a reliable and distinctive descriptor

Harris Corners: The Basic Idea

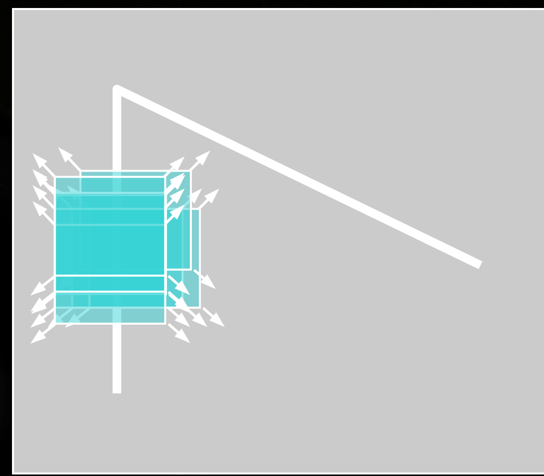
- We should easily recognize the point by looking through a small window
- Shifting a window in any direction should give a large change in intensity



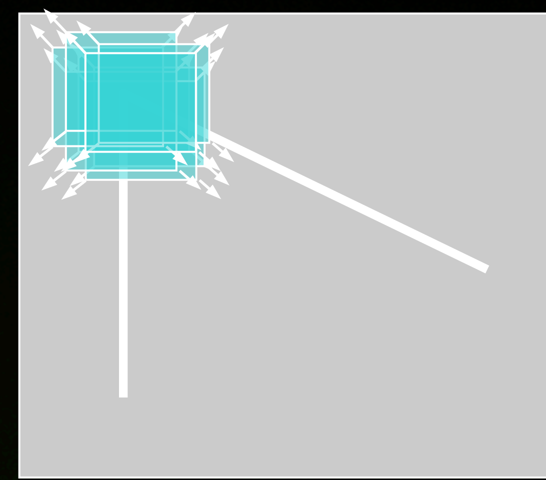
Harris Detector: Basic Idea



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

Harris Detector: Mathematics

Window-averaged change of intensity for the shift [*u,v*]:

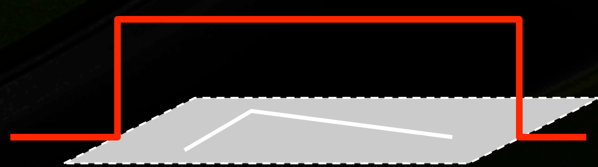
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function

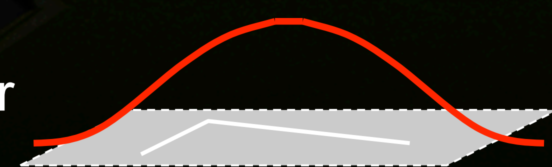
Shifted intensity

Intensity

Window function $w(x,y)$ =



or



1 in window, 0 outside

Gaussian

Harris Detector: Mathematics

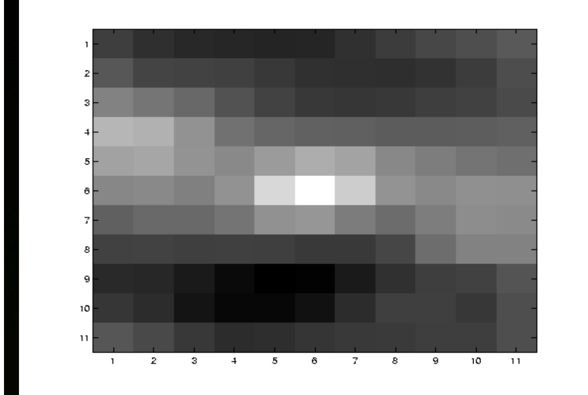
Expanding *E(u,v)* in a 2nd order Taylor series expansion, we have, for small shifts [*u,v*], a *bilinear* approximation:

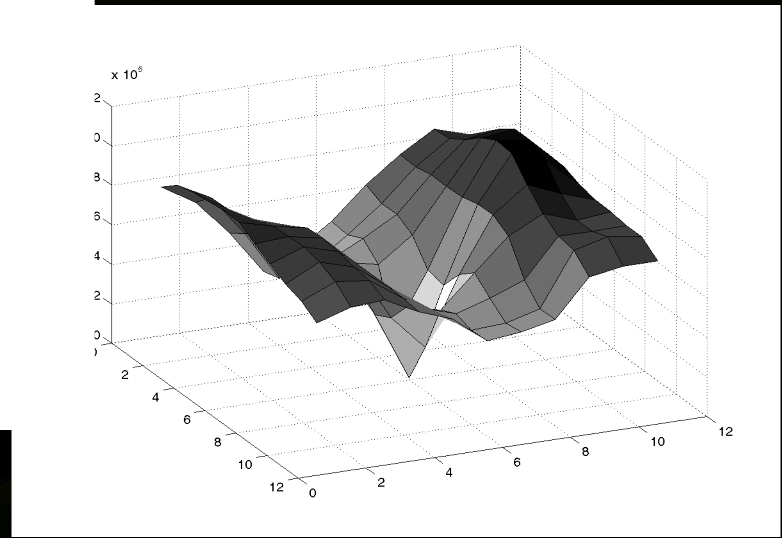
$$E(u,v) \cong [u,v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

where *M* is a 2×2 matrix computed from image derivatives:

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

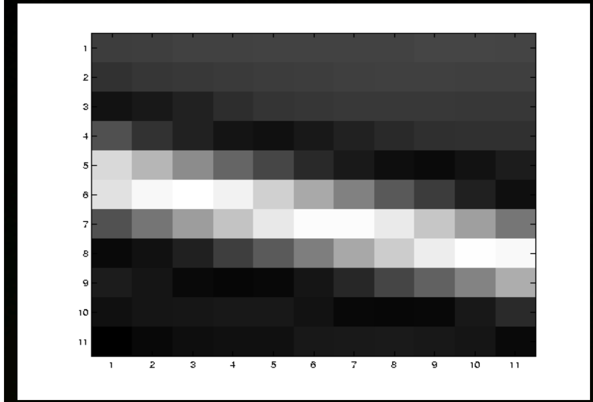
Eigenvalues λ_1, λ_2 of M at different locations

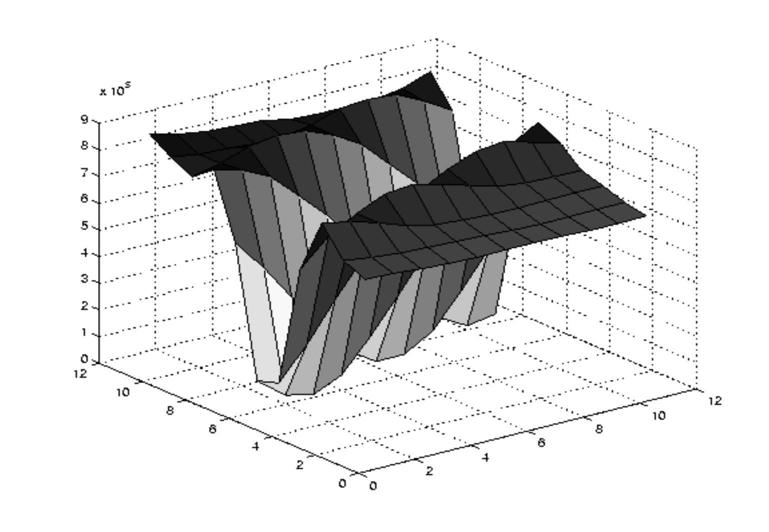




λ_1 and λ_2 are large

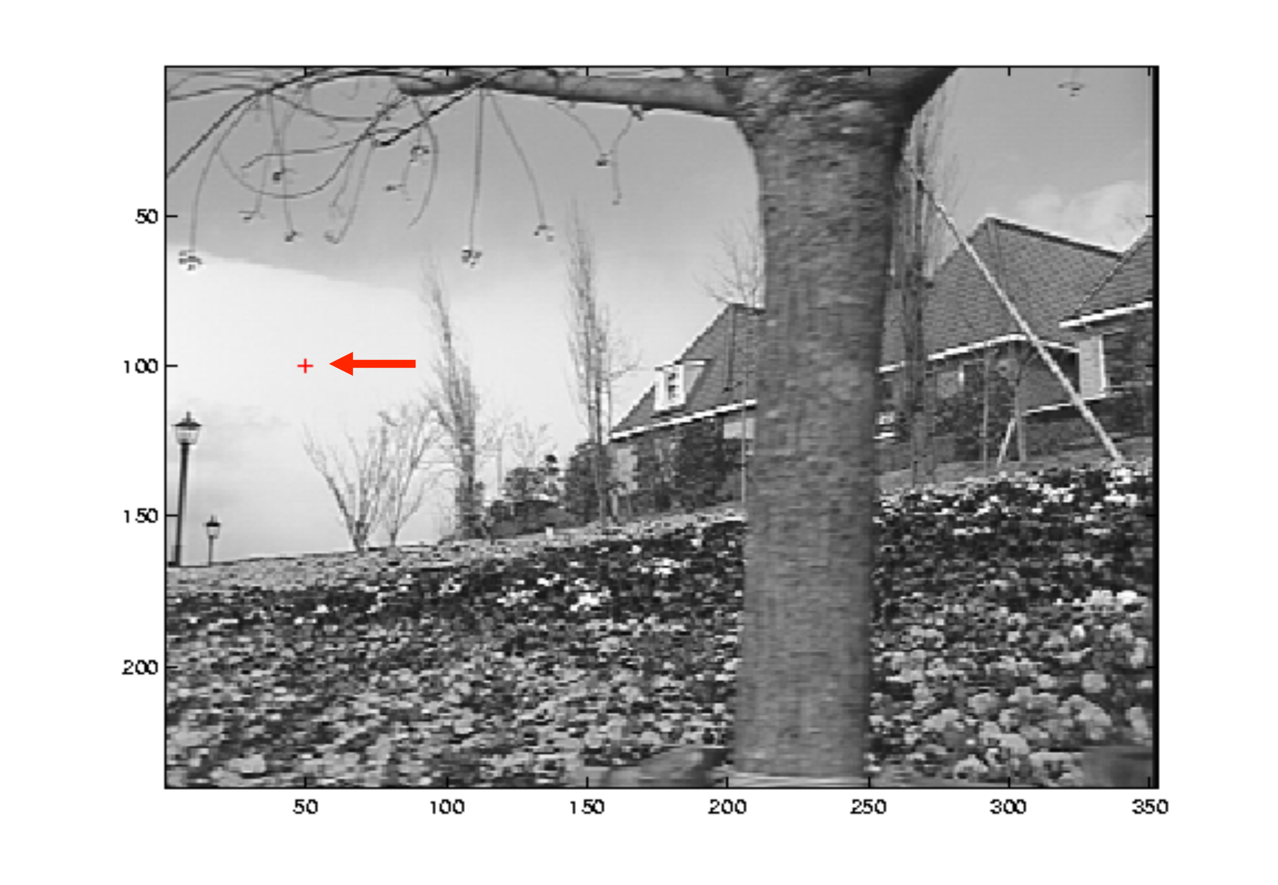
Eigenvalues λ_1, λ_2 of M at different locations

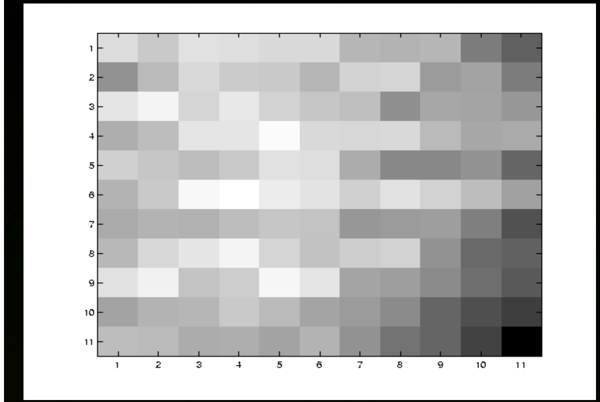


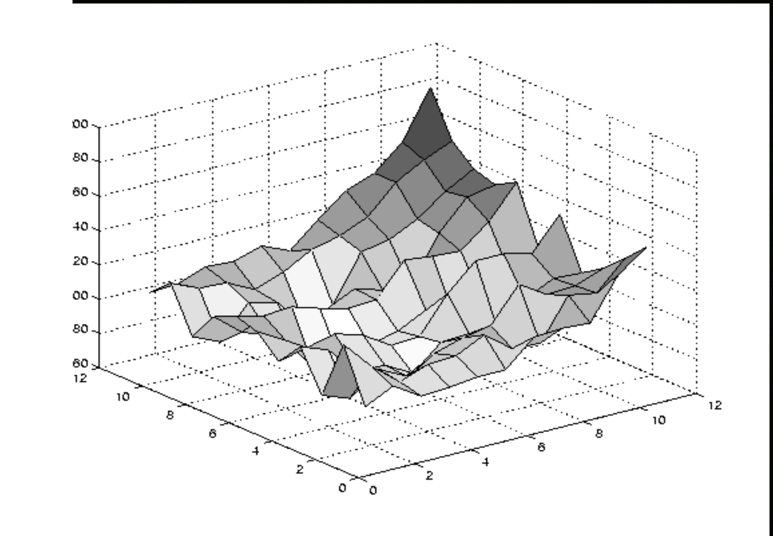


large λ_1, small λ_2

Eigenvalues λ_1, λ_2 of M at different locations



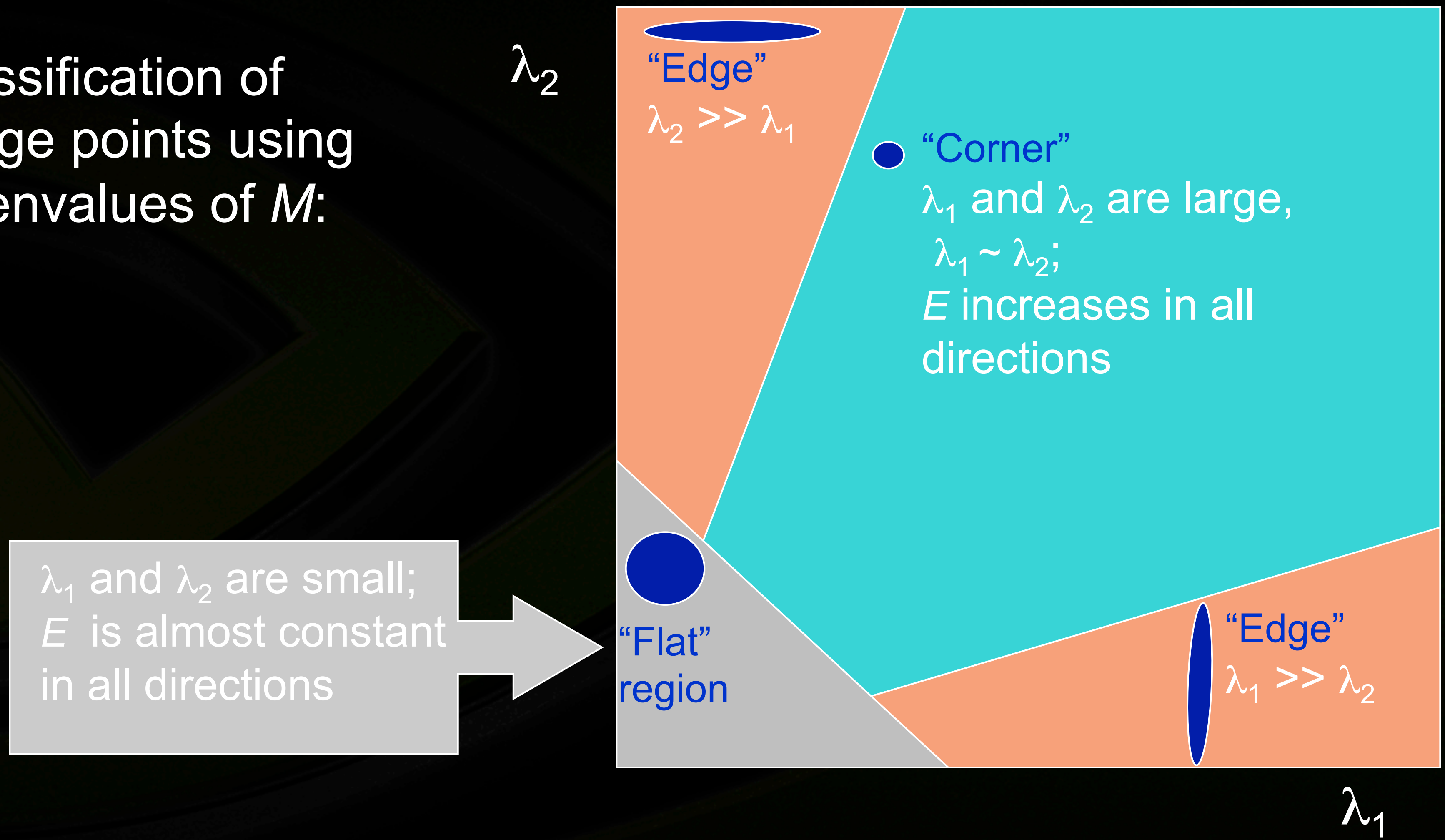




small λ_1, small λ_2

Harris Detector: Mathematics

Classification of image points using eigenvalues of *M*:



Harris Detector: Mathematics

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

Measure of corner response:

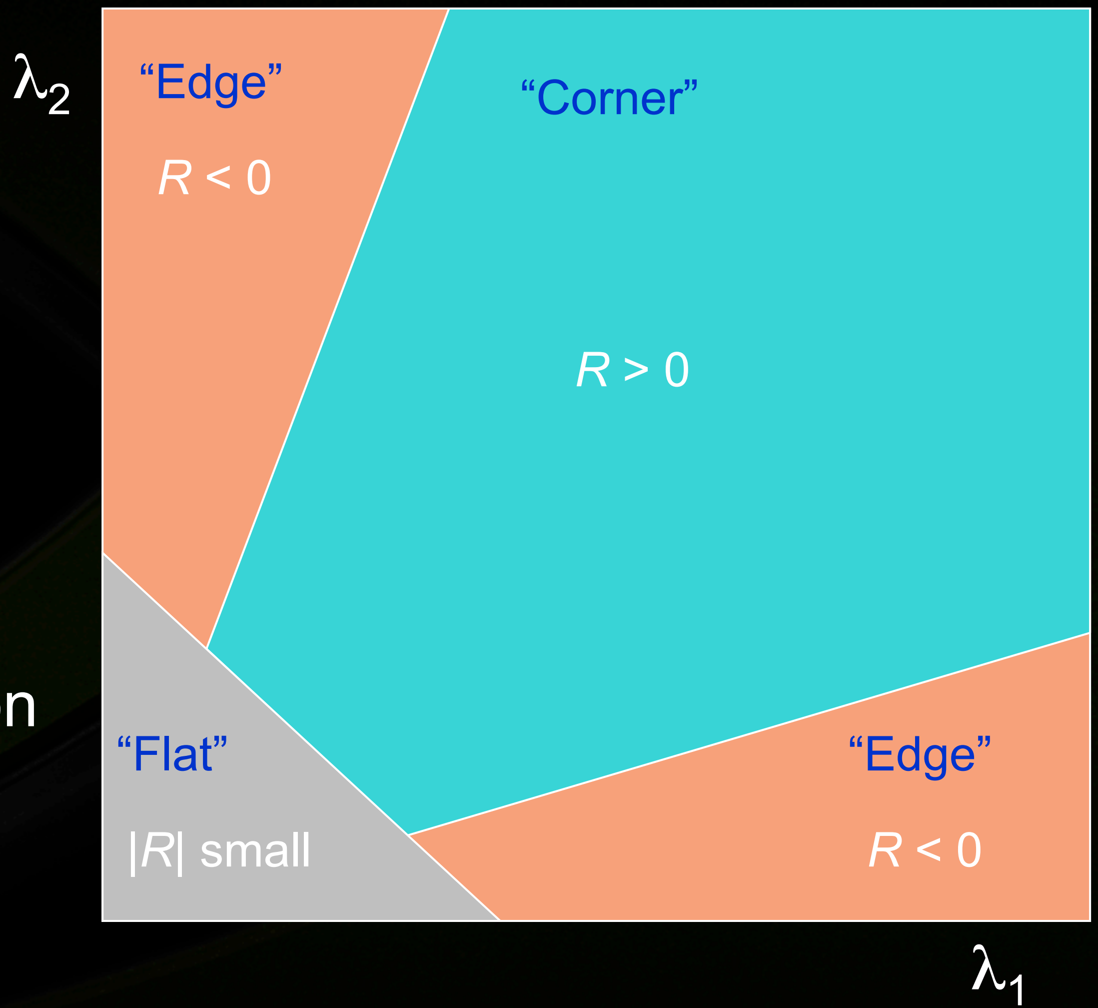
$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

(k – empirical constant, k = 0.04 - 0.06)

