

# UNIT I

Computer Vision

ECSCI24402

# Syllabus

Unit	Content	Hrs.
1	<p>Overview of computer vision and its applications:</p> <p>Computer Vision and Computer Graphics: What is Computer Vision – Low level, Mid-level, High-level, Overview of Diverse Computer Vision Applications: Document Image Analysis, Biometrics, Object Recognition, Tracking, Medical Image Analysis, Content-Based Image Retrieval, Video Data Processing, Multimedia, Virtual Reality and Augmented Reality.</p>	5

# Topics to cover

- Overview of computer vision and its applications:
- Computer Vision and Computer Graphics:
  - What is Computer Vision – Low level, Mid-level, High-level,
- Overview of Diverse Computer Vision Applications:
  - Document Image Analysis, Biometrics, Object Recognition, Tracking, Medical Image Analysis, Content-Based Image Retrieval, Video Data Processing, Multimedia, Virtual Reality and Augmented Reality.

# What is Computer Vision?

- Definition: Computer Vision (CV) is an interdisciplinary field of Artificial Intelligence (AI) focused on enabling computers and systems to extract meaningful information from images, videos, and other visual inputs.
- Analogy to Human Vision:
  - Human vision: Eyes + brain (retina, optic nerve, cortex).
  - Computer vision: Cameras + algorithms + data.
- Objective: Mimic human perception to recognize, interpret, and act on visual inputs.

## Machine Vision



input



sensing device



machine



output

## Human Vision



»

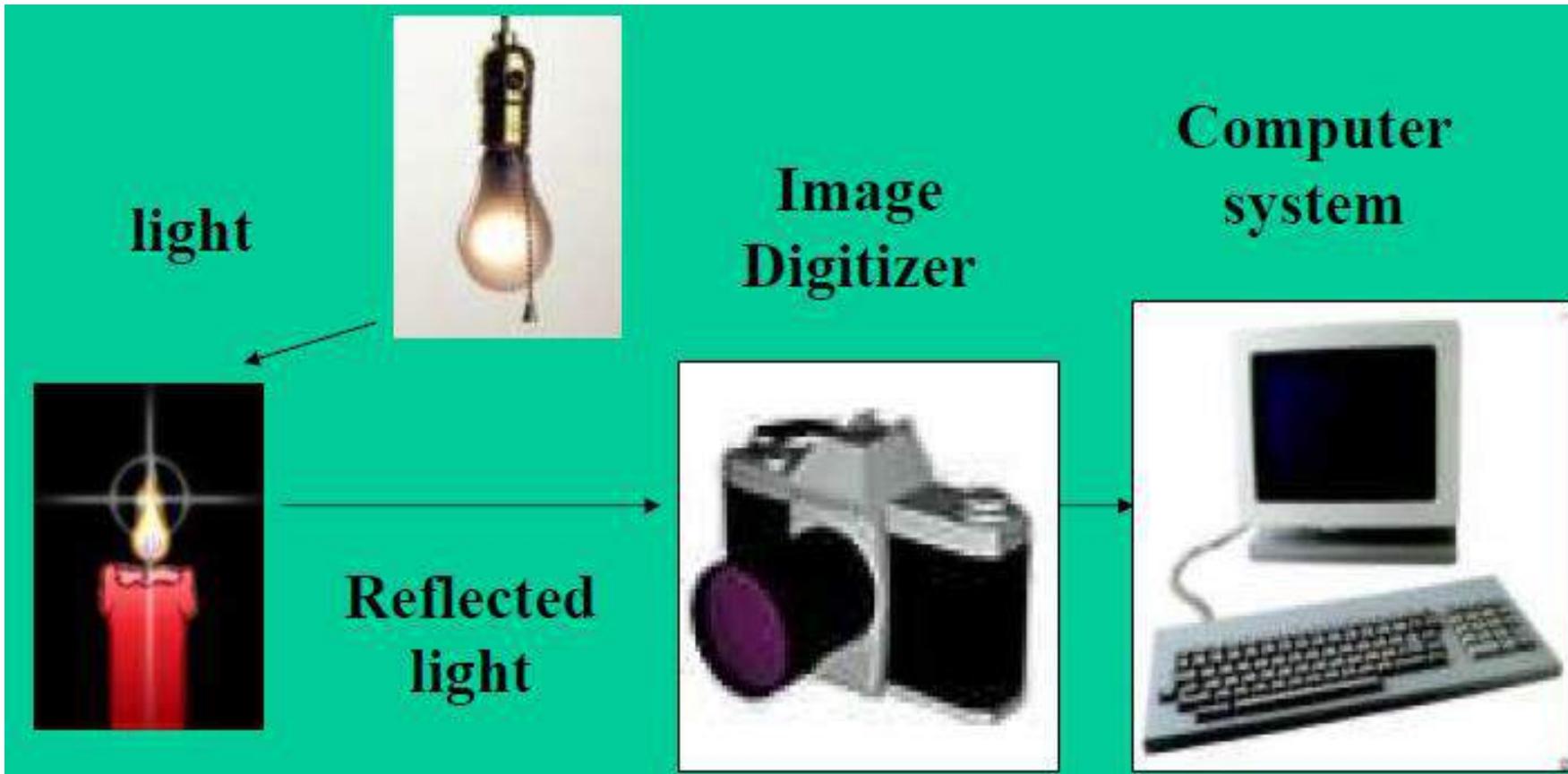


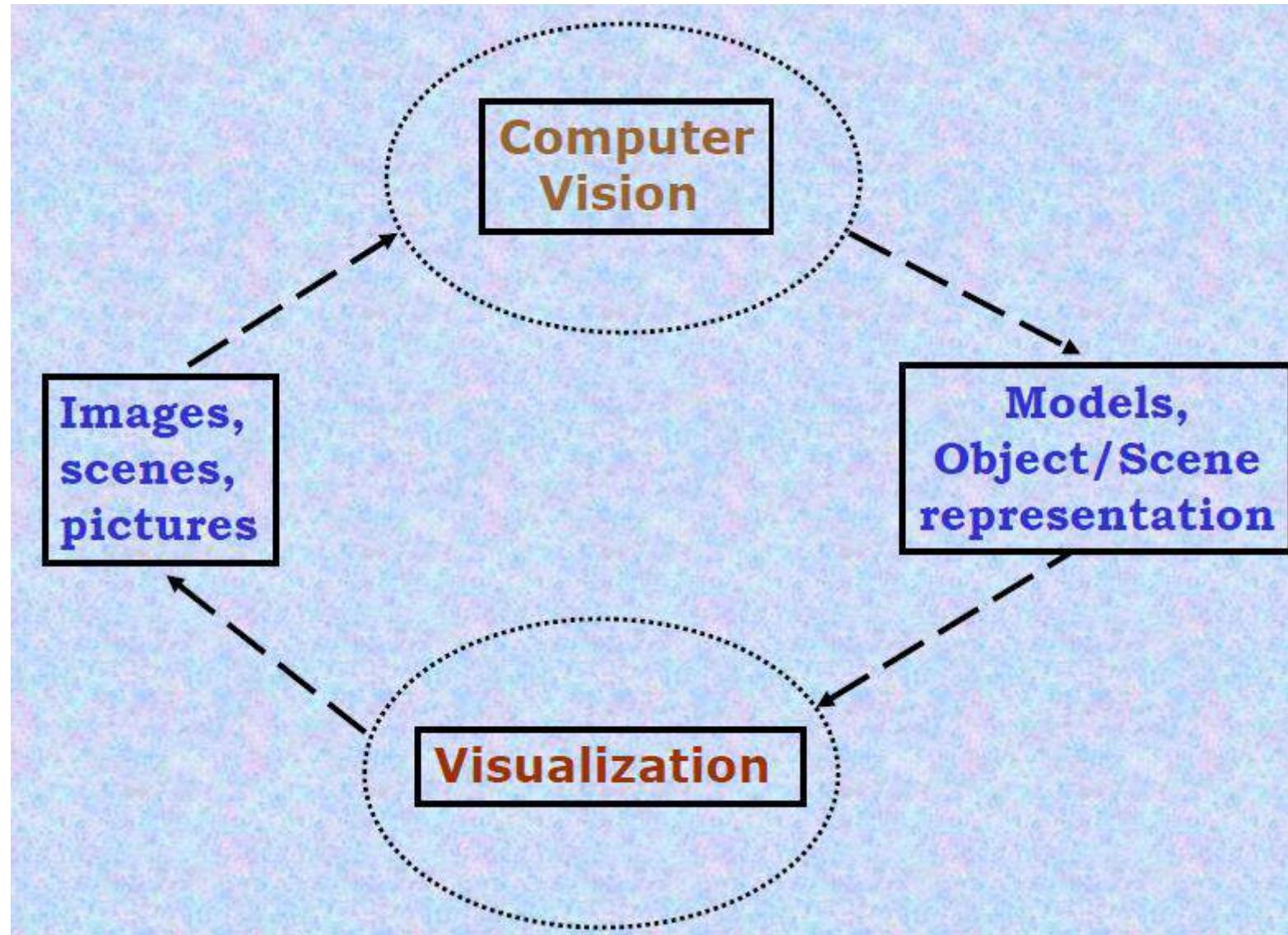
»



»

# Computer Vision System





# Why is Computer Vision Important?

- Automation & Efficiency: Enhances decision-making and process automation in real-time environments.
- Scalability: Can deploy “digital eyes” at any scale—drones, satellites, surveillance systems, etc.

# Wide Applications

- **Medical Imaging:** MRI, CT, X-ray analysis for disease detection.
- **Autonomous Vehicles:** Lane detection, obstacle avoidance, pedestrian tracking.
- **Retail & Fashion:** Visual search, virtual try-on systems, Amazon StyleSnap.
- **Security & Surveillance:** Intrusion detection, facial recognition.
- **Agriculture:** Crop health monitoring using drones.
- **Industry 4.0:** Automated defect detection in manufacturing.
- **Self-driving cars (Tesla Autopilot).**
- **Social media filters (Instagram, Snapchat).**

# Common Computer Vision Tasks

## Classification



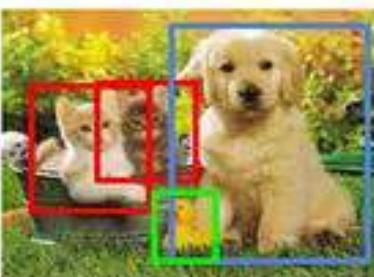
CAT

## Classification + Localization



CAT

## Object Detection



CAT, DOG, DUCK

Instance Segmentation



## CAT, DOG, DUCK

Semantic Segmentation



GRASS, CAT,  
TREE, SKY

### Single object

## Multiple objects

No objects, just pixels

## Object Tracking



## Face Detection

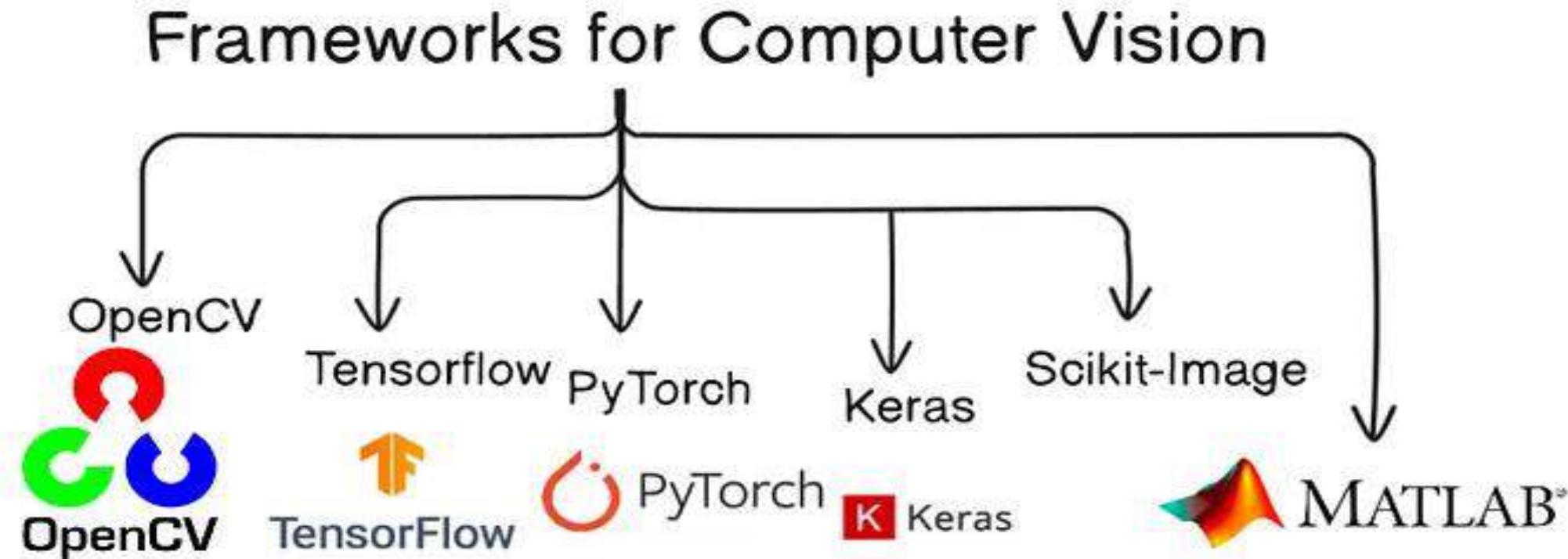


## Face Recognition

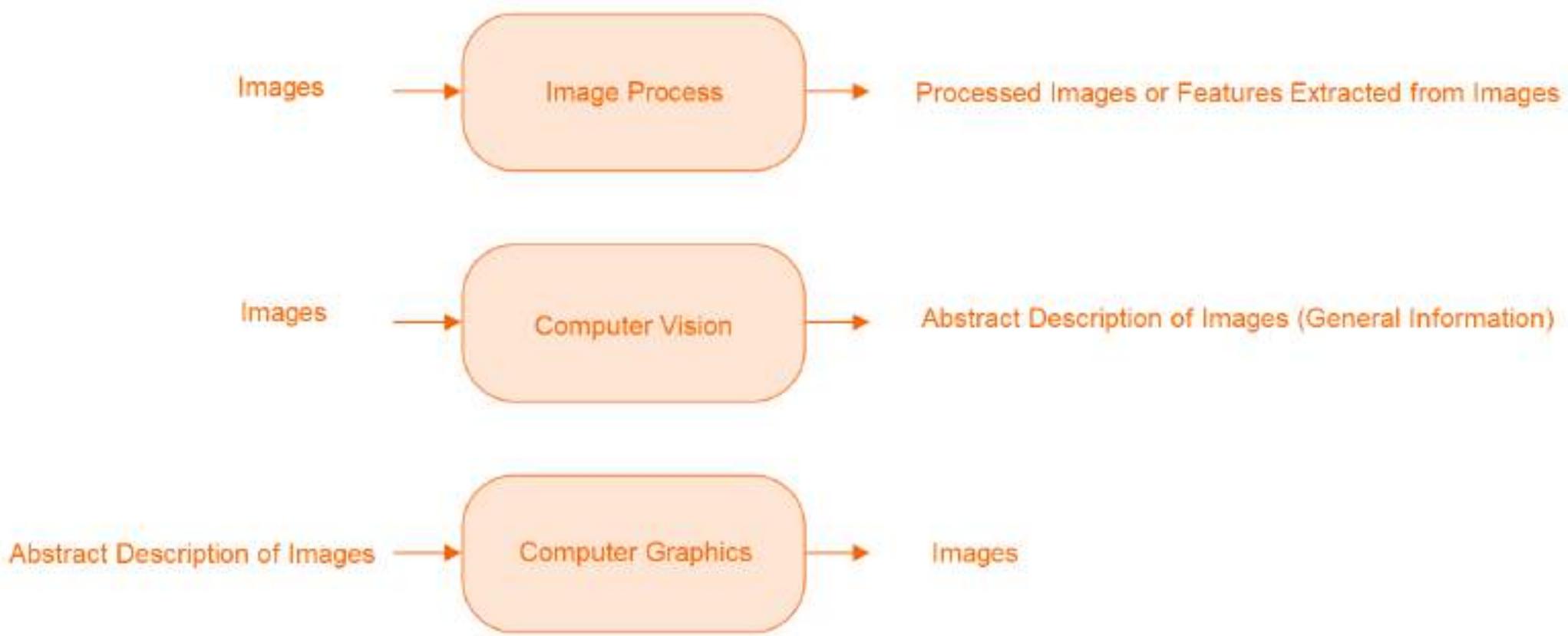
# Common Computer Vision Tasks

- Image Classification: Assign a label to an entire image.
- Object Detection: Locate and classify multiple objects within an image.
- Semantic Segmentation: Assign a class label to each pixel (e.g., background vs object).
- Instance Segmentation: Distinguish between different instances of the same object class.
- Pose Estimation: Detect body keypoints (skeleton tracking).
- OCR (Text Recognition): Extract printed/handwritten text from images.
- Image Captioning: Generate textual description of an image.

# Frameworks for Computer Vision



# Computer Vision and Computer Graphics



## The Three Stages of Computer Vision

- low-level (image processing)

image → image

- mid-level (feature extraction)

image → features

- high-level (the intelligent part)

features → analysis

# Computer Vision Hierarchy

## Low-Level Vision

- **Goal:** Process raw image data to extract **basic visual features**.
- **Tasks:**
  - Edge detection
  - Corner detection
  - Optical flow estimation
- **Output:** Primitive features such as edges, motion vectors, or gradients.
- **Use:** Acts as the foundation for higher-level interpretation.

# Computer Vision Hierarchy

## Mid-Level Vision

- **Goal:** Use features from low-level vision to perform **structured analysis**.
- **Tasks:**
  - Object recognition
  - Motion analysis
  - 3D reconstruction
- **Output:** Geometrical or grouped information like object boundaries, shapes, or trajectories.
- **Use:** Critical in tasks like tracking, detection, and environment modeling.

# Computer Vision Hierarchy

## High-Level Vision

- **Goal:** Interpret and reason about the visual content semantically.
- **Tasks:**
  - Scene understanding
  - Activity recognition
  - Behavior/intention prediction
- **Characteristics:**
  - Directs mid and low-level vision operations dynamically.
  - Adds **semantic context** to the observed scene.
- **Use:** Surveillance, autonomous systems, cognitive robotics.

## Low-Level



original image

Canny  
edge  
operator  
→



edge image

## Mid-Level (Lines and Curves)



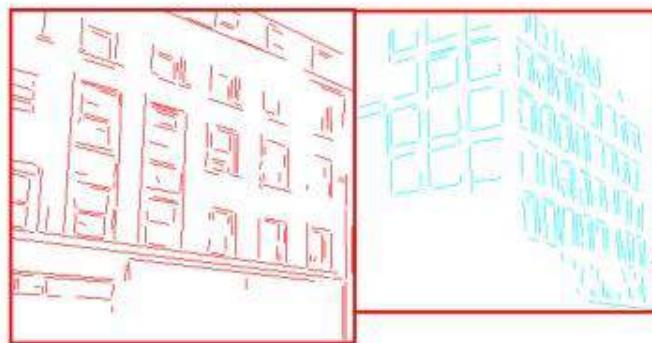
edge image

ORT  
line &  
circle  
extraction  
→  
data  
structure



circular arcs and line segments<sup>3</sup>

# Low- to High-Level



low-level



edge image

mid-level

high-level

consistent  
line clusters

## Building Recognition

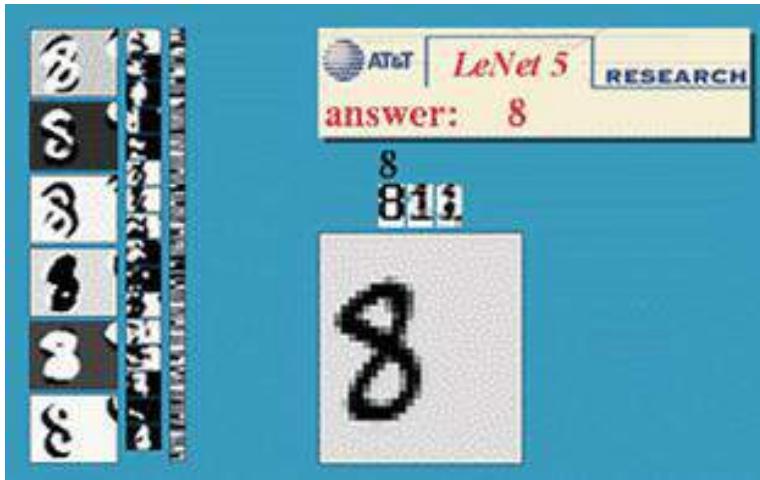
# Diverse Computer Vision Applications

# Document Image Analysis

- Extracts text and structure from scanned documents using OCR, layout detection.
- Used in digitization, archiving, and automation.

# Optical character recognition (OCR)

- If you have a scanner, it probably came with OCR software



Digit recognition, AT&T labs (1990's)  
<http://yann.lecun.com/exdb/lenet/>



License plate readers  
[http://en.wikipedia.org/wiki/Automatic\\_number\\_plate\\_recognition](http://en.wikipedia.org/wiki/Automatic_number_plate_recognition)



Automatic check processing



Sudoku grabber  
<http://sudokugrab.blogspot.com/>

# Biometrics

- Uses facial, iris, fingerprint or gait recognition for identification and verification in secure systems.

# Login without a password



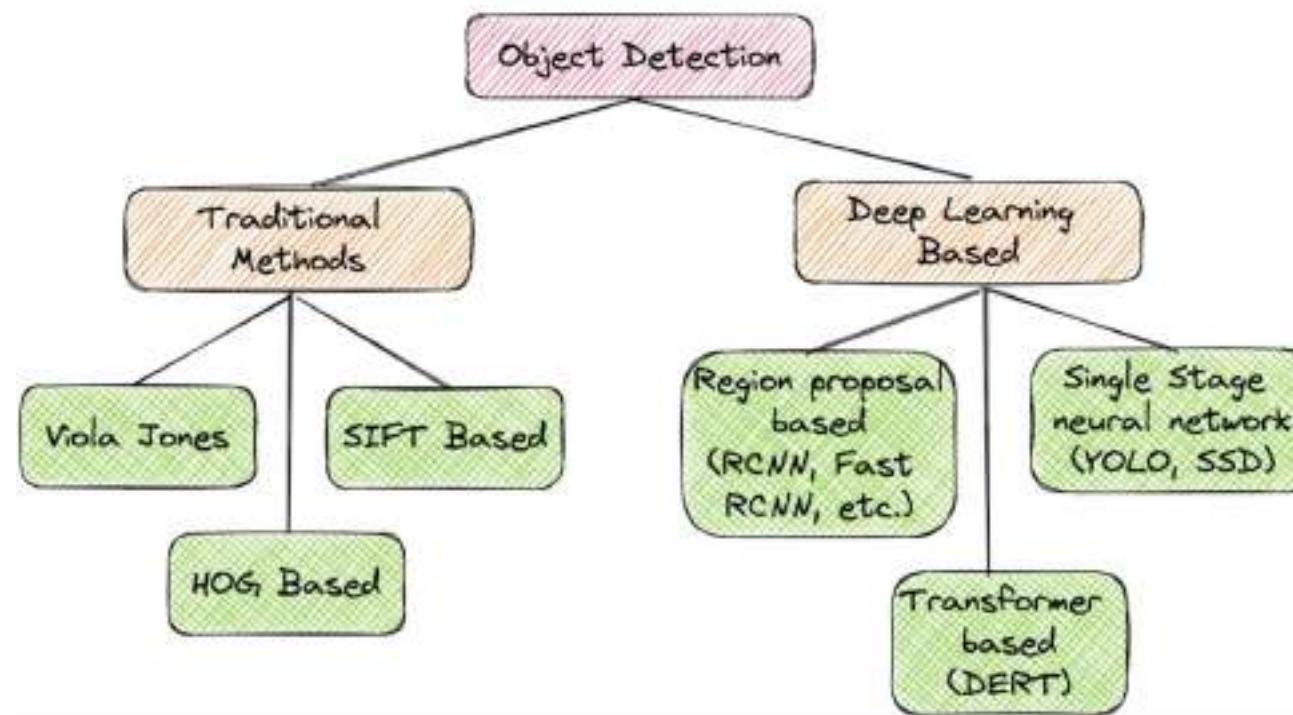
Fingerprint scanners on  
many new smartphones  
and other devices



Face unlock on Apple iPhone X  
See also <http://www.sensiblevision.com/>

# Object Recognition

- Classifies and locates objects in images using deep learning models.
- Applied in robotics, retail, and self-driving vehicles.



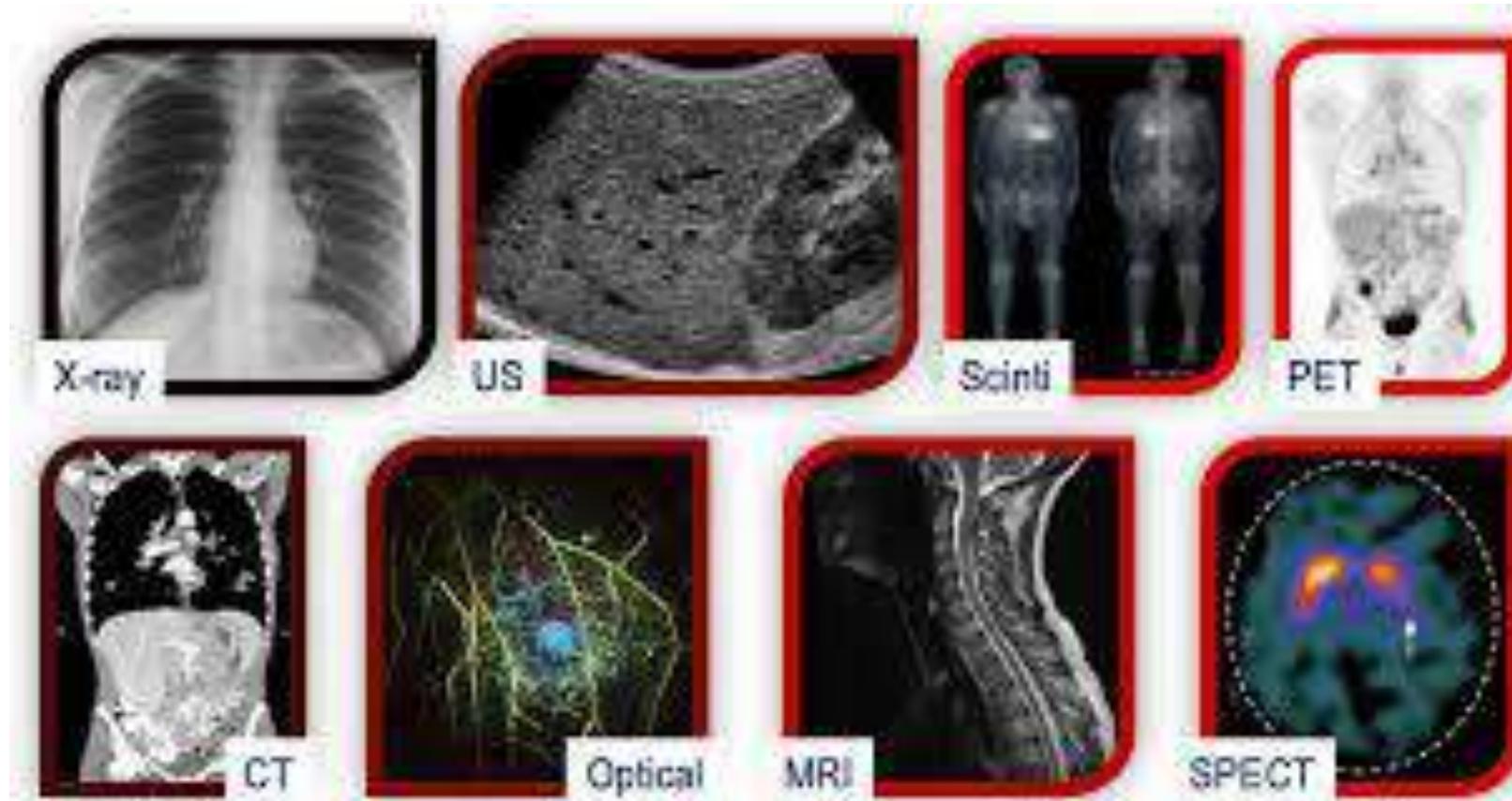


# Tracking

- Monitors moving objects across video frames. Used in surveillance, sports analytics, and UAV navigation.

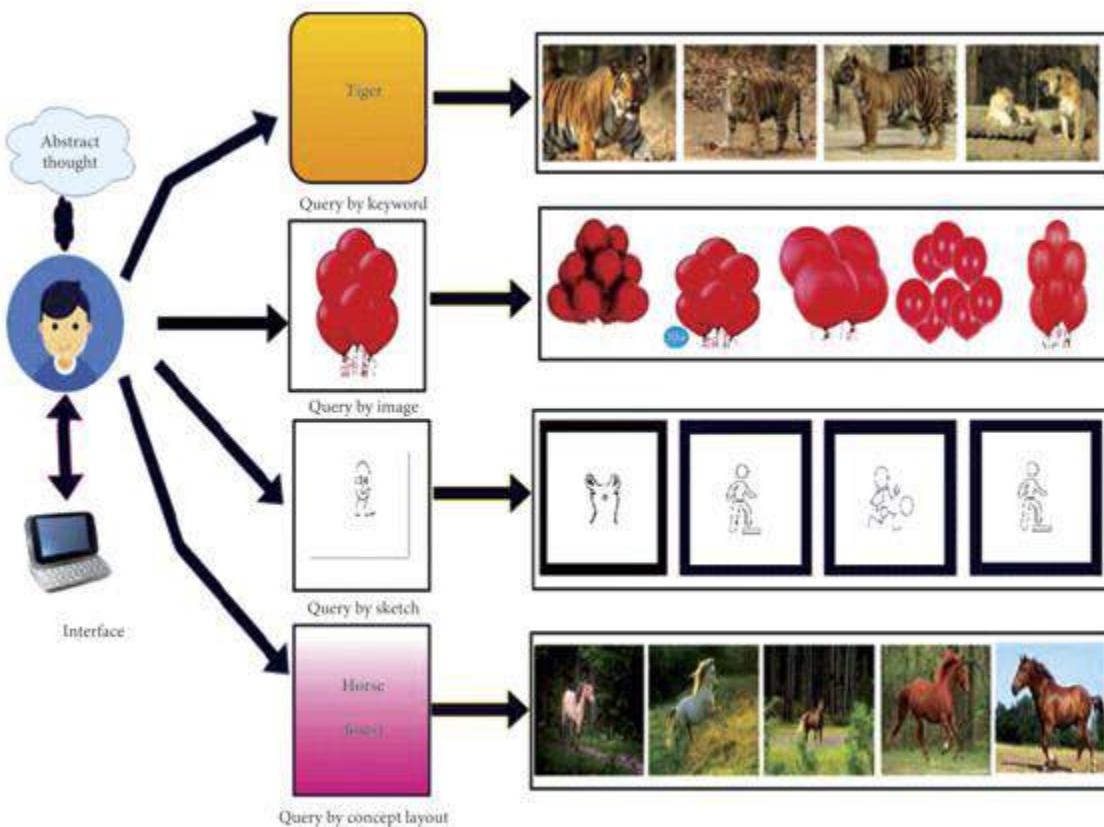
# Medical Image Analysis

- Analyzes MRI, CT, and X-rays for disease detection and surgical planning. Enhances diagnostic accuracy.



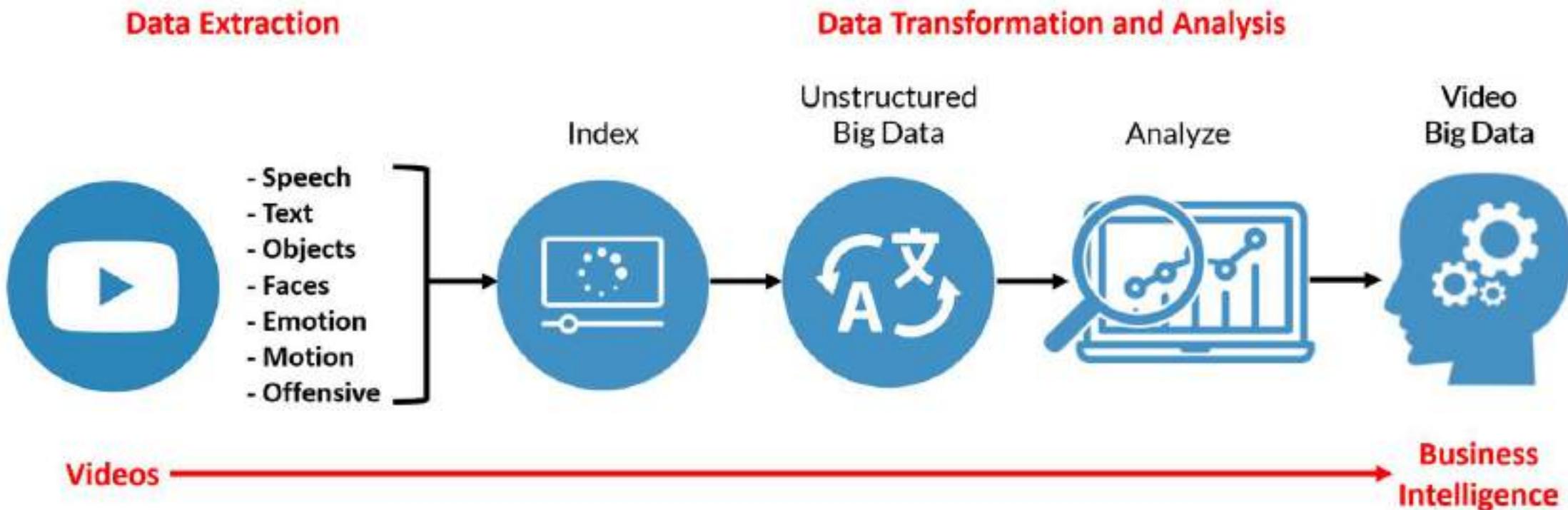
# Content-Based Image Retrieval (CBIR)

- Searches for images using visual features like color and shape rather than text metadata.



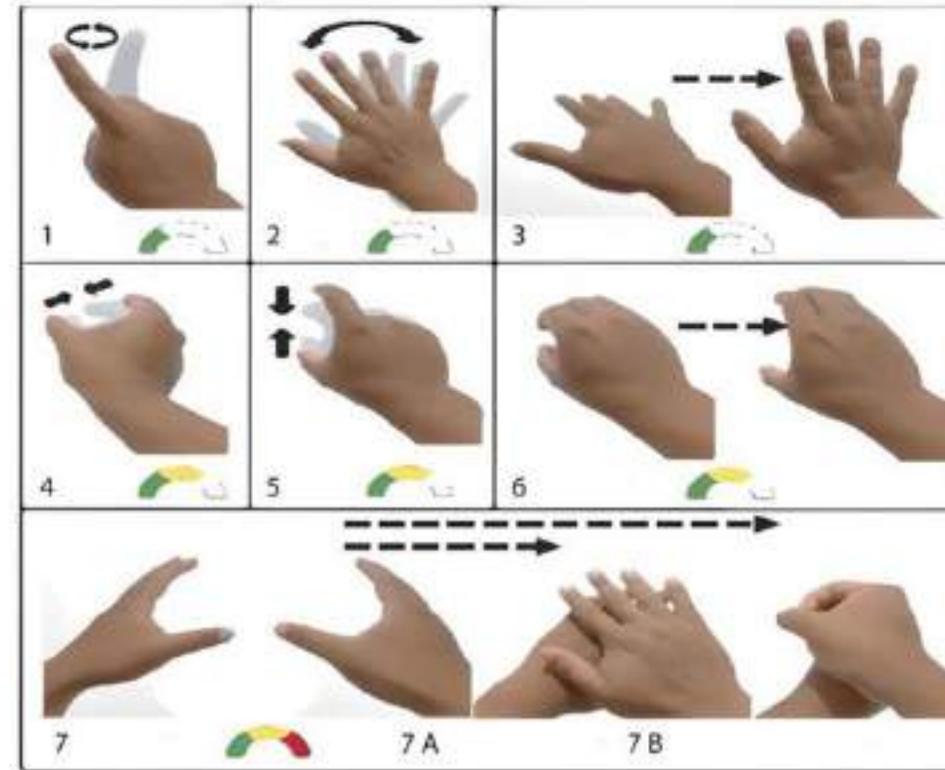
# Video Data Processing

- Processes frames in videos for activity recognition, summarization, and surveillance event detection.



# Multimedia

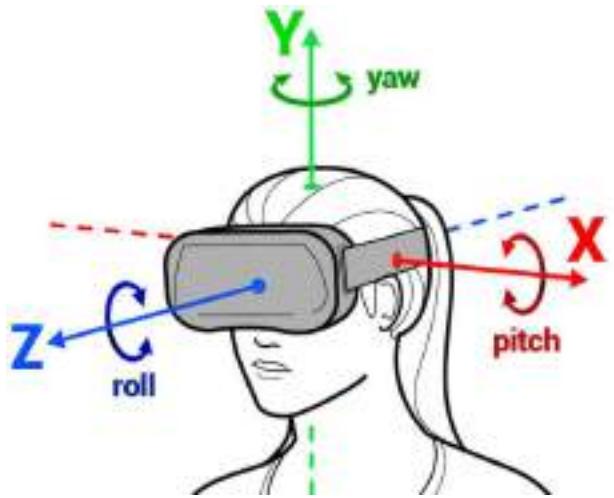
- Applies computer vision in media applications like filters, effects, gesture control, and video enhancement.



# Virtual Reality (VR) & Augmented Reality (AR)

- VR offers immersive 3D scenes, AR overlays information on real views. Used in gaming, education, and virtual try-ons.

# Virtual & Augmented Reality



6DoF head tracking



Hand & body tracking



3D scene understanding



3D-360 video capture

**Quiz!!!**

# Computer Vision Quiz (MCQ)

Choose the correct option from the given alternatives.

**1. Which of the following best defines Computer Vision?**

- A. Study of digital sound
- B. Automatic interpretation of images and videos
- C. Creating 3D models manually
- D. Writing text descriptions of algorithms



**Answer: B**

## **2. What is the main difference between Computer Graphics and Computer Vision?**

- A. Graphics uses images, vision uses text
- B. Computer Graphics generates images, Computer Vision analyzes images
- C. Vision uses only grayscale, Graphics uses color
- D. They are the same



**Answer: B**

# Which level of computer vision involves edge detection and corner detection?

A. High-level

B. Mid-level

C. Low-level

D. All of the above

# Which of the following is primarily a Mid-level vision task?

- A. Scene interpretation
- B. 3D reconstruction
- C. Edge filtering
- D. Object categorization

# What is the primary focus of high-level vision?

- A. Feature extraction
- B. Motion detection
- C. Scene interpretation
- D. 3D modeling

# Which technique is essential for extracting text from scanned documents?

- A. Biometrics
- B. Object Tracking
- C. Document Image Analysis
- D. Augmented Reality

# Which application uses facial geometry and fingerprint analysis?

- A. Medical Image Analysis
- B. Biometrics
- C. Object Detection
- D. Multimedia

# Which application uses feature matching across video frames?

A. Document Analysis

B. Tracking

C. Content Retrieval

D. VR

# **CBIR stands for:**

- A. Computer-Based Image Rendering**
- B. Content-Based Image Retrieval**
- C. Common Biometric Image Recognition**
- D. Camera-Based Inference Recognition**

# Which application is heavily used in diagnostic imaging?

- A. Object Recognition
- B. Medical Image Analysis
- C. Tracking
- D. Virtual Reality

# Which of the following enhances user experience through artificial interaction?

- A. Object Detection
- B. CBIR
- C. Virtual and Augmented Reality
- D. Medical Imaging

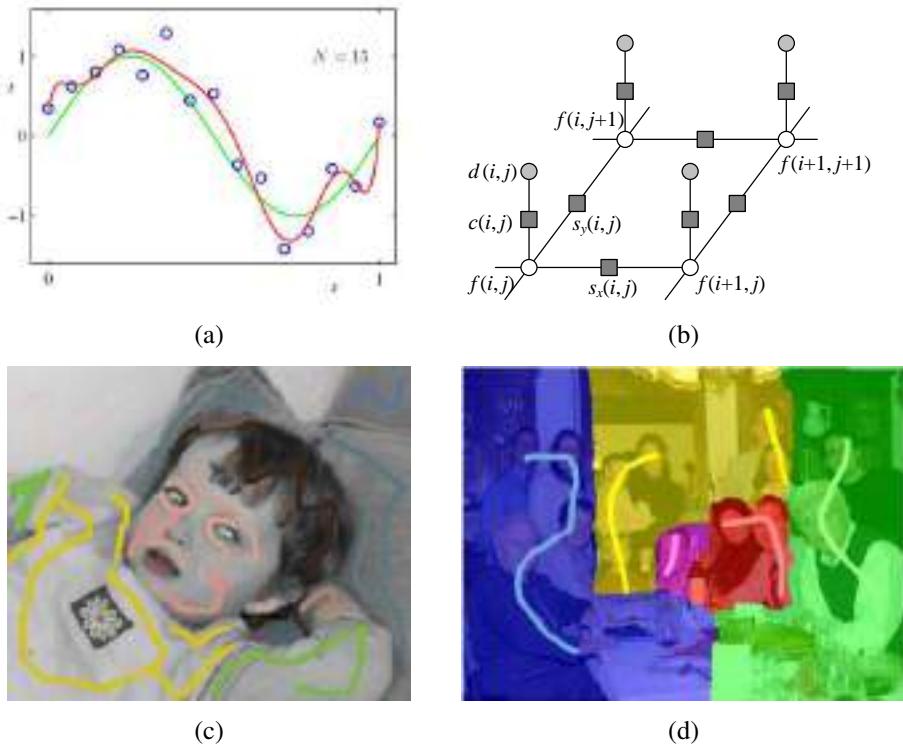
# Which of the following links computer vision with 3D rendering techniques?

- A. Biometrics
- B. Computer Vision and Graphics
- C. Content-Based Retrieval
- D. Medical Analysis

# Chapter 4

## Model fitting and optimization

4.1	Scattered data interpolation . . . . .	194
4.1.1	Radial basis functions . . . . .	196
4.1.2	Overfitting and underfitting . . . . .	199
4.1.3	Robust data fitting . . . . .	202
4.2	Variational methods and regularization . . . . .	204
4.2.1	Discrete energy minimization . . . . .	206
4.2.2	Total variation . . . . .	210
4.2.3	Bilateral solver . . . . .	210
4.2.4	<i>Application:</i> Interactive colorization . . . . .	211
4.3	Markov random fields . . . . .	212
4.3.1	Conditional random fields . . . . .	222
4.3.2	<i>Application:</i> Interactive segmentation . . . . .	227
4.4	Additional reading . . . . .	230
4.5	Exercises . . . . .	232



**Figure 4.1** Examples of data interpolation and global optimization: (a) scattered data interpolation (curve fitting) (Bishop 2006) © 2006 Springer; (b) graphical model interpretation of first-order regularization; (c) colorization using optimization (Levin, Lischinski, and Weiss 2004) © 2004 ACM; (d) multi-image photomontage formulated as an unordered label MRF (Agarwala, Dontcheva et al. 2004) © 2004 ACM.

In the previous chapter, we covered a large number of image processing operators that take as input one or more images and produce some filtered or transformed version of these images. In many situations, however, we are given *incomplete* data as input, such as depths at a sparse number of locations, or user scribbles suggesting how an image should be colorized or segmented (Figure 4.1c–d).

The problem of interpolating a complete image (or more generally a *function* or *field*) from incomplete or varying quality data is often called *scattered data interpolation*. We begin this chapter with a review of techniques in this area, since in addition to being widely used in computer vision, they also form the basis of most machine learning algorithms, which we will study in the next chapter.

Instead of doing an exhaustive survey, we present in Section 4.1 some easy-to-use techniques, such as triangulation, spline interpolation, and radial basis functions. While these techniques are widely used, they cannot easily be modified to provide *controlled continuity*, i.e., to produce the kinds of piecewise continuous reconstructions we expect when estimating depth maps, label maps, or even color images.

For this reason, we introduce in Section 4.2 *variational methods*, which formulate the interpolation problem as the recovery of a piecewise smooth function subject to exact or approximate data constraints. Because the smoothness is controlled using penalties formulated as norms of the function, this class of techniques are often called *regularization* or *energy-based* approaches. To find the minimum-energy solutions to these problems, we discretize them (typically on a pixel grid), resulting in a discrete energy, which can then be minimized using sparse linear systems or related iterative techniques.

In the last part of this chapter, Section 4.3, we show how such energy-based formulations are related to Bayesian inference techniques formulated as *Markov random fields*, which are a special case of general probabilistic *graphical models*. In these formulations, data constraints can be interpreted as noisy and/or incomplete measurements, and piecewise smoothness constraints as *prior assumptions* or *models* over the solution space. Such formulations are also often called *generative models*, since we can, in principle, generate random samples from the prior distribution to see if they conform with our expectations. Because the prior models can be more complex than simple smoothness constraints, and because the solution space can have multiple local minima, more sophisticated optimization techniques have been developed, which we discuss in this section.

## 4.1 Scattered data interpolation

The goal of *scattered data interpolation* is to produce a (usually continuous and smooth) function  $\mathbf{f}(\mathbf{x})$  that passes *through* a set of data points  $\mathbf{d}_k$  placed at locations  $\mathbf{x}_k$  such that

$$\mathbf{f}(\mathbf{x}_k) = \mathbf{d}_k. \quad (4.1)$$

The related problem of *scattered data approximation* only requires the function to pass *near* the data points (Amidror 2002; Wendland 2004; Anjyo, Lewis, and Pighin 2014). This is usually formulated using a penalty function such as

$$E_D = \sum_k \|\mathbf{f}(\mathbf{x}_k) - \mathbf{d}_k\|^2, \quad (4.2)$$

with the squared norm in the above formula sometimes replaced by a different norm or robust function (Section 4.1.3). In statistics and machine learning, the problem of predicting an output function given a finite number of samples is called *regression* (Section 5.1). The  $\mathbf{x}$  vectors are called the *inputs* and the outputs  $\mathbf{y}$  are called the *targets*. Figure 4.1a shows an example of one-dimensional scattered data interpolation, while Figures 4.2 and 4.8 show some two-dimensional examples.

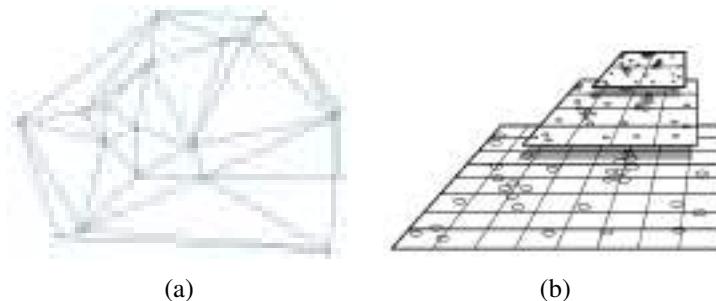
At first glance, scattered data interpolation seems closely related to *image interpolation*, which we studied in Section 3.5.1. However, unlike images, which are regularly gridded, the data points in scattered data interpolation are irregularly placed throughout the domain, as shown in Figure 4.2. This requires some adjustments to the interpolation methods we use.

If the domain  $\mathbf{x}$  is two-dimensional, as is the case with images, one simple approach is to *triangulate* the domain  $\mathbf{x}$  using the data locations  $\mathbf{x}_k$  as the triangle vertices. The resulting triangular network, shown in Figure 4.2a, is called a *triangular irregular network* (TIN), and was one of the early techniques used to produce elevation maps from scattered field measurements collected by surveys.

The triangulation in Figure 4.2a was produced using a *Delaunay triangulation*, which is the most widely used planar triangulation technique due to its attractive computational properties, such as the avoidance of long skinny triangles. Algorithms for efficiently computing such triangulation are readily available<sup>1</sup> and covered in textbooks on computational geometry (Preparata and Shamos 1985; de Berg, Cheong *et al.* 2008). The Delaunay triangulation can be extended to higher-dimensional domains using the property of circumscribing spheres, i.e., the requirement that all selected simplices (triangles, tetrahedra, etc.) have no other vertices inside their circumscribing spheres.

---

<sup>1</sup>For example, <https://docs.scipy.org/doc/scipy/reference/tutorial/spatial.html>



**Figure 4.2** Some simple scattered data interpolation and approximation algorithms: (a) a Delaunay triangulation defined over a set of data point locations; (b) data structure and intermediate results for the pull-push algorithm (Gortler, Grzeszczuk et al. 1996) © 1996 ACM.

Once the triangulation has been defined, it is straightforward to define a piecewise-linear interpolant over each triangle, resulting in an interpolant that is  $C_0$  but not generally  $C_1$  continuous. The formulas for the function inside each triangle are usually derived using *barycentric coordinates*, which attain their maximal values at the vertices and sum up to one (Farin 2002; Amidror 2002).

If a smoother surface is desired as the interpolant, we can replace the piecewise linear functions on each triangle with higher-order *splines*, much as we did for image interpolation (Section 3.5.1). However, since these splines are now defined over irregular triangulations, more sophisticated techniques must be used (Farin 2002; Amidror 2002). Other, more recent interpolators based on geometric modeling techniques in computer graphics include subdivision surfaces (Peters and Reif 2008).

An alternative to triangulating the data points is to use a regular  $n$ -dimensional grid, as shown in Figure 4.2b. Splines defined on such domains are often called *tensor product splines* and have been used to interpolate scattered data (Lee, Wolberg, and Shin 1997).

An even faster, but less accurate, approach is called the *pull-push* algorithm and was originally developed for interpolating missing 4D lightfield samples in a Lumigraph (Gortler, Grzeszczuk et al. 1996). The algorithm proceeds in three phases, as schematically illustrated in Figure 4.2b.

First, the irregular data samples are *splatted* onto (i.e., spread across) the nearest grid vertices, using the same approach we discussed in Section 3.6.1 on parametric image transformations. The splatting operations accumulate both values and weights at nearby vertices. In the second, *pull*, phase, values and weights are computed at a hierarchical set of lower resolution grids by combining the coefficient values from the higher resolution grids. In the lower

resolution grids, the gaps (regions where the weights are low) become smaller. In the third, *push*, phase, information from each lower resolution grid is combined with the next higher resolution grid, filling in the gaps while not unduly blurring the higher resolution information already computed. Details of these three stages can be found in (Gortler, Grzeszczuk *et al.* 1996).

The pull-push algorithm is very fast, since it is essentially linear in the number of input data points and fine-level grid samples.

### 4.1.1 Radial basis functions

While the mesh-based representations I have just described can provide good-quality interpolants, they are typically limited to low-dimensional domains, because the size of the mesh grows combinatorially with the dimensionality of the domain. In higher dimensions, it is common to use *mesh-free* approaches that define the desired interpolant as a weighted sum of basis functions, similar to the formulation used in image interpolation (3.64). In machine learning, such approaches are often called *kernel functions* or *kernel regression* (Bishop 2006, Chapter 6; Murphy 2012, Chapter 14; Schölkopf and Smola 2001).

In more detail, the interpolated function  $f$  is a weighted sum (or *superposition*) of basis functions centered at each input data point

$$\mathbf{f}(\mathbf{x}) = \sum_k \mathbf{w}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|), \quad (4.3)$$

where the  $\mathbf{x}_k$  are the locations of the scattered data points, the  $\phi$ s are the *radial basis functions* (or kernels), and  $\mathbf{w}_k$  are the local *weights* associated with each kernel. The basis functions  $\phi()$  are called *radial* because they are applied to the radial distance between a data sample  $\mathbf{x}_k$  and an evaluation point  $\mathbf{x}$ . The choice of  $\phi$  determines the smoothness properties of the interpolant, while the choice of weights  $w_k$  determines how closely the function approximates the input.

Some commonly used basis functions (Anjyo, Lewis, and Pighin 2014) include

$$\text{Gaussian} \qquad \qquad \qquad \phi(r) = \exp(-r^2/c^2) \quad (4.4)$$

$$\text{Hardy multiquadric} \qquad \qquad \qquad \phi(r) = \sqrt{(r^2 + c^2)} \quad (4.5)$$

$$\text{Inverse multiquadric} \qquad \qquad \qquad \phi(r) = 1/\sqrt{(r^2 + c^2)} \quad (4.6)$$

$$\text{Thin plate spline} \qquad \qquad \qquad \phi(r) = r^2 \log r. \quad (4.7)$$

In these equations,  $r$  is the radial distance and  $c$  is a scale parameter that controls the size (radial falloff) of the basis functions, and hence its smoothness (more compact bases lead to

“peakier” solutions). The thin plate spline equation holds for two dimensions (the general  $n$ -dimensional spline is called the *polyharmonic spline* and is given in (Anjyo, Lewis, and Pighin 2014)) and is the analytic solution to the second degree variational spline derived in (4.19).

If we want our function to exactly interpolate the data values, we solve the linear system of equations (4.1), i.e.,

$$\mathbf{f}(\mathbf{x}_k) = \sum_l \mathbf{w}_l \phi(\|\mathbf{x}_k - \mathbf{x}_l\|) = \mathbf{d}_k, \quad (4.8)$$

to obtain the desired set of weights  $\mathbf{w}_k$ . Note that for large amounts of basis function overlap (large values of  $c$ ), these equations may be quite *ill-conditioned*, i.e., small changes in data values or locations can result in large changes in the interpolated function. Note also that the solution of such a system of equations is in general  $O(m^3)$ , where  $m$  is the number of data points (unless we use basis functions with finite extent to obtain a sparse set of equations).

A more prudent approach is to solve the *regularized data approximation problem*, which involves minimizing the data constraint energy (4.2) together with a weight penalty (*regularizer*) of the form

$$E_W = \sum_k \|\mathbf{w}_k\|^p, \quad (4.9)$$

and to then minimize the regularized least squares problem

$$E(\{w_k\}) = E_D + \lambda E_W \quad (4.10)$$

$$= \sum_k \left\| \sum_l \mathbf{w}_l \phi(\|\mathbf{x}_k - \mathbf{x}_l\|) - \mathbf{d}_k \right\|^2 + \lambda \sum_k \|\mathbf{w}_k\|^p. \quad (4.11)$$

When  $p = 2$  (quadratic weight penalty), the resulting energy is a pure least squares problem, and can be solved using the *normal equations* (Appendix A.2), where the  $\lambda$  value gets added along the diagonal to stabilize the system of equations.