Harris Detector: Mathematics

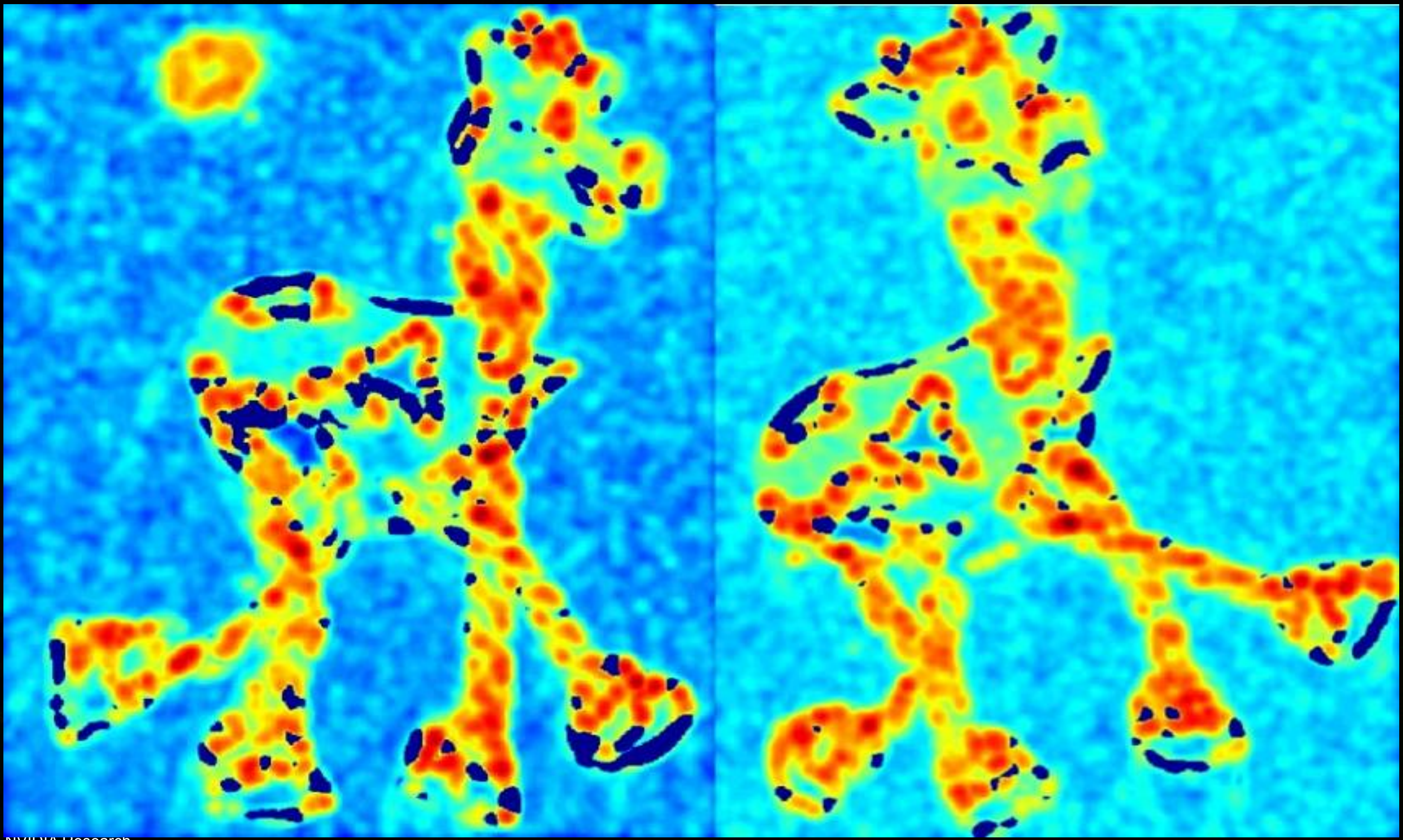
- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



Harris Detector: Workflow

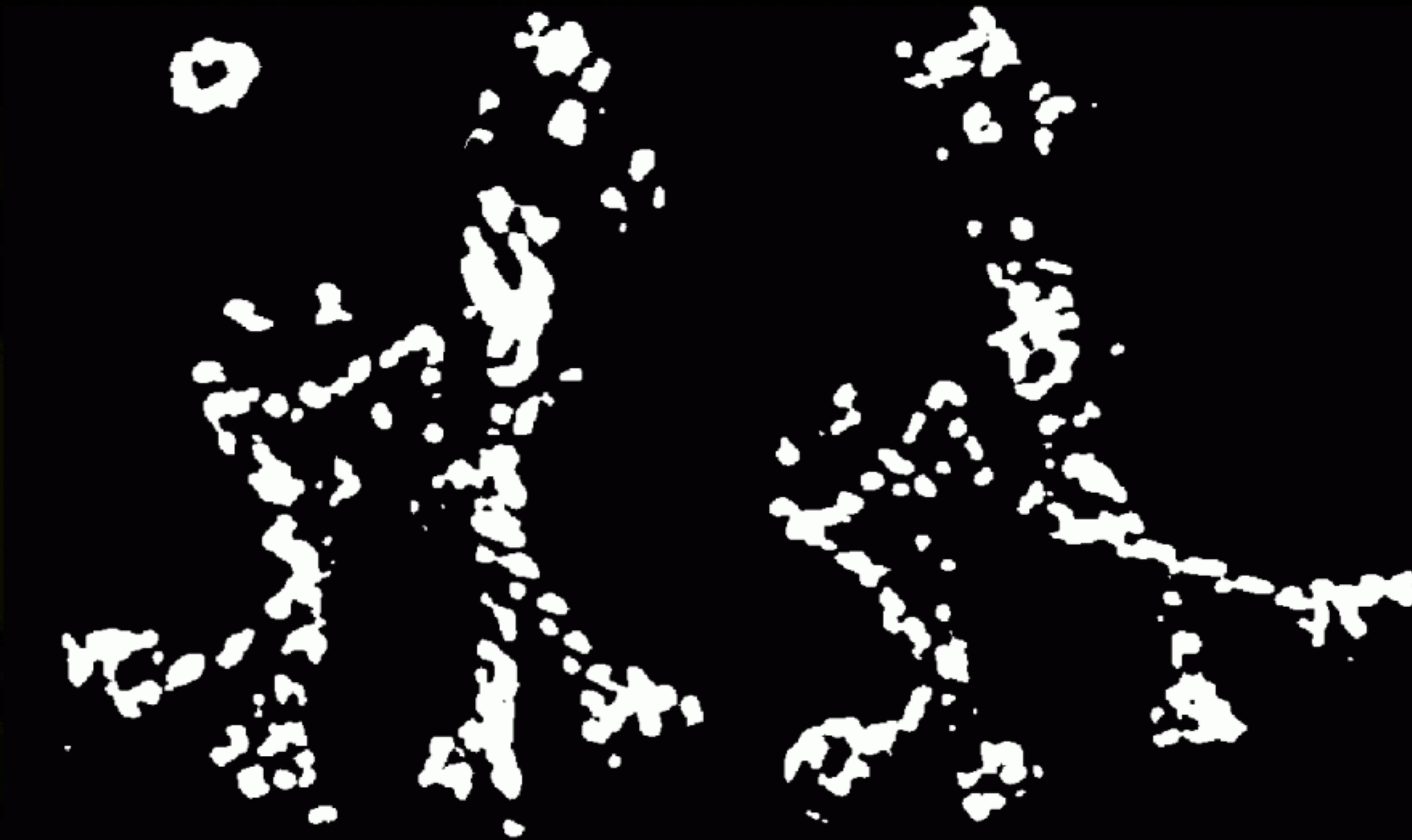
Harris Detector: Workflow

Compute corner response R



Harris Detector: Workflow

Find points with large corner response: R > threshold



Harris Detector: Workflow

Take only the points of local maxima of R

Harris Detector: Workflow

Harris Detector: Summary

- Average intensity change in direction [*u*,*v*] can be expressed as a bilinear form:

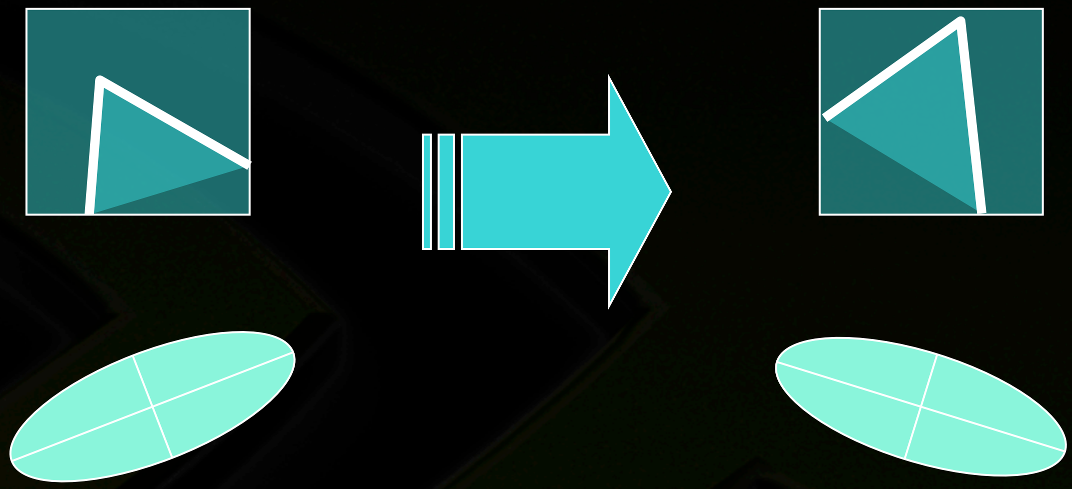
$$E(u,v) \cong [u,v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

- Describe a point in terms of eigenvalues of *M*: *measure of corner response*

$$R = \lambda_1 \lambda_2 - k\left(\lambda_1 + \lambda_2\right)^2$$

- A good (corner) point should have a *large intensity change* in *all directions*, i.e., *R* should be large positive

Harris Detector: Invariant to rotation



Ellipse rotates
but its shape (i.e., eigenvalues) remains the same

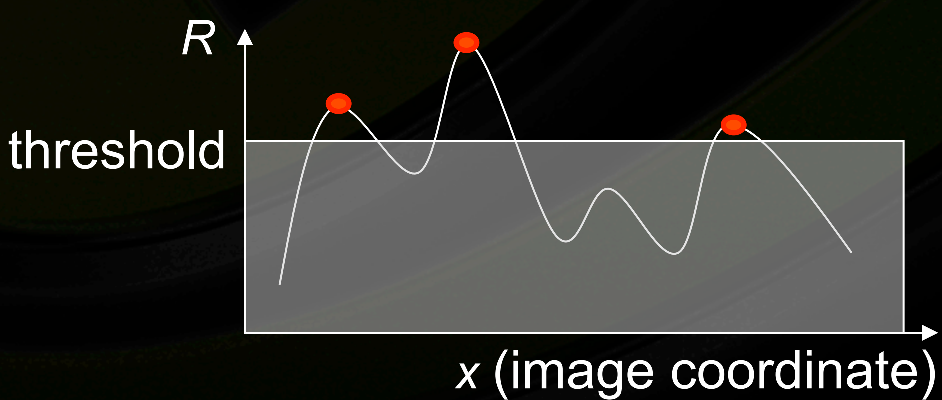
Corner response R is invariant to image rotation

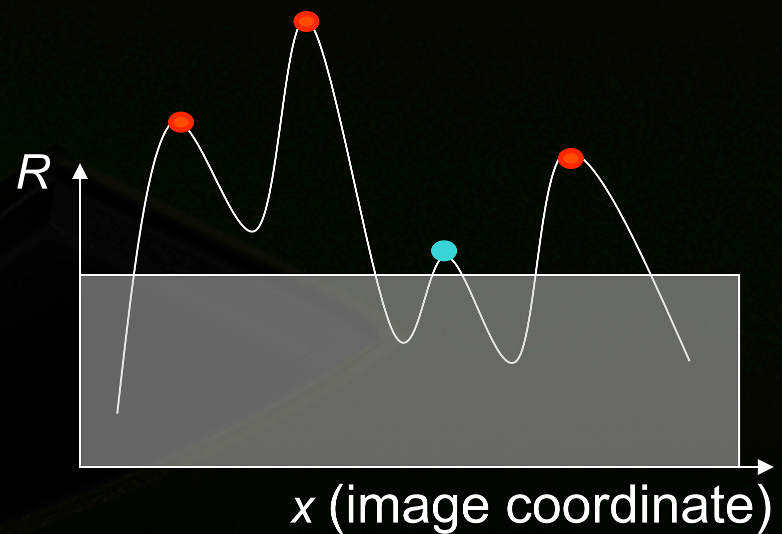
Almost invariant to intensity change

- **Partial invariance**
  - ✓ Only derivatives are used

    =>

    invariance to intensity shift $I \rightarrow I + b$
  - ✓ Intensity scale: $I \rightarrow a\, I$





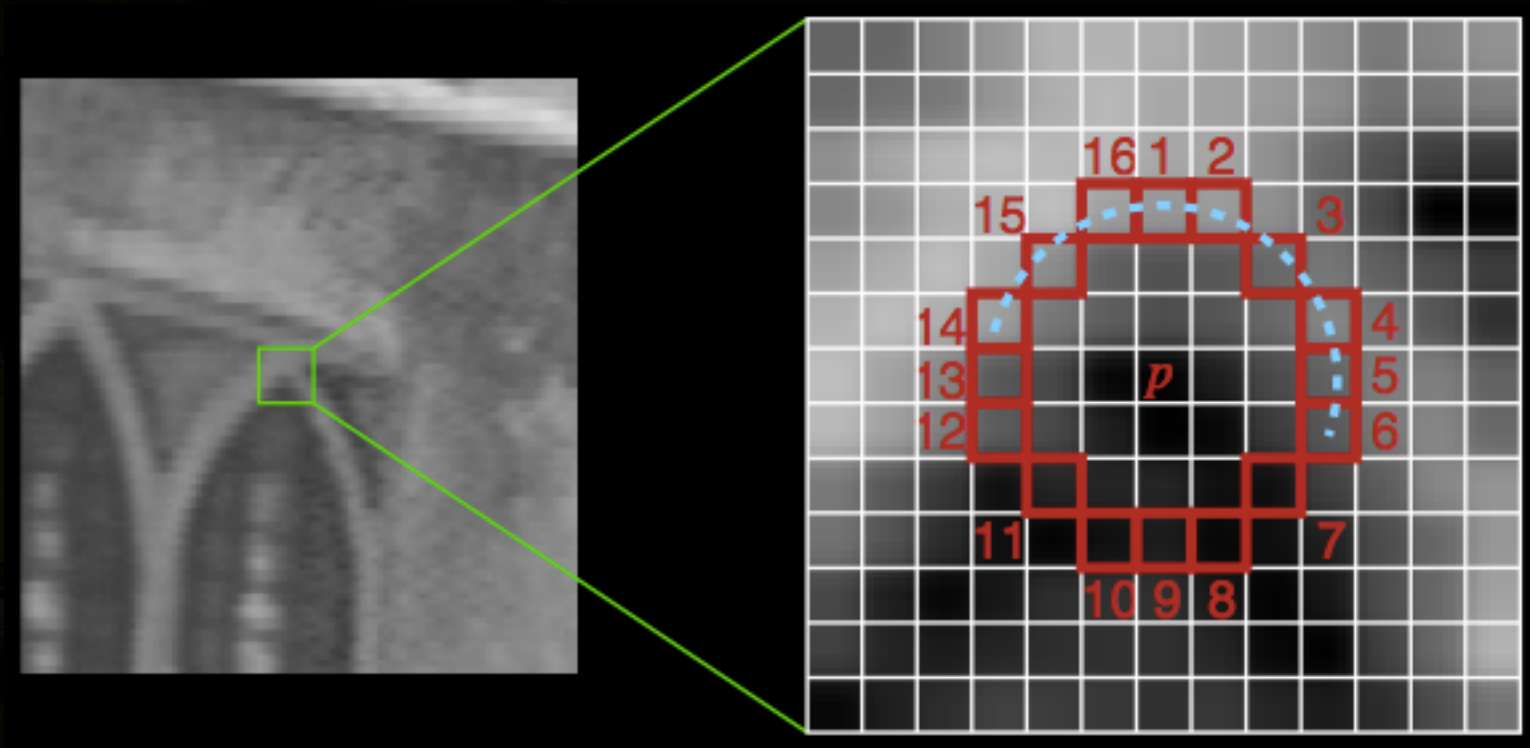
Not invariant to image scale!

All points will be classified as edges

Corner !

FAST Corners

- Look for a contiguous arc of N pixels
  - all much darker (or brighter) than the central pixel *p*

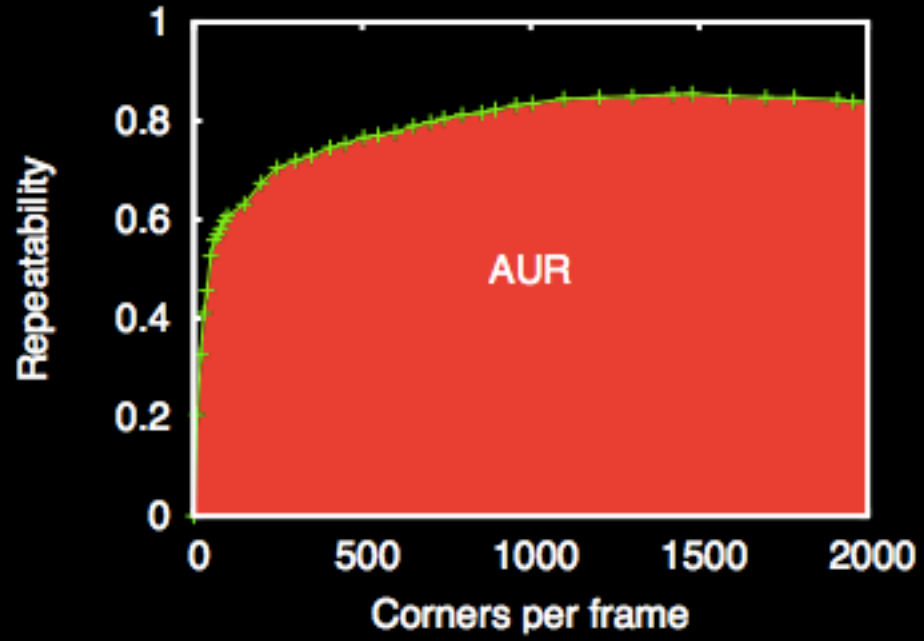


How FAST?

Detector	Set 1 Pixel rate (MPix/s)	Set 1 %	Set 2 MPix/s	Set 2 %
FAST $n = 9$	188	4.90	179	5.15
FAST $n = 12$	158	5.88	154	5.98
Original FAST ($n = 12$)	79.0	11.7	82.2	11.2
FAST-ER	75.4	12.2	67.5	13.7
SUSAN	12.3	74.7	13.6	67.9
Harris	8.05	115	7.90	117
Shi-Tomasi	6.50	142	6.50	142
DoG	4.72	195	5.10	179

How repeatable?

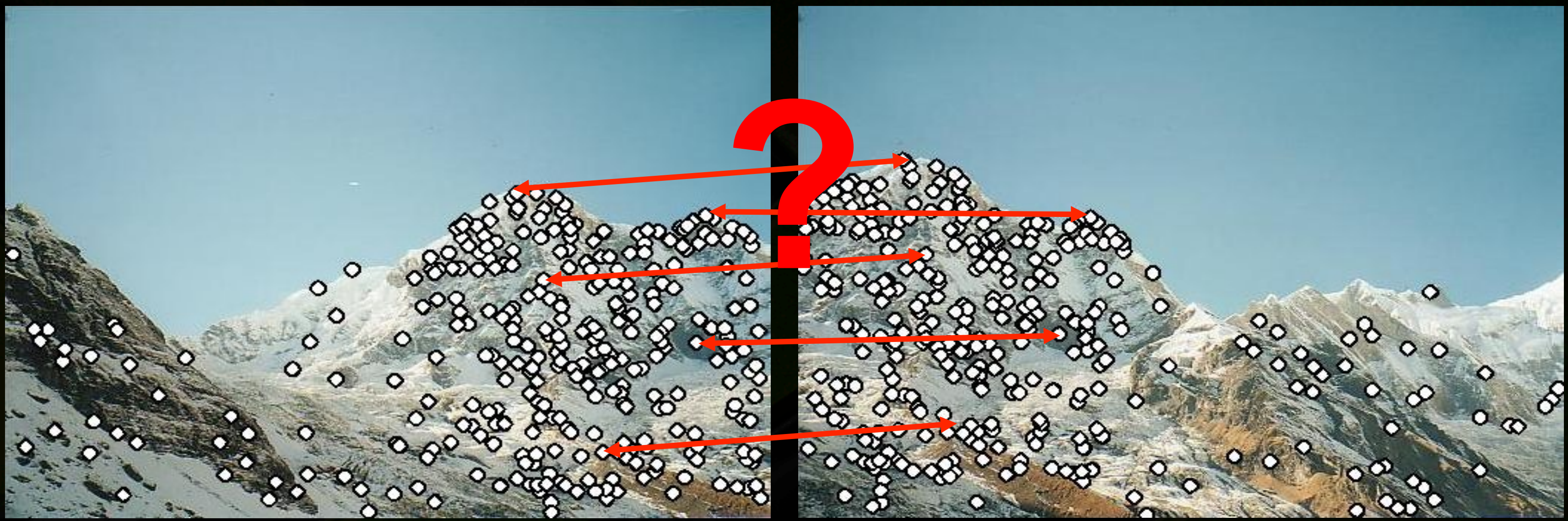
Detector	AUR
FAST-ER	1313.6
FAST-9	1304.57
DoG	1275.59
Shi & Tomasi	1219.08
Harris	1195.2
Harris-Laplace	1153.13
FAST-12	1121.53
SUSAN	1116.79
Random	271.73



Point Descriptors

- We know how to detect points
- Next question:

How to match them?



Point descriptor should be:

1. Invariant
2. Distinctive

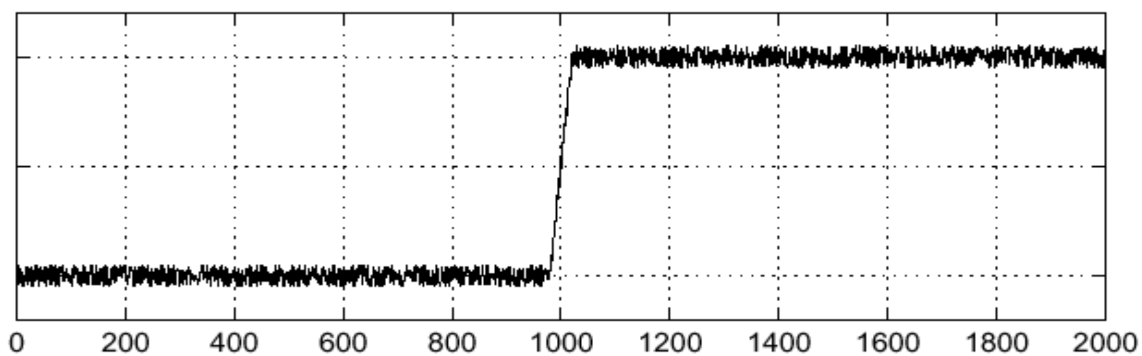
SIFT – Scale Invariant Feature Transform

Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affi ne transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

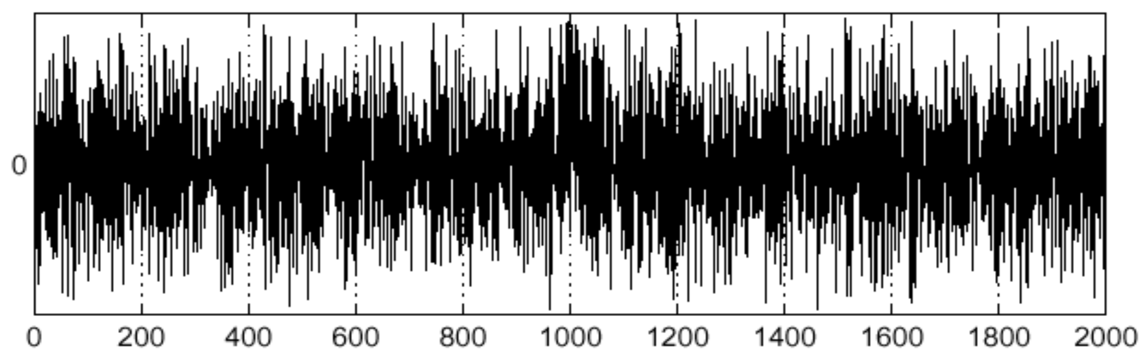
Effects of Noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

$$f(x)$$



$$\frac{d}{dx}f(x)$$



Where is the edge?

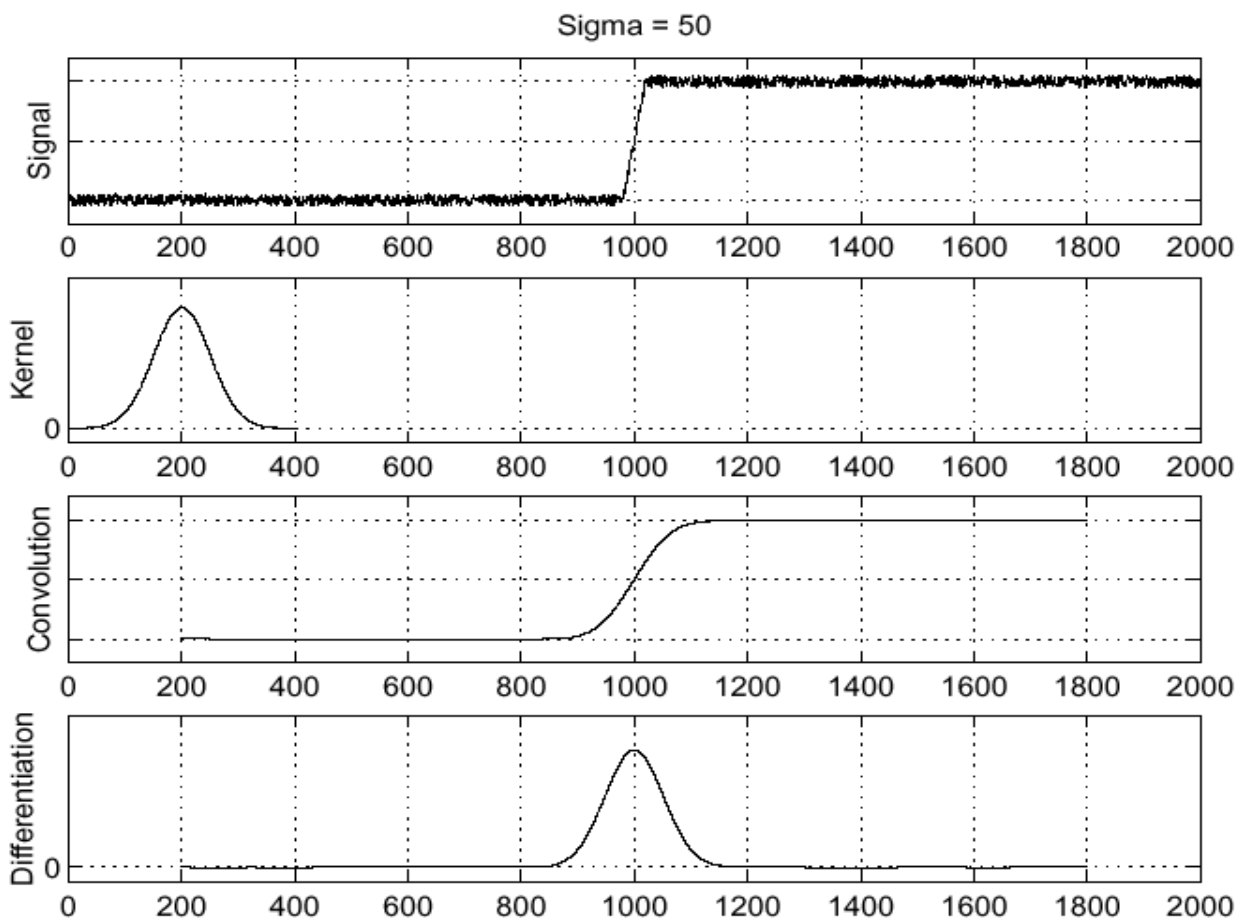
Solution: Smooth First

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



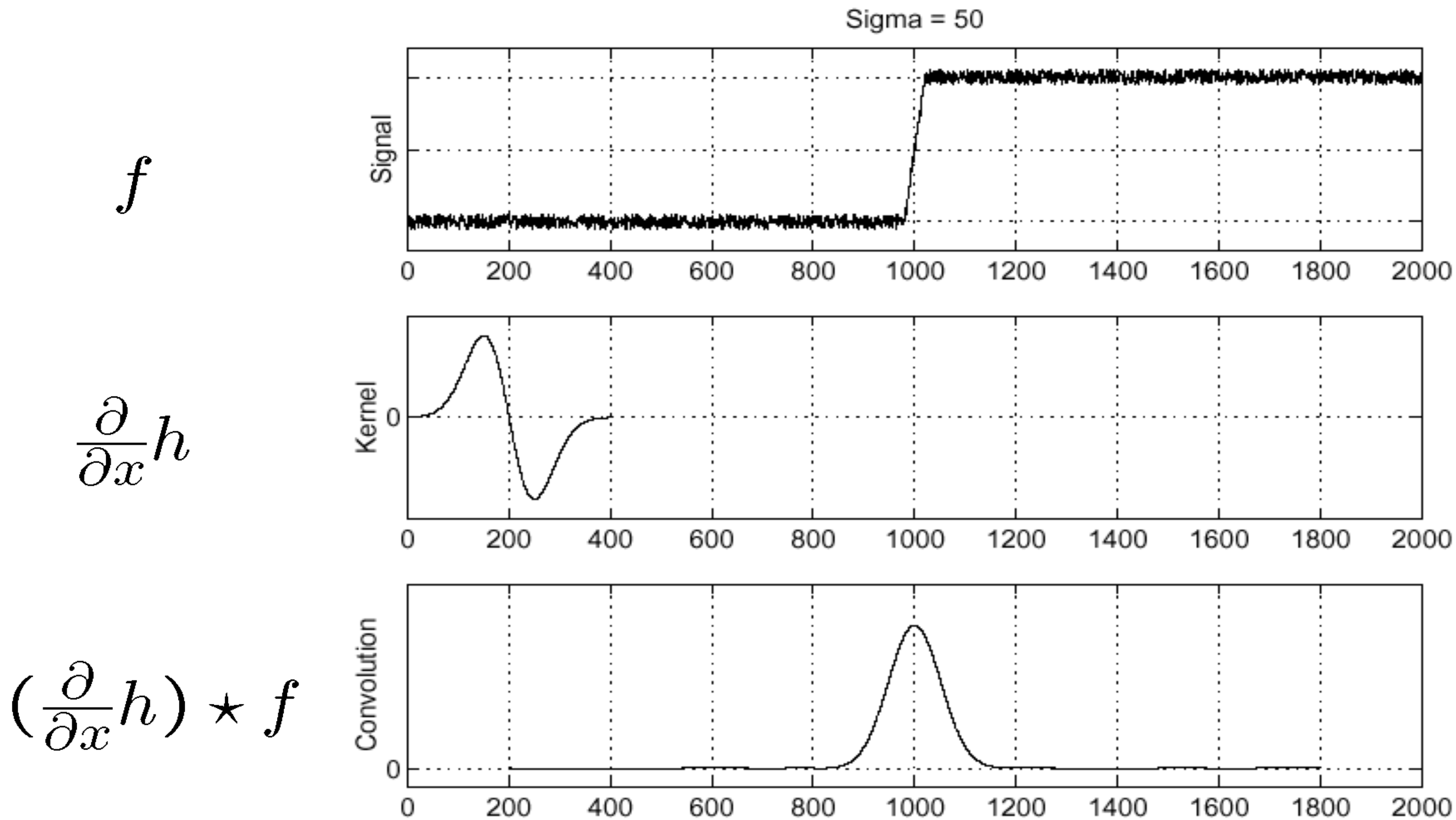
Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Associative Property of Convolution

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:



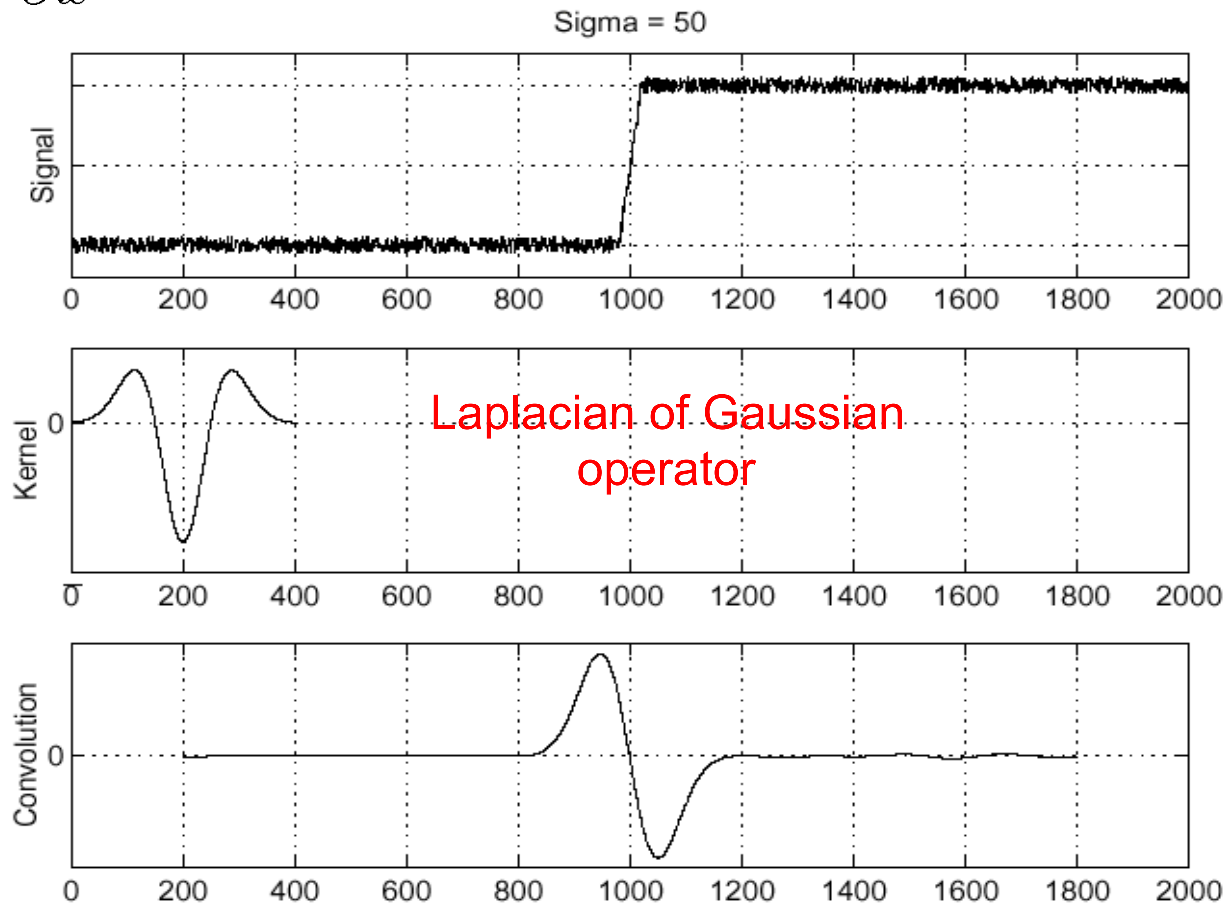
Laplacian of Gaussian

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

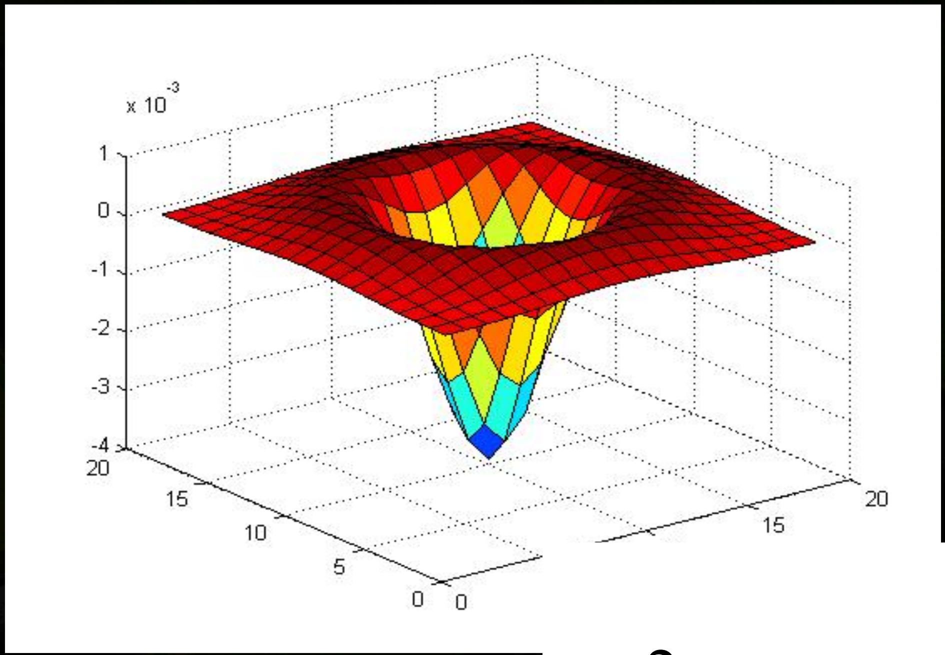
$(\frac{\partial^2}{\partial x^2}h) \star f$

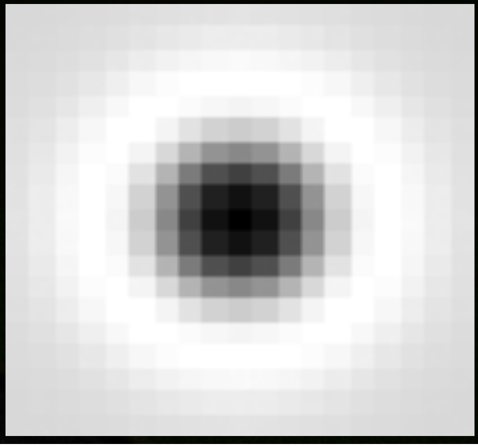


Where is the edge? Zero-crossings of bottom graph

Blob detection in 2D

- **Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D**



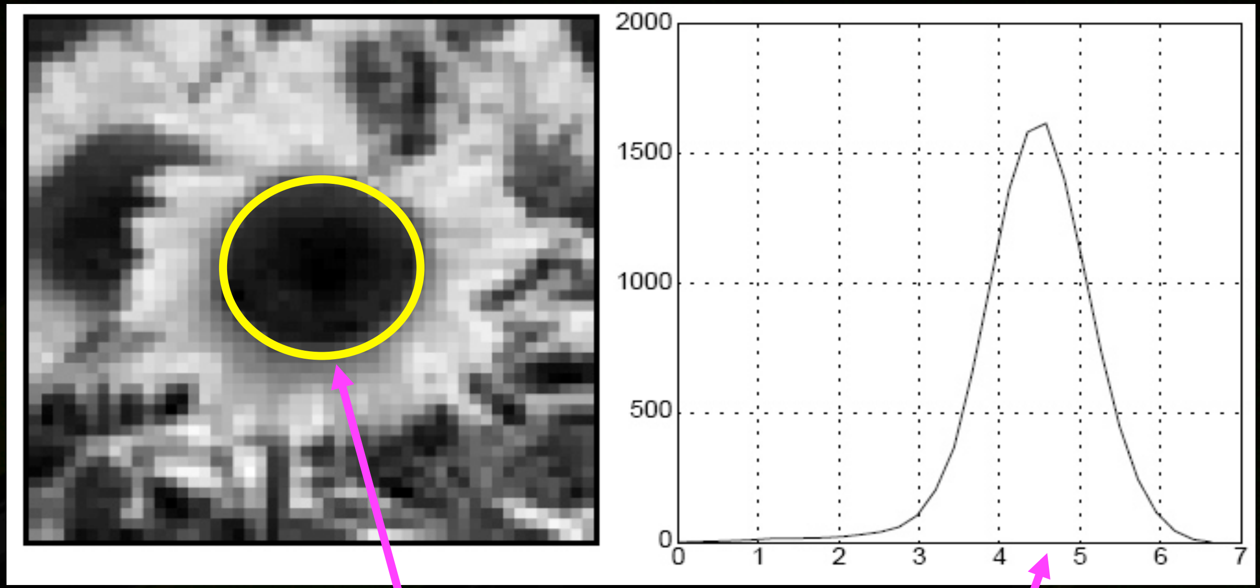


Scale-normalized:

$$\nabla^2_{\text{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

Characteristic scale

- **We define the characteristic scale as the scale that produces peak of Laplacian response**

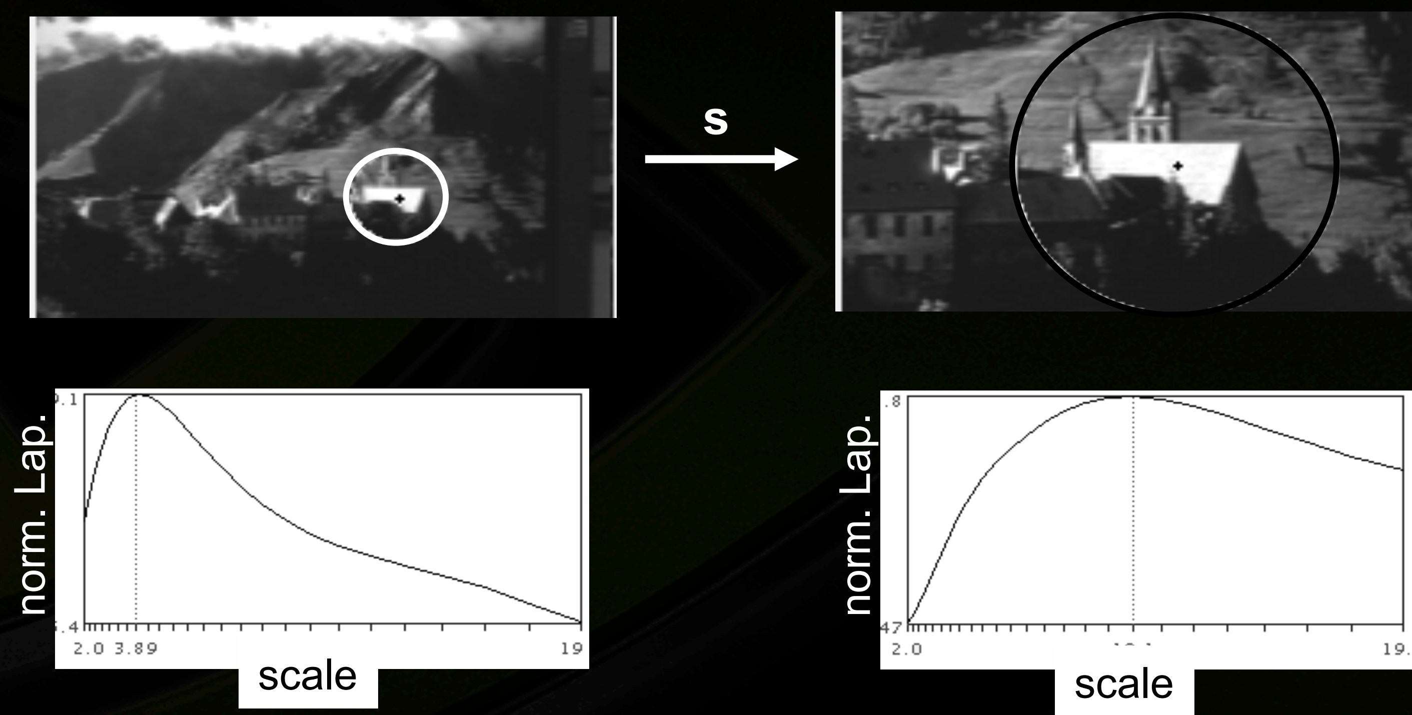


characteristic scale

T. Lindeberg (1998). Feature detection with automatic scale selection. *International Journal of Computer Vision* **30** (2): pp 77--116.