In statistics and machine learning, the quadratic (regularized least squares) problem is called *ridge regression*. In neural networks, adding a quadratic penalty on the weights is called *weight decay*, because it encourages weights to decay towards zero (Section 5.3.3). When  $p = 1$ , the technique is called *lasso* (least absolute shrinkage and selection operator), since for sufficiently large values of  $\lambda$ , many of the weights  $\mathbf{w}_k$  get driven to zero (Tibshirani 1996; Bishop 2006; Murphy 2012; Deisenroth, Faisal, and Ong 2020). This results in a *sparse* set of basis functions being used in the interpolant, which can greatly speed up the computation of new values of  $\mathbf{f}(\mathbf{x})$ . We will have more to say on sparse kernel techniques in the section on Support Vector Machines (Section 5.1.4).

An alternative to solving a set of equations to determine the weights  $\mathbf{w}_k$  is to simply set them to the input data values  $\mathbf{d}_k$ . However, this fails to interpolate the data, and instead

produces higher values in higher density regions. This can be useful if we are trying to estimate a probability density function from a set of samples. In this case, the resulting density function, obtained after normalizing the sum of sample-weighted basis functions to have a unit integral, is called the *Parzen window* or *kernel* approach to probability density estimation (Duda, Hart, and Stork 2001, Section 4.3; Bishop 2006, Section 2.5.1). Such probability densities can be used, among other things, for (spatially) clustering color values together for image segmentation in what is known as the *mean shift* approach (Comaniciu and Meer 2002) (Section 7.5.2).

If, instead of just estimating a density, we wish to actually interpolate a set of data values  $\mathbf{d}_k$ , we can use a related technique known as *kernel regression* or the *Nadaraya-Watson* model, in which we divide the data-weighted summed basis functions by the sum of all the basis functions,

$$\mathbf{f}(\mathbf{x}) = \frac{\sum_k \mathbf{d}_k \phi(\|\mathbf{x} - \mathbf{x}_k\|)}{\sum_l \phi(\|\mathbf{x} - \mathbf{x}_l\|)}. \quad (4.12)$$

Note how this operation is similar, in concept, to the *splatting* method for forward rendering we discussed in Section 3.6.1, except that here, the bases can be much wider than the nearest-neighbor bilinear bases used in graphics (Takeda, Farsiu, and Milanfar 2007).

Kernel regression is equivalent to creating a new set of spatially varying normalized shifted basis functions

$$\phi'_k(\mathbf{x}) = \frac{\phi(\|\mathbf{x} - \mathbf{x}_k\|)}{\sum_l \phi(\|\mathbf{x} - \mathbf{x}_l\|)}, \quad (4.13)$$

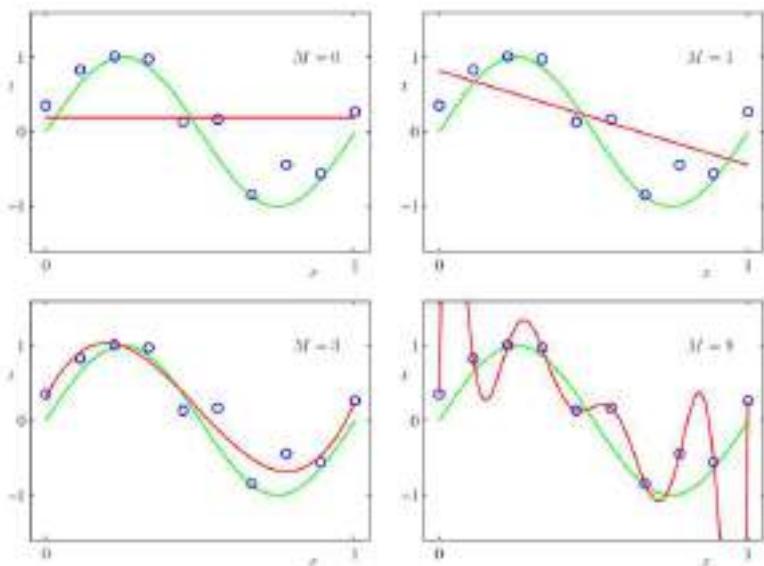
which form a *partition of unity*, i.e., sum up to 1 at every location (Anjyo, Lewis, and Pighin 2014). While the resulting interpolant can now be written more succinctly as

$$\mathbf{f}(\mathbf{x}) = \sum_k \mathbf{d}_k \phi'_k(\|\mathbf{x} - \mathbf{x}_k\|), \quad (4.14)$$

in most cases, it is more expensive to precompute and store the  $K$   $\phi'_k$  functions than to evaluate (4.12).

While not that widely used in computer vision, kernel regression techniques have been applied by Takeda, Farsiu, and Milanfar (2007) to a number of low-level image processing operations, including state-of-the-art handheld multi-frame super-resolution (Wronski, Garcia-Dorado *et al.* 2019).

One last scattered data interpolation technique worth mentioning is *moving least squares*, where a weighted subset of nearby points is used to compute a local smooth surface. Such techniques are mostly widely used in 3D computer graphics, especially for point-based surface modeling, as discussed in Section 13.4 and (Alexa, Behr *et al.* 2003; Pauly, Keiser *et al.* 2003; Anjyo, Lewis, and Pighin 2014).



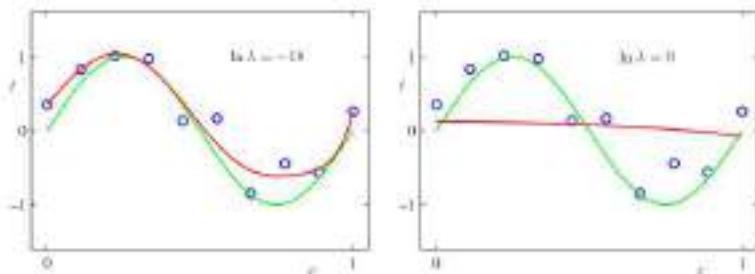
**Figure 4.3** Polynomial curve fitting to the blue circles, which are noisy samples from the green sine curve (Bishop 2006) © 2006 Springer. The four plots show the 0th order constant function, the first order linear fit, the  $M = 3$  cubic polynomial, and the 9th degree polynomial. Notice how the first two curves exhibit underfitting, while the last curve exhibits overfitting, i.e., excessive wobble.

### 4.1.2 Overfitting and underfitting

When we introduced weight regularization in (4.9), we said that it was usually preferable to approximate the data but we did not explain why. In most data fitting problems, the samples  $\mathbf{d}_k$  (and sometimes even their locations  $\mathbf{x}_k$ ) are noisy, so that fitting them exactly makes no sense. In fact, doing so can introduce a lot of spurious wiggles, when the true solution is likely to be smoother.

To delve into this phenomenon, let us start with a simple polynomial fitting example taken from (Bishop 2006, Chapter 1.1). Figure 4.3 shows a number of polynomial curves of different orders  $M$  fit to the blue circles, which are noisy samples from the underlying green sine curve. Notice how the low-order ( $M = 0$  and  $M = 1$ ) polynomials severely *underfit* the underlying data, resulting in curves that are too flat, while the  $M = 9$  polynomial, which exactly fits the data, exhibits far more wiggle than is likely.

How can we quantify this amount of underfitting and overfitting, and how can we get just the right amount? This topic is widely studied in machine learning and covered in a number of



**Figure 4.4** Regularized  $M = 9$  polynomial fitting for two different values of  $\lambda$  (Bishop 2006) © 2006 Springer. The left plot shows a reasonable amount of regularization, resulting in a plausible fit, while the larger value of  $\lambda$  on the right causes underfitting.

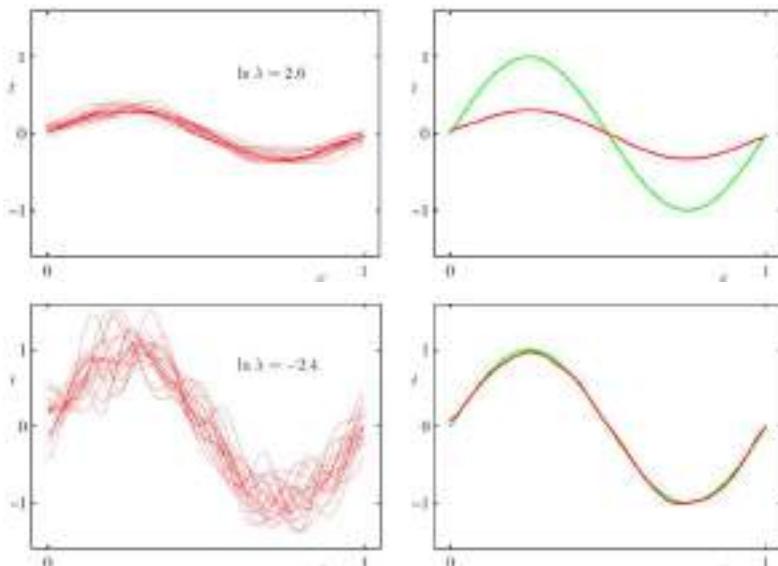


**Figure 4.5** Fitting (training) and validation errors as a function of the amount of regularization or smoothing © Glassner (2018). The less regularized solutions on the right, while exhibiting lower fitting error, perform less well on the validation data.

texts, including Bishop (2006, Chapter 1.1), Glassner (2018, Chapter 9), Deisenroth, Faisal, and Ong (2020, Chapter 8), and Zhang, Lipton *et al.* (2021, Section 4.4.3).

One approach is to use regularized least squares, introduced in (4.11). Figure 4.4 shows an  $M = 9$ th degree polynomial fit obtained by minimizing (4.11) with the polynomial basis functions  $\phi_k(x) = x^k$  for two different values of  $\lambda$ . The left plot shows a reasonable amount of regularization, resulting in a plausible fit, while the larger value of  $\lambda$  on the right causes underfitting. Note that the  $M = 9$  interpolant shown in the lower right quadrant of Figure 4.3 corresponds to the unregularized  $\lambda = 0$  case.

If we were to now measure the difference between the red (estimated) and green (noise-free) curves, we see that choosing a good intermediate value of  $\lambda$  will produce the best result.



**Figure 4.6** The more heavily regularized solution  $\log \lambda = 2.6$  exhibits higher bias (deviation from original curve) than the less heavily regularized version ( $\log \lambda = -2.4$ ), which has much higher variance (Bishop 2006) © 2006 Springer. The red curves on the left are  $M = 24$  Gaussian basis fits to 25 randomly sampled points on the green curve. The red curve on the right is their mean.

In practice, however, we never have access to samples from the noise-free data.

Instead, if we are given a set of samples to interpolate, we can save some in a *validation* set in order to see if the function we compute is underfitting or overfitting. When we vary a parameter such as  $\lambda$  (or use some other measure to control smoothness), we typically obtain a curve such as the one shown in Figure 4.5. In this figure, the blue curve denotes the fitting error, which in this case is called the *training error*, since in machine learning, we usually split the given data into a (typically larger) training set and a (typically smaller) validation set.

To obtain an even better estimate of the ideal amount of regularization, we can repeat the process of splitting our sample data into training and validation sets several times. One well-known technique, called *cross-validation* (Craven and Wahba 1979; Wahba and Wendelberger 1980; Bishop 2006, Section 1.3; Murphy 2012, Section 1.4.8; Deisenroth, Faisal, and Ong 2020, Chapter 8; Zhang, Lipton *et al.* 2021, Section 4.4.2), splits the training data into  $K$  *folds* (equal sized pieces). You then put aside each fold, in turn, and train on the remaining

data. You can then estimate the best regularization parameter by averaging over all  $K$  training runs. While this generally works well ( $K = 5$  is often used), it may be too expensive when training large neural networks because of the long training times involved.

Cross-validation is just one example of a class of *model selection* techniques that estimate *hyperparameters* in a training algorithm to achieve good performance. Additional methods include *information criteria* such as the Bayesian information criterion (BIC) (Torr 2002) and the Akaike information criterion (AIC) (Kanatani 1998), and Bayesian modeling approaches (Szeliski 1989; Bishop 2006; Murphy 2012).

One last topic worth mention with regard to data fitting, since it comes up often in discussions of statistical machine learning techniques, is the *bias-variance tradeoff* (Bishop 2006, Section 3.2). As you can see in Figure 4.6, using a large amount of regularization (top row) results in much lower variance between different random sample solutions, but much higher bias away from the true solution. Using insufficient regularization increases the variance dramatically, although an average over a large number of samples has low bias. The trick is to determine a reasonable compromise in terms of regularization so that any individual solution has a good expectation of being close to the *ground truth* (original clean continuous) data.

### 4.1.3 Robust data fitting

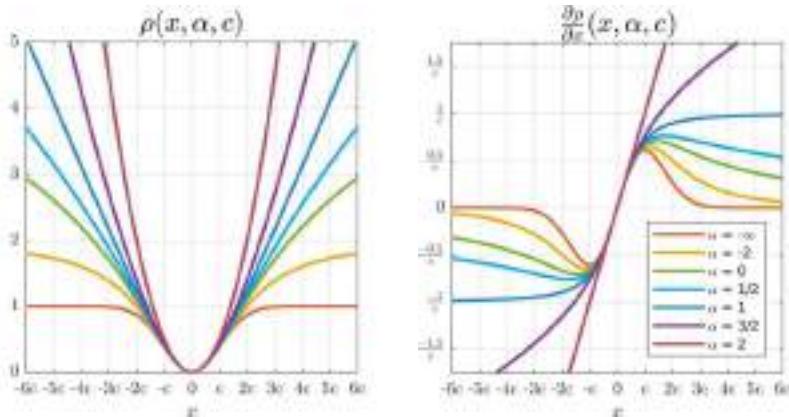
When we added a regularizer on the weights in (4.9), we noted that it did not have to be a quadratic penalty and could, instead, be a lower-order monomial that encouraged *sparsity* in the weights.

This same idea can be applied to data terms such as (4.2), where, instead of using a quadratic penalty, we can use a *robust loss function*  $\rho()$ ,

$$E_R = \sum_k \rho(\|\mathbf{r}_k\|), \quad \text{with } \mathbf{r}_k = \mathbf{f}(\mathbf{x}_k) - \mathbf{d}_k, \quad (4.15)$$

which gives lower weights to larger data fitting errors, which are more likely to be outlier measurements. (The fitting error term  $\mathbf{r}_k$  is called the *residual error*.)

Some examples of loss functions from (Barron 2019) are shown in Figure 4.7 along with their derivatives. The regular quadratic ( $\alpha = 2$ ) penalty gives full (linear) weight to each error, whereas the  $\alpha = 1$  loss gives equal weight to all larger residuals, i.e., it behaves as an  $L_1$  loss for large residuals, and  $L_2$  for small ones. Even larger values of  $\alpha$  discount large errors (outliers) even more, although they result in optimization problems that are *non-convex*, i.e., that can have multiple local minima. We will discuss techniques for finding good initial guesses for such problems later on in Section 8.1.4.

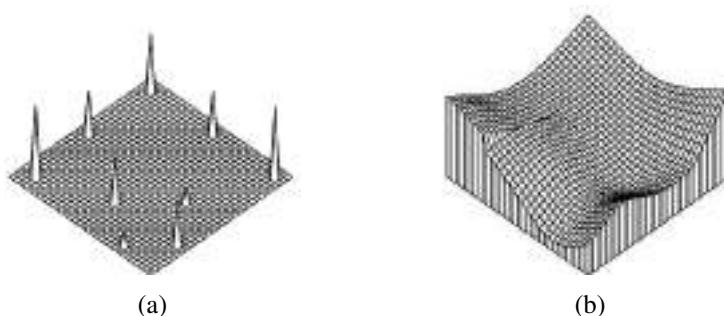


**Figure 4.7** A general and adaptive loss function (left) and its gradient (right) for different values of its shape parameter  $\alpha$  (Barron 2019) © 2019 IEEE. Several values of  $\alpha$  reproduce existing loss functions:  $L_2$  loss ( $\alpha = 2$ ), Charbonnier loss ( $\alpha = 1$ ), Cauchy loss ( $\alpha = 0$ ), Geman-McClure loss ( $\alpha = -2$ ), and Welsch loss ( $\alpha = -1$ ).

In statistics, minimizing non-quadratic loss functions to deal with potential outlier measurements is known as *M-estimation* (Huber 1981; Hampel, Ronchetti *et al.* 1986; Black and Rangarajan 1996; Stewart 1999). Such estimation problems are often solved using *iteratively reweighted least squares*, which we discuss in more detail in Section 8.1.4 and Appendix B.3. The Appendix also discusses the relationship between robust statistics and non-Gaussian probabilistic models.

The generalized loss function introduced by Barron (2019) has two free parameters. The first one,  $\alpha$ , controls how drastically outlier residuals are downweighted. The second (scale) parameter  $c$  controls the width of the quadratic well near the minimum, i.e., what range of residual values roughly corresponds to inliers. Traditionally, the choice of  $\alpha$ , which corresponds to a variety of previously published loss functions, was determined heuristically, based on the expected shape of the outlier distribution and computational considerations (e.g., whether a convex loss was desired). The scale parameter  $c$  could be estimated using a robust measure of variance, as discussed in Appendix B.3.

In his paper, Barron (2019) discusses how both parameters can be determined at run time by maximizing the likelihood (or equivalently, minimizing the negative log-likelihood) of the given residuals, making such an algorithm self-tuning to a wide variety of noise levels and outlier distributions.



**Figure 4.8** A simple surface interpolation problem: (a) nine data points of various heights scattered on a grid; (b) second-order, controlled-continuity, thin-plate spline interpolator, with a tear along its left edge and a crease along its right (Szeliski 1989) © 1989 Springer.

## 4.2 Variational methods and regularization

The theory of regularization we introduced in the previous section was first developed by statisticians trying to fit models to data that severely underconstrained the solution space (Tikhonov and Arsenin 1977; Engl, Hanke, and Neubauer 1996). Consider, for example, finding a smooth surface that passes through (or near) a set of measured data points (Figure 4.8). Such a problem is described as *ill-posed* because many possible surfaces can fit this data. Since small changes in the input can sometimes lead to large changes in the fit (e.g., if we use polynomial interpolation), such problems are also often *ill-conditioned*. Since we are trying to recover the unknown function  $f(x, y)$  from which the data points  $d(x_i, y_i)$  were sampled, such problems are also often called *inverse problems*. Many computer vision tasks can be viewed as inverse problems, since we are trying to recover a full description of the 3D world from a limited set of images.

In the previous section, we attacked this problem using basis functions placed at the data points, or other heuristics such as the pull-push algorithm. While such techniques can provide reasonable solutions, they do not let us directly *quantify* and hence *optimize* the amount of smoothness in the solution, nor do they give us local control over where the solution should be discontinuous (Figure 4.8).

To do this, we use norms (measures) on function derivatives (described below) to formulate the problem and then find minimal energy solutions to these norms. Such techniques are often called *energy-based* or *optimization-based* approaches to computer vision. They are also often called *variational*, since we can use the *calculus of variations* to find the optimal solutions. Variational methods have been widely used in computer vision since the early

1980s to pose and solve a number of fundamental problems, including optical flow (Horn and Schunck 1981; Black and Anandan 1993; Brox, Bruhn *et al.* 2004; Werlberger, Pock, and Bischof 2010), segmentation (Kass, Witkin, and Terzopoulos 1988; Mumford and Shah 1989; Chan and Vese 2001), denoising (Rudin, Osher, and Fatemi 1992; Chan, Osher, and Shen 2001; Chan and Shen 2005), and multi-view stereo (Faugeras and Keriven 1998; Pons, Keriven, and Faugeras 2007; Kolev, Klodt *et al.* 2009). A more detailed list of relevant papers can be found in the Additional Reading section at the end of this chapter.

In order to quantify what it means to find a smooth solution, we can define a norm on the solution space. For one-dimensional functions  $f(x)$ , we can integrate the squared first derivative of the function,

$$\mathcal{E}_1 = \int f_x^2(x) dx \quad (4.16)$$

or perhaps integrate the squared second derivative,

$$\mathcal{E}_2 = \int f_{xx}^2(x) dx. \quad (4.17)$$

(Here, we use subscripts to denote differentiation.) Such energy measures are examples of *functionals*, which are operators that map functions to scalar values. They are also often called *variational methods*, because they measure the variation (non-smoothness) in a function.

In two dimensions (e.g., for images, flow fields, or surfaces), the corresponding smoothness functionals are

$$\mathcal{E}_1 = \int f_x^2(x, y) + f_y^2(x, y) dx dy = \int \|\nabla f(x, y)\|^2 dx dy \quad (4.18)$$

and

$$\mathcal{E}_2 = \int f_{xx}^2(x, y) + 2f_{xy}^2(x, y) + f_{yy}^2(x, y) dx dy, \quad (4.19)$$

where the mixed  $2f_{xy}^2$  term is needed to make the measure rotationally invariant (Grimson 1983).

The first derivative norm is often called the *membrane*, since interpolating a set of data points using this measure results in a tent-like structure. (In fact, this formula is a small-deflection approximation to the surface area, which is what soap bubbles minimize.) The second-order norm is called the *thin-plate spline*, since it approximates the behavior of thin plates (e.g., flexible steel) under small deformations. A blend of the two is called the *thin-plate spline under tension* (Terzopoulos 1986b).

The regularizers (smoothness functions) we have just described force the solution to be smooth and  $C_0$  and/or  $C_1$  continuous everywhere. In most computer vision applications, however, the fields we are trying to model or recover are only piecewise continuous, e.g.,

depth maps and optical flow fields jump at object discontinuities. Color images are even more discontinuous, since they also change appearance at albedo (surface color) and shading discontinuities.

To better model such functions, Terzopoulos (1986b) introduced *controlled-continuity splines*, where each derivative term is multiplied by a local weighting function,

$$\begin{aligned}\mathcal{E}_{\text{CC}} = \int \rho(x, y) & \{ [1 - \tau(x, y)][f_x^2(x, y) + f_y^2(x, y)] \\ & + \tau(x, y)[f_{xx}^2(x, y) + 2f_{xy}^2(x, y) + f_{yy}^2(x, y)] \} dx dy.\end{aligned}\quad (4.20)$$

Here,  $\rho(x, y) \in [0, 1]$  controls the *continuity* of the surface and  $\tau(x, y) \in [0, 1]$  controls the local *tension*, i.e., how flat the surface wants to be. Figure 4.8 shows a simple example of a controlled-continuity interpolator fit to nine scattered data points. In practice, it is more common to find first-order smoothness terms used with images and flow fields (Section 9.3) and second-order smoothness associated with surfaces (Section 13.3.1).

In addition to the smoothness term, variational problems also require a data term (or *data penalty*). For scattered data interpolation (Nielson 1993), the data term measures the distance between the function  $f(x, y)$  and a set of data points  $d_i = d(x_i, y_i)$ ,

$$\mathcal{E}_{\text{D}} = \sum_i [f(x_i, y_i) - d_i]^2. \quad (4.21)$$

For a problem like noise removal, a continuous version of this measure can be used,

$$\mathcal{E}_{\text{D}} = \int [f(x, y) - d(x, y)]^2 dx dy. \quad (4.22)$$

To obtain a global energy that can be minimized, the two energy terms are usually added together,

$$\mathcal{E} = \mathcal{E}_{\text{D}} + \lambda \mathcal{E}_{\text{S}}, \quad (4.23)$$

where  $\mathcal{E}_{\text{S}}$  is the *smoothness penalty* ( $\mathcal{E}_1$ ,  $\mathcal{E}_2$  or some weighted blend such as  $\mathcal{E}_{\text{CC}}$ ) and  $\lambda$  is the *regularization parameter*, which controls the smoothness of the solution. As we saw in Section 4.1.2, good values for the regularization parameter can be estimated using techniques such as cross-validation.

### 4.2.1 Discrete energy minimization

In order to find the minimum of this continuous problem, the function  $f(x, y)$  is usually first discretized on a regular grid.<sup>2</sup> The most principled way to perform this discretization is to use

---

<sup>2</sup>The alternative of using *kernel basis functions* centered on the data points (Boult and Kender 1986; Nielson 1993) is discussed in more detail in Section 13.3.1.

*finite element analysis*, i.e., to approximate the function with a piecewise continuous spline, and then perform the analytic integration (Bathe 2007).

Fortunately, for both the first-order and second-order smoothness functionals, the judicious selection of appropriate finite elements results in particularly simple discrete forms (Terzopoulos 1983). The corresponding *discrete* smoothness energy functions become

$$\begin{aligned} E_1 = & \sum_{i,j} s_x(i,j)[f(i+1,j) - f(i,j) - g_x(i,j)]^2 \\ & + s_y(i,j)[f(i,j+1) - f(i,j) - g_y(i,j)]^2 \end{aligned} \quad (4.24)$$

and

$$\begin{aligned} E_2 = & h^{-2} \sum_{i,j} c_x(i,j)[f(i+1,j) - 2f(i,j) + f(i-1,j)]^2 \\ & + 2c_m(i,j)[f(i+1,j+1) - f(i+1,j) - f(i,j+1) + f(i,j)]^2 \\ & + c_y(i,j)[f(i,j+1) - 2f(i,j) + f(i,j-1)]^2, \end{aligned} \quad (4.25)$$

where  $h$  is the size of the finite element grid. The  $h$  factor is only important if the energy is being discretized at a variety of resolutions, as in coarse-to-fine or multigrid techniques.

The optional smoothness weights  $s_x(i,j)$  and  $s_y(i,j)$  control the location of horizontal and vertical tears (or weaknesses) in the surface. For other problems, such as colorization (Levin, Lischinski, and Weiss 2004) and interactive tone mapping (Lischinski, Farbman *et al.* 2006), they control the smoothness in the interpolated chroma or exposure field and are often set inversely proportional to the local luminance gradient strength. For second-order problems, the crease variables  $c_x(i,j)$ ,  $c_m(i,j)$ , and  $c_y(i,j)$  control the locations of creases in the surface (Terzopoulos 1988; Szeliski 1990a).

The data values  $g_x(i,j)$  and  $g_y(i,j)$  are gradient data terms (constraints) used by algorithms, such as photometric stereo (Section 13.1.1), HDR tone mapping (Section 10.2.1) (Fattal, Lischinski, and Werman 2002), Poisson blending (Section 8.4.4) (Pérez, Gangnet, and Blake 2003), gradient-domain blending (Section 8.4.4) (Levin, Zomet *et al.* 2004), and Poisson surface reconstruction (Section 13.5.1) (Kazhdan, Bolitho, and Hoppe 2006; Kazhdan and Hoppe 2013). They are set to zero when just discretizing the conventional first-order smoothness functional (4.18). Note how separate smoothness and curvature terms can be imposed in the  $x$ ,  $y$ , and mixed directions to produce local tears or creases (Terzopoulos 1988; Szeliski 1990a).

The two-dimensional discrete data energy is written as

$$E_D = \sum_{i,j} c(i,j)[f(i,j) - d(i,j)]^2, \quad (4.26)$$

where the local confidence weights  $c(i, j)$  control how strongly the data constraint is enforced. These values are set to zero where there is no data and can be set to the inverse variance of the data measurements when there is data (as discussed by Szeliski (1989) and in Section 4.3).