Scale selection

- Scale invariance of the characteristic scale



Difference of Gaussians (DoG)

- **Laplacian of Gaussian can be approximated by the difference between two different Gaussians**

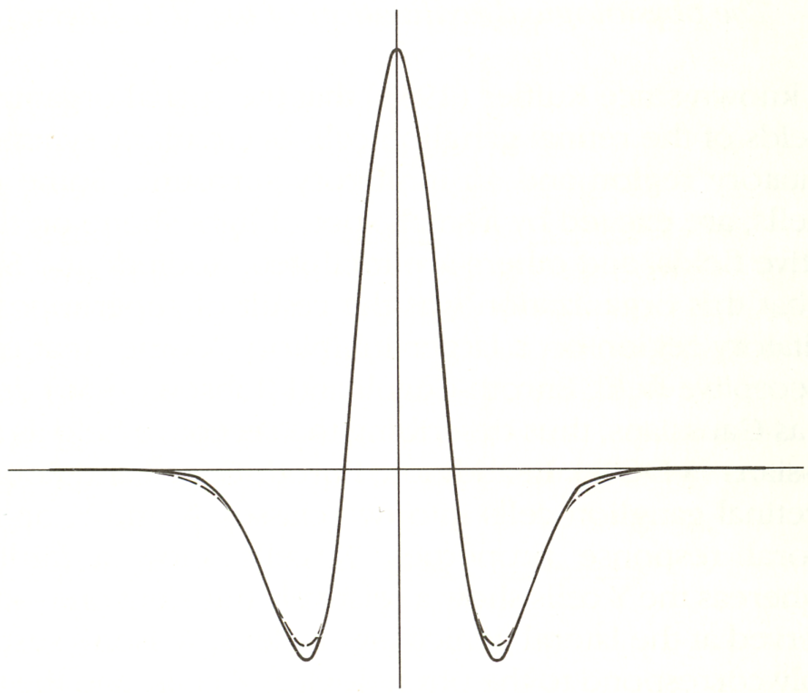
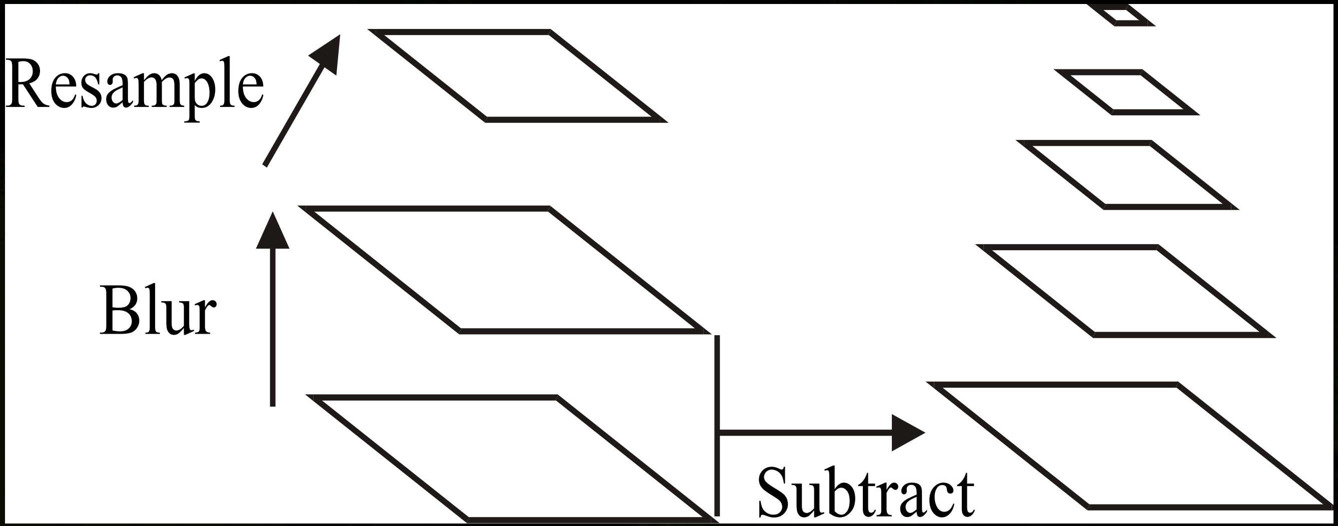
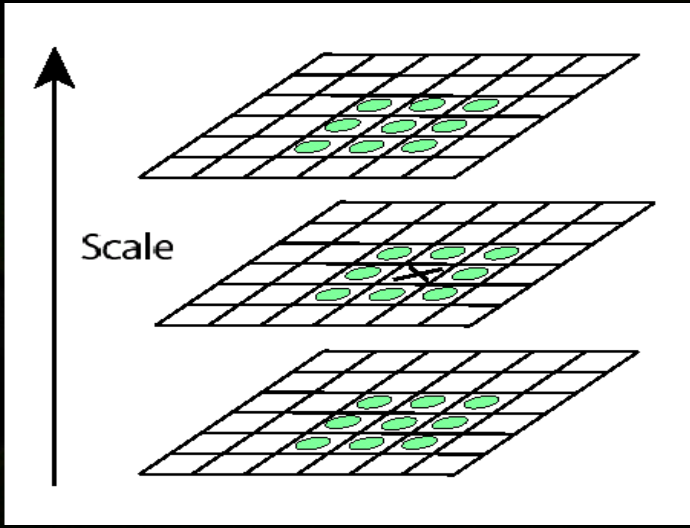


Figure 2–16. The best engineering approximation to $\nabla^2 G$ (shown by the continuous line), obtained by using the difference of two Gaussians (DOG), occurs when the ratio of the inhibitory to excitatory space constraints is about 1:1.6. The DOG is shown here dotted. The two profiles are very similar. (Reprinted by permission from D. Marr and E. Hildreth, "Theory of edge detection," *Proc. R. Soc. Lond. B 204,* pp. 301–328.)

SIFT – Scale Invariant Feature Transform

- **Descriptor overview:**
  - **Determine scale (by maximizing DoG in scale and in space)**

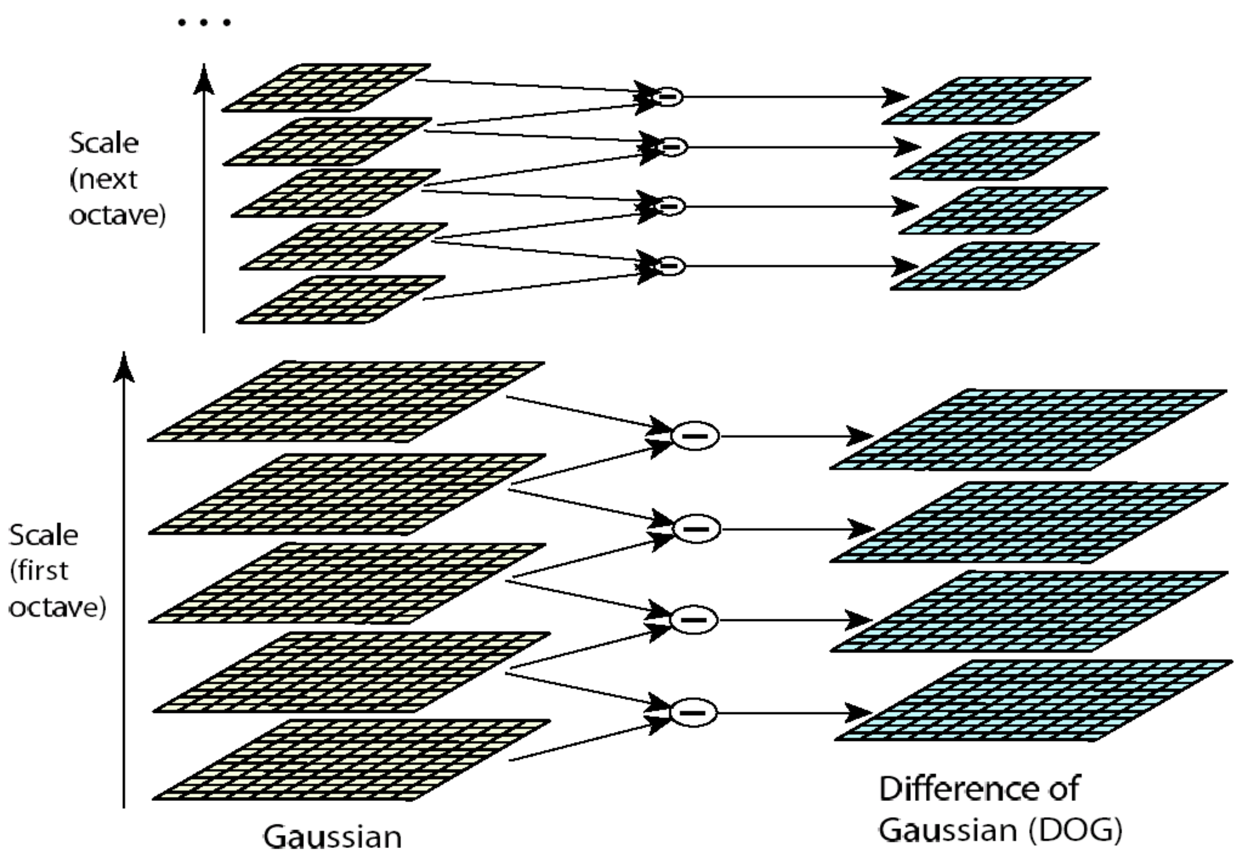




D. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints” IJCV 2004

DOG detector

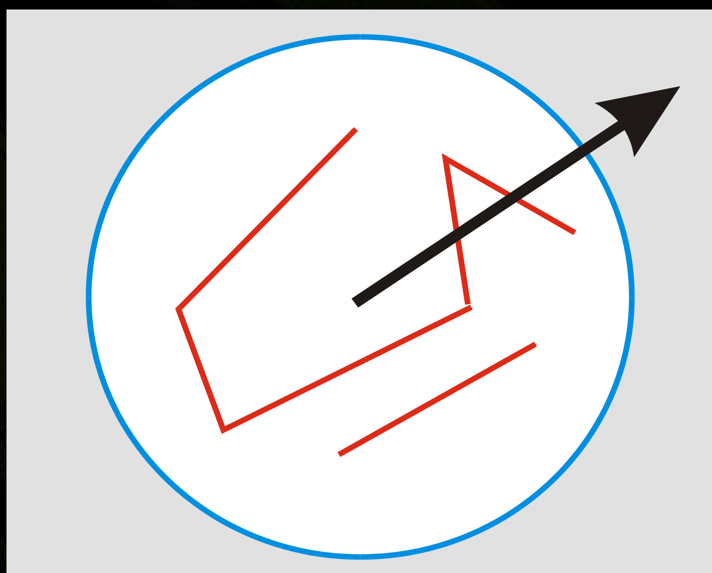
- Fast computation, scale space processed one octave at a time

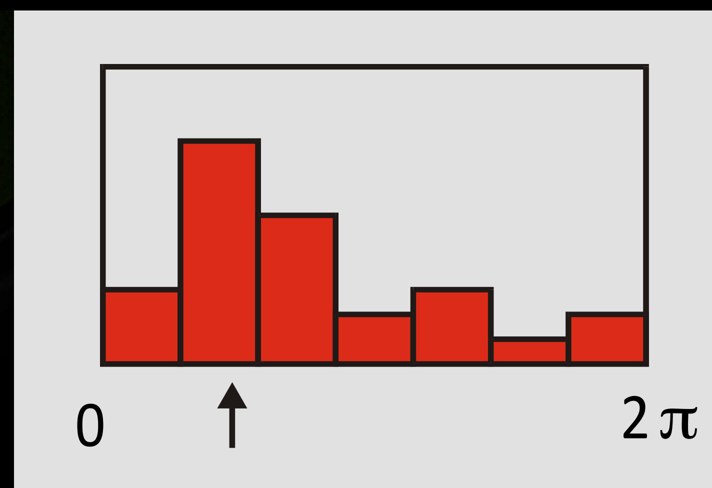


David G. Lowe. "Distinctive image features from scale-invariant keypoints."I*JCV* 60 (2).

SIFT – Scale Invariant Feature Transform

- **Descriptor overview:**
  - **Determine scale (by maximizing DoG in scale and in space), local orientation as the dominant gradient direction**
  - **Use this scale and orientation to make all further computations invariant to scale and rotation**



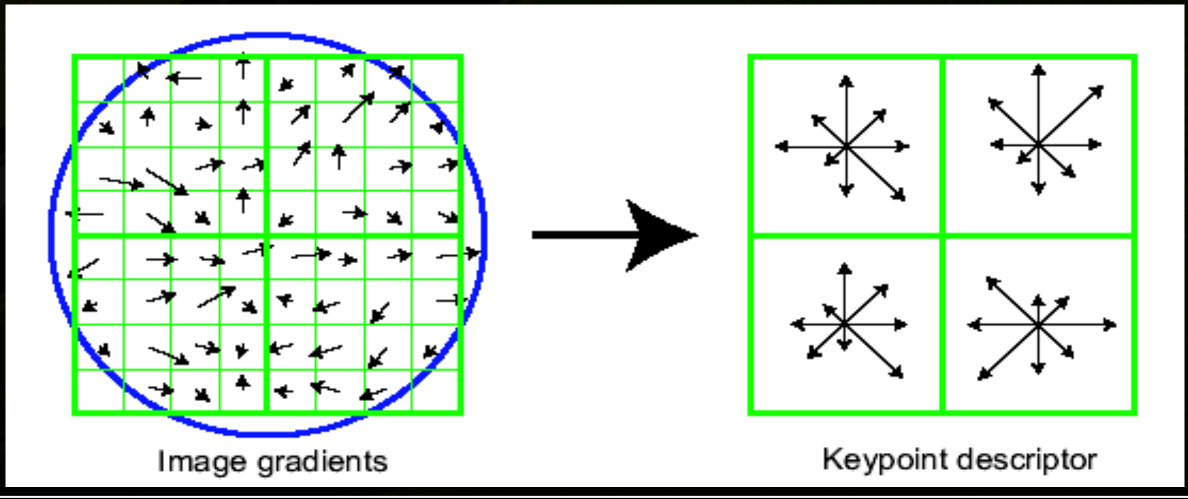


D. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints” IJCV 2004

Affine invariant regions - Example

SIFT – Scale Invariant Feature Transform

- Descriptor overview:
  - Determine scale (by maximizing DoG in scale and in space), local orientation as the dominant gradient direction
  - Use this scale and orientation to make all further computations invariant to scale and rotation
  - Compute gradient orientation histograms of several small windows (128 values for each point)
  - Normalize the descriptor to make it invariant to intensity change



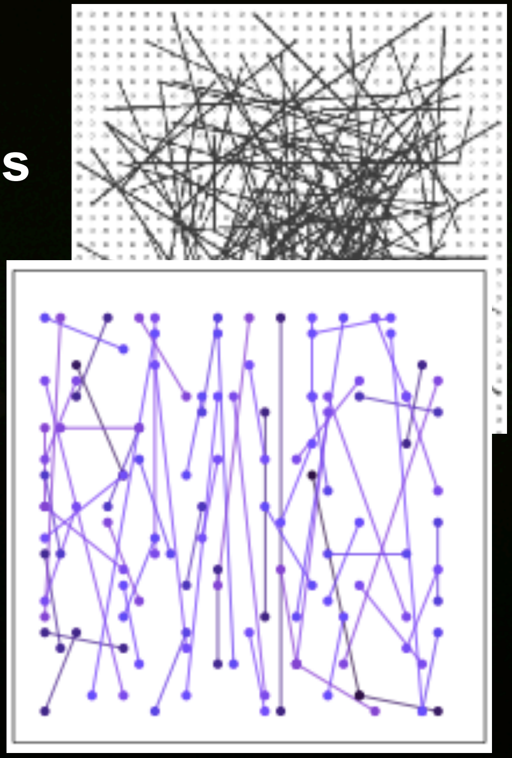
D. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints" IJCV 2004

ORB (Oriented FAST and Rotated BRIEF)

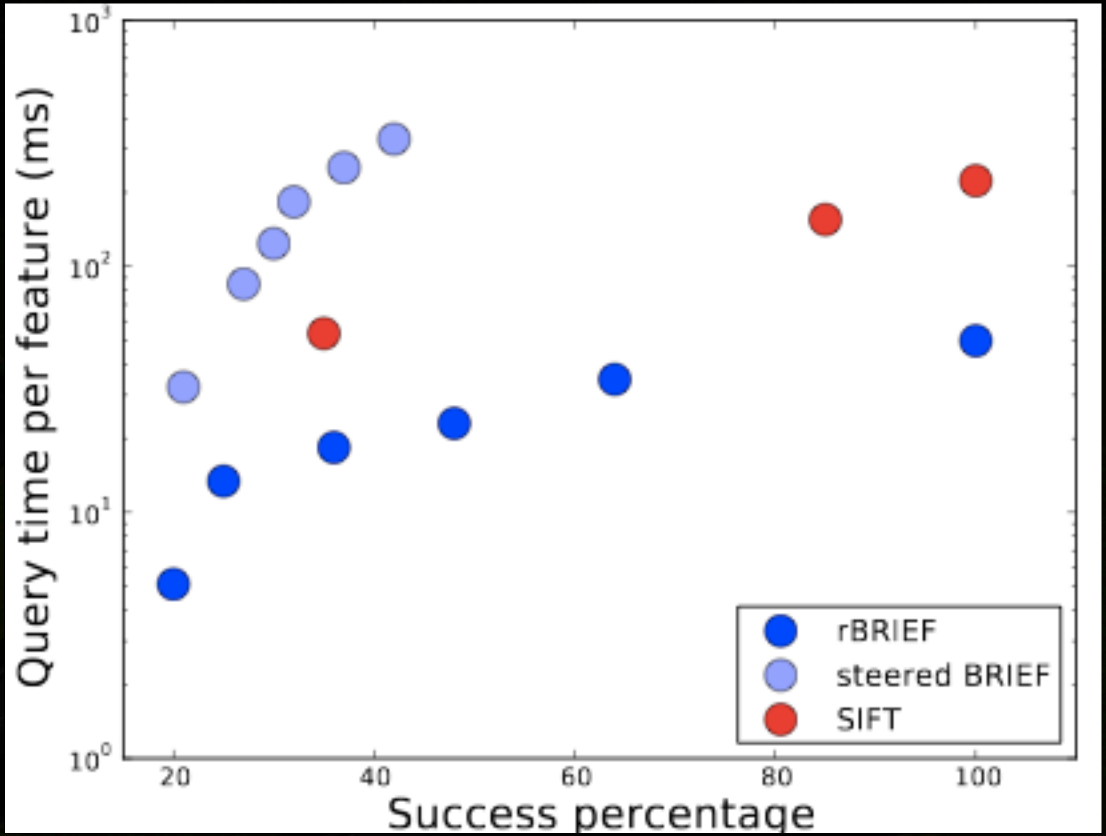
- **Use FAST-9**
  - use Harris measure to order them
- **Find orientation**
  - calculate weighted new center

$$\left(\frac{\sum xI(x,y)}{\sum I(x,y)}, \frac{\sum yI(x,y)}{\sum I(x,y)} \right)$$

  - reorient image so that gradients vary vertically
- **BRIEF**
  - Binary Robust Independent Elementary Features
  - choose pixels to compare, result creates 0 or 1
  - combine to a binary vector, compare using Hamming distance (XOR + pop count)
- **Rotated BRIEF**
  - train a good set of pixels to compare



rBRIEF vs. SIFT



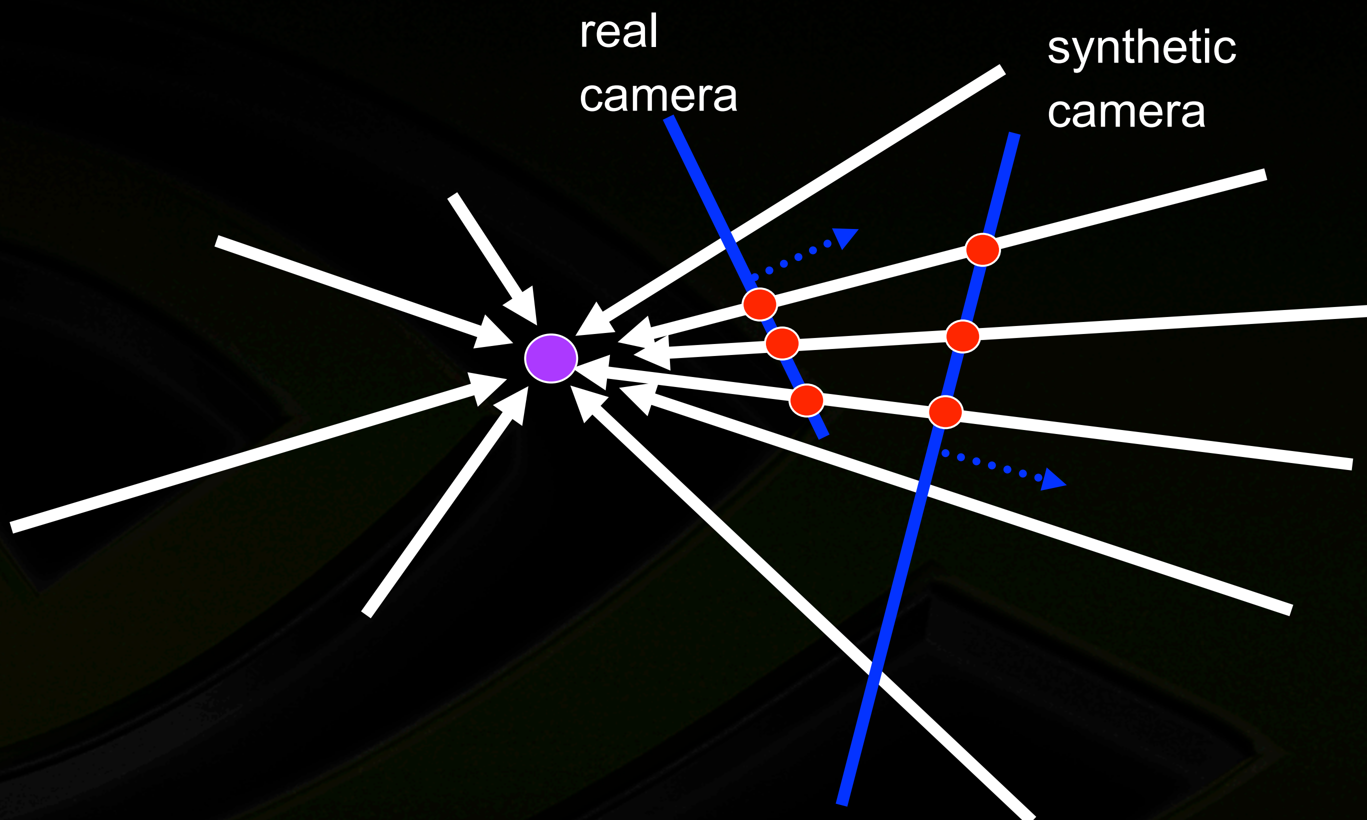
Aligning images: Translation?

left on top

right on top

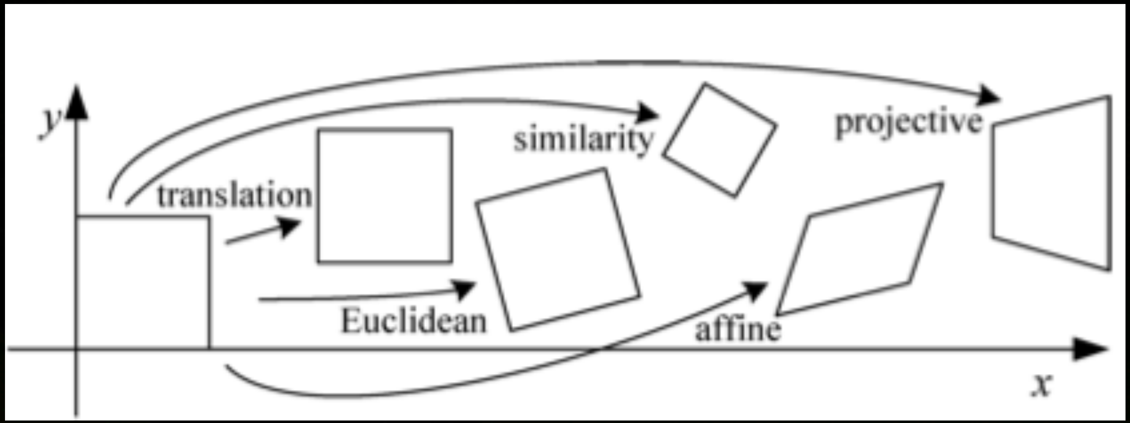
Translations are not enough to align the images

A pencil of rays contains all views

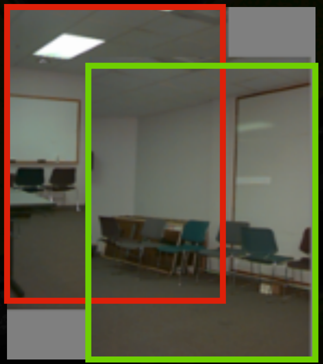


Can generate any synthetic camera view
as long as it has **the same center of projection**!
… and scene geometry does not matter …

Which transform to use?

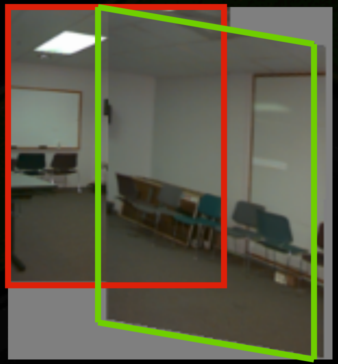


Translation



2 unknowns

Affine

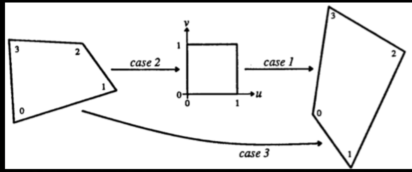


6 unknowns

Perspective

8 unknowns

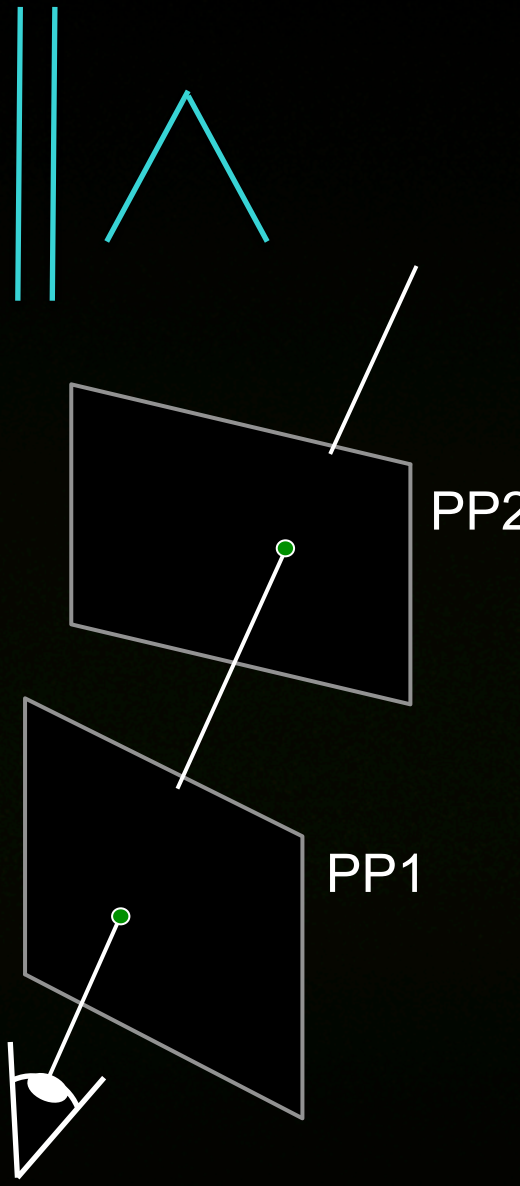
Homography



- **Projective mapping between any two PPs with the same center of projection**
  - **rectangle should map to arbitrary quadrilateral**
  - **parallel lines aren't**
  - **but must preserve straight lines**

is called a Homography

$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$



- **To apply a homography H**
  - **compute p' = Hp** ***(regular matric multiply)***
  - **convert p' from homogeneous to image coordinates [x', y']** ***(divide by w)***

Homography from mapping quads

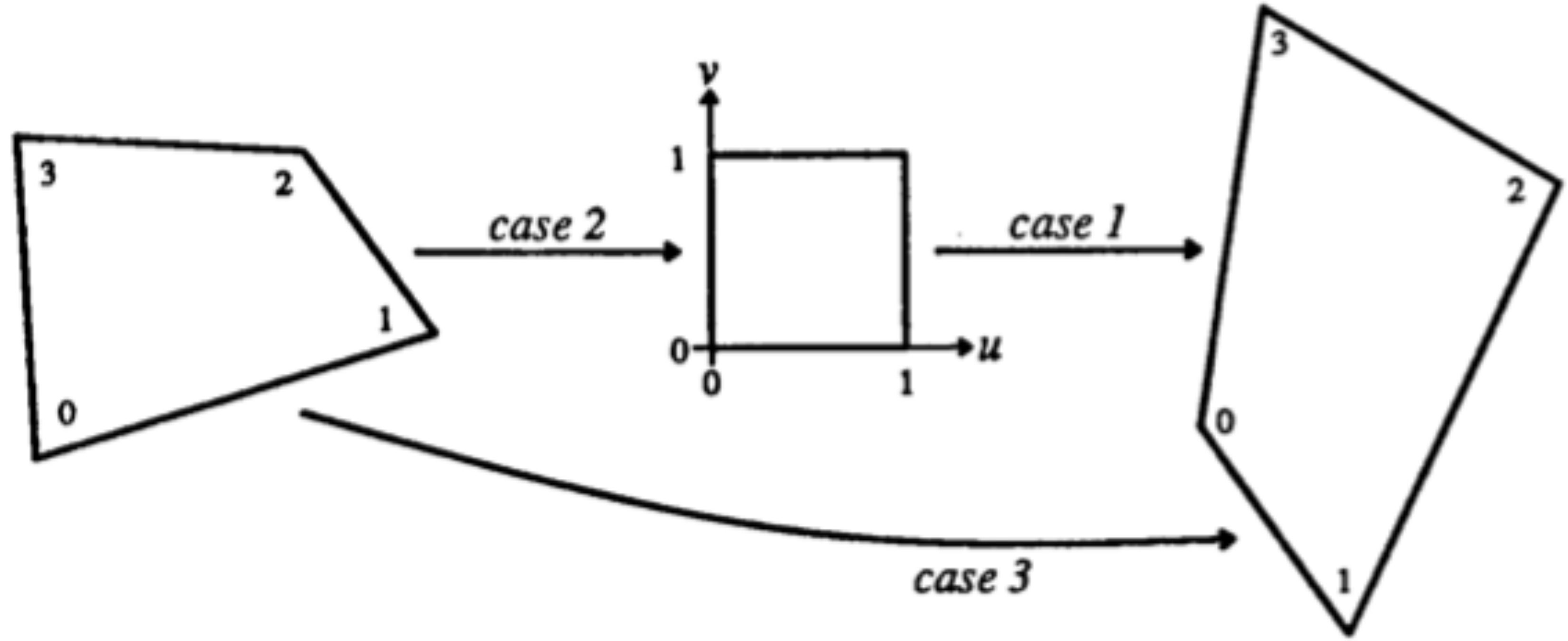


Figure 2.8: *Quadrilateral to quadrilateral mapping as a composition of simpler mappings.*

Fundamentals of Texture Mapping and Image Warping
Paul Heckbert, M.Sc. thesis, U.C. Berkeley, June 1989, 86 pp.
http://www.cs.cmu.edu/~ph/texfund/texfund.pdf

Homography from *n* point pairs (x,y ; x',y')

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

p' H p

- **Multiply out**
  - $wx' = h_{11}\, x + h_{12}\, y + h_{13}$
  - $wy' = h_{21}\, x + h_{22}\, y + h_{23}$
  - $w \;\; = h_{31}\, x + h_{32}\, y + h_{33}$
- **Get rid of w**
  - $(h_{31}\, x + h_{32}\, y + h_{33})x' - (h_{11}\, x + h_{12}\, y + h_{13}) = 0$
  - $(h_{31}\, x + h_{32}\, y + h_{33})y' - (h_{21}\, x + h_{22}\, y + h_{23}) = 0$
- **Create a new system Ah = 0**
  - Each point constraint gives two rows of A
    - $[-x \quad -y \quad -1 \quad 0 \quad 0 \quad 0 \quad xx' \quad yx' \quad x']$
    - $[\;0 \quad 0 \quad 0 \quad -x \quad -y \quad -1 \quad xy' \quad yy' \quad y']$

$$h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

- **Solve with singular value decomposition of $A = USV^T$**
  - solution is in the nullspace of A
  - the last column of V (= last row of V^T)

Python test code

```
from numpy import *

# create 4 random homogen. points
X = ones([3,4])              # the points are on columns
X[:2,:] = random.rand(2,4) # first row x coord, second y coord, third w = 1
x,y = X[0],X[1]
# create projective matrix
H = random.rand(3,3)
# create the target points
Y = dot(H,X)
# homogeneous division
YY = (Y / Y[2])[:2,:]
u,v = YY[0],YY[1]

A = zeros([8,9])
for i in range(4):
    A[2*i  ] = [-x[i], -y[i], -1,     0,     0,  0, x[i] * u[i], y[i] * u[i], u[i]]
    A[2*i+1] = [    0,     0,  0, -x[i], -y[i], -1, x[i] * v[i], y[i] * v[i], v[i]]

[u,s,vt] = linalg.svd(A)

# reorder the last row of vt to 3x3 matrix
HH = vt[-1,:].reshape([3,3])

# test that the matrices are the same (within a multiplicative factor)
print H - HH * (H[2,2] / HH[2,2])
```

Example

common picture plane of mosaic image

perspective reprojection

Pics: Marc Levoy

Reprojecting an image onto a different picture plane

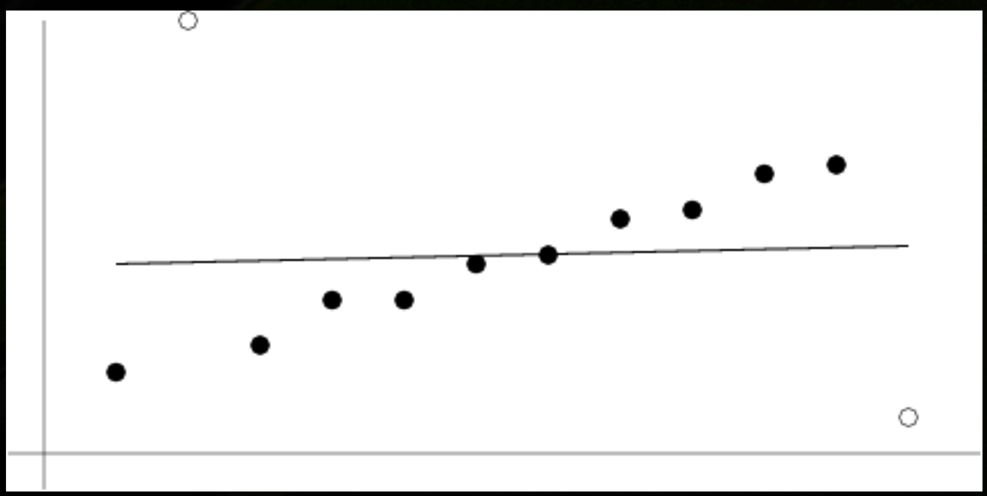
the sidewalk art of Julian Beever

- **the view on any picture plane can be projected onto any other plane in 3D without changing its appearance as seen from the center of projection**

What to do with outliers?