The total energy of the discretized problem can now be written as a *quadratic form*

$$E = E_D + \lambda E_S = \mathbf{x}^T \mathbf{A} \mathbf{x} - 2\mathbf{x}^T \mathbf{b} + c, \quad (4.27)$$

where  $\mathbf{x} = [f(0, 0) \dots f(m-1, n-1)]$  is called the *state vector*.<sup>3</sup>

The sparse symmetric positive-definite matrix  $\mathbf{A}$  is called the *Hessian* since it encodes the second derivative of the energy function.<sup>4</sup> For the one-dimensional, first-order problem,  $\mathbf{A}$  is tridiagonal; for the two-dimensional, first-order problem, it is multi-banded with five non-zero entries per row. We call  $\mathbf{b}$  the *weighted data vector*. Minimizing the above quadratic form is equivalent to solving the sparse linear system

$$\mathbf{A} \mathbf{x} = \mathbf{b}, \quad (4.28)$$

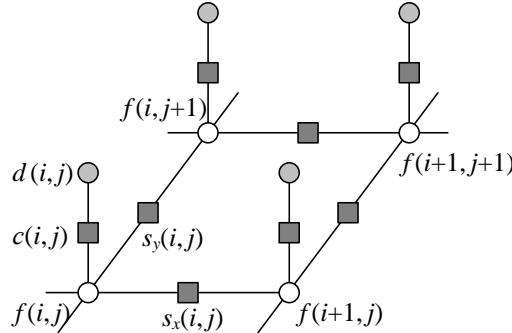
which can be done using a variety of sparse matrix techniques, such as multigrid (Briggs, Henson, and McCormick 2000) and hierarchical preconditioners (Szeliski 2006b; Krishnan and Szeliski 2011; Krishnan, Fattal, and Szeliski 2013), as described in Appendix A.5 and illustrated in Figure 4.11. Using such techniques is essential to obtaining reasonable runtimes, since properly preconditioned sparse linear systems have convergence times that are *linear* in the number of pixels.

While regularization was first introduced to the vision community by Poggio, Torre, and Koch (1985) and Terzopoulos (1986b) for problems such as surface interpolation, it was quickly adopted by other vision researchers for such varied problems as edge detection (Section 7.2), optical flow (Section 9.3), and shape from shading (Section 13.1) (Poggio, Torre, and Koch 1985; Horn and Brooks 1986; Terzopoulos 1986b; Bertero, Poggio, and Torre 1988; Brox, Bruhn *et al.* 2004). Poggio, Torre, and Koch (1985) also showed how the discrete energy defined by Equations (4.24–4.26) could be implemented in a resistive grid, as shown in Figure 4.9. In computational photography (Chapter 10), regularization and its variants are commonly used to solve problems such as high-dynamic range tone mapping (Fattal, Lischinski, and Werman 2002; Lischinski, Farbman *et al.* 2006), Poisson and gradient-domain blending (Pérez, Gangnet, and Blake 2003; Levin, Zomet *et al.* 2004; Agarwala, Dontcheva *et al.*

---

<sup>3</sup>We use  $\mathbf{x}$  instead of  $\mathbf{f}$  because this is the more common form in the numerical analysis literature (Golub and Van Loan 1996).

<sup>4</sup>In numerical analysis,  $\mathbf{A}$  is called the *coefficient* matrix (Saad 2003); in finite element analysis (Bathe 2007), it is called the *stiffness* matrix.



**Figure 4.9** Graphical model interpretation of first-order regularization. The white circles are the unknowns  $f(i, j)$  while the dark circles are the input data  $d(i, j)$ . In the resistive grid interpretation, the  $d$  and  $f$  values encode input and output voltages and the black squares denote resistors whose conductance is set to  $s_x(i, j)$ ,  $s_y(i, j)$ , and  $c(i, j)$ . In the spring-mass system analogy, the circles denote elevations and the black squares denote springs. The same graphical model can be used to depict a first-order Markov random field (Figure 4.12).

2004), colorization (Levin, Lischinski, and Weiss 2004), and natural image matting (Levin, Lischinski, and Weiss 2008).

### Robust regularization

While regularization is most commonly formulated using quadratic ( $L_2$ ) norms, i.e., the squared derivatives in (4.16–4.19) and squared differences in (4.24–4.25), it can also be formulated using the non-quadratic *robust* penalty functions first introduced in Section 4.1.3 and discussed in more detail in Appendix B.3. For example, (4.24) can be generalized to

$$E_{1R} = \sum_{i,j} s_x(i, j) \rho(f(i + 1, j) - f(i, j)) \\ + s_y(i, j) \rho(f(i, j + 1) - f(i, j)), \quad (4.29)$$

where  $\rho(x)$  is some monotonically increasing penalty function. For example, the family of norms  $\rho(x) = |x|^p$  is called  $p$ -norms. When  $p < 2$ , the resulting smoothness terms become more piecewise continuous than totally smooth, which can better model the discontinuous nature of images, flow fields, and 3D surfaces.

An early example of robust regularization is the *graduated non-convexity* (GNC) algorithm of Blake and Zisserman (1987). Here, the norms on the data and derivatives are

clamped,

$$\rho(x) = \min(x^2, V). \quad (4.30)$$

Because the resulting problem is highly non-convex (it has many local minima), a *continuation* method is proposed, where a quadratic norm (which is convex) is gradually replaced by the non-convex robust norm (Allgower and Georg 2003). (Around the same time, Terzopoulos (1988) was also using continuation to infer the tear and crease variables in his surface interpolation problems.)

### 4.2.2 Total variation

Today, many regularized problems are formulated using the  $L_1$  ( $p = 1$ ) norm, which is often called *total variation* (Rudin, Osher, and Fatemi 1992; Chan, Osher, and Shen 2001; Chambolle 2004; Chan and Shen 2005; Tschumperlé and Deriche 2005; Tschumperlé 2006; Cremers 2007; Kaftory, Schechner, and Zeevi 2007; Kolev, Klodt *et al.* 2009; Werlberger, Pock, and Bischof 2010). The advantage of this norm is that it tends to better preserve discontinuities, but still results in a convex problem that has a globally unique solution. Other norms, for which the *influence* (derivative) more quickly decays to zero, are presented by Black and Rangarajan (1996), Black, Sapiro *et al.* (1998), and Barron (2019) and discussed in Section 4.1.3 and Appendix B.3.

Even more recently, *hyper-Laplacian* norms with  $p < 1$  have gained popularity, based on the observation that the log-likelihood distribution of image derivatives follows a  $p \approx 0.5 - 0.8$  slope and is therefore a hyper-Laplacian distribution (Simoncelli 1999; Levin and Weiss 2007; Weiss and Freeman 2007; Krishnan and Fergus 2009). Such norms have an even stronger tendency to prefer large discontinuities over small ones. See the related discussion in Section 4.3 (4.43).

While least squares regularized problems using  $L_2$  norms can be solved using linear systems, other  $p$ -norms require different iterative techniques, such as iteratively reweighted least squares (IRLS), Levenberg–Marquardt, alternation between local non-linear subproblems and global quadratic regularization (Krishnan and Fergus 2009), or primal-dual algorithms (Chambolle and Pock 2011). Such techniques are discussed in Section 8.1.3 and Appendices A.3 and B.3.

### 4.2.3 Bilateral solver

In our discussion of variational methods, we have focused on energy minimization problems based on gradients and higher-order derivatives, which in the discrete setting involves

evaluating weighted errors between neighboring pixels. As we saw previously in our discussion of bilateral filtering in Section 3.3.2, we can often get better results by looking at a larger spatial neighborhood and combining pixels with similar colors or grayscale values. To extend this idea to a variational (energy minimization) setting, Barron and Poole (2016) propose replacing the usual first-order nearest-neighbor smoothness penalty (4.24) with a wider-neighborhood, bilaterally weighted version

$$E_B = \sum_{i,j} \sum_{k,l} \hat{w}(i,j,k,l) [f(k,l) - f(i,j)]^2, \quad (4.31)$$

where

$$\hat{w}(i,j,k,l) = \frac{w(i,j,k,l)}{\sum_{m,n} w(i,j,m,n)}, \quad (4.32)$$

is the *bistochastized* (normalized) version of the *bilateral weight function* given in (3.37), which may depend on an input guide image, but not on the estimated values of  $f$ .<sup>5</sup>

To efficiently solve the resulting set of equations (which are much denser than nearest-neighbor versions), the authors use the same approach originally used to accelerate bilateral filtering, i.e., solving a related problem on a (spatially coarser) bilateral grid. The sequence of operations resembles those used for bilateral filtering, except that after splatting and before slicing, an iterative least squares solver is used instead of a multi-dimensional Gaussian blur. To further speed up the conjugate gradient solver, Barron and Poole (2016) use a multi-level preconditioner inspired by previous work on image-adapted preconditioners (Szeliski 2006b; Krishnan, Fattal, and Szeliski 2013).

Since its introduction, the bilateral solver has been used in a number of video processing and 3D reconstruction applications, including the stitching of binocular omnidirectional panoramic videos (Anderson, Gallup *et al.* 2016). The smartphone AR system developed by Valentin, Kowdle *et al.* (2018) extends the bilateral solver to have local *planar* models and uses a hardware-friendly real-time implementation (Mazumdar, Alaghi *et al.* 2017) to produce dense occlusion effects.

#### 4.2.4 Application: Interactive colorization

A good use of edge-aware interpolation techniques is in *colorization*, i.e., manually adding colors to a “black and white” (grayscale) image. In most applications of colorization, the user draws some scribbles indicating the desired colors in certain regions (Figure 4.10a) and the system interpolates the specified chrominance ( $u, v$ ) values to the whole image, which

---

<sup>5</sup>Note that in their paper, Barron and Poole (2016) use different  $\sigma_r$  values for the luminance and chrominance components of pixel color differences.



**Figure 4.10** Colorization using optimization (Levin, Lischinski, and Weiss 2004) © 2004 ACM: (a) grayscale image with some color scribbles overlaid; (b) resulting colorized image; (c) original color image from which the grayscale image and the chrominance values for the scribbles were derived. Original photograph by Rotem Weiss.

are then re-combined with the input luminance channel to produce a final colorized image, as shown in Figure 4.10b. In the system developed by Levin, Lischinski, and Weiss (2004), the interpolation is performed using locally weighted regularization (4.24), where the local smoothness weights are inversely proportional to luminance gradients. This approach to locally weighted regularization has inspired later algorithms for high dynamic range tone mapping (Lischinski, Farbman *et al.* 2006)(Section 10.2.1, as well as other applications of the weighted least squares (WLS) formulation (Farbman, Fattal *et al.* 2008). These techniques have benefitted greatly from image-adapted regularization techniques, such as those developed in Szeliski (2006b), Krishnan and Szeliski (2011), Krishnan, Fattal, and Szeliski (2013), and Barron and Poole (2016), as shown in Figure 4.11. An alternative approach to performing the sparse chrominance interpolation based on geodesic (edge-aware) distance functions has been developed by Yatziv and Sapiro (2006). Neural networks can also be used to implement *deep priors* for image colorization (Zhang, Zhu *et al.* 2017).

### 4.3 Markov random fields

As we have just seen, regularization, which involves the minimization of energy functionals defined over (piecewise) continuous functions, can be used to formulate and solve a variety of low-level computer vision problems. An alternative technique is to formulate a *Bayesian* or *generative* model, which separately models the noisy image formation (*measurement*) process, as well as assuming a statistical *prior* model over the solution space (Bishop 2006, Section 1.5.4). In this section, we look at priors based on Markov random fields, whose log-likelihood can be described using local neighborhood interaction (or penalty) terms (Kin-



**Figure 4.11** Speeding up the inhomogeneous least squares colorization solver using locally adapted hierarchical basis preconditioning (Szeliski 2006b) © 2006 ACM: (a) input gray image with color strokes overlaid; (b) solution after 20 iterations of conjugate gradient; (c) using one iteration of hierarchical basis function preconditioning; (d) using one iteration of locally adapted hierarchical basis functions.

dermann and Snell 1980; Geman and Geman 1984; Marroquin, Mitter, and Poggio 1987; Li 1995; Szeliski, Zabih *et al.* 2008; Blake, Kohli, and Rother 2011).

The use of Bayesian modeling has several potential advantages over regularization (see also Appendix B). The ability to model measurement processes statistically enables us to extract the maximum information possible from each measurement, rather than just guessing what weighting to give the data. Similarly, the parameters of the prior distribution can often be *learned* by observing samples from the class we are modeling (Roth and Black 2007a; Tappen 2007; Li and Huttenlocher 2008). Furthermore, because our model is probabilistic, it is possible to estimate (in principle) complete probability *distributions* over the unknowns being recovered and, in particular, to model the *uncertainty* in the solution, which can be useful in later processing stages. Finally, Markov random field models can be defined over *discrete* variables, such as image labels (where the variables have no proper ordering), for which regularization does not apply.

According to Bayes' rule (Appendix B.4), the *posterior* distribution  $p(\mathbf{x}|\mathbf{y})$  over the unknowns  $\mathbf{x}$  given the measurements  $\mathbf{y}$  can be obtained by multiplying the measurement likelihood  $p(\mathbf{y}|\mathbf{x})$  by the prior distribution  $p(\mathbf{x})$  and normalizing,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}, \quad (4.33)$$

where  $p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$  is a normalizing constant used to make the  $p(\mathbf{x}|\mathbf{y})$  distribution *proper* (integrate to 1). Taking the negative logarithm of both sides of (4.33), we get

$$-\log p(\mathbf{x}|\mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) - \log p(\mathbf{x}) + C, \quad (4.34)$$

which is the *negative posterior log likelihood*.

To find the most likely (*maximum a posteriori* or MAP) solution  $\mathbf{x}$  given some measurements  $\mathbf{y}$ , we simply minimize this negative log likelihood, which can also be thought of as an *energy*,

$$E(\mathbf{x}, \mathbf{y}) = E_D(\mathbf{x}, \mathbf{y}) + E_P(\mathbf{x}). \quad (4.35)$$

(We drop the constant  $C$  because its value does not matter during energy minimization.) The first term  $E_D(\mathbf{x}, \mathbf{y})$  is the *data energy* or *data penalty*; it measures the negative log likelihood that the data were observed given the unknown state  $\mathbf{x}$ . The second term  $E_P(\mathbf{x})$  is the *prior energy*; it plays a role analogous to the smoothness energy in regularization. Note that the MAP estimate may not always be desirable, as it selects the “peak” in the posterior distribution rather than some more stable statistic—see the discussion in Appendix B.2 and by Levin, Weiss *et al.* (2009).

For the remainder of this section, we focus on Markov random fields, which are probabilistic models defined over two or three-dimensional pixel or voxel grids. Before we dive into this, however, we should mention that MRFs are just one special case of the more general family of *graphical models* (Bishop 2006, Chapter 8; Koller and Friedman 2009; Nowozin and Lampert 2011; Murphy 2012, Chapters 10, 17, 19), which have sparse interactions between variables that can be captured in a *factor graph* (Dellaert and Kaess 2017; Dellaert 2021), such as the one shown in Figure 4.12. Graphical models come in a wide variety of topologies, including chains (used for audio and speech processing), trees (often used for modeling kinematic chains in tracking people (e.g., Felzenszwalb and Huttenlocher 2005)), stars (simplified models for people; Dalal and Triggs 2005; Felzenszwalb, Girshick *et al.* 2010, and constellations (Fergus, Perona, and Zisserman 2007). Such models were widely used for part-based recognition, as discussed in Section 6.2.1. For graphs that are acyclic, efficient linear-time inference algorithms based on dynamic programming can be used.

For image processing applications, the unknowns  $\mathbf{x}$  are the set of output pixels

$$\mathbf{x} = [f(0, 0) \dots f(m - 1, n - 1)], \quad (4.36)$$

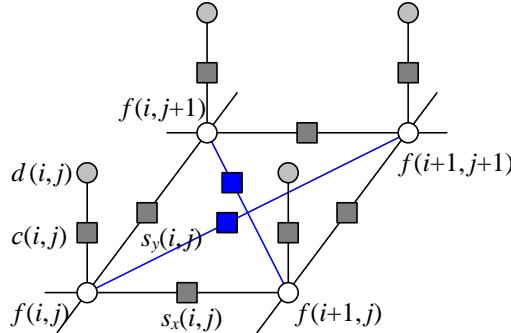
and the data are (in the simplest case) the input pixels

$$\mathbf{y} = [d(0, 0) \dots d(m - 1, n - 1)] \quad (4.37)$$

as shown in Figure 4.12.

For a Markov random field, the probability  $p(\mathbf{x})$  is a *Gibbs* or *Boltzmann distribution*, whose negative log likelihood (according to the Hammersley–Clifford theorem) can be written as a sum of pairwise interaction potentials,

$$E_P(\mathbf{x}) = \sum_{\{(i,j),(k,l)\} \in \mathcal{N}(i,j)} V_{i,j,k,l}(f(i, j), f(k, l)), \quad (4.38)$$



**Figure 4.12** Graphical model for an  $\mathcal{N}_4$  neighborhood Markov random field. (The blue edges are added for an  $\mathcal{N}_8$  neighborhood.) The white circles are the unknowns  $f(i, j)$ , while the dark circles are the input data  $d(i, j)$ . The  $s_x(i, j)$  and  $s_y(i, j)$  black boxes denote arbitrary interaction potentials between adjacent nodes in the random field, and the  $c(i, j)$  denote the data penalty functions. The same graphical model can be used to depict a discrete version of a first-order regularization problem (Figure 4.9).

where  $\mathcal{N}(i, j)$  denotes the *neighbors* of pixel  $(i, j)$ . In fact, the general version of the theorem says that the energy may have to be evaluated over a larger set of *cliques*, which depend on the *order* of the Markov random field (Kindermann and Snell 1980; Geman and Geman 1984; Bishop 2006; Kohli, Ladický, and Torr 2009; Kohli, Kumar, and Torr 2009).

The most commonly used neighborhood in Markov random field modeling is the  $\mathcal{N}_4$  neighborhood, where each pixel in the field  $f(i, j)$  interacts only with its immediate neighbors. The model in Figure 4.12, which we previously used in Figure 4.9 to illustrate the discrete version of first-order regularization, shows an  $\mathcal{N}_4$  MRF. The  $s_x(i, j)$  and  $s_y(i, j)$  black boxes denote arbitrary *interaction potentials* between adjacent nodes in the random field and the  $c(i, j)$  denote the data penalty functions. These square nodes can also be interpreted as *factors* in a *factor graph* version of the (undirected) graphical model (Bishop 2006; Dellaert and Kaess 2017; Dellaert 2021), which is another name for interaction potentials. (Strictly speaking, the factors are (improper) probability functions whose product is the (un-normalized) posterior distribution.)

As we will see in (4.41–4.42), there is a close relationship between these interaction potentials and the discretized versions of regularized image restoration problems. Thus, to a first approximation, we can view energy minimization being performed when solving a regularized problem and the maximum *a posteriori* inference being performed in an MRF as equivalent.

While  $\mathcal{N}_4$  neighborhoods are most commonly used, in some applications  $\mathcal{N}_8$  (or even higher order) neighborhoods perform better at tasks such as image segmentation because they can better model discontinuities at different orientations (Boykov and Kolmogorov 2003; Rother, Kohli *et al.* 2009; Kohli, Ladický, and Torr 2009; Kohli, Kumar, and Torr 2009).

## Binary MRFs

The simplest possible example of a Markov random field is a binary field. Examples of such fields include 1-bit (black and white) scanned document images as well as images segmented into foreground and background regions.

To denoise a scanned image, we set the data penalty to reflect the agreement between the scanned and final images,

$$E_D(i, j) = w\delta(f(i, j), d(i, j)) \quad (4.39)$$

and the smoothness penalty to reflect the agreement between neighboring pixels

$$E_P(i, j) = s\delta(f(i, j), f(i + 1, j)) + s\delta(f(i, j), f(i, j + 1)). \quad (4.40)$$

Once we have formulated the energy, how do we minimize it? The simplest approach is to perform gradient descent, flipping one state at a time if it produces a lower energy. This approach is known as *contextual classification* (Kittler and Föglein 1984), *iterated conditional modes* (ICM) (Besag 1986), or *highest confidence first* (HCF) (Chou and Brown 1990) if the pixel with the largest energy decrease is selected first.

Unfortunately, these downhill methods tend to get easily stuck in local minima. An alternative approach is to add some randomness to the process, which is known as *stochastic gradient descent* (Metropolis, Rosenbluth *et al.* 1953; Geman and Geman 1984). When the amount of noise is decreased over time, this technique is known as *simulated annealing* (Kirkpatrick, Gelatt, and Vecchi 1983; Carnevali, Coletti, and Patarnello 1985; Wolberg and Pavlidis 1985; Swendsen and Wang 1987) and was first popularized in computer vision by Geman and Geman (1984) and later applied to stereo matching by Barnard (1989), among others.

Even this technique, however, does not perform that well (Boykov, Veksler, and Zabih 2001). For binary images, a much better technique, introduced to the computer vision community by Boykov, Veksler, and Zabih (2001) is to re-formulate the energy minimization as a *max-flow/min-cut* graph optimization problem (Greig, Porteous, and Seheult 1989). This technique has informally come to be known as *graph cuts* in the computer vision community (Boykov and Kolmogorov 2011). For simple energy functions, e.g., those where the penalty for non-identical neighboring pixels is a constant, this algorithm is guaranteed to produce the

*global minimum.* Kolmogorov and Zabih (2004) formally characterize the class of binary energy potentials (*regularity conditions*) for which these results hold, while newer work by Komodakis, Tziritas, and Paragios (2008) and Rother, Kolmogorov *et al.* (2007) provide good algorithms for the cases when they do not, i.e., for energy functions that are not *regular* or *sub-modular*.

In addition to the above mentioned techniques, a number of other optimization approaches have been developed for MRF energy minimization, such as (loopy) belief propagation and dynamic programming (for one-dimensional problems). These are discussed in more detail in Appendix B.5 as well as the comparative survey papers by Szeliski, Zabih *et al.* (2008) and Kappes, Andres *et al.* (2015), which have associated benchmarks and code at <https://vision.middlebury.edu/MRF> and <http://hciweb2.iwr.uni-heidelberg.de/opengm>.

## Ordinal-valued MRFs

In addition to binary images, Markov random fields can be applied to ordinal-valued labels such as grayscale images or depth maps. The term “ordinal” indicates that the labels have an implied ordering, e.g., that higher values are lighter pixels. In the next section, we look at unordered labels, such as source image labels for image compositing.

In many cases, it is common to extend the binary data and smoothness prior terms as

$$E_D(i, j) = c(i, j)\rho_d(f(i, j) - d(i, j)) \quad (4.41)$$

and

$$E_P(i, j) = s_x(i, j)\rho_p(f(i, j) - f(i + 1, j)) + s_y(i, j)\rho_p(f(i, j) - f(i, j + 1)), \quad (4.42)$$

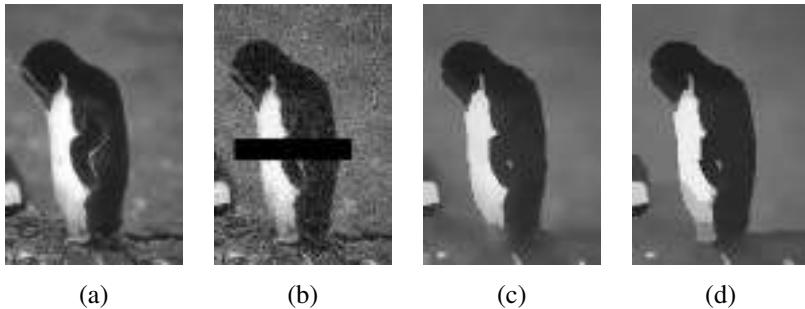
which are robust generalizations of the quadratic penalty terms (4.26) and (4.24), first introduced in (4.29). As before, the  $c(i, j)$ ,  $s_x(i, j)$ , and  $s_y(i, j)$  weights can be used to locally control the data weighting and the horizontal and vertical smoothness. Instead of using a quadratic penalty, however, a general monotonically increasing penalty function  $\rho()$  is used. (Different functions can be used for the data and smoothness terms.) For example,  $\rho_p$  can be a hyper-Laplacian penalty

$$\rho_p(d) = |d|^p, \quad p < 1, \quad (4.43)$$

which better encodes the distribution of gradients (mainly edges) in an image than either a quadratic or linear (total variation) penalty.<sup>6</sup> Levin and Weiss (2007) use such a penalty to

---

<sup>6</sup>Note that, unlike a quadratic penalty, the sum of the horizontal and vertical derivative  $p$ -norms is not rotationally invariant. A better approach may be to locally estimate the gradient direction and to impose different norms on the perpendicular and parallel components, which Roth and Black (2007b) call a *steerable random field*.



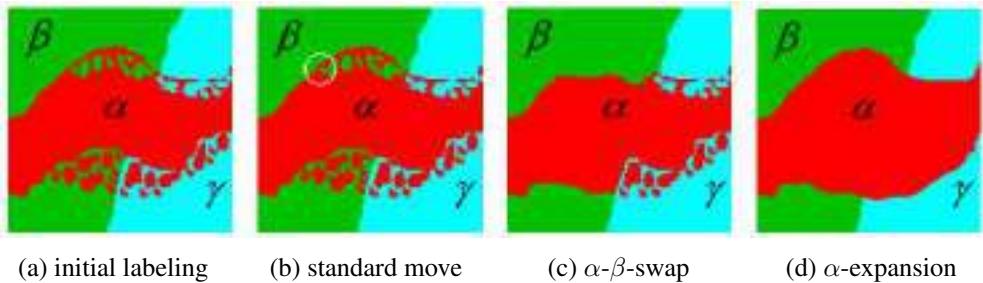
**Figure 4.13** Grayscale image denoising and inpainting: (a) original image; (b) image corrupted by noise and with missing data (black bar); (c) image restored using loopy belief propagation; (d) image restored using expansion move graph cuts. Images are from <https://vision.middlebury.edu/MRF/results> (Szeliski, Zabih et al. 2008).

separate a transmitted and reflected image (Figure 9.16) by encouraging gradients to lie in one or the other image, but not both. Levin, Fergus *et al.* (2007) use the hyper-Laplacian as a prior for image deconvolution (deblurring) and Krishnan and Fergus (2009) develop a faster algorithm for solving such problems. For the data penalty,  $\rho_d$  can be quadratic (to model Gaussian noise) or the log of a *contaminated Gaussian* (Appendix B.3).

When  $\rho_p$  is a quadratic function, the resulting Markov random field is called a Gaussian Markov random field (GMRF) and its minimum can be found by sparse linear system solving (4.28). When the weighting functions are uniform, the GMRF becomes a special case of Wiener filtering (Section 3.4.1). Allowing the weighting functions to depend on the input image (a special kind of conditional random field, which we describe below) enables quite sophisticated image processing algorithms to be performed, including colorization (Levin, Lischinski, and Weiss 2004), interactive tone mapping (Lischinski, Farbman *et al.* 2006), natural image matting (Levin, Lischinski, and Weiss 2008), and image restoration (Tappen, Liu *et al.* 2007).

When  $\rho_p$  or  $\rho_d$  are non-quadratic functions, gradient descent techniques such as non-linear least squares or iteratively re-weighted least squares can sometimes be used (Appendix A.3). However, if the search space has lots of local minima, as is the case for stereo matching (Barnard 1989; Boykov, Veksler, and Zabih 2001), more sophisticated techniques are required.

The extension of graph cut techniques to multi-valued problems was first proposed by Boykov, Veksler, and Zabih (2001). In their paper, they develop two different algorithms, called the *swap move* and the *expansion move*, which iterate among a series of binary labeling



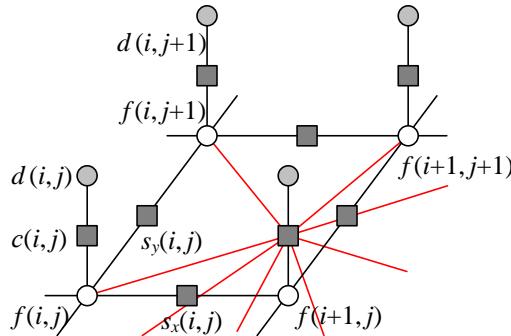
**Figure 4.14** Multi-level graph optimization from Boykov, Veksler, and Zabih (2001) © 2001 IEEE: (a) initial problem configuration; (b) the standard move only changes one pixel; (c) the  $\alpha$ - $\beta$ -swap optimally exchanges all  $\alpha$  and  $\beta$ -labeled pixels; (d) the  $\alpha$ -expansion move optimally selects among current pixel values and the  $\alpha$  label.

sub-problems to find a good solution (Figure 4.14). Note that a global solution is generally not achievable, as the problem is provably NP-hard for general energy functions. Because both these algorithms use a binary MRF optimization inside their inner loop, they are subject to the kind of constraints on the energy functions that occur in the binary labeling case (Kolmogorov and Zabih 2004).

Another MRF inference technique is *belief propagation* (BP). While belief propagation was originally developed for inference over trees, where it is exact (Pearl 1988), it has more recently been applied to graphs with loops such as Markov random fields (Freeman, Pasztor, and Carmichael 2000; Yedidia, Freeman, and Weiss 2001). In fact, some of the better performing stereo-matching algorithms use loopy belief propagation (LBP) to perform their inference (Sun, Zheng, and Shum 2003). LBP is discussed in more detail in comparative survey papers on MRF optimization (Szeliski, Zabih *et al.* 2008; Kappes, Andres *et al.* 2015).

Figure 4.13 shows an example of image denoising and inpainting (hole filling) using a non-quadratic energy function (non-Gaussian MRF). The original image has been corrupted by noise and a portion of the data has been removed (the black bar). In this case, the loopy belief propagation algorithm computes a slightly lower energy and also a smoother image than the alpha-expansion graph cut algorithm.

Of course, the above formula (4.42) for the smoothness term  $E_P(i, j)$  just shows the simplest case. In follow-on work, Roth and Black (2009) propose a *Field of Experts* (FoE) model, which sums up a large number of exponentiated local filter outputs to arrive at the smoothness penalty. Weiss and Freeman (2007) analyze this approach and compare it to the simpler hyper-Laplacian model of natural image statistics. Lyu and Simoncelli (2009) use



**Figure 4.15** Graphical model for a Markov random field with a more complex measurement model. The additional colored edges show how combinations of unknown values (say, in a sharp image) produce the measured values (a noisy blurred image). The resulting graphical model is still a classic MRF and is just as easy to sample from, but some inference algorithms (e.g., those based on graph cuts) may not be applicable because of the increased network complexity, since state changes during the inference become more entangled and the posterior MRF has much larger cliques.

Gaussian Scale Mixtures (GSMs) to construct an inhomogeneous multi-scale MRF, with one (positive exponential) GMRF modulating the variance (amplitude) of another Gaussian MRF.

It is also possible to extend the *measurement* model to make the sampled (noise-corrupted) input pixels correspond to blends of unknown (latent) image pixels, as in Figure 4.15. This is the commonly occurring case when trying to deblur an image. While this kind of a model is still a traditional generative Markov random field, i.e., we can in principle generate random samples from the prior distribution, finding an optimal solution can be difficult because the clique sizes get larger. In such situations, gradient descent techniques, such as iteratively reweighted least squares, can be used (Joshi, Zitnick *et al.* 2009). Exercise 4.4 has you explore some of these issues.

### Unordered labels

Another case with multi-valued labels where Markov random fields are often applied is that of *unordered labels*, i.e., labels where there is no semantic meaning to the numerical difference between the values of two labels. For example, if we are classifying terrain from aerial imagery, it makes no sense to take the numerical difference between the labels assigned to forest, field, water, and pavement. In fact, the adjacencies of these various kinds of terrain



**Figure 4.16** An unordered label MRF (Agarwala, Dontcheva et al. 2004) © 2004 ACM: Strokes in each of the source images on the left are used as constraints on an MRF optimization, which is solved using graph cuts. The resulting multi-valued label field is shown as a color overlay in the middle image, and the final composite is shown on the right.

each have different likelihoods, so it makes more sense to use a prior of the form

$$E_P(i, j) = s_x(i, j)V(l(i, j), l(i + 1, j)) + s_y(i, j)V(l(i, j), l(i, j + 1)), \quad (4.44)$$

where  $V(l_0, l_1)$  is a general *compatibility* or *potential* function. (Note that we have also replaced  $f(i, j)$  with  $l(i, j)$  to make it clearer that these are labels rather than function samples.) An alternative way to write this prior energy (Boykov, Veksler, and Zabih 2001; Szeliski, Zabih et al. 2008) is

$$E_P = \sum_{(p, q) \in \mathcal{N}} V_{p, q}(l_p, l_q), \quad (4.45)$$

where the  $(p, q)$  are neighboring pixels and a spatially varying potential function  $V_{p, q}$  is evaluated for each neighboring pair.

An important application of unordered MRF labeling is seam finding in image compositing (Davis 1998; Agarwala, Dontcheva et al. 2004) (see Figure 4.16, which is explained in more detail in Section 8.4.2). Here, the compatibility  $V_{p, q}(l_p, l_q)$  measures the quality of the visual appearance that would result from placing a pixel  $p$  from image  $l_p$  next to a pixel  $q$  from image  $l_q$ . As with most MRFs, we assume that  $V_{p, q}(l, l) = 0$ . For different labels, however, the compatibility  $V_{p, q}(l_p, l_q)$  may depend on the values of the underlying pixels  $I_{l_p}(p)$  and  $I_{l_q}(q)$ .

Consider, for example, where one image  $I_0$  is all sky blue, i.e.,  $I_0(p) = I_0(q) = B$ , while the other image  $I_1$  has a transition from sky blue,  $I_1(p) = B$ , to forest green,  $I_1(q) = G$ .

$$I_0 : \boxed{p \mid q} \quad \boxed{p \mid q} : I_1$$

In this case,  $V_{p, q}(1, 0) = 0$  (the colors agree), while  $V_{p, q}(0, 1) > 0$  (the colors disagree).

### 4.3.1 Conditional random fields

In a classic Bayesian model (4.33–4.35),

$$p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}), \quad (4.46)$$

the prior distribution  $p(\mathbf{x})$  is independent of the observations  $\mathbf{y}$ . Sometimes, however, it is useful to modify our prior assumptions, say about the smoothness of the field we are trying to estimate, in response to the sensed data. Whether this makes sense from a probability viewpoint is something we discuss once we have explained the new model.

Consider an interactive image segmentation system such as the one described in Boykov and Funka-Lea (2006). In this application, the user draws foreground and background strokes, and the system then solves a binary MRF labeling problem to estimate the extent of the foreground object. In addition to minimizing a data term, which measures the pointwise similarity between pixel colors and the inferred region distributions (Section 4.3.2), the MRF is modified so that the smoothness terms  $s_x(x, y)$  and  $s_y(x, y)$  in Figure 4.12 and (4.42) depend on the magnitude of the gradient between adjacent pixels.<sup>7</sup>

Since the smoothness term now depends on the data, Bayes' rule (4.46) no longer applies. Instead, we use a direct model for the posterior distribution  $p(\mathbf{x}|\mathbf{y})$ , whose negative log likelihood can be written as

$$\begin{aligned} E(\mathbf{x}|\mathbf{y}) &= E_D(\mathbf{x}, \mathbf{y}) + E_S(\mathbf{x}, \mathbf{y}) \\ &= \sum_p V_p(x_p, \mathbf{y}) + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(x_p, x_q, \mathbf{y}), \end{aligned} \quad (4.47)$$

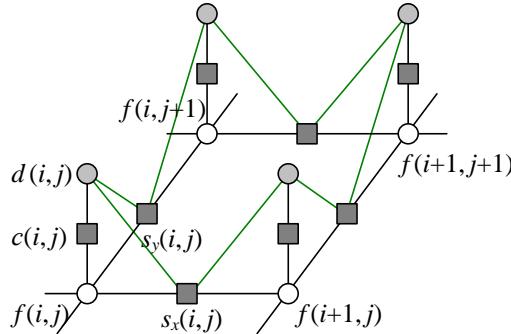
using the notation introduced in (4.45). The resulting probability distribution is called a *conditional random field* (CRF) and was first introduced to the computer vision field by Kumar and Hebert (2003), based on earlier work in text modeling by Lafferty, McCallum, and Pereira (2001).

Figure 4.17 shows a graphical model where the smoothness terms depend on the data values. In this particular model, each smoothness term depends only on its adjacent pair of data values, i.e., terms are of the form  $V_{p,q}(x_p, x_q, y_p, y_q)$  in (4.47).

The idea of modifying smoothness terms in response to input data is not new. For example, Boykov and Jolly (2001) used this idea for interactive segmentation, and it is now widely used in image segmentation (Section 4.3.2) (Blake, Rother *et al.* 2004; Rother, Kolmogorov, and Blake 2004), denoising (Tappen, Liu *et al.* 2007), and object recognition (Section 6.4) (Winn and Shotton 2006; Shotton, Winn *et al.* 2009).

---

<sup>7</sup>An alternative formulation that also uses detected edges to modulate the smoothness of a depth or motion field and hence to integrate multiple lower level vision modules is presented by Poggio, Gamble, and Little (1988).



**Figure 4.17** Graphical model for a conditional random field (CRF). The additional green edges show how combinations of sensed data influence the smoothness in the underlying MRF prior model, i.e.,  $s_x(i, j)$  and  $s_y(i, j)$  in (4.42) depend on adjacent  $d(i, j)$  values. These additional links (factors) enable the smoothness to depend on the input data. However, they make sampling from this MRF more complex.

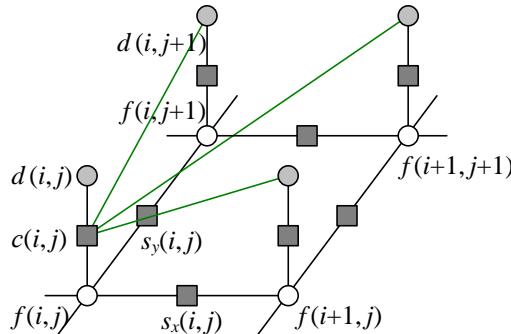
In stereo matching, the idea of encouraging disparity discontinuities to coincide with intensity edges goes back even further to the early days of optimization and MRF-based algorithms (Poggio, Gamble, and Little 1988; Fua 1993; Bobick and Intille 1999; Boykov, Veksler, and Zabih 2001) and is discussed in more detail in (Section 12.5).

In addition to using smoothness terms that adapt to the input data, Kumar and Hebert (2003) also compute a neighborhood function over the input data for each  $V_p(x_p, \mathbf{y})$  term, as illustrated in Figure 4.18, instead of using the classic unary MRF data term  $V_p(x_p, y_p)$  shown in Figure 4.12.<sup>8</sup> Because such neighborhood functions can be thought of as *discriminant* functions (a term widely used in machine learning (Bishop 2006)), they call the resulting graphical model a *discriminative random field* (DRF). In their paper, Kumar and Hebert (2006) show that DRFs outperform similar CRFs on a number of applications, such as structure detection and binary image denoising.

Here again, one could argue that previous stereo correspondence algorithms also look at a neighborhood of input data, either explicitly, because they compute correlation measures (Criminisi, Cross *et al.* 2006) as data terms, or implicitly, because even pixel-wise disparity costs look at several pixels in either the left or right image (Barnard 1989; Boykov, Veksler, and Zabih 2001).

What then are the advantages and disadvantages of using conditional or discriminative

<sup>8</sup>Kumar and Hebert (2006) call the unary potentials  $V_p(x_p, \mathbf{y})$  *association potentials* and the pairwise potentials  $V_{p,q}(x_p, y_q, \mathbf{y})$  *interaction potentials*.



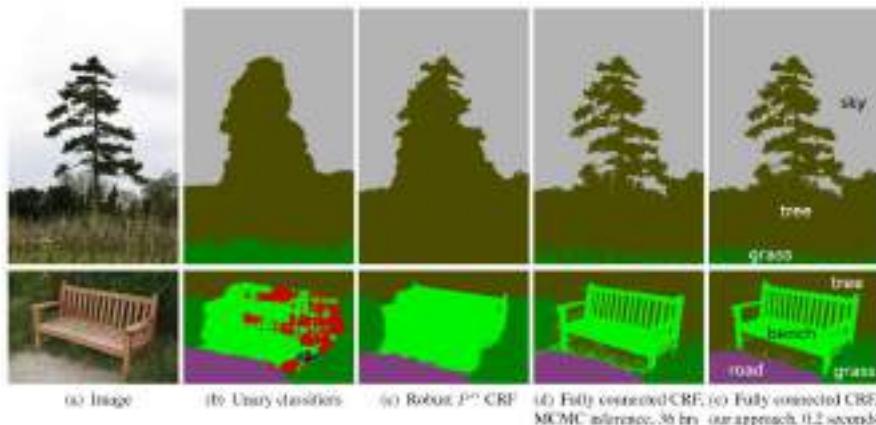
**Figure 4.18** Graphical model for a discriminative random field (DRF). The additional green edges show how combinations of sensed data, e.g.,  $d(i, j + 1)$ , influence the data term for  $f(i, j)$ . The generative model is therefore more complex, i.e., we cannot just apply a simple function to the unknown variables and add noise.

random fields instead of MRFs?

Classic Bayesian inference (MRF) assumes that the prior distribution of the data is independent of the measurements. This makes a lot of sense: if you see a pair of sixes when you first throw a pair of dice, it would be unwise to assume that they will always show up thereafter. However, if after playing for a long time you detect a statistically significant bias, you may want to adjust your prior. What CRFs do, in essence, is to select or modify the prior model based on observed data. This can be viewed as making a partial inference over additional hidden variables or correlations between the unknowns (say, a label, depth, or clean image) and the knowns (observed images).

In some cases, the CRF approach makes a lot of sense and is, in fact, the only plausible way to proceed. For example, in grayscale image colorization (Section 4.2.4) (Levin, Lischinski, and Weiss 2004), a commonly used way to transfer the continuity information from the input grayscale image to the unknown color image is to modify the local smoothness constraints. Similarly, for simultaneous segmentation and recognition (Winn and Shotton 2006; Shotton, Winn *et al.* 2009), it makes a lot of sense to permit strong color edges to increase the likelihood of semantic image label discontinuities.

In other cases, such as image denoising, the situation is more subtle. Using a non-quadratic (robust) smoothness term as in (4.42) plays a qualitatively similar role to setting the smoothness based on local gradient information in a Gaussian MRF (GMRF) (Tappen, Liu *et al.* 2007; Tanaka and Okutomi 2008). The advantage of Gaussian MRFs, when the smoothness can be correctly inferred, is that the resulting quadratic energy can be minimized



**Figure 4.19** *Pixel-level classification with a fully connected CRF, from © Krähenbühl and Koltun (2011). The labels in each column describe the image or algorithm being run, which include a robust  $P^n$  CRF (Kohli, Ladický, and Torr 2009) and a very slow MCMC optimization algorithm.*

in a single step, i.e., by solving a sparse set of linear equations. However, for situations where the discontinuities are not self-evident in the input data, such as for piecewise-smooth sparse data interpolation (Blake and Zisserman 1987; Terzopoulos 1988), classic robust smoothness energy minimization may be preferable. Thus, as with most computer vision algorithms, a careful analysis of the problem at hand and desired robustness and computation constraints may be required to choose the best technique.

Perhaps the biggest advantage of CRFs and DRFs, as argued by Kumar and Hebert (2006), Tappen, Liu *et al.* (2007), and Blake, Rother *et al.* (2004), is that learning the model parameters is more principled and sometimes easier. While learning parameters in MRFs and their variants is not a topic that we cover in this book, interested readers can find more details in publications by Kumar and Hebert (2006), Roth and Black (2007a), Tappen, Liu *et al.* (2007), Tappen (2007), and Li and Huttenlocher (2008).

## Dense Conditional Random Fields (CRFs)

As with regular Markov random fields, conditional random fields (CRFs) are normally defined over small neighborhoods, e.g., the  $\mathcal{N}_4$  neighborhood shown in Figure 4.17. However, images often contain longer-range interactions, e.g., pixels of similar colors may belong to

related classes (Figure 4.19). In order to model such longer-range interactions, Krähenbühl and Koltun (2011) introduced what they call a *fully connected CRF*, which many people now call a *dense CRF*.

As with traditional conditional random fields (4.47), their energy function consists of both unary terms and pairwise terms

$$E(\mathbf{x}|\mathbf{y}) = \sum_p V_p(x_p, \mathbf{y}) + \sum_{(p,q)} V_{p,q}(x_p, x_q, y_p, y_q), \quad (4.48)$$

where the  $(p, q)$  summation is now taken over *all* pairs of pixels, and not just adjacent ones.<sup>9</sup> The  $\mathbf{y}$  denotes the input (guide) image over which the random field is conditioned. The pairwise interaction potentials have a restricted form

$$V_{p,q}(x_p, x_q, y_p, y_q) = \mu(x_p, x_q) \sum_{m=1}^M s_m w_m(p, q) \quad (4.49)$$

that is the product of a spatially invariant *label compatibility function*  $\mu(x_p, x_q)$  and a sum of  $M$  Gaussian kernels of the same form (3.37) as is used in bilateral filtering and the bilateral solver. In their seminal paper, Krähenbühl and Koltun (2011) use two kernels, the first of which is an *appearance kernel* similar to (3.37) and the second is a spatial-only *smoothness kernel*.

Because of the special form of the long-range interaction potentials, which encapsulate all spatial and color similarity terms into a bilateral form, higher-dimensional filtering algorithms similar to those used in fast bilateral filters and solvers (Adams, Baek, and Davis 2010) can be used to efficiently compute a *mean field approximation* to the posterior conditional distribution (Krähenbühl and Koltun 2011). Figure 4.19 shows a comparison of their results (rightmost column) with previous approaches, including using simple unary terms, a robust CRF (Kohli, Ladický, and Torr 2009), and a very slow MCMC (Markov chain Monte Carlo) inference algorithm. As you can see, the fully connected CRF with a mean field solver produces dramatically better results in a very short time.

Since the publication of this paper, provably convergent and more efficient inference algorithms have been developed both by the original authors (Krähenbühl and Koltun 2013) and others (Vineet, Warrell, and Torr 2014; Desmaison, Bunel *et al.* 2016). Dense CRFs have seen widespread use in image segmentation problems and also as a “clean-up” stage for deep neural networks, as in the widely cited DeepLab paper by Chen, Papandreou *et al.* (2018).

---

<sup>9</sup>In practice, as with bilateral filtering and the bilateral solver, the spatial extent may be over a large but finite region.

### 4.3.2 Application: Interactive segmentation

The goal of image segmentation algorithms is to group pixels that have similar appearance (statistics) and to have the boundaries between pixels in different regions be of short length and across visible discontinuities. If we restrict the boundary measurements to be between immediate neighbors and compute region membership statistics by summing over pixels, we can formulate this as a classic pixel-based energy function using either a *variational formulation* (Section 4.2) or as a binary Markov random field (Section 4.3).

Examples of the continuous approach include Mumford and Shah (1989), Chan and Vese (2001), Zhu and Yuille (1996), and Tabb and Ahuja (1997) along with the level set approaches discussed in Section 7.3.2. An early example of a discrete labeling problem that combines both region-based and boundary-based energy terms is the work of Leclerc (1989), who used minimum description length (MDL) coding to derive the energy function being minimized. Boykov and Funka-Lea (2006) present a wonderful survey of various energy-based techniques for binary object segmentation, some of which we discuss below.

As we saw earlier in this chapter, the energy corresponding to a segmentation problem can be written (c.f. Equations (4.24) and (4.35–4.42)) as

$$E(f) = \sum_{i,j} E_R(i,j) + E_P(i,j), \quad (4.50)$$

where the region term

$$E_R(i,j) = C(I(i,j); R(f(i,j))) \quad (4.51)$$

is the negative log likelihood that pixel intensity (or color)  $I(i,j)$  is consistent with the statistics of region  $R(f(i,j))$  and the boundary term

$$E_P(i,j) = s_x(i,j)\delta(f(i,j), f(i+1,j)) + s_y(i,j)\delta(f(i,j), f(i,j+1)) \quad (4.52)$$

measures the inconsistency between  $\mathcal{N}_4$  neighbors modulated by local horizontal and vertical smoothness terms  $s_x(i,j)$  and  $s_y(i,j)$ .

Region statistics can be something as simple as the mean gray level or color (Leclerc 1989), in which case

$$C(I; \mu_k) = \|I - \mu_k\|^2. \quad (4.53)$$

Alternatively, they can be more complex, such as region intensity histograms (Boykov and Jolly 2001) or color Gaussian mixture models (Rother, Kolmogorov, and Blake 2004). For smoothness (boundary) terms, it is common to make the strength of the smoothness  $s_x(i,j)$  inversely proportional to the local edge strength (Boykov, Veksler, and Zabih 2001).

Originally, energy-based segmentation problems were optimized using iterative gradient descent techniques, which were slow and prone to getting trapped in local minima. Boykov



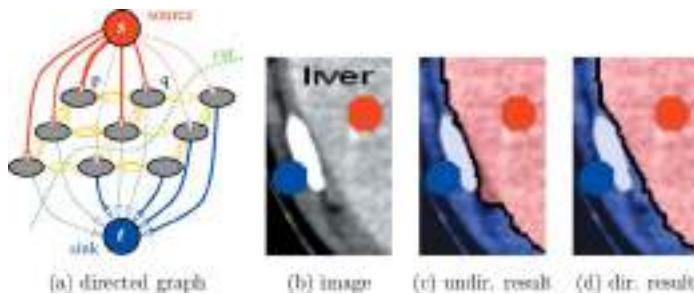
**Figure 4.20** *GrabCut image segmentation (Rother, Kolmogorov, and Blake 2004) © 2004 ACM:* (a) the user draws a bounding box in red; (b) the algorithm guesses color distributions for the object and background and performs a binary segmentation; (c) the process is repeated with better region statistics.

and Jolly (2001) were the first to apply the binary MRF optimization algorithm developed by Greig, Porteous, and Seheult (1989) to binary object segmentation.

In this approach, the user first delineates pixels in the background and foreground regions using a few strokes of an image brush. These pixels then become the *seeds* that tie nodes in the *S–T graph* to the source and sink labels  $S$  and  $T$ . Seed pixels can also be used to estimate foreground and background region statistics (intensity or color histograms).

The capacities of the other edges in the graph are derived from the region and boundary energy terms, i.e., pixels that are more compatible with the foreground or background region get stronger connections to the respective source or sink; adjacent pixels with greater smoothness also get stronger links. Once the minimum-cut/maximum-flow problem has been solved using a polynomial time algorithm (Goldberg and Tarjan 1988; Boykov and Kolmogorov 2004), pixels on either side of the computed cut are labeled according to the source or sink to which they remain connected. While graph cuts is just one of several known techniques for MRF energy minimization, it is still the one most commonly used for solving binary MRF problems.

The basic binary segmentation algorithm of Boykov and Jolly (2001) has been extended in a number of directions. The *GrabCut* system of Rother, Kolmogorov, and Blake (2004) iteratively re-estimates the region statistics, which are modeled as a mixtures of Gaussians in color space. This allows their system to operate given minimal user input, such as a single bounding box (Figure 4.20a)—the background color model is initialized from a strip of pixels around the box outline. (The foreground color model is initialized from the interior pixels, but quickly converges to a better estimate of the object.) The user can also place additional strokes to refine the segmentation as the solution progresses. Cui, Yang *et al.* (2008) use color and edge models derived from previous segmentations of similar objects to improve the local models used in GrabCut. Graph cut algorithms and other variants of Markov and conditional



**Figure 4.21** Segmentation with a directed graph cut (Boykov and Funka-Lea 2006) © 2006 Springer: (a) directed graph; (b) image with seed points; (c) the undirected graph incorrectly continues the boundary along the bright object; (d) the directed graph correctly segments the light gray region from its darker surround.

random fields have been applied to the *semantic segmentation* problem (Shotton, Winn *et al.* 2009; Krähenbühl and Koltun 2011), an example of which is shown in Figure 4.19 and which we study in more detail in Section 6.4.

Another major extension to the original binary segmentation formulation is the addition of *directed edges*, which allows boundary regions to be oriented, e.g., to prefer light to dark transitions or *vice versa* (Kolmogorov and Boykov 2005). Figure 4.21 shows an example where the directed graph cut correctly segments the light gray liver from its dark gray surround. The same approach can be used to measure the *flux* exiting a region, i.e., the signed gradient projected normal to the region boundary. Combining oriented graphs with larger neighborhoods enables approximating continuous problems such as those traditionally solved using level sets in the globally optimal graph cut framework (Boykov and Kolmogorov 2003; Kolmogorov and Boykov 2005).

More recent developments in graph cut-based segmentation techniques include the addition of connectivity priors to force the foreground to be in a single piece (Vicente, Kolmogorov, and Rother 2008) and shape priors to use knowledge about an object’s shape during the segmentation process (Lempitsky and Boykov 2007; Lempitsky, Blake, and Rother 2008).

While optimizing the binary MRF energy (4.50) requires the use of combinatorial optimization techniques, such as maximum flow, an approximate solution can be obtained by converting the binary energy terms into quadratic energy terms defined over a continuous  $[0, 1]$  random field, which then becomes a classical membrane-based regularization problem (4.24–4.27). The resulting quadratic energy function can then be solved using standard linear system solvers (4.27–4.28), although if speed is an issue, you should use multigrid or one

of its variants (Appendix A.5). Once the continuous solution has been computed, it can be thresholded at 0.5 to yield a binary segmentation.

The  $[0, 1]$  continuous optimization problem can also be interpreted as computing the probability at each pixel that a *random walker* starting at that pixel ends up at one of the labeled seed pixels, which is also equivalent to computing the potential in a resistive grid where the resistors are equal to the edge weights (Grady 2006; Sinop and Grady 2007).  $K$ -way segmentations can also be computed by iterating through the seed labels, using a binary problem with one label set to 1 and all the others set to 0 to compute the relative membership probabilities for each pixel. In follow-on work, Grady and Ali (2008) use a precomputation of the eigenvectors of the linear system to make the solution with a novel set of seeds faster, which is related to the Laplacian matting problem presented in Section 10.4.3 (Levin, Acha, and Lischinski 2008). Couprie, Grady *et al.* (2009) relate the random walker to watersheds and other segmentation techniques. Singaraju, Grady, and Vidal (2008) add directed-edge constraints in order to support flux, which makes the energy piecewise quadratic and hence not solvable as a single linear system. The random walker algorithm can also be used to solve the Mumford–Shah segmentation problem (Grady and Alvino 2008) and to compute fast multi-grid solutions (Grady 2008). A nice review of these techniques is given by Singaraju, Grady *et al.* (2011).

An even faster way to compute a continuous  $[0, 1]$  approximate segmentation is to compute *weighted geodesic distances* between the 0 and 1 seed regions (Bai and Sapiro 2009), which can also be used to estimate soft alpha mattes (Section 10.4.3). A related approach by Criminisi, Sharp, and Blake (2008) can be used to find fast approximate solutions to general binary Markov random field optimization problems.