- **Least squares OK when error has Gaussian distribution**
- **But it breaks with *outliers***
  - **data points that are not drawn from the same distribution**
- **Mis-matched points are outliers to the Gaussian error distribution**
  - **severely disturbs the Homography**

Line fitting using regression is biased by outliers

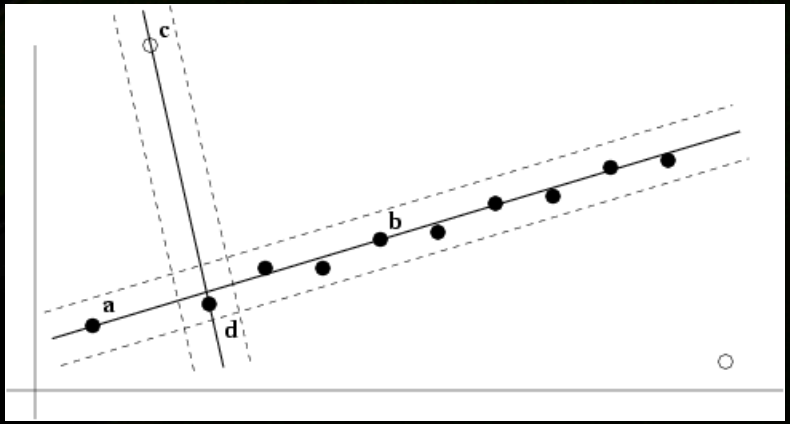


RANSAC

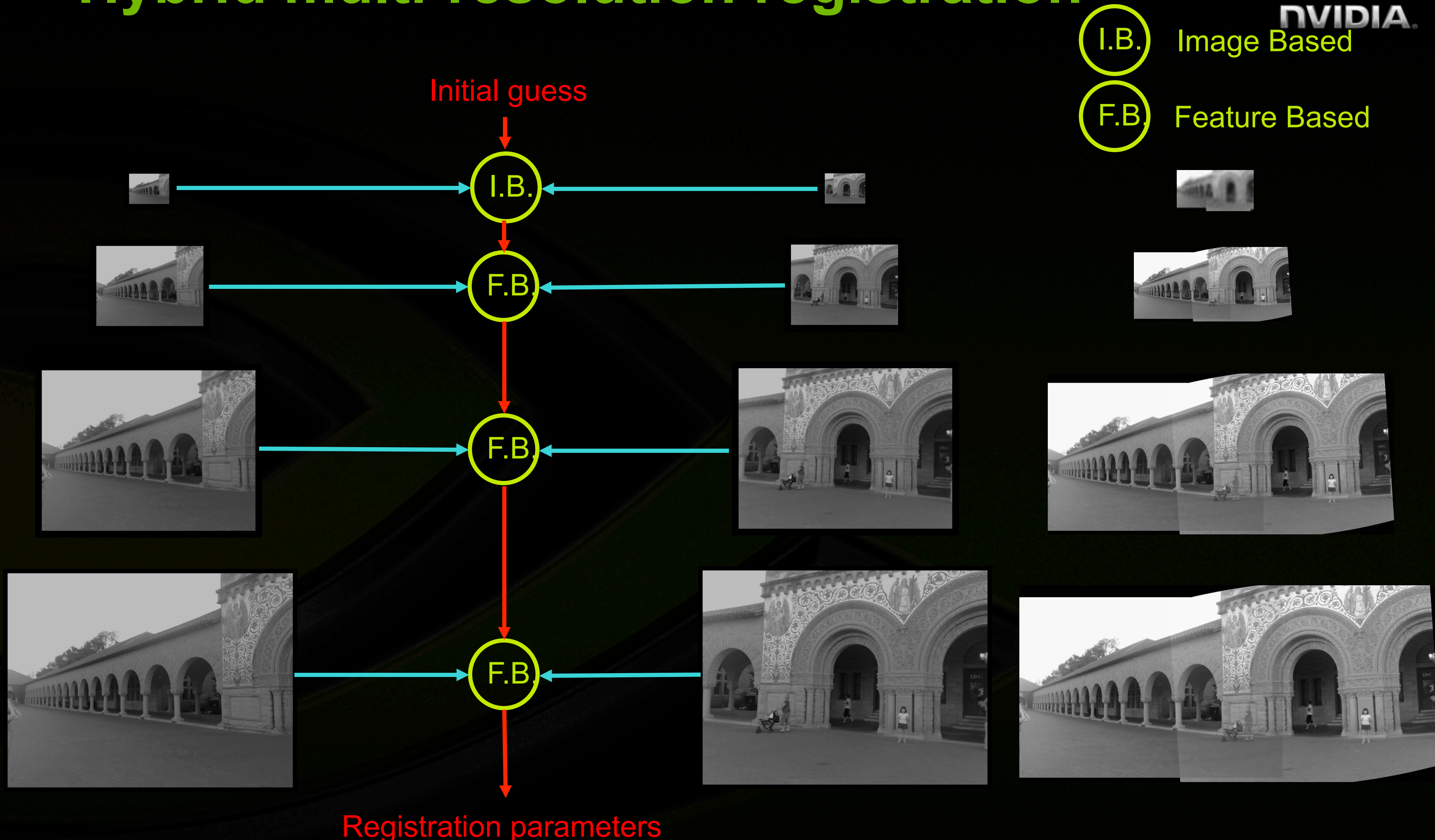
- **RANdom SAmple Consensus**

1. **Randomly choose a subset of data points to fit model (a *sample*)**
2. **Points within some distance threshold t of model are a *consensus set***
   Size of consensus set is model's *support*
3. **Repeat for N samples; model with biggest support is most robust fit**
   - **Points within distance t of best model are inliers**
   - **Fit final model to all inliers**

Two samples
and their supports
for line-fitting



Hybrid multi-resolution registration



K. Pulli, M. Tico, Y. Xiong, X. Wang, C-K. Liang, “Panoramic Imaging System for Camera Phones”, ICCE 2010

Progression of multi-resolution registration

Actual

size

Applied

to hi-res

Feature-based registration

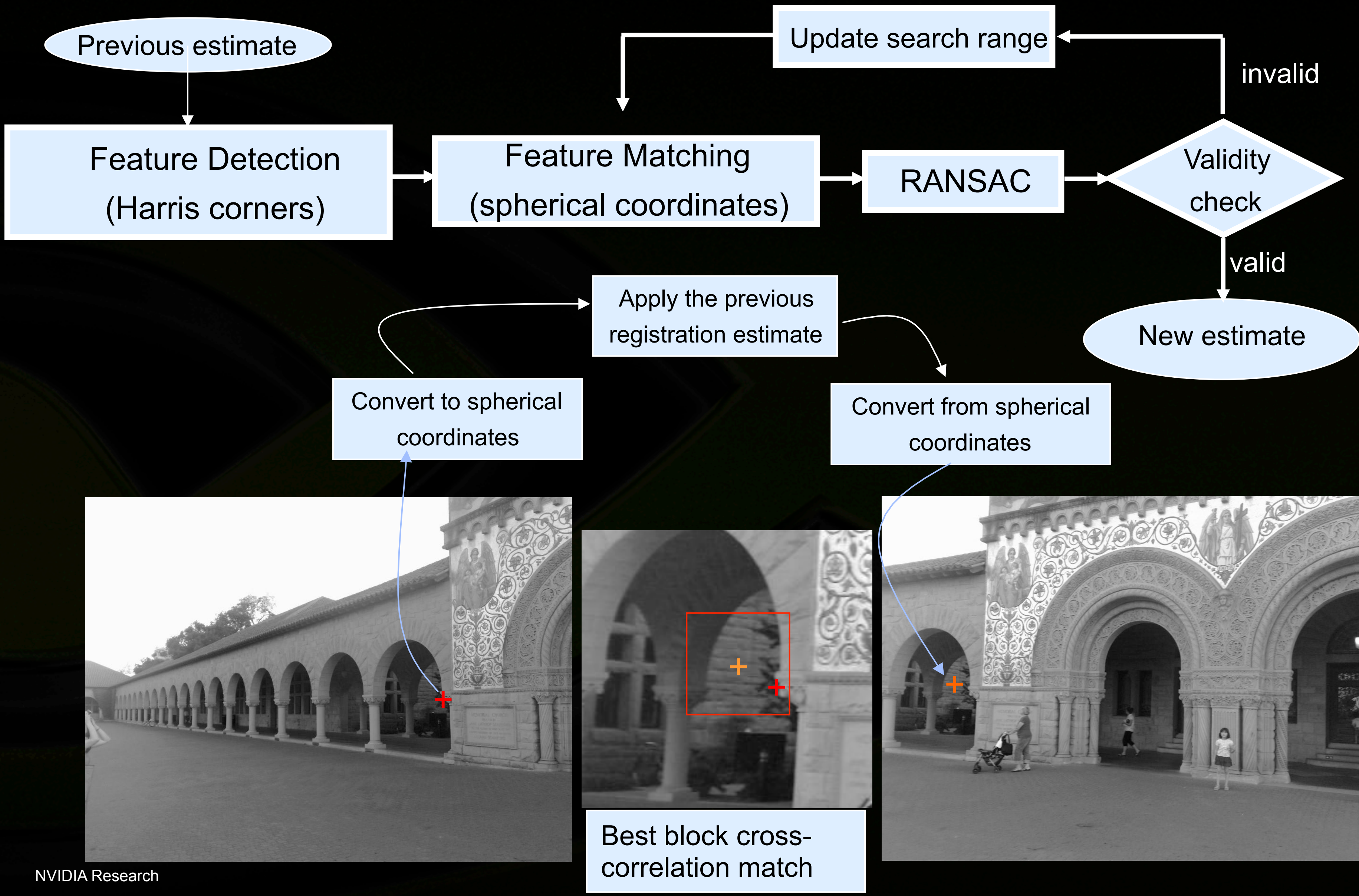
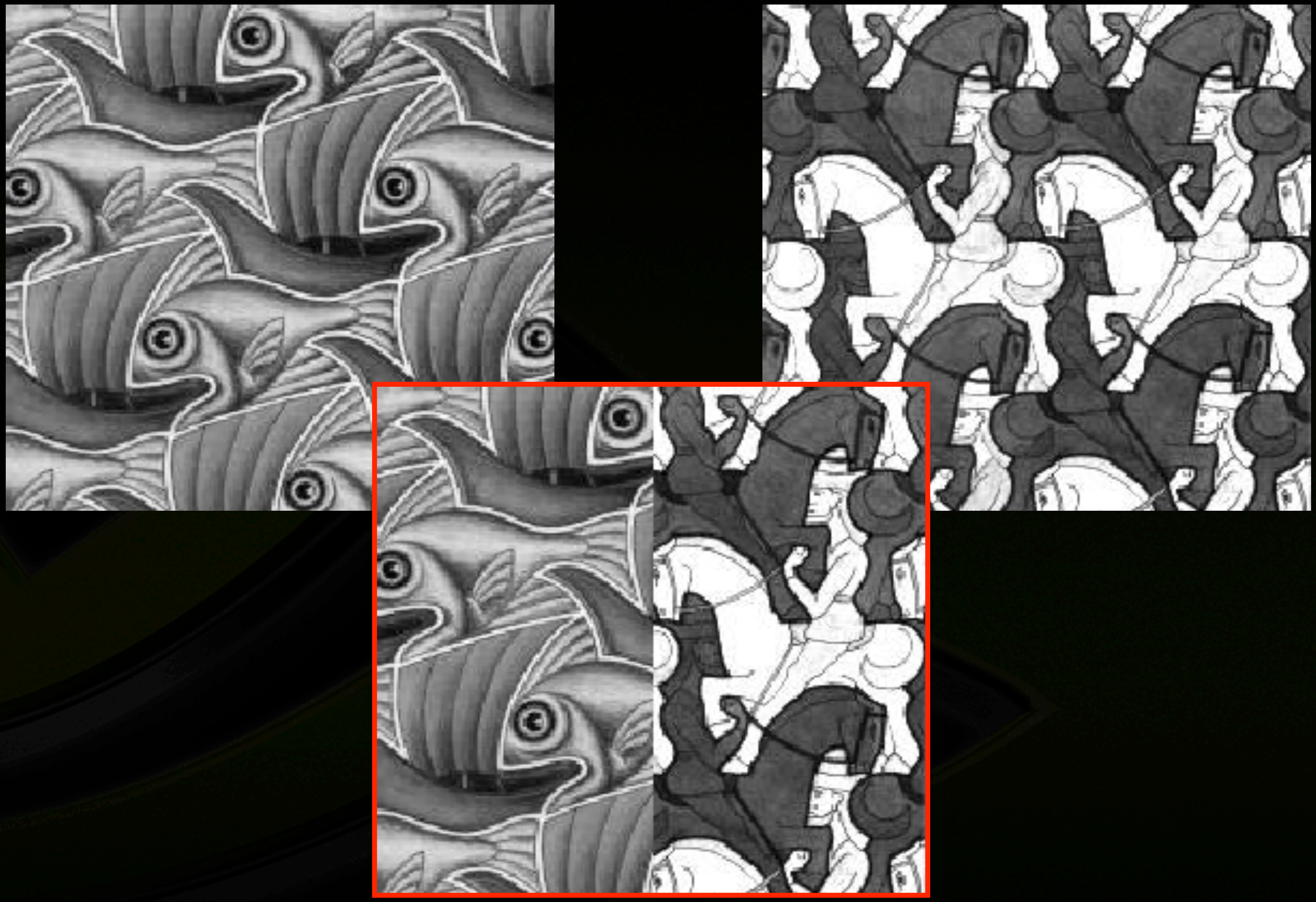


Image blending

- **Directly averaging the overlapped pixels results in ghosting artifacts**
  - **Moving objects, errors in registration, parallax, etc.**

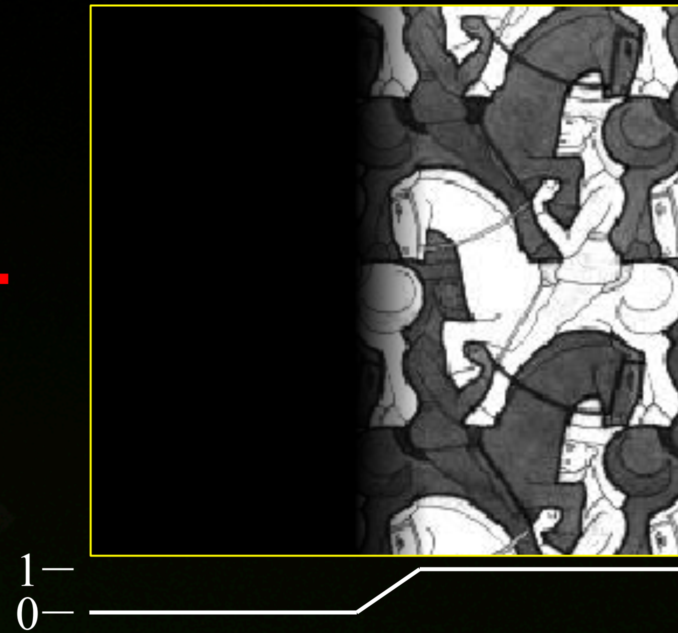
Photo by Chia-Kai Liang

Alpha Blending / Feathering



Alpha Blending / Feathering

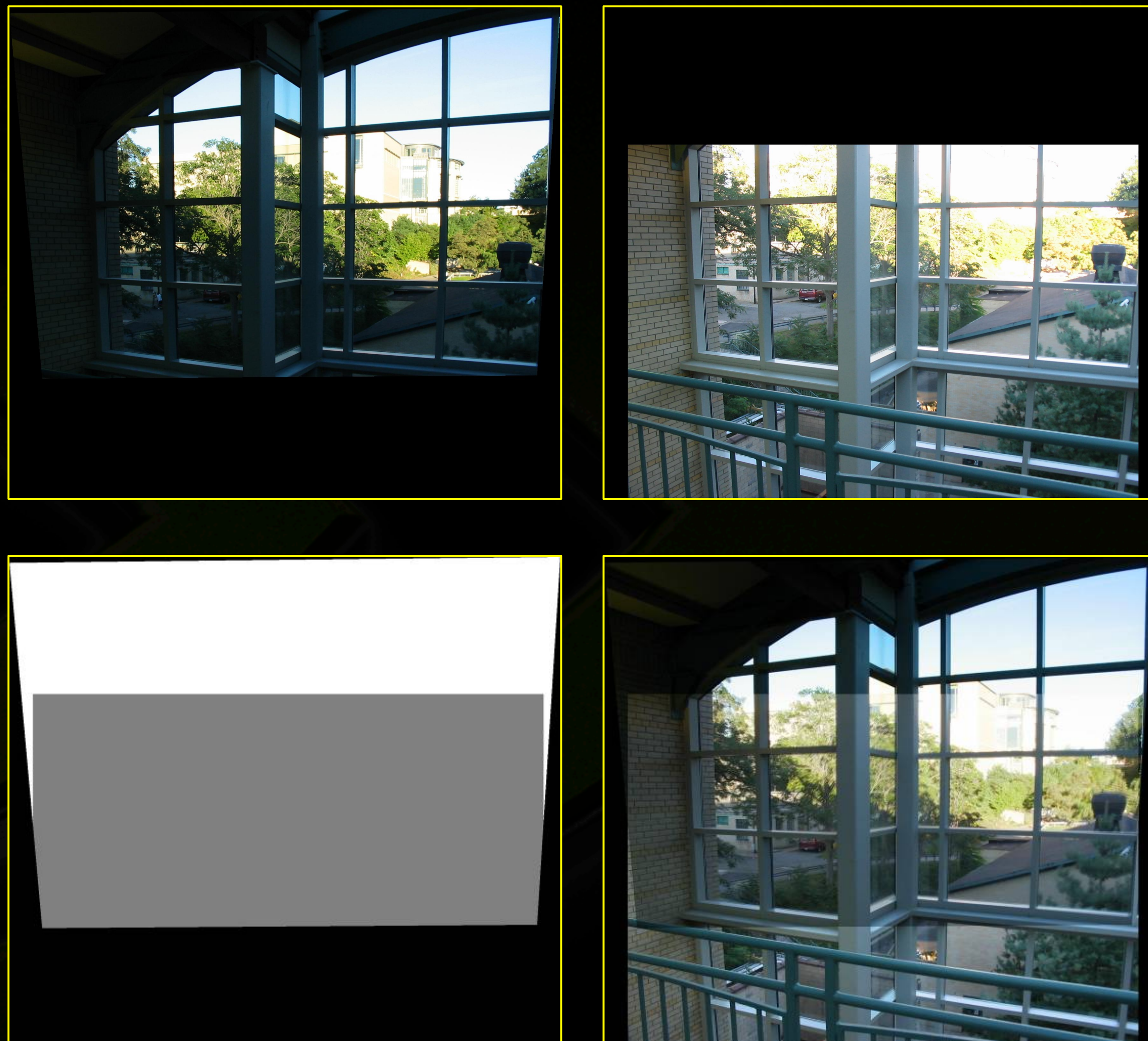
+



=

$$I_{blend} = \alpha I_{left} + (1-\alpha) I_{right}$$

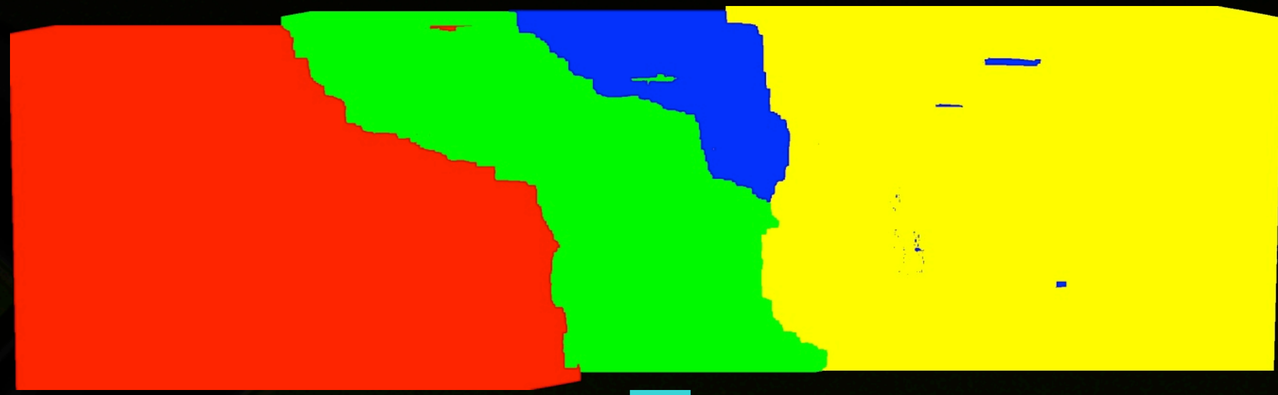
Setting alpha: simple averaging

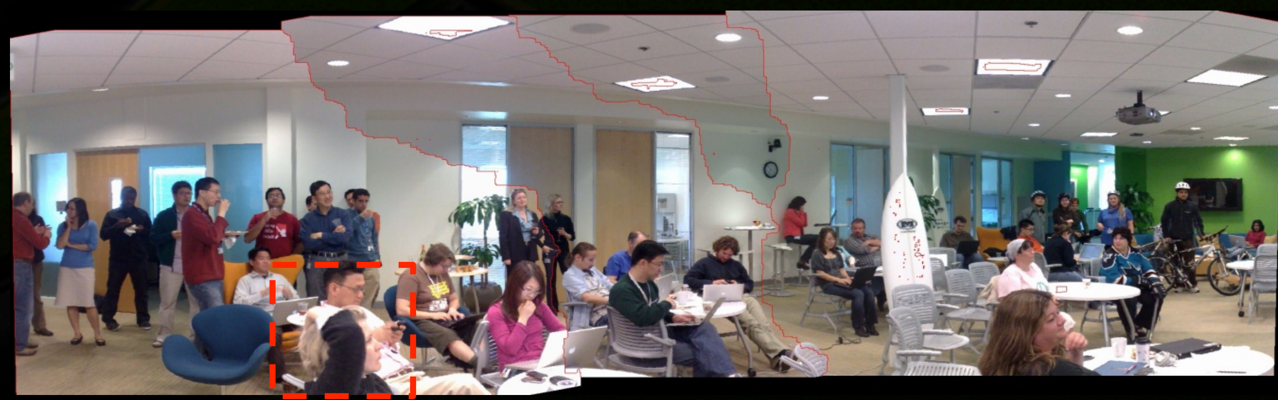


Alpha = .5 in overlap region

Solution for ghosting: Image labeling

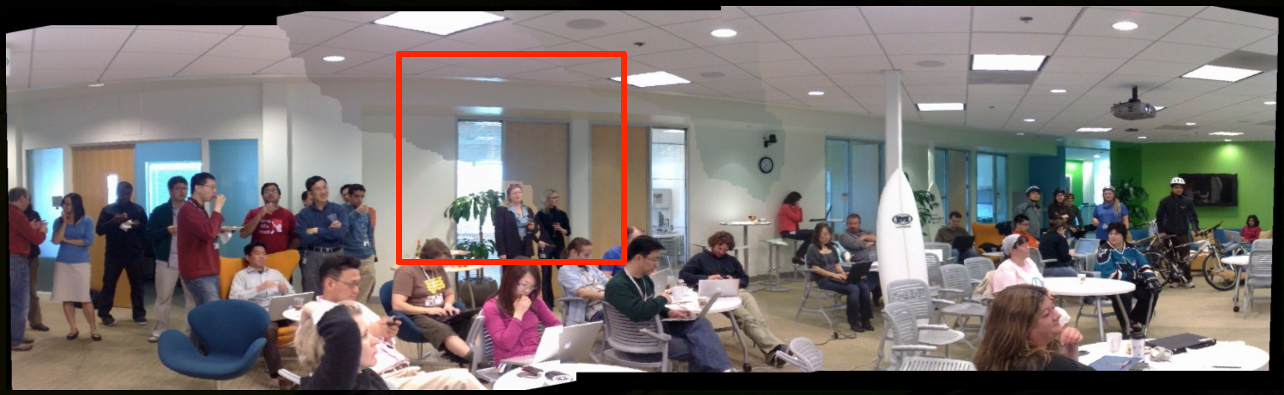
- Assign one input image each output pixel
  - Optimal assignment can be found by graph cut [Agarwala et al. 2004]