## 4.4 Additional reading

Scattered data interpolation and approximation techniques are fundamental to many different branches of applied mathematics. Some good introductory texts and articles include Amidror (2002), Wendland (2004), and Anjyo, Lewis, and Pighin (2014). These techniques are also related to geometric modeling techniques in computer graphics, which continues to be a very active research area. A nice introduction to basic spline techniques for curves and surfaces can be found in Farin (2002), while more recent approaches using subdivision surfaces are covered in Peters and Reif (2008).

Data interpolation and approximation also lie at the heart of *regression techniques*, which form the mathematical basis for most of the machine learning techniques we study in the next chapter. You can find good introductions to this topic (as well as underfitting, overfitting,

# Computer Vision

---

## **UNIT II**

**Multi-Level Vision (Low, Mid, High):**

**LOW: Feature Detection and Matching, Edge Detection,  
(Chap 7)**

**MID: Image Alignment(Chap 8)**

**HIGH: Instance Recognition, Object Detection, Image  
Classification, Semantic Segmentation, Video  
Understanding, Vision and Language.(Chap 6)**

# Today

---

- Image matching
- Features
- Features detection
- Feature descriptors
- Features Matching

## Readings

- Szeliski, Ch 7
- (optional) K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors. In PAMI 27(10):1615-1630
  - [http://www.robots.ox.ac.uk/~vgg/research/affine/det\\_eval\\_files/mikolajczyk](http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/mikolajczyk)

# Image matching

---

Trevi fountain-Rome



by [Diva Sian](#)



by [swashford](#)

# Harder case

---



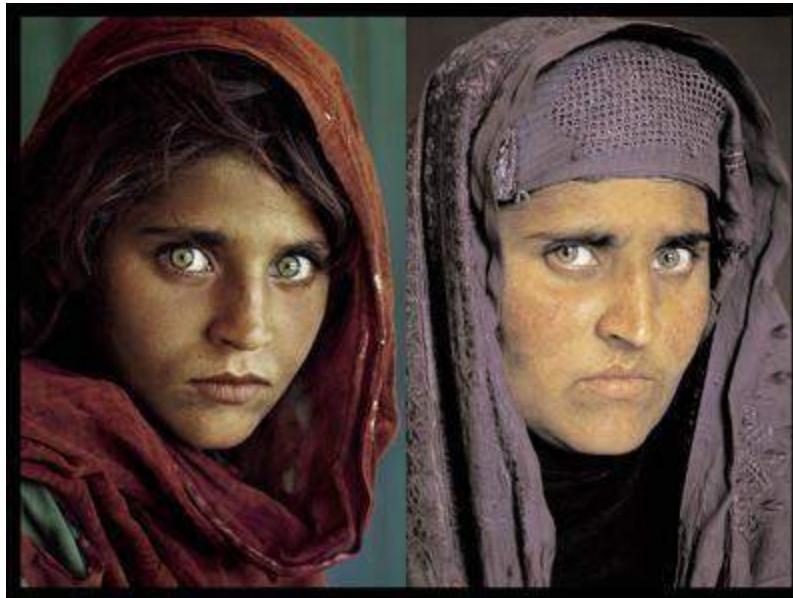
by [Diva Sian](#)



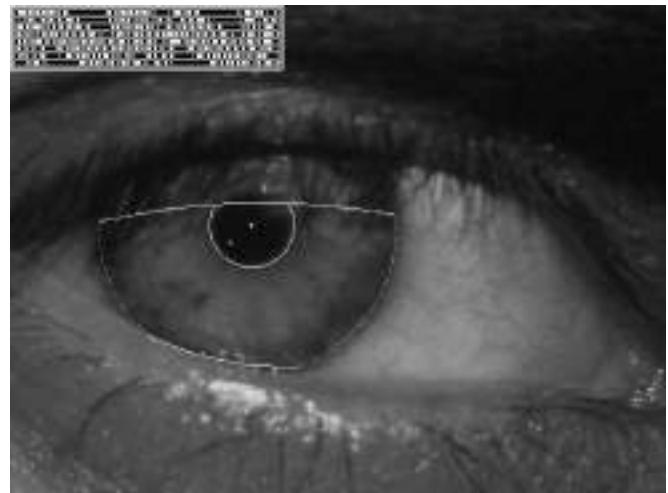
by [scgbt](#)

# Even harder case

---

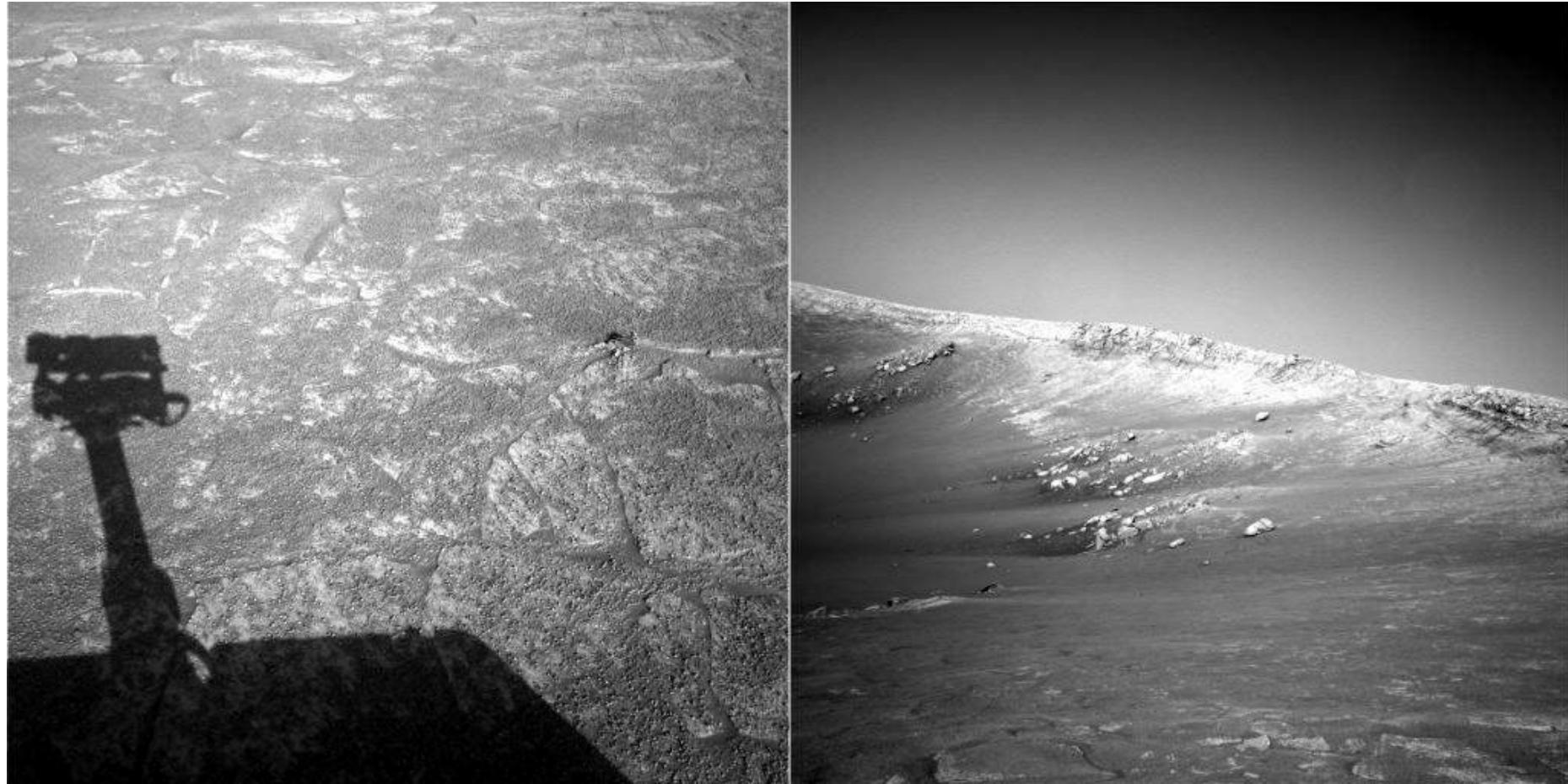


***“How the Afghan Girl was Identified by Her Iris Patterns”*** Read the [story](#)



# Harder still?

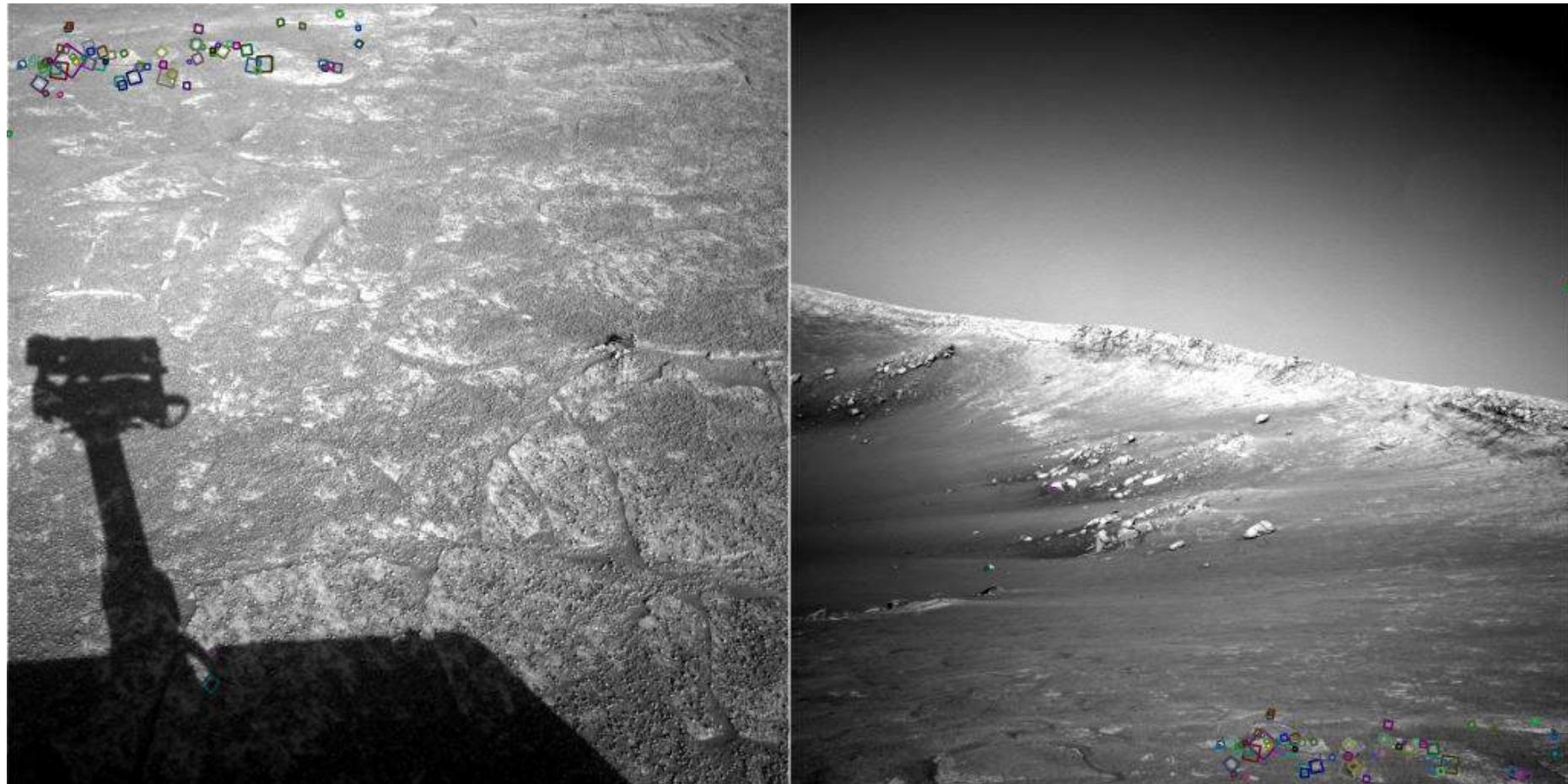
---



NASA Mars Rover images

# Answer below (look for tiny colored squares...)

---

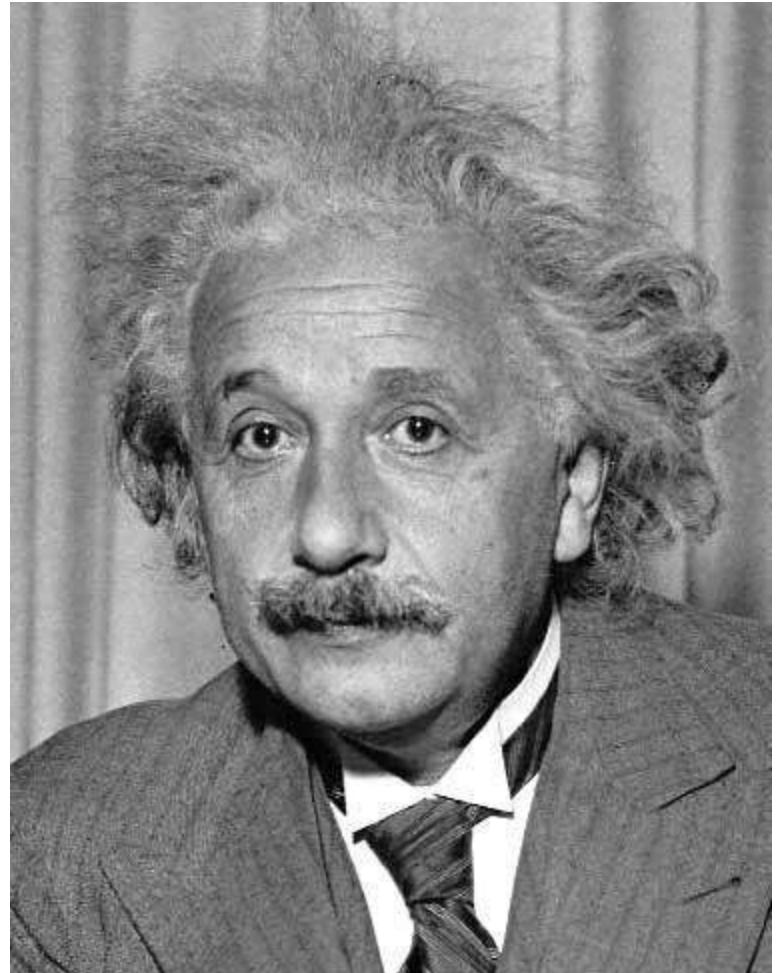


NASA Mars Rover images  
with SIFT feature matches  
Figure by Noah Snavely

# Template matching

---

- Goal: find  in image
- Main challenge: What is a good similarity or distance measure between two patches?
  - Correlation
  - Zero-mean correlation
  - Sum Square Difference
  - Normalized Cross Correlation

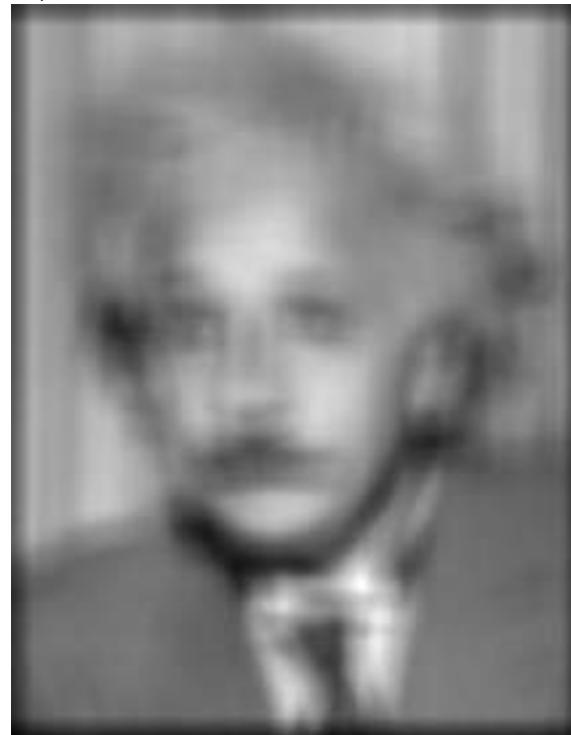


# Matching with filters

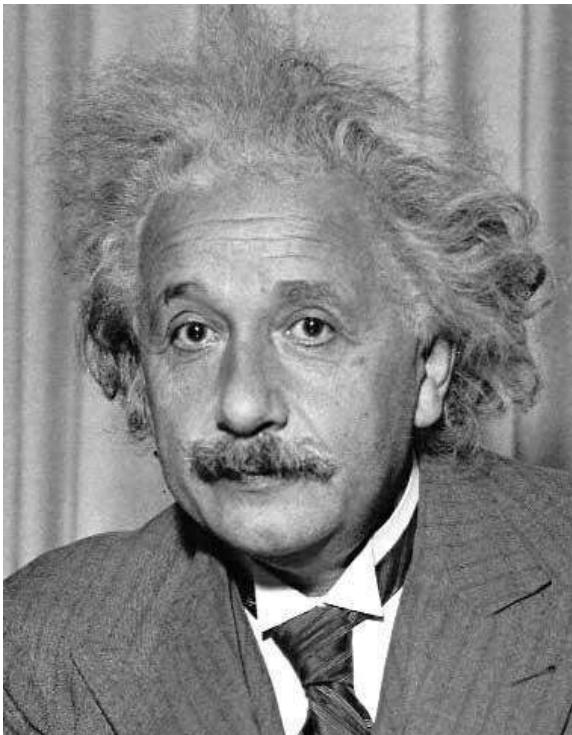
---

- Goal: find  in image
- Method 0: filter the image with eye patch

$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$



**f = image**  
**g = filter**



**Input**

**Filtered Image**

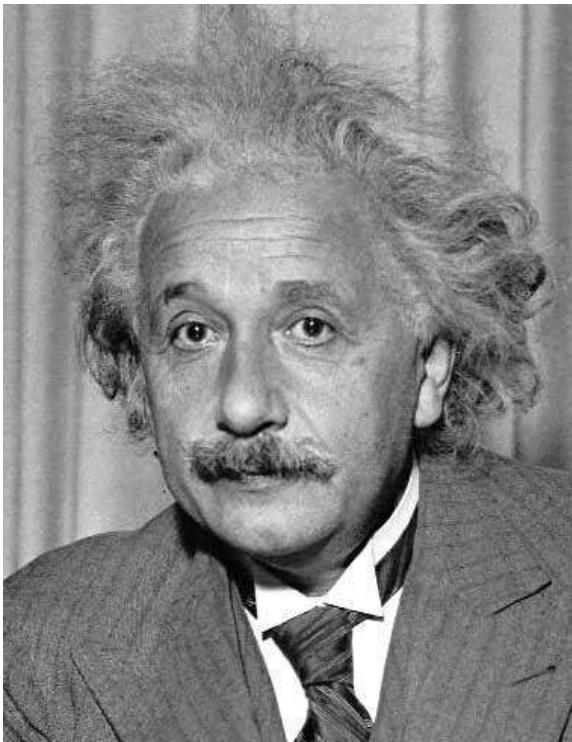
**What went wrong?**  
**Problem:** response is  
stronger for higher  
intensity

# Matching with filters

---

- Goal: find  in image
- Method 1: filter the image with zero-mean eye

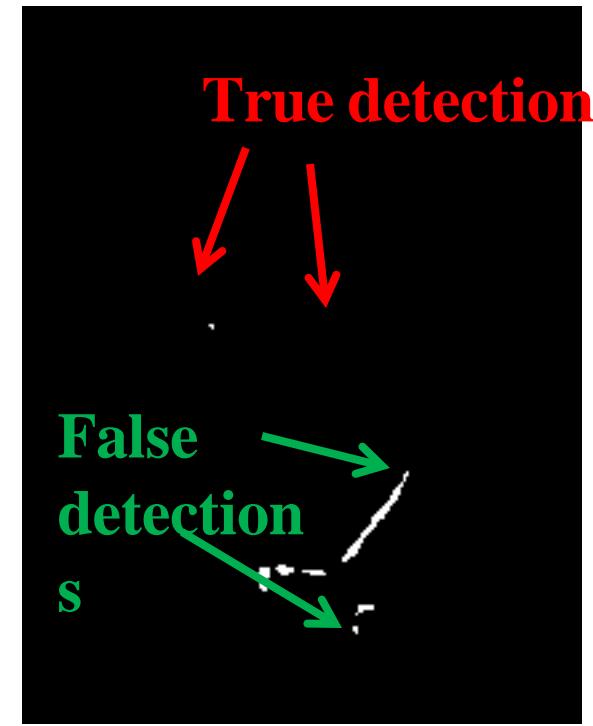
$$h[m, n] = \sum_{k, l} (f[k, l] - \bar{f}) \underbrace{(g[m + k, n + l])}_{\text{mean of } f}$$



Input



Filtered Image (scaled)



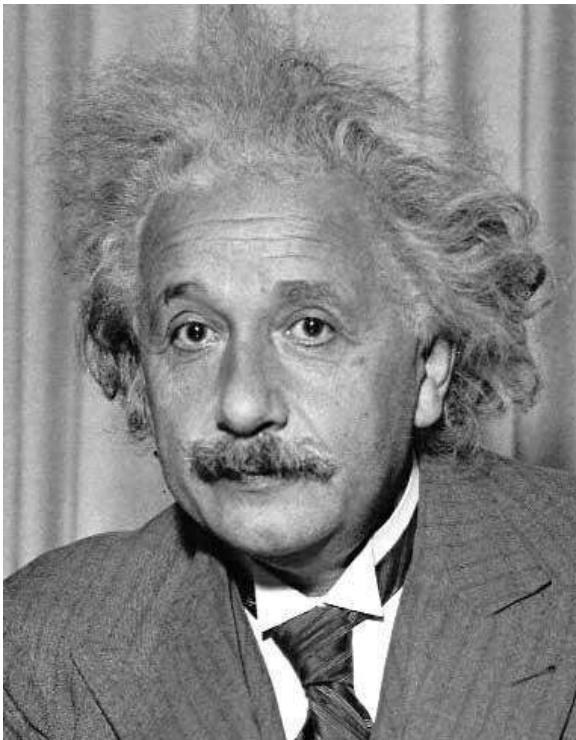
Thresholded Image

# Matching with filters

---

- Goal: find  in image
- Method 2: SSD

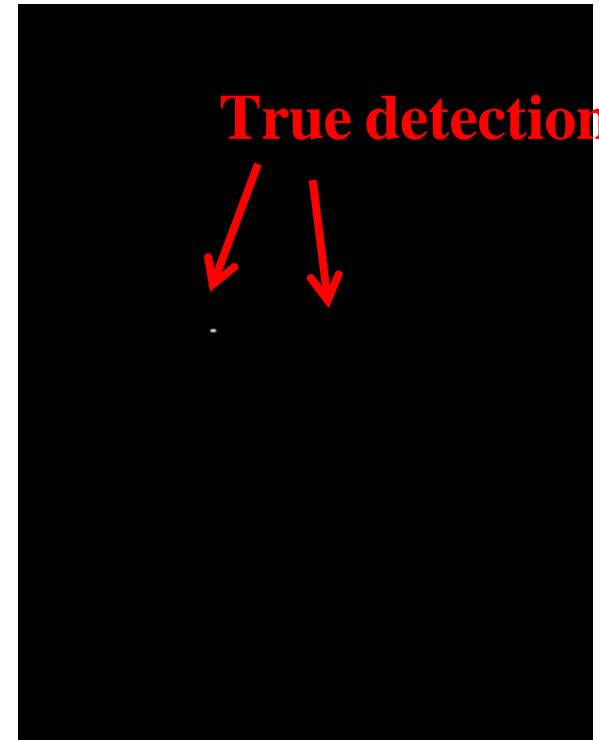
$$h[m, n] = \sum_{k, l} (g[k, l] - f[m + k, n + l])^2$$



Input



1 -  $\text{sqrt}(\text{SSD})$



Thresholded Image

True detections

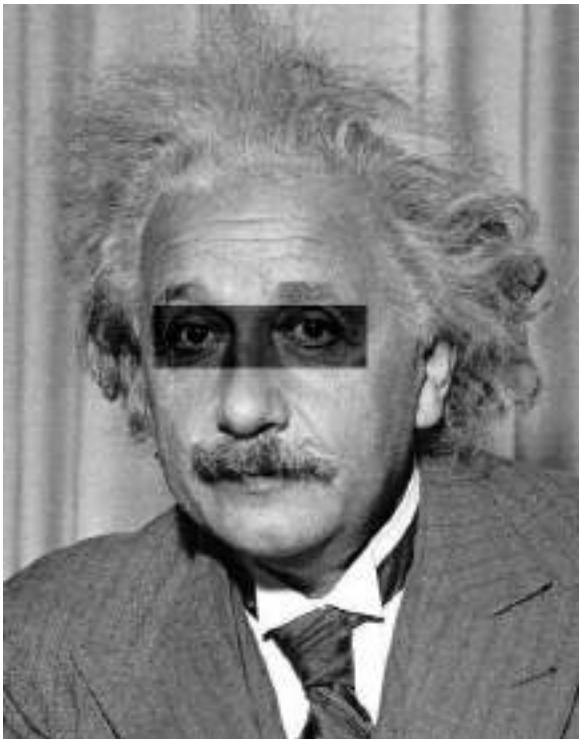
# Matching with filters

---

- Goal: find  in image
- Method 2: SSD

**What's the potential downside of SSD?**  
: SSD sensitive to average intensity

$$h[m, n] = \sum_{k, l} (g[k, l] - f[m + k, n + l])^2$$



**Input**



**1- sqrt(SSD)**

# Matching with filters

---

- Goal: find  in image
- Method 3: Normalized cross-correlation

$$h[m, n] = \frac{\sum_{k,l} (g[k, l] - \bar{g})(f[m - k, n - l] - \bar{f}_{m,n})}{\left( \sum_{k,l} (g[k, l] - \bar{g})^2 \sum_{k,l} (f[m - k, n - l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

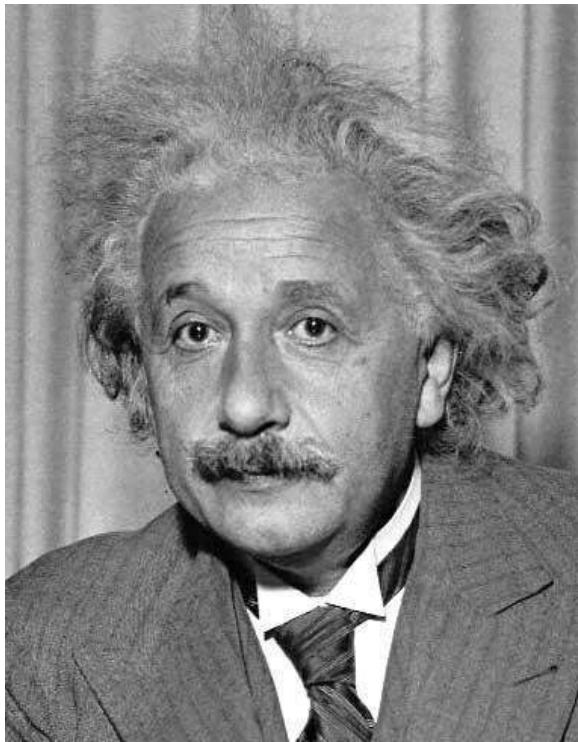
mean template                            mean image patch