New artifacts

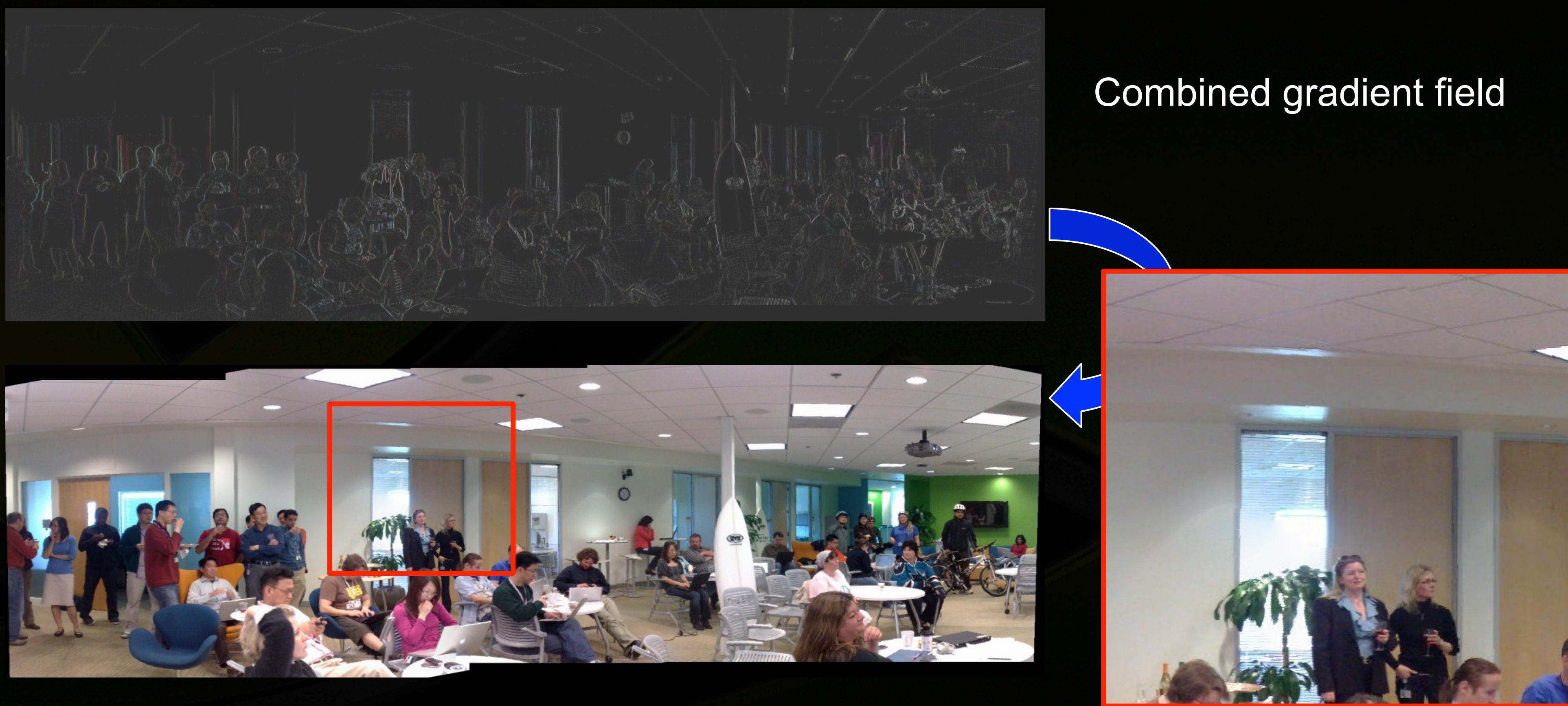
- Inconsistency between pixels from different input images
  - Different exposure/white balance settings
  - Photometric distortions (e.g., vignetting)



Solution: Poisson blending

- Copy the gradient field from the input image
- Reconstruct the final image by solving a Poisson equation

Combined gradient field

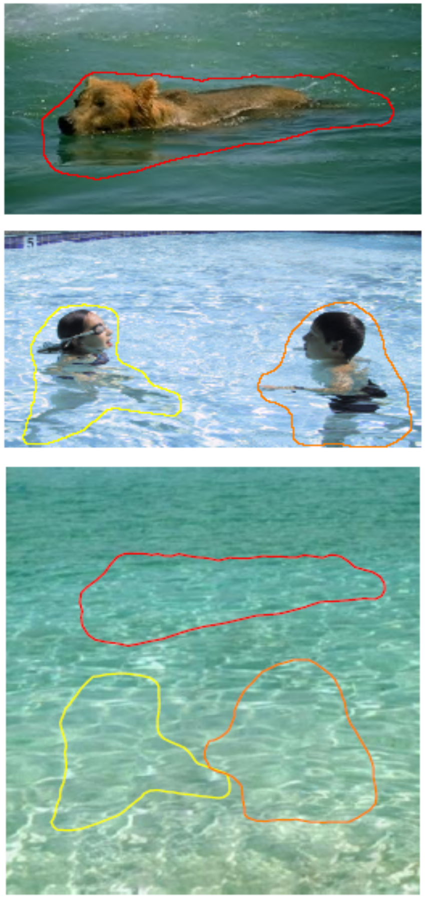


Problems with direct cloning

P. Pérez, M. Gangnet, A. Blake. Poisson image editing. SIGGRAPH 2003

http://www.irisa.fr/vista/Papers/2003_siggraph_perez.pdf

Solution: clone gradient, integrate colors

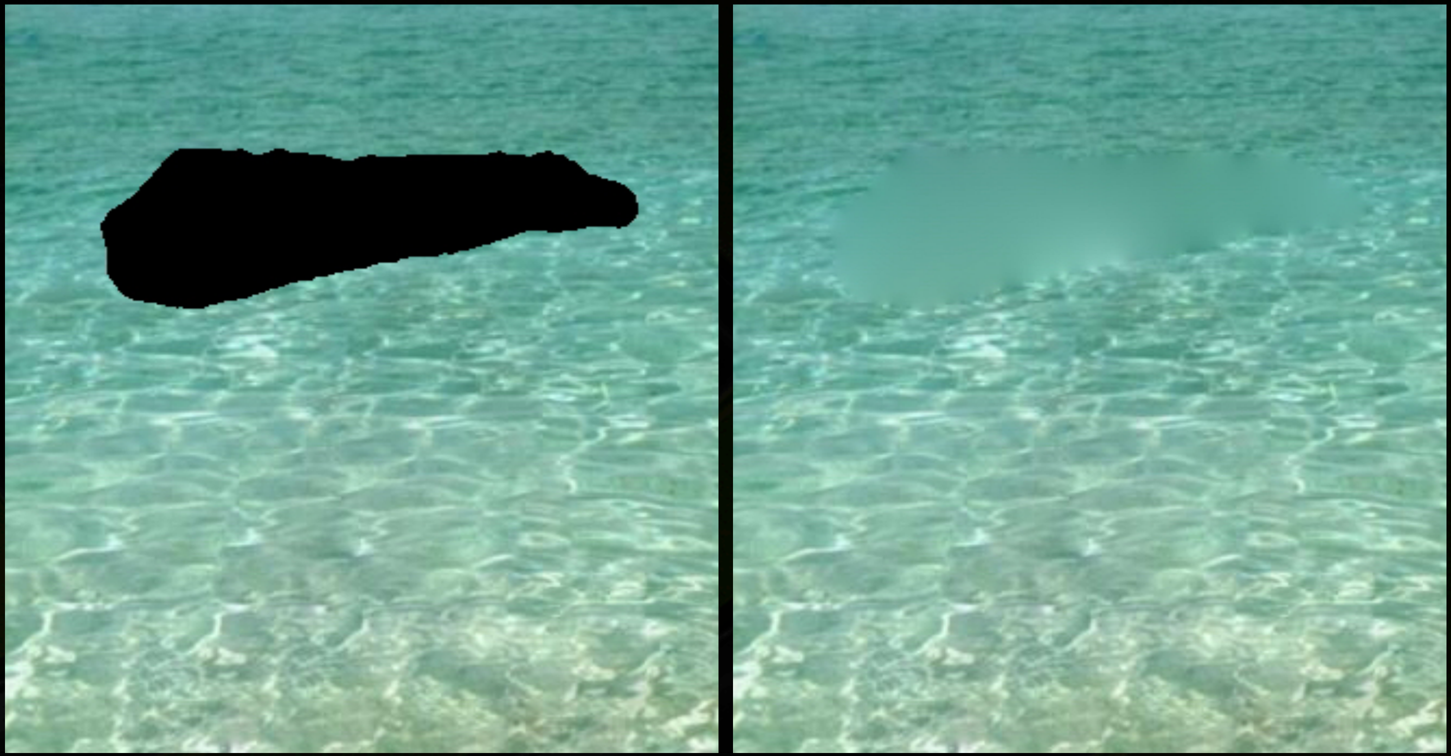


sources/destinations

cloning

seamless cloning

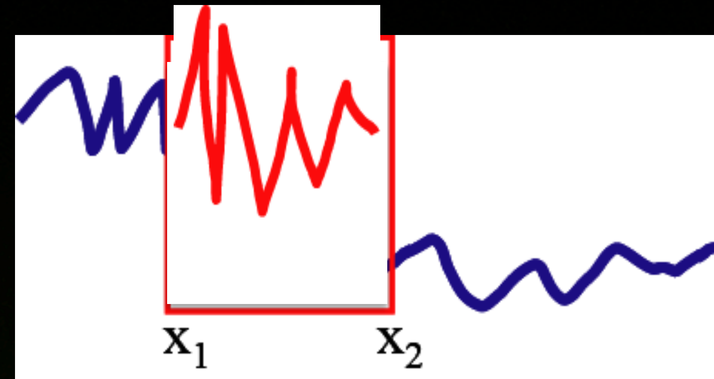
Membrane interpolation



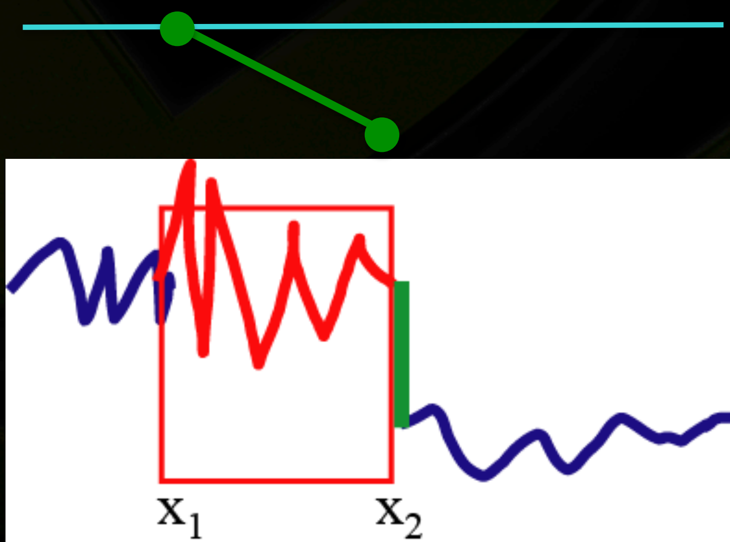
Copy the details

Seamlessly paste

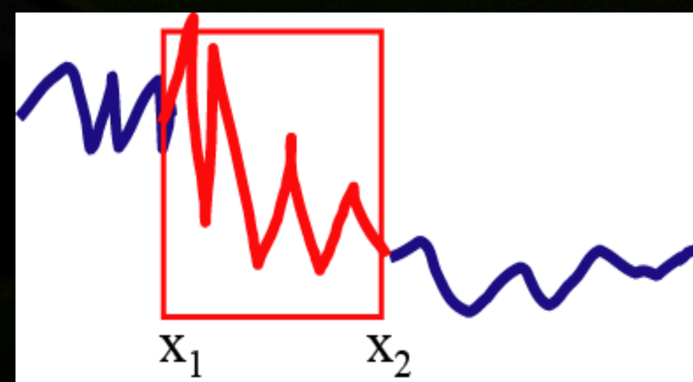
onto



Just add a linear function so that the boundary condition is respected



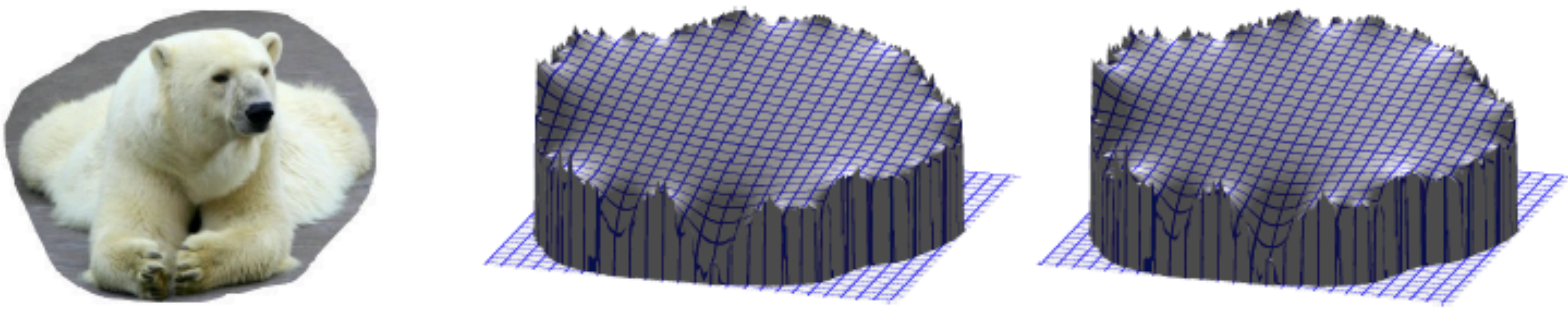
Gradients didn't change much, and function is continuous



Coordinates for Instant Image Cloning

SIGGRAPH 2009

Zeev Farbman, Hebrew University
Gil Hoffer, Tel Aviv University
Yaron Lipman, Princeton University
Daniel Cohen-Or, Tel Aviv University
Dani Lischinski, Hebrew University



(a) Source patch (b) Laplace membrane (c) Mean-value membrane

(d) Target image (e) Poisson cloning (f) Mean-value cloning

Figure 1: *Poisson cloning smoothly interpolates the error along the boundary of the source and the target regions across the entire cloned region (the resulting membrane is shown in (b)), yielding a seamless composite (e). A qualitatively similar membrane (c) may be achieved via transfinite interpolation, without solving a linear system. (f) Seamless cloning obtained instantly using the mean-value interpolant.*

Mean-value coordinates for smooth interpolation

these coordinates may be used to smoothly interpolate any function f defined at the boundary vertices:

$$\tilde{f}(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) f(\mathbf{p}_i). \quad (3)$$

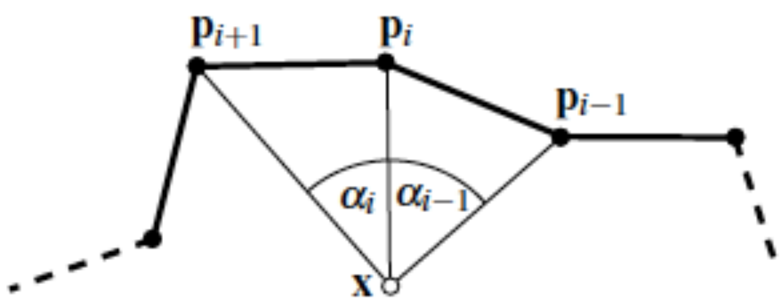


Figure 2: *Angle definitions for mean-value coordinates.*

- Evaluate them sparsely…
  - … and interpolate within triangles

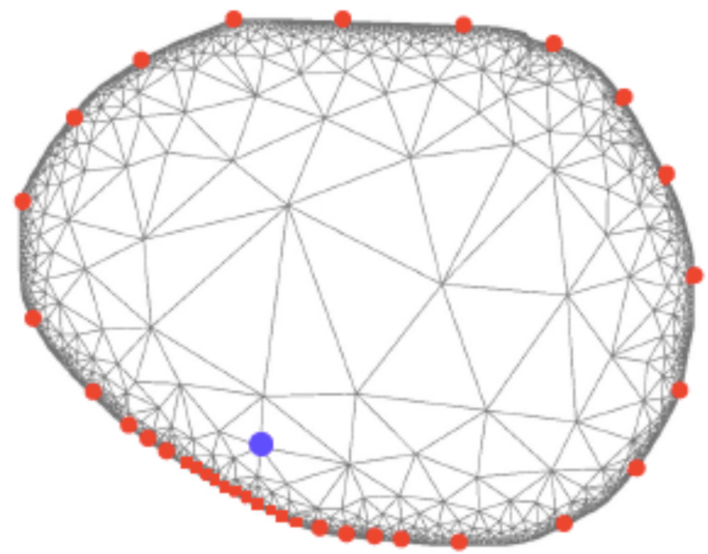


Figure 3: *An adaptive triangular mesh constructed over the region to be cloned. The red dots on the boundary show the positions of boundary vertices that were selected by adaptive hierarchical subsampling for the mesh vertex indicated in blue.*

Interactive Poisson cloning!

Table 1: *Performance statistics for MVC cloning. Times exclude disk I/O and sending the images to the graphics subsystem. Cloning rate is the number of region updates per second.*

#cloned pixels	#bdry pixels	#mesh vertices	coords /vertex	prep. time(s)	cloning rate CPU	cloning rate GPU
51,820	1,113	2,063	38.63	0.15	199.0	163
133,408	1,562	2,963	44.21	0.30	92.1	134
465,134	2,683	5,323	45.50	0.63	22.6	82
1,076,572	4,145	8,241	44.59	1.16	9.7	44
4,248,461	8,133	16,369	57.71	3.63	2.7	26
12,328,289	14,005	28,240	58.68	8.99	0.94	–

- Faster than “real” Poisson

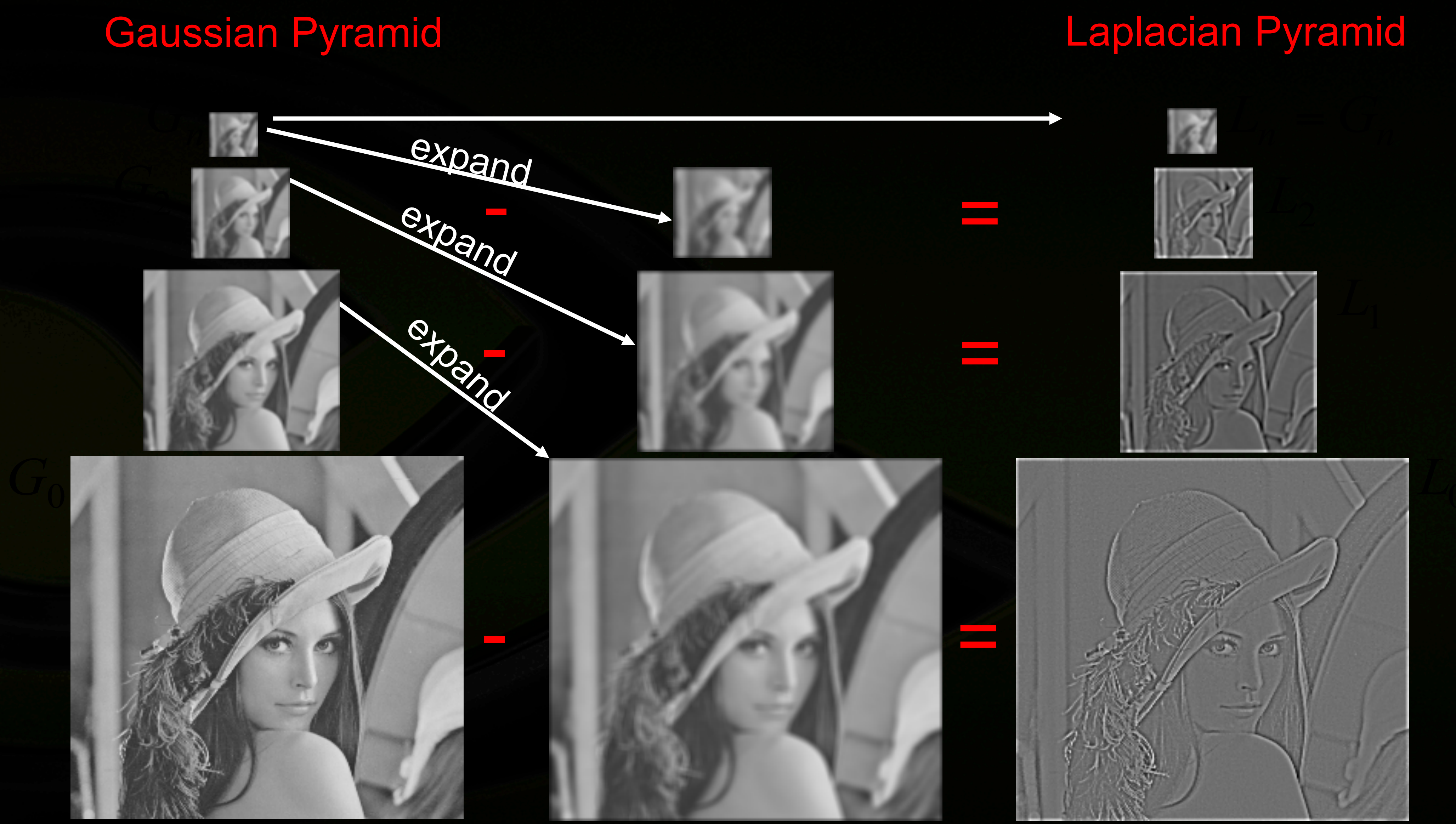
2008], as well as our own experiments, indicate that common Poisson solvers on the CPU are able to handle regions with 256^2 pixels at a rate of 3–5 solutions per second. Another possibility, which we have not seen mentioned in the literature, is to precompute a factorization of the Poisson equation matrix during the preprocessing stage, and then quickly compute the solution via back-substitution at each target location. In our experiments, for a region with 125K pixels, computing the back-substitution takes 0.3 seconds. Thus, all of the above are significantly slower than the rates we are able to achieve.