Matlab: `normxcorr2(template, im)`

# Matching with filters

---

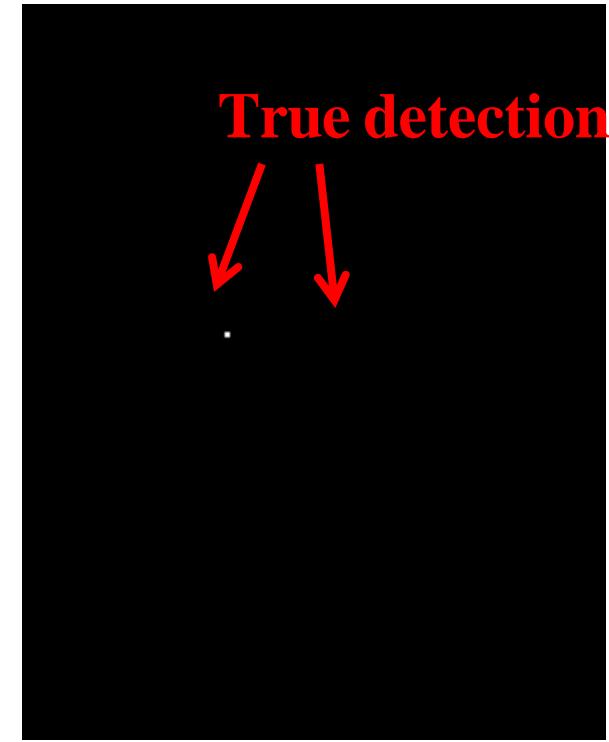
- Goal: find  in image
- Method 3: Normalized cross-correlation



**Input**



**Normalized X-Correlation**

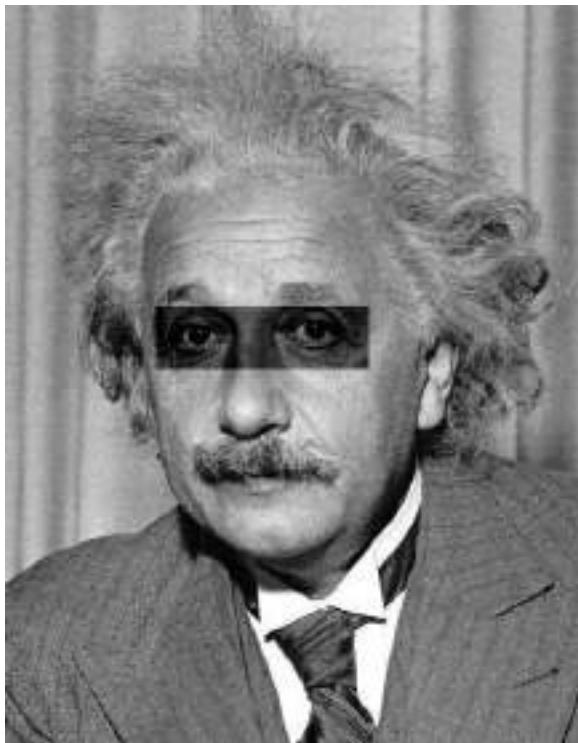


**Thresholded Image**

# Matching with filters

---

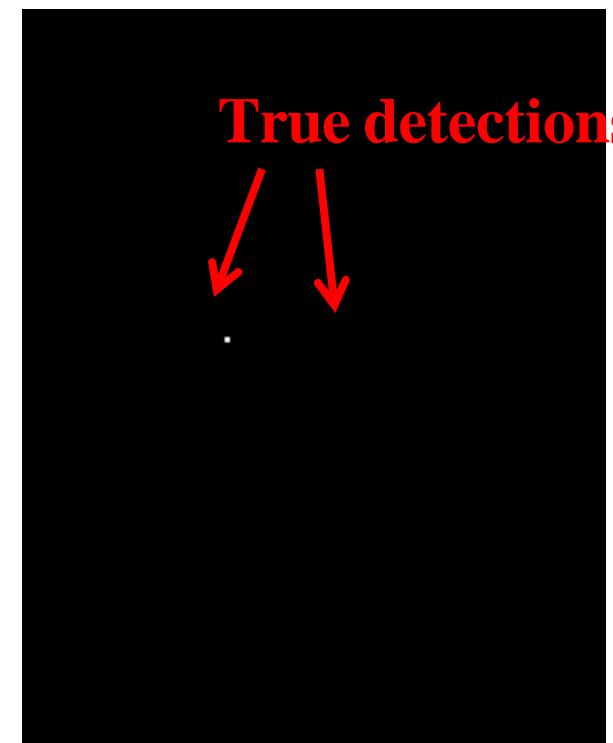
- Goal: find  in image
- Method 3: Normalized cross-correlation



Input



Normalized X-Correlation Thresholded Image



True detections

# Q: What is the best method to use?

---

A: Depends

- SSD: faster, sensitive to overall intensity
- Normalized cross-correlation: slower, invariant to local average intensity and contrast

Q: What if we want to find larger or smaller eyes?

---

- A: Image Pyramid

# Review of Sampling

---



# Gaussian pyramid

---



Source: Forsyth

# Template Matching with Image Pyramids

---

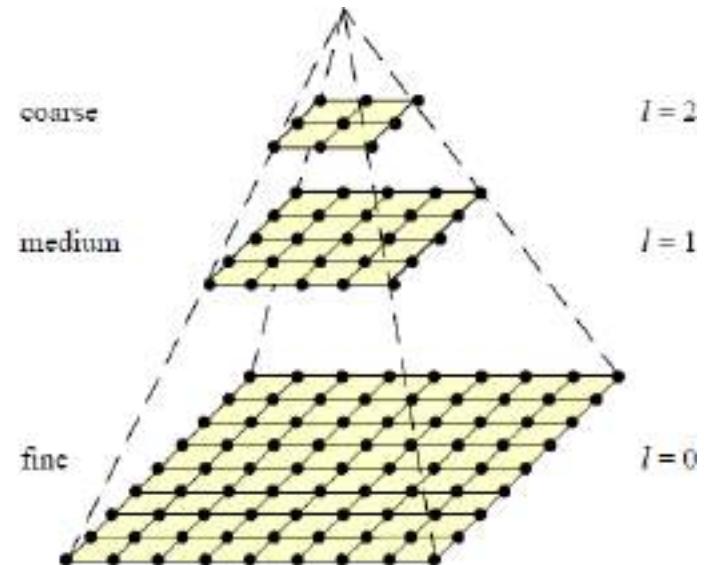
Input: Image, Template

1. Match template at current scale
2. Downsample image
3. Repeat 1-2 until image is very small
4. Take responses above some threshold, perhaps with non-maxima suppression

# Coarse-to-fine Image Registration

---

1. Compute Gaussian pyramid
2. Align with coarse pyramid
3. Successively align with finer pyramids
  - Search smaller range

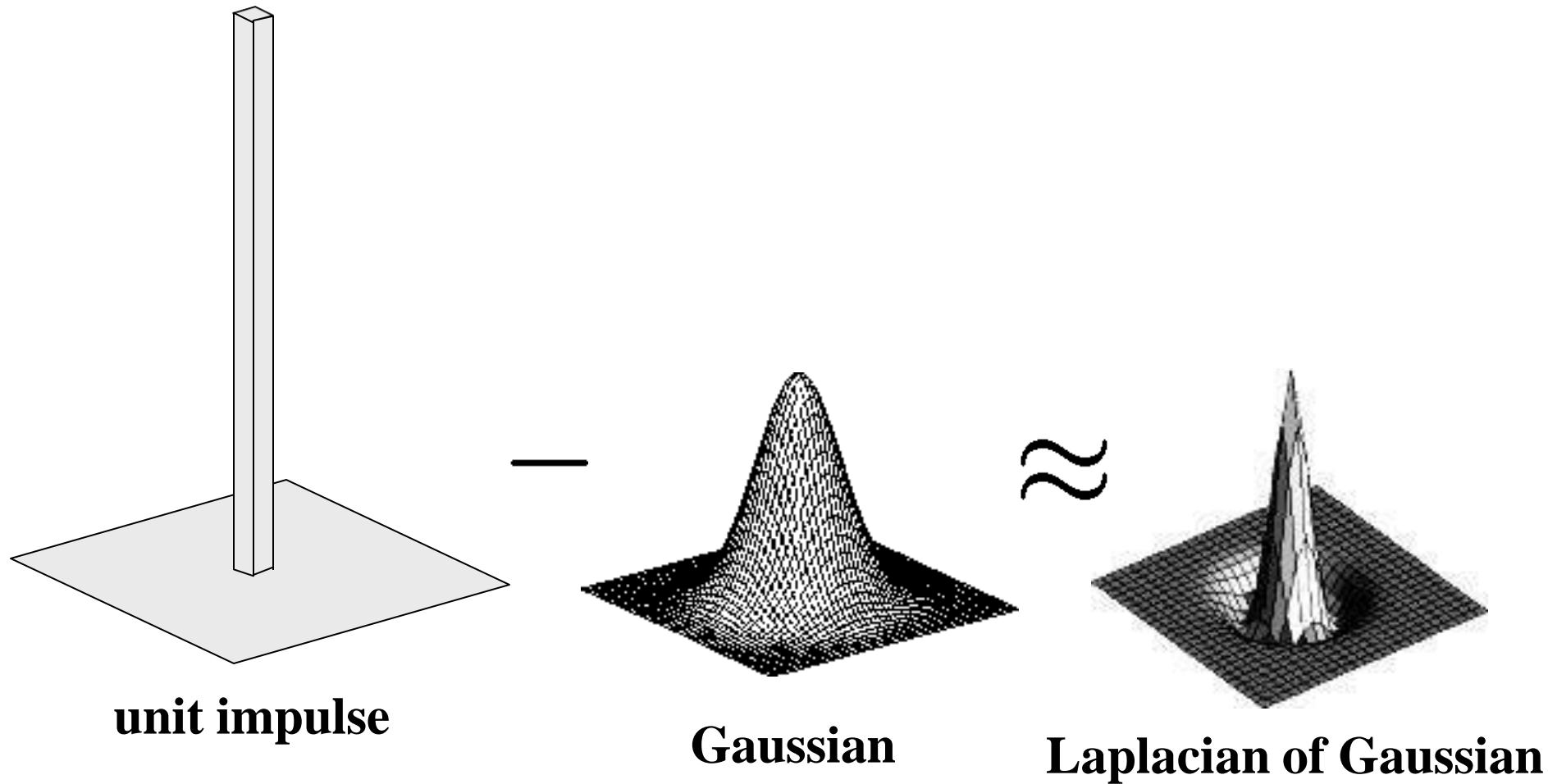


Why is this faster?

Are we guaranteed to get  
the same result?

# Laplacian filter

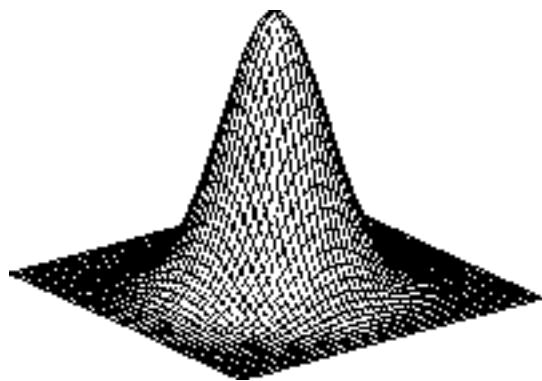
---



Source: Lazebnik

# 2D edge detection filters

---



**Gaussian**

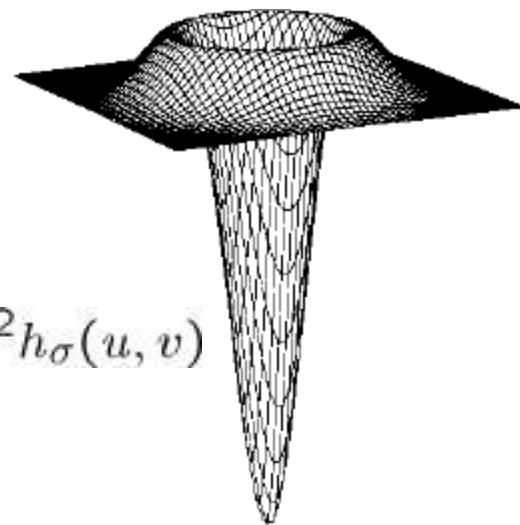
$$h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



**derivative of Gaussian**

$$\frac{\partial}{\partial x} h_\sigma(u, v)$$

**Laplacian of Gaussian**



$\nabla^2$  is the **Laplacian operator**:

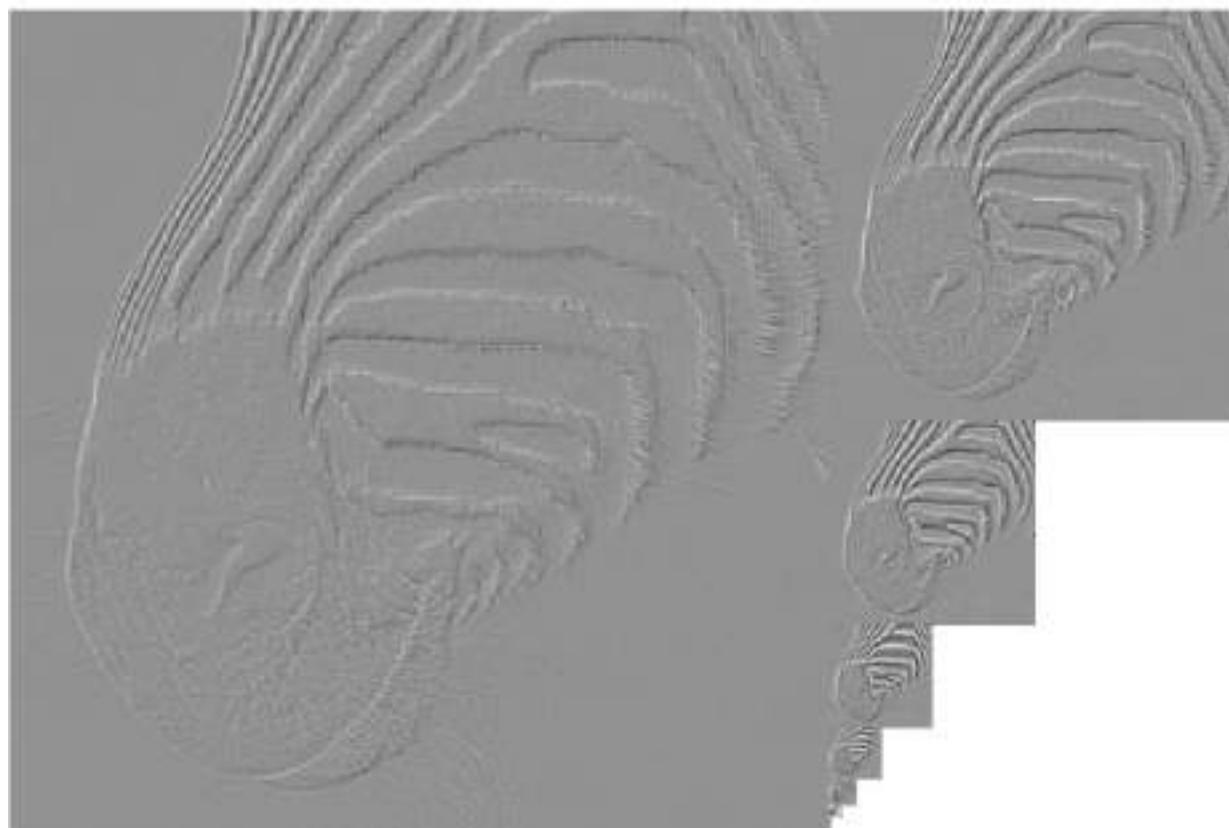
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

# Laplacian pyramid

---

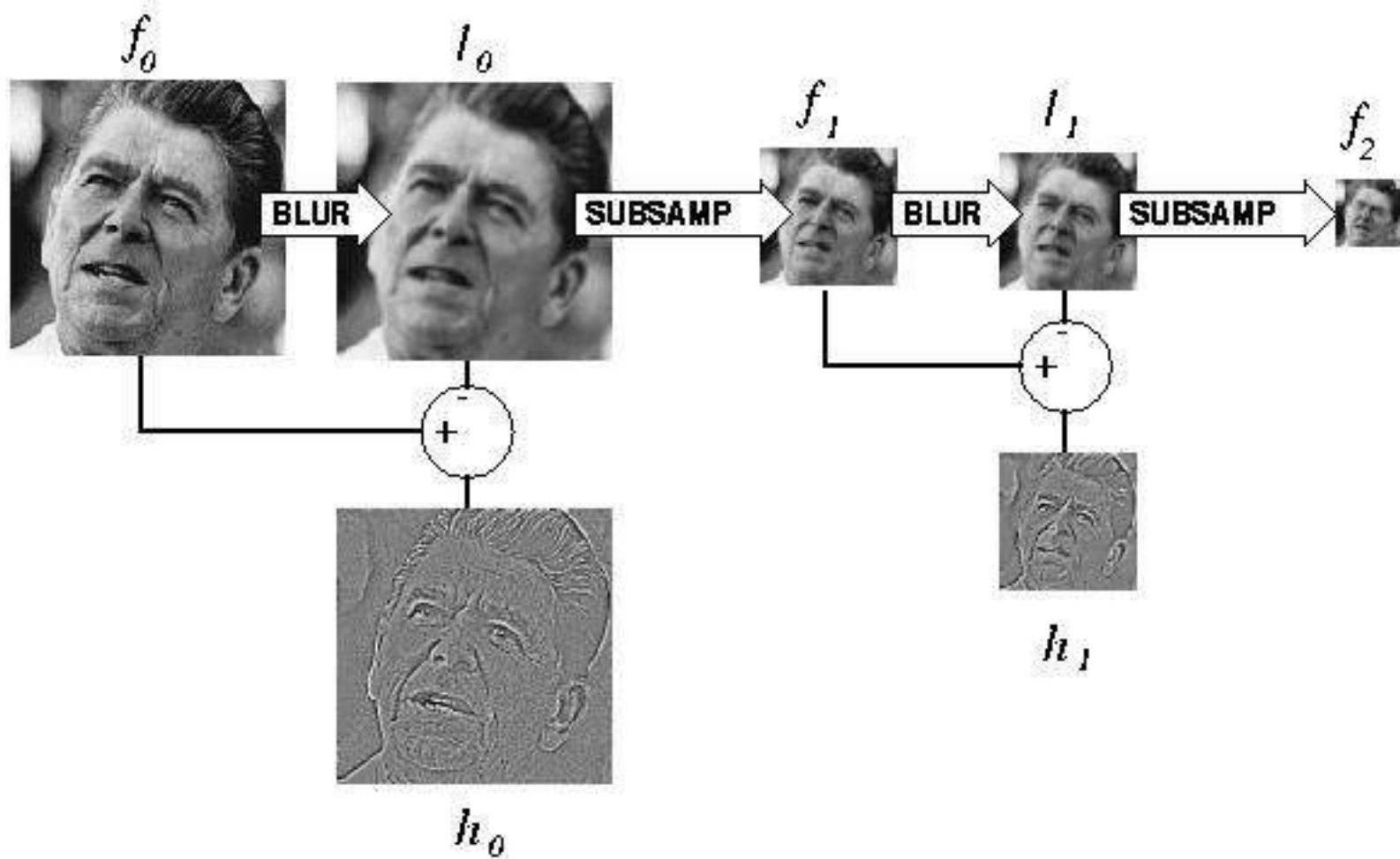


512      256      128      64      32      16      8



Source: Forsyth

# Computing Gaussian/Laplacian Pyramid



# Hybrid Image

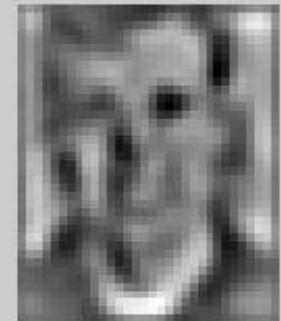
---



# Hybrid Image in Laplacian Pyramid

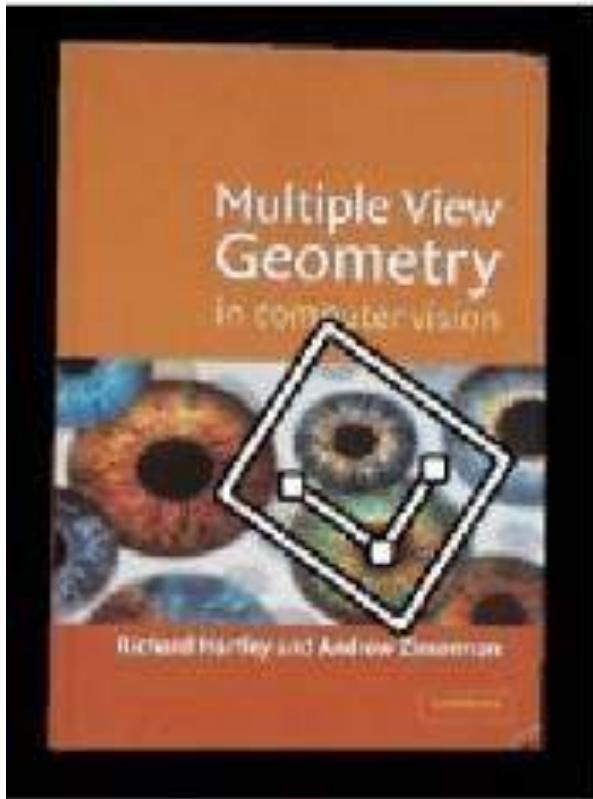
---

High frequency → Low frequency



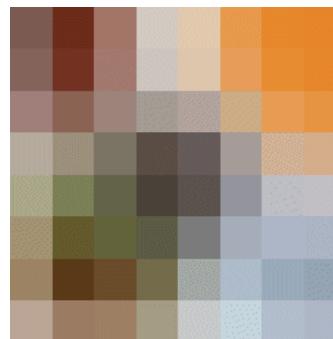
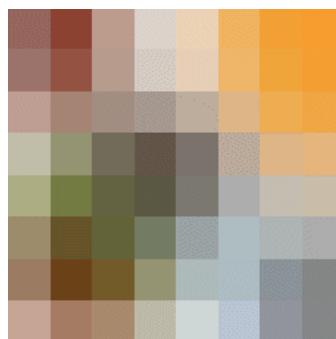
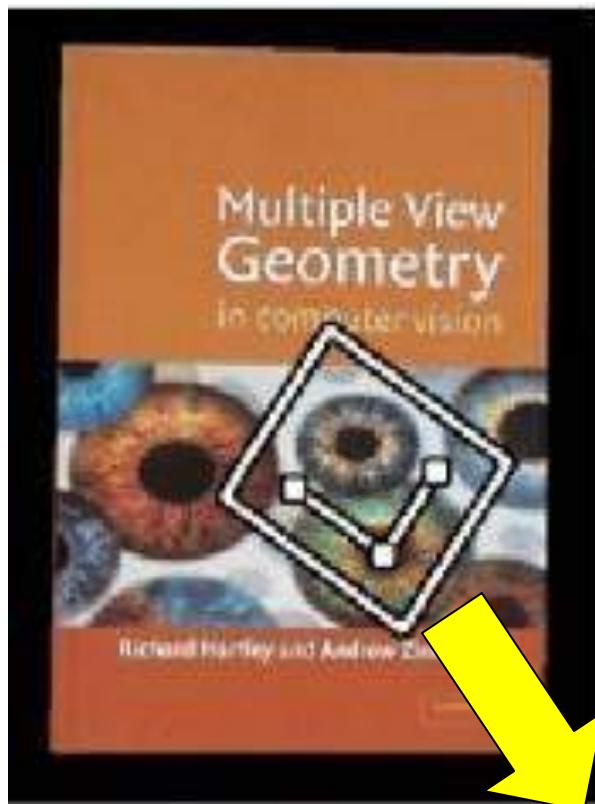
# Image Matching

---



# Image Matching

---

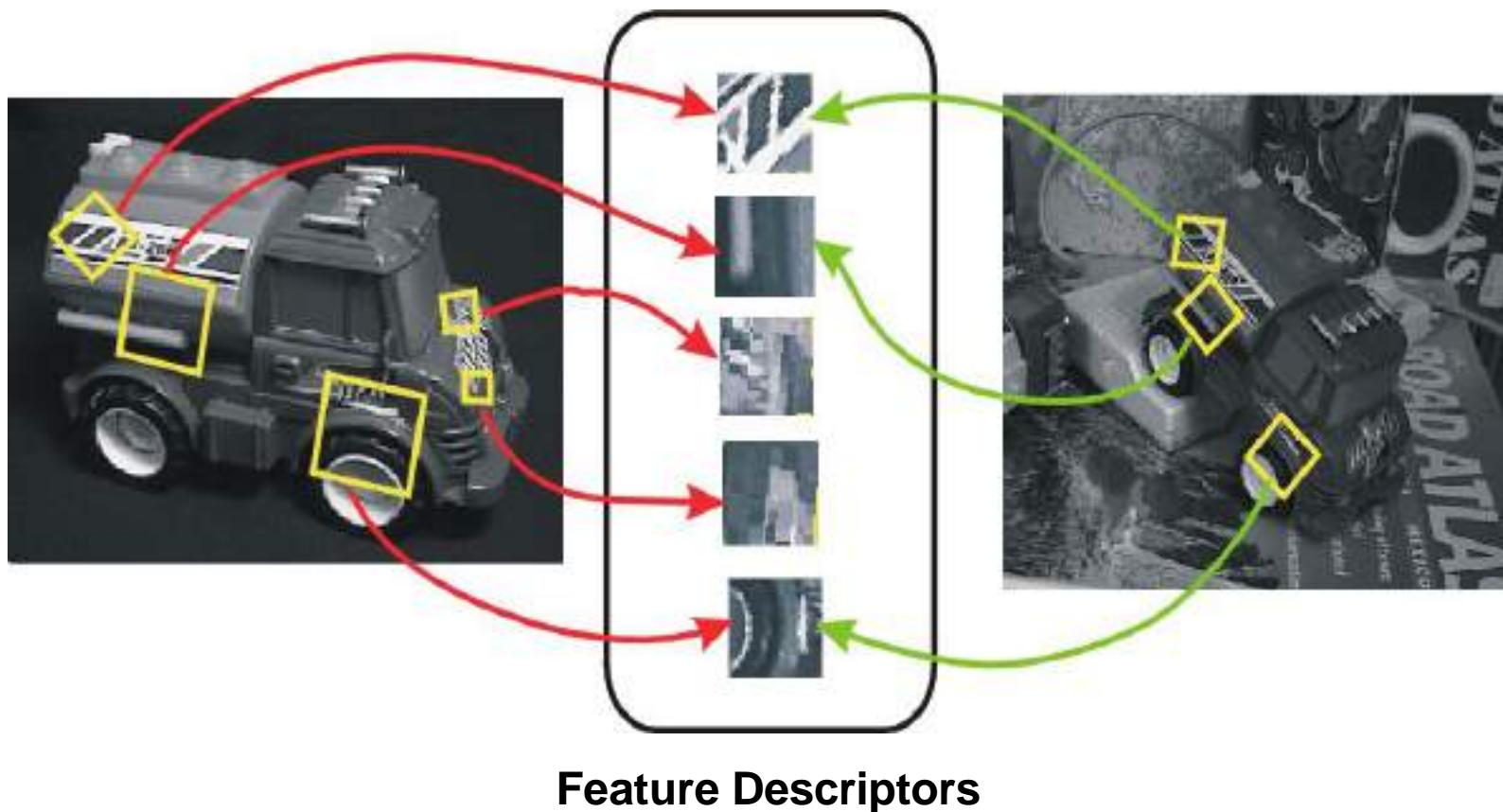


# Invariant local features

---

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, ...