Alpha blending

After labeling

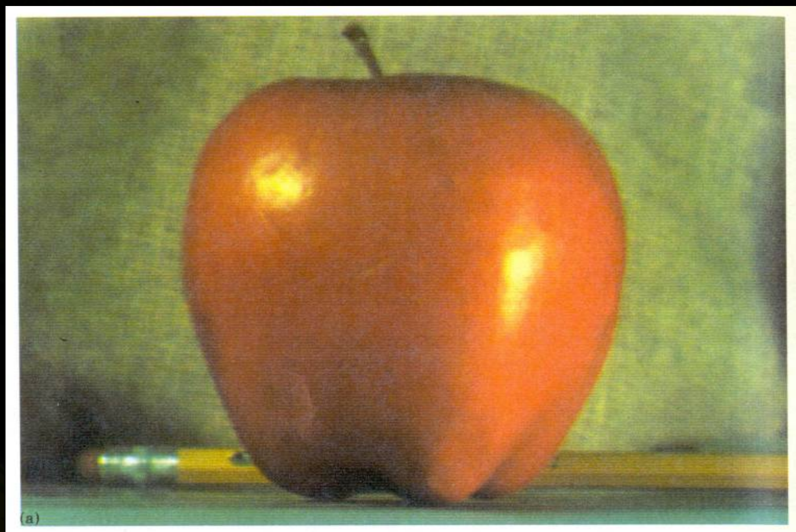
Poisson blending

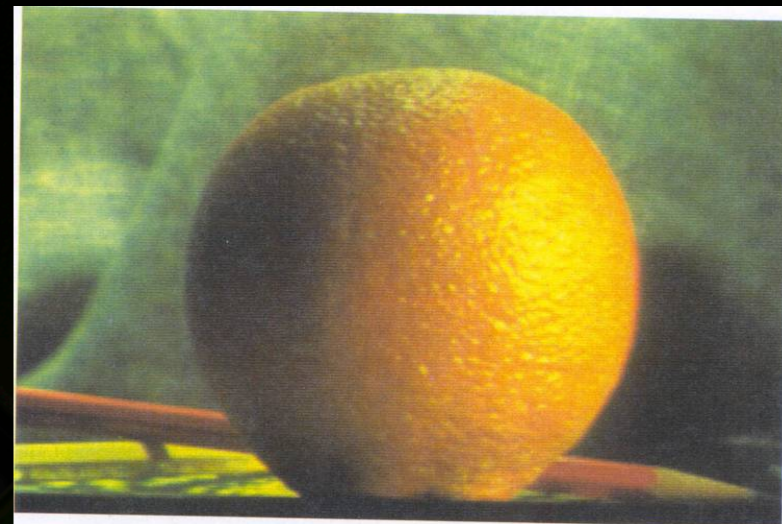
The Laplacian pyramid

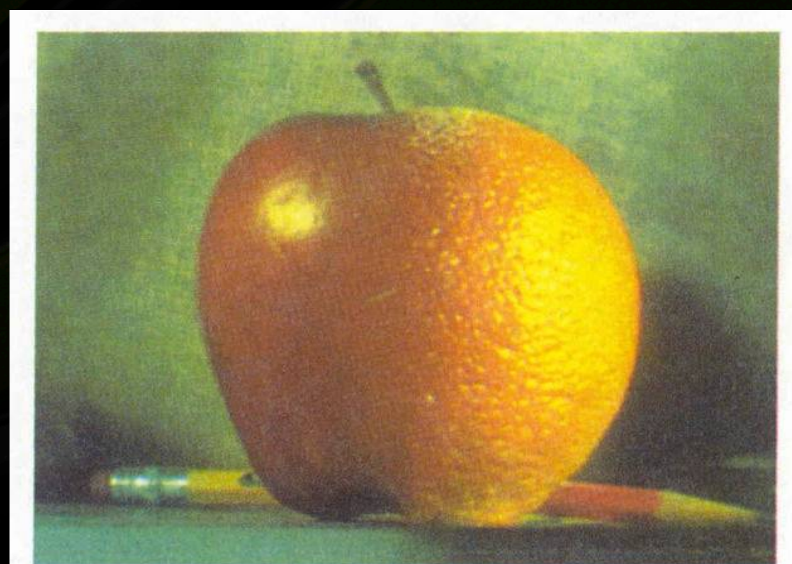


```
void createLaplacePyr( const Mat &img, int num_levels, std::vector<Mat> &pyr )
{
    pyr.resize(num_levels + 1);
    Mat downNext, lvl_up, lvl_down;
    Mat current = img;
    pyrDown(img, downNext);

    for( int i = 1; i < num_levels; ++i )
    {
        pyrDown( downNext, lvl_down );
        pyrUp( downNext, lvl_up, current.size() );
        subtract( current, lvl_up, pyr[i-1], noArray(), CV_16S );

        current  = downNext;
        downNext = lvl_down;
    }

    pyrUp( downNext, lvl_up, current.size() );
    subtract( current, lvl_up, pyr[num_levels-1], noArray(), CV_16S );
    downNext.convertTo( pyr[num_levels], CV_16S );
}
```

Pyramid Blending





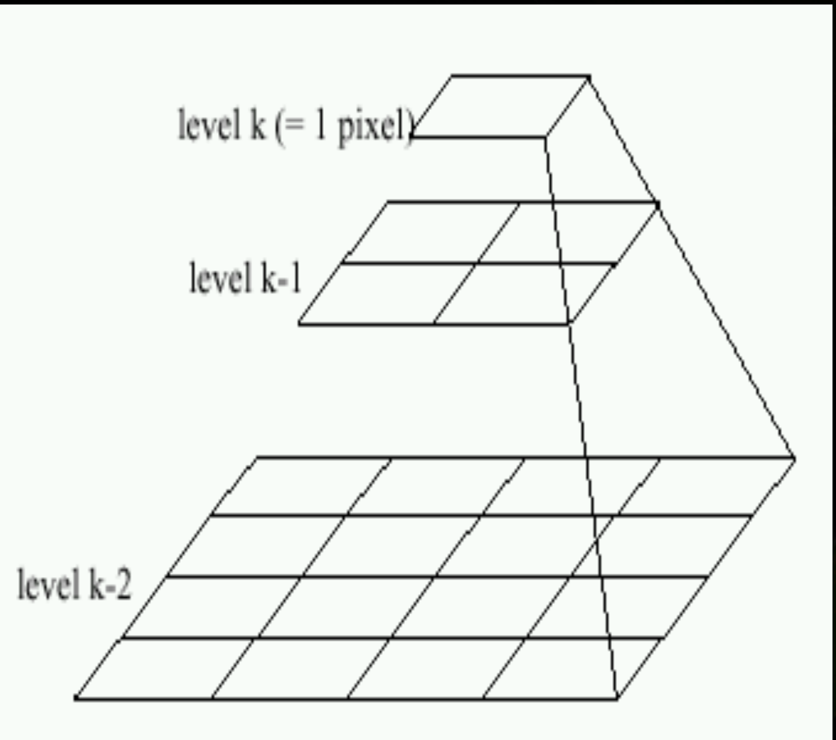


(1)

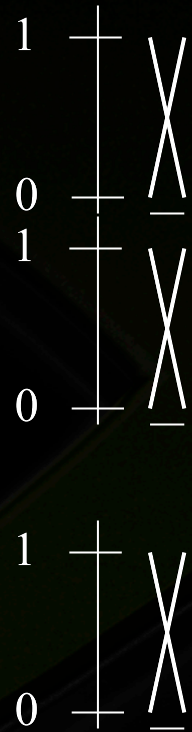
Laplacian Pyramid: Blending

- General Approach:
  1. Build Laplacian pyramids *LA* and *LB* from images *A* and *B*
  2. Build a Gaussian pyramid *GR* from selected region *R*
  3. Form a combined pyramid *LS* from *LA* and *LB* using nodes of *GR* as weights:
     - *LS(i,j) = GR(I,j,) * LA(I,j) + (1-GR(I,j)) * LB(I,j)*
  4. Collapse the *LS* pyramid to get the final blended image

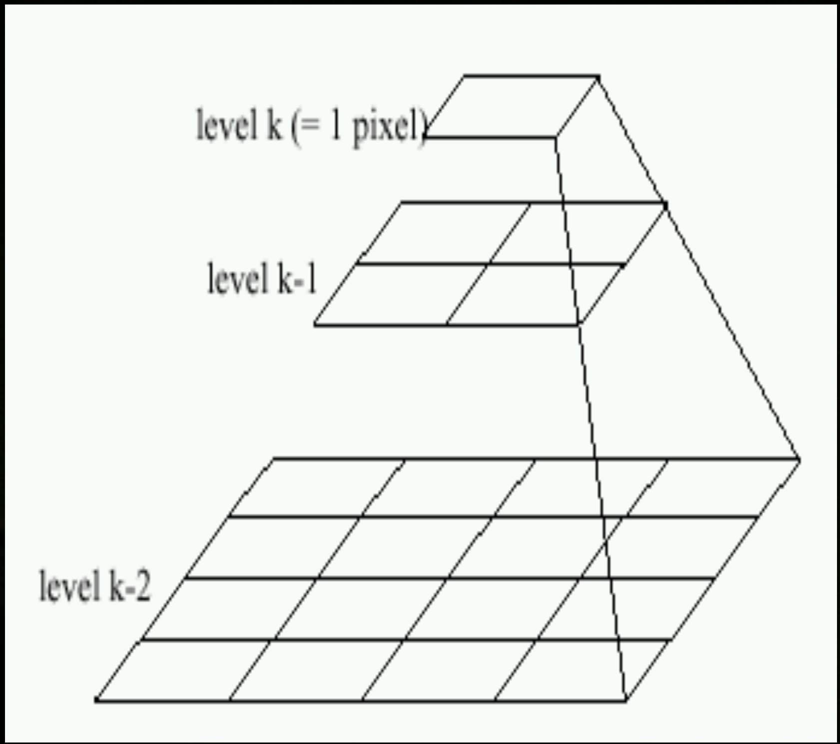
Pyramid Blending



Left pyramid



blend

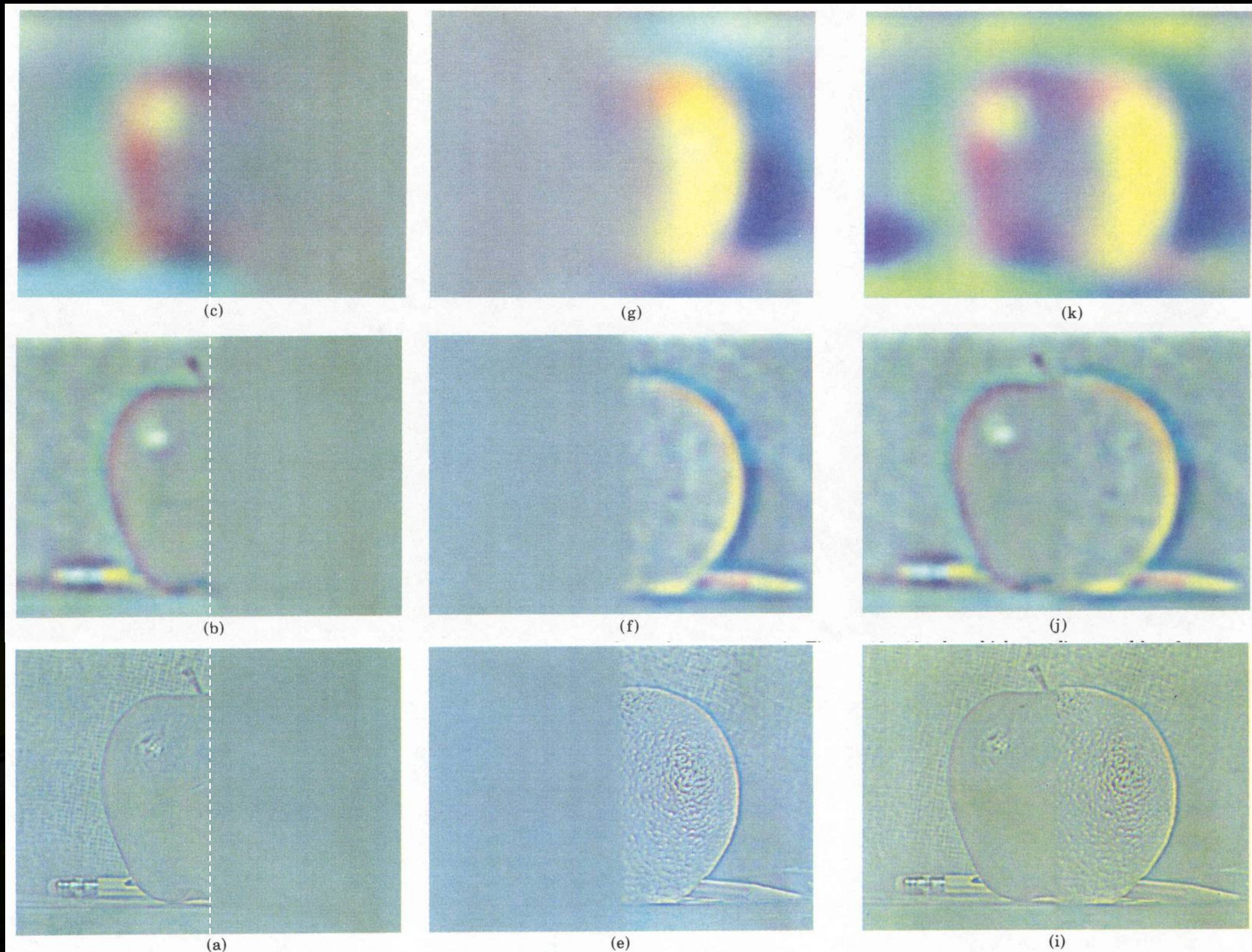


Right pyramid

Laplacian level 4

Laplacian level 2

Laplacian level 0

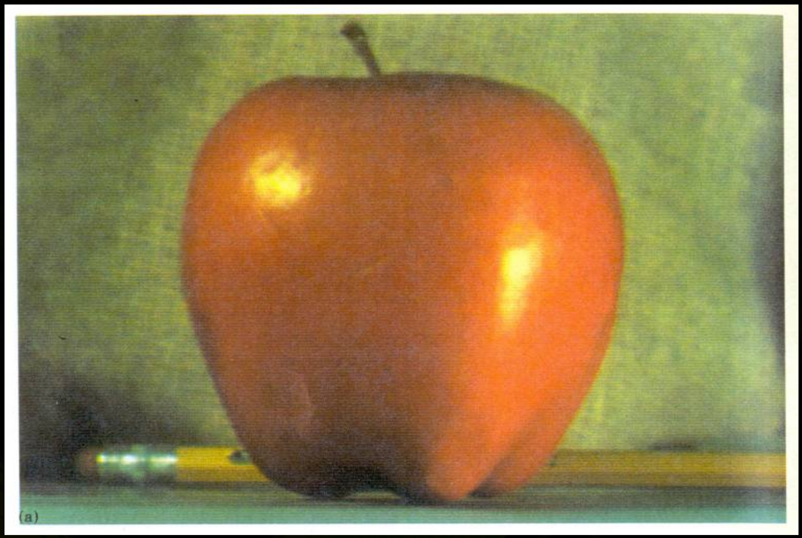


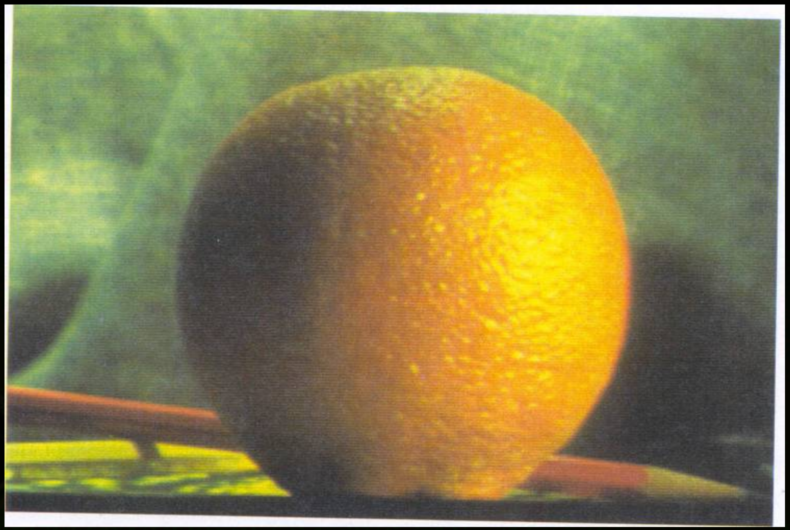
left pyramid

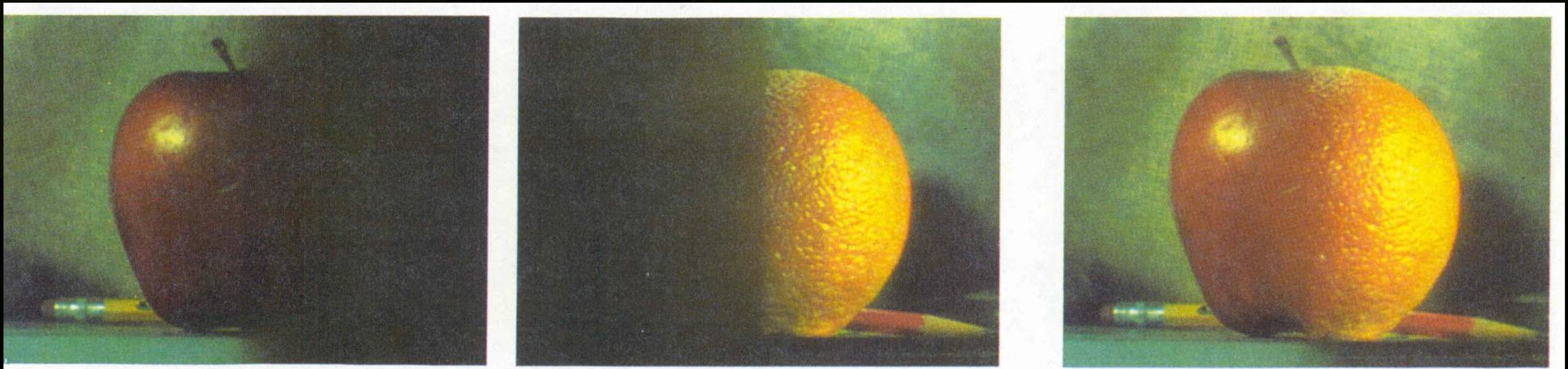
right pyramid

blended pyramid

Pyramid Blending







(d) (h) (l)

Simplification: Two-band Blending

- **Brown & Lowe, 2003**
  - **Only use two bands: high freq. and low freq.**
  - **Blends low freq. smoothly**

2-band Blending

Low frequency (λ > 2 pixels)

High frequency (λ < 2 pixels)

Linear Blending

2-band Blending