# Invariant Local Feature Methods

---

Method	Geometric Invariance	Photometric Invariance
Harris Corner Detector	Translation	✗
SIFT (Scale-Invariant Feature Transform)	Translation, Rotation, Scale	Partial brightness invariance
SURF	Translation, Rotation, Scale	Partial
ORB	Translation, Rotation, Scale	Partial
BRISK	Translation, Rotation, Scale	Partial

# Advantages of local features

---

## Locality

- features are local, so robust to occlusion and clutter

## Distinctiveness:

- can differentiate a large database of objects

## Quantity

- hundreds or thousands in a single image

## Efficiency

- real-time performance achievable

## Generality

- exploit different types of features in different situations

# More motivation...

---

Feature points are used for:

- Image alignment
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- ... other

# What makes a good feature?

---



# Want uniqueness

---

Look for image regions that are unusual

- Lead to unambiguous matches in other images

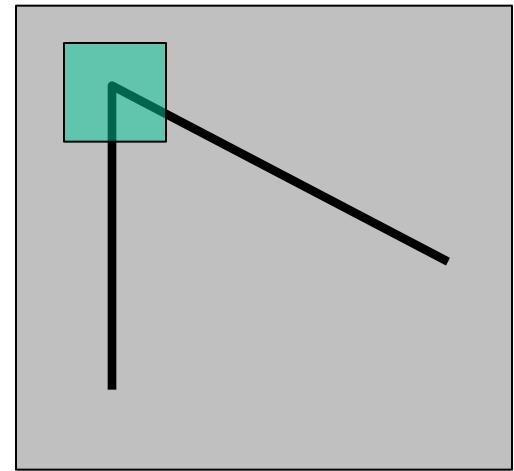
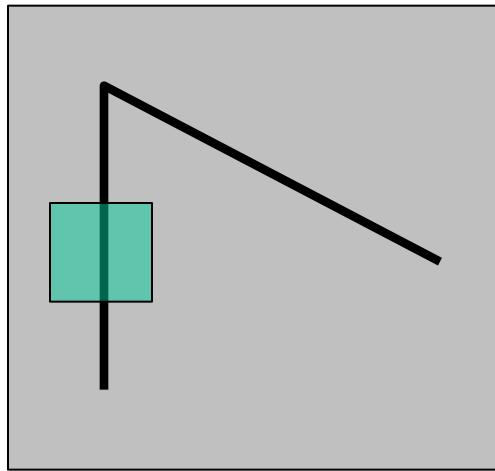
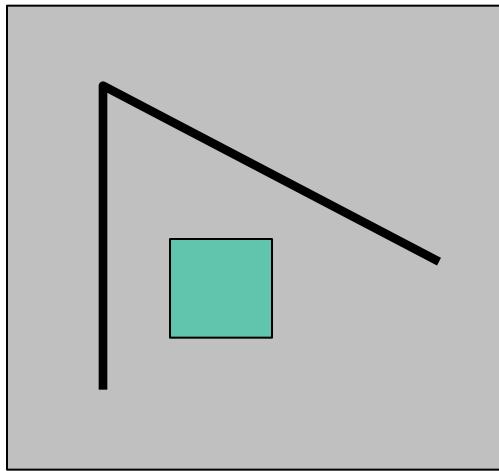
How to define “unusual”?

# Local measures of uniqueness

---

Suppose we only consider a small window of pixels

- What defines whether a feature is a good or bad candidate?

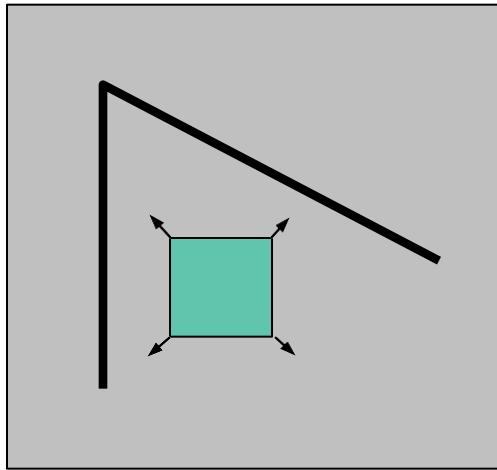


# Feature detection

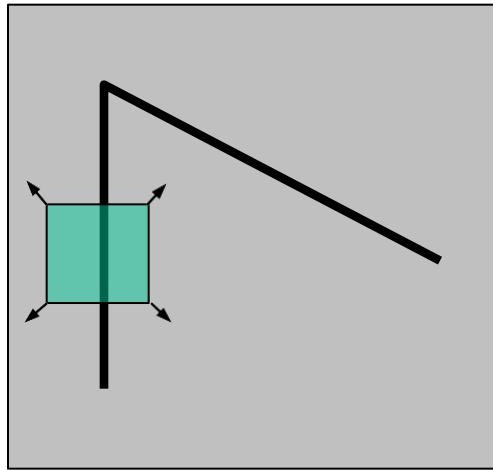
---

Local measure of feature uniqueness

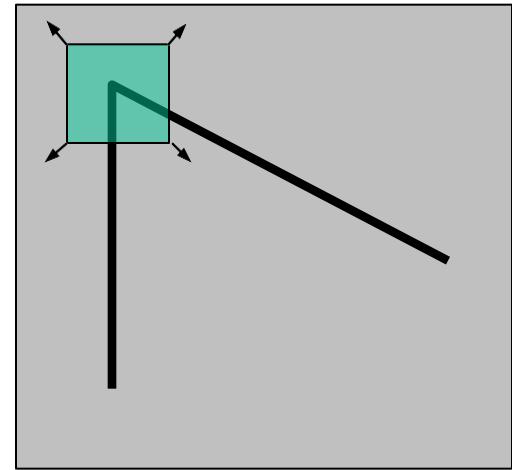
- How does the window change when you shift it?
- Shifting the window in *any direction* causes a *big change*



“flat” region:  
no change in all  
directions



“edge”:  
no change along  
the edge direction



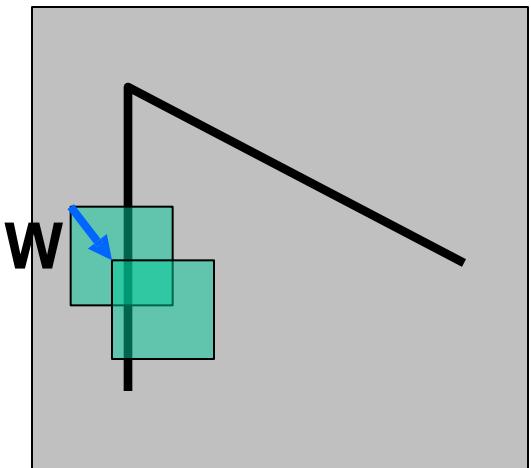
“corner”:  
significant change  
in all directions

# Feature detection: the math

---

Consider shifting the window  $\mathbf{W}$  by  $(u,v)$

- how do the pixels in  $\mathbf{W}$  change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD “error” of  $E(u,v)$ :



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

# Small motion assumption

---

Taylor Series expansion of I:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion  $(u, v)$  is small, then first order approx is good

$$I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

shorthand:  $I_x = \frac{\partial I}{\partial x}$

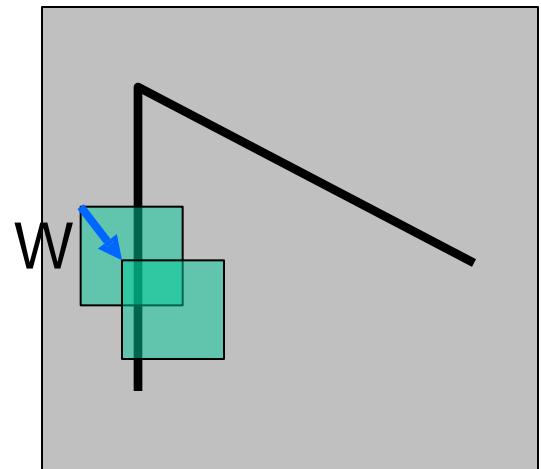
Plugging this into the formula on the previous slide...

# Feature detection: the math

---

Consider shifting the window  $W$  by  $(u, v)$

- how do the pixels in  $W$  change?
- compare each pixel before and after by summing up the squared differences
- this defines an “error” of  $E(u, v)$ :



$$\begin{aligned} E(u, v) &= \sum_{(x,y) \in W} [I(x + u, y + v) - I(x, y)]^2 \\ &\approx \sum_{(x,y) \in W} [I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix}^2 - I(x, y)] \\ &\approx \sum_{(x,y) \in W} \left[ [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \right]^2 \end{aligned}$$

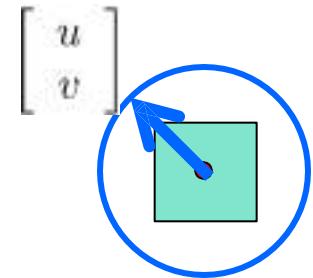
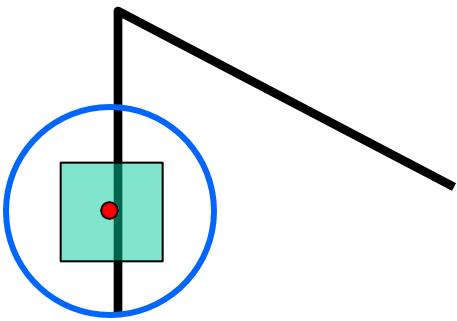
# Feature detection: the math

---

This can be rewritten:

$$E(u, v) = \sum_{(x,y) \in W} [u \ v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$H$



For the example above

- You can move the center of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of  $H$

# Quick eigenvalue/eigenvector review

---

The **eigenvectors** of a matrix  $\mathbf{A}$  are the vectors  $\mathbf{x}$  that satisfy:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

The scalar  $\lambda$  is the **eigenvalue** corresponding to  $\mathbf{x}$

- The eigenvalues are found by solving:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

- In our case,  $\mathbf{A} = \mathbf{H}$  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:

$$\lambda_{\pm} = \frac{1}{2} \left[ (h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2} \right]$$

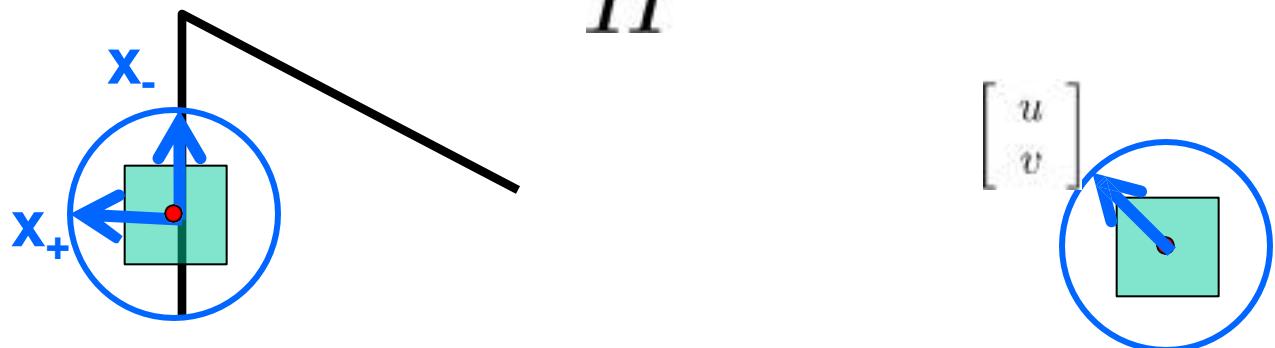
Once you know  $\lambda$ , you find  $\mathbf{x}$  by solving

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

# Feature detection: the math

This can be rewritten:

$$E(u, v) = \sum_{(x,y) \in W} [u \ v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$



## Eigenvalues and eigenvectors of $H$

- Define shifts with the smallest and largest change (E value)
- $x_+$  = direction of **largest** increase in E.
- $\lambda_+$  = amount of increase in direction  $x_+$
- $x_-$  = direction of **smallest** increase in E.
- $\lambda_-$  = amount of increase in direction  $x_-$

$$Hx_+ = \lambda_+ x_+$$

$$Hx_- = \lambda_- x_-$$

# Feature detection: the math

---

How are  $\lambda_+$ ,  $\mathbf{x}_+$ ,  $\lambda_-$ , and  $\mathbf{x}_-$  relevant for feature detection?

- What's our feature scoring function?

# Feature detection: the math

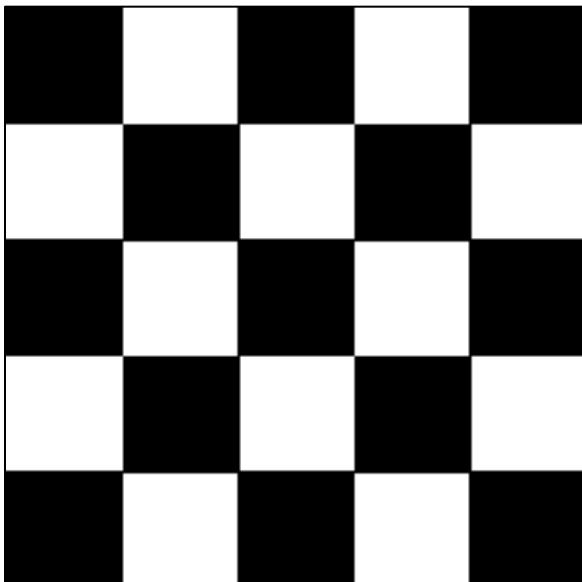
---

How are  $\lambda_+$ ,  $\mathbf{x}_+$ ,  $\lambda_-$ , and  $\mathbf{x}_-$  relevant for feature detection?

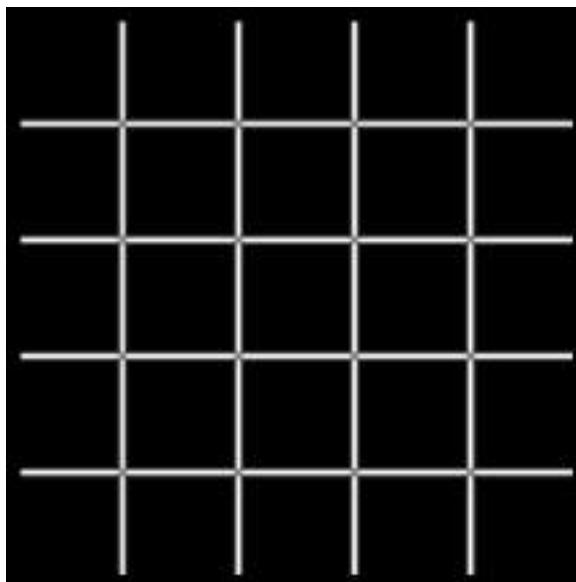
- What's our feature scoring function?

Want  $E(u,v)$  to be **large** for small shifts in **all** directions

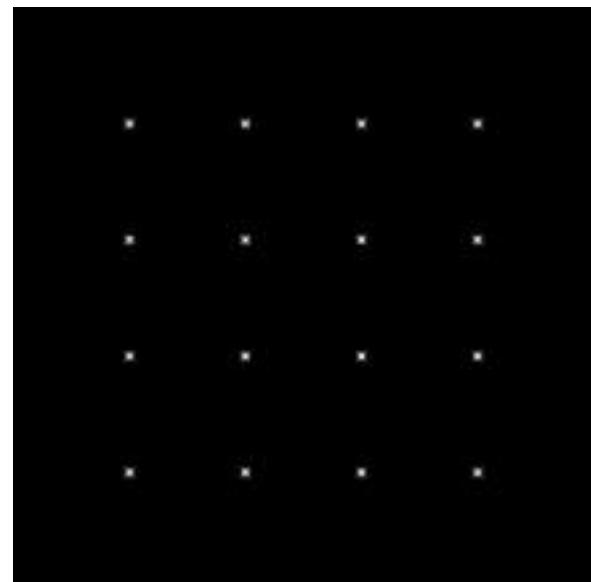
- the *minimum* of  $E(u,v)$  should be large, over all unit vectors  $[u \ v]$
- this minimum is given by the smaller eigenvalue ( $\lambda_-$ ) of  $\mathbf{H}$



$I$



$\lambda_+$



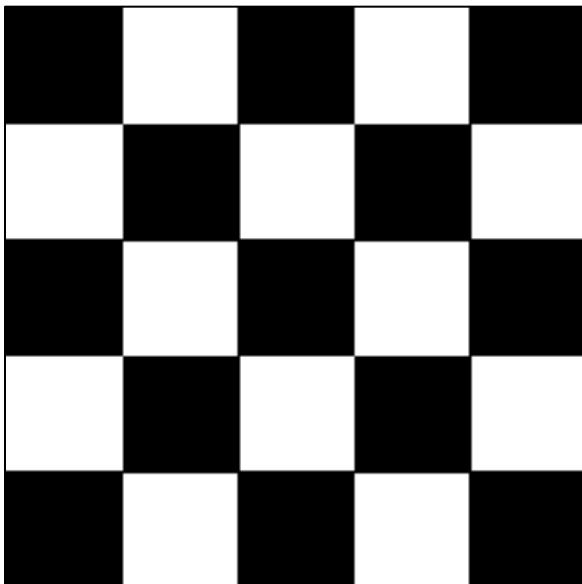
$\lambda_-$

# Feature detection summary ( $\lambda_-$ )

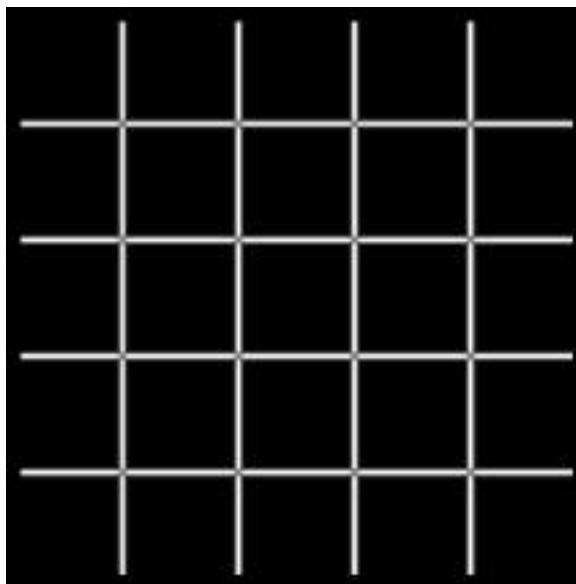
---

Here's what you do

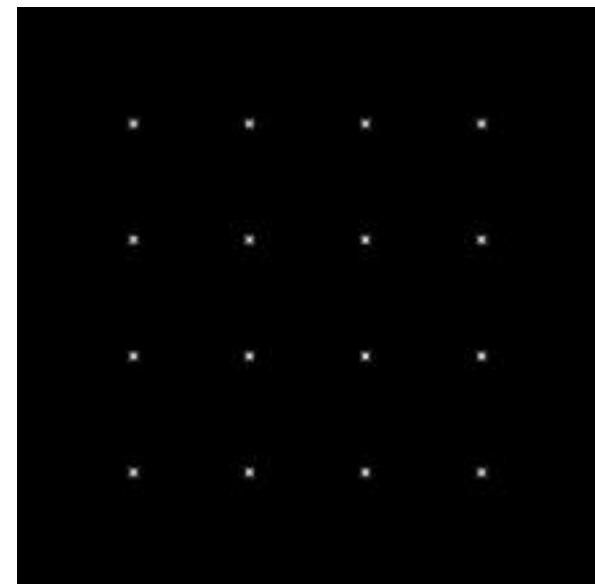
- Compute the gradient at each point in the image
- Create the  $H$  matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response ( $\lambda_- > \text{threshold}$ )
- Choose those points where  $\lambda_-$  is a local maximum as features



$I$



$\lambda_+$



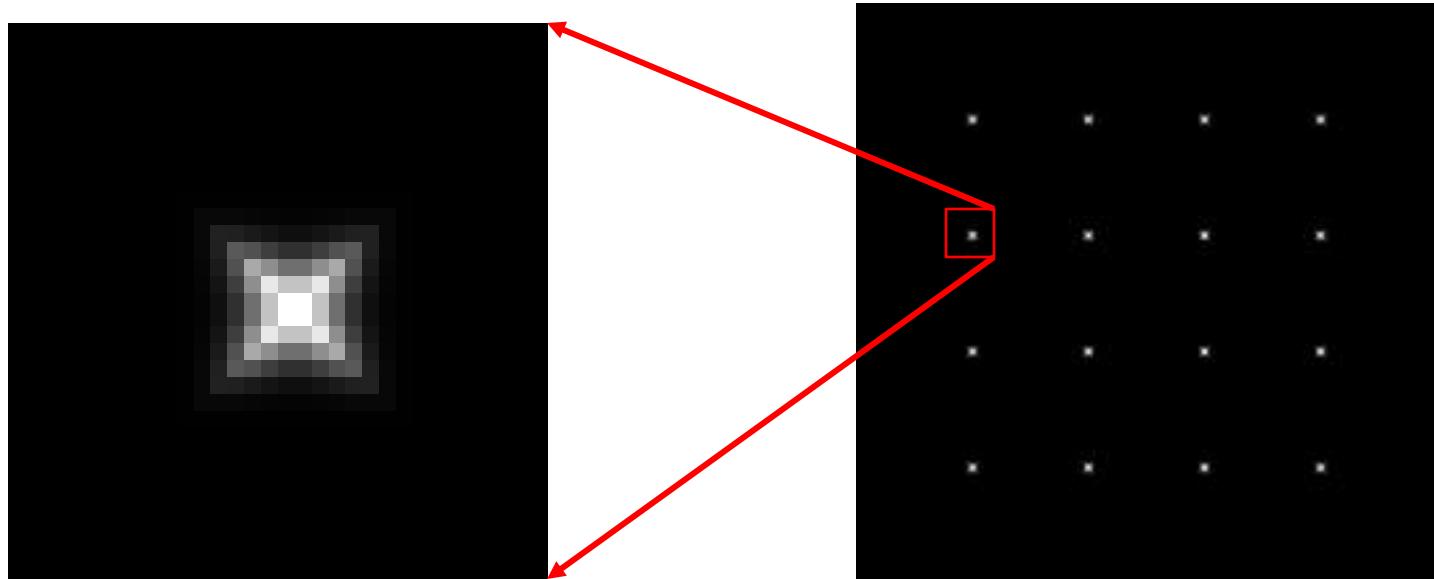
$\lambda_-$

# Feature detection summary ( $\lambda_-$ )

---

Here's what you do

- Compute the gradient at each point in the image
- Create the  $H$  matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response ( $\lambda_- > \text{threshold}$ )
- Choose those points where  $\lambda_-$  is a local maximum as features



$\lambda_-$

# The Harris operator

---

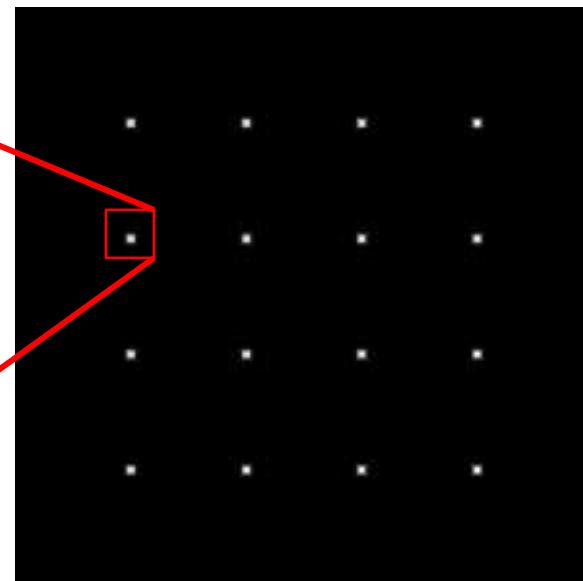
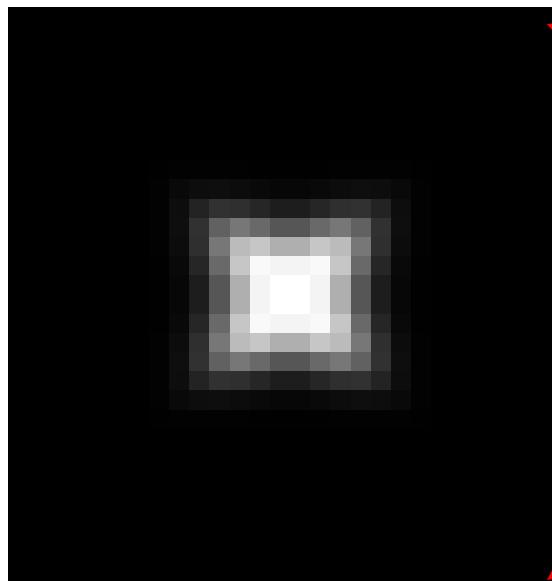
$\lambda_{\perp}$  is a variant of the “Harris operator” for feature detection

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$
$$= \frac{\text{determinant}(H)}{\text{trace}(H)}$$

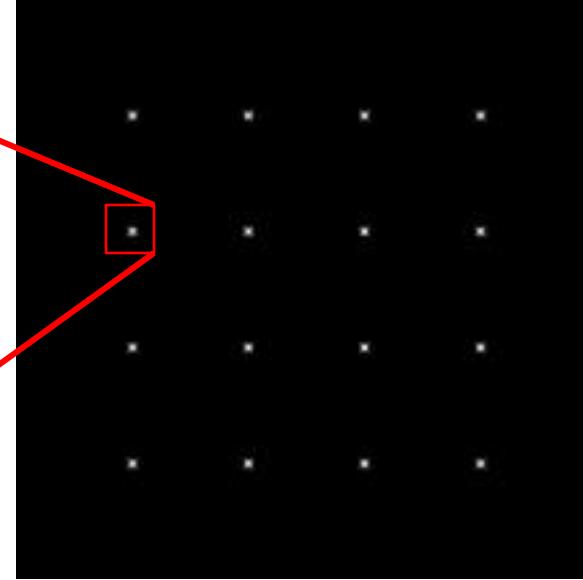
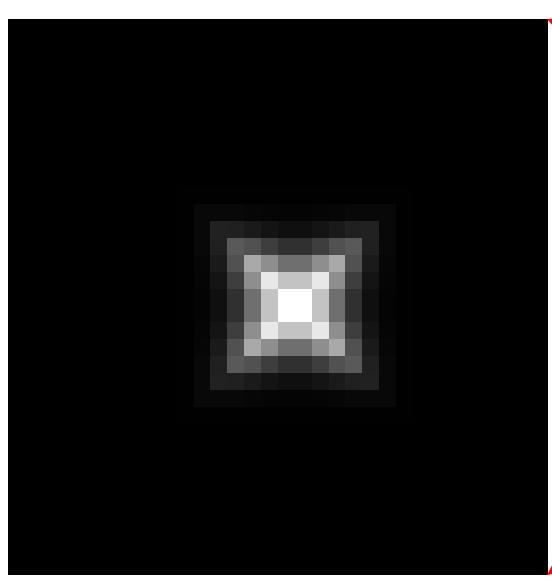
- The *trace* is the sum of the diagonals, i.e.,  $\text{trace}(H) = h_{11} + h_{22}$
- Very similar to  $\lambda_{\perp}$  but less expensive (no square root)
- Called the “Harris Corner Detector” or “Harris Operator”
- Lots of other detectors, this is one of the most popular

# The Harris operator

---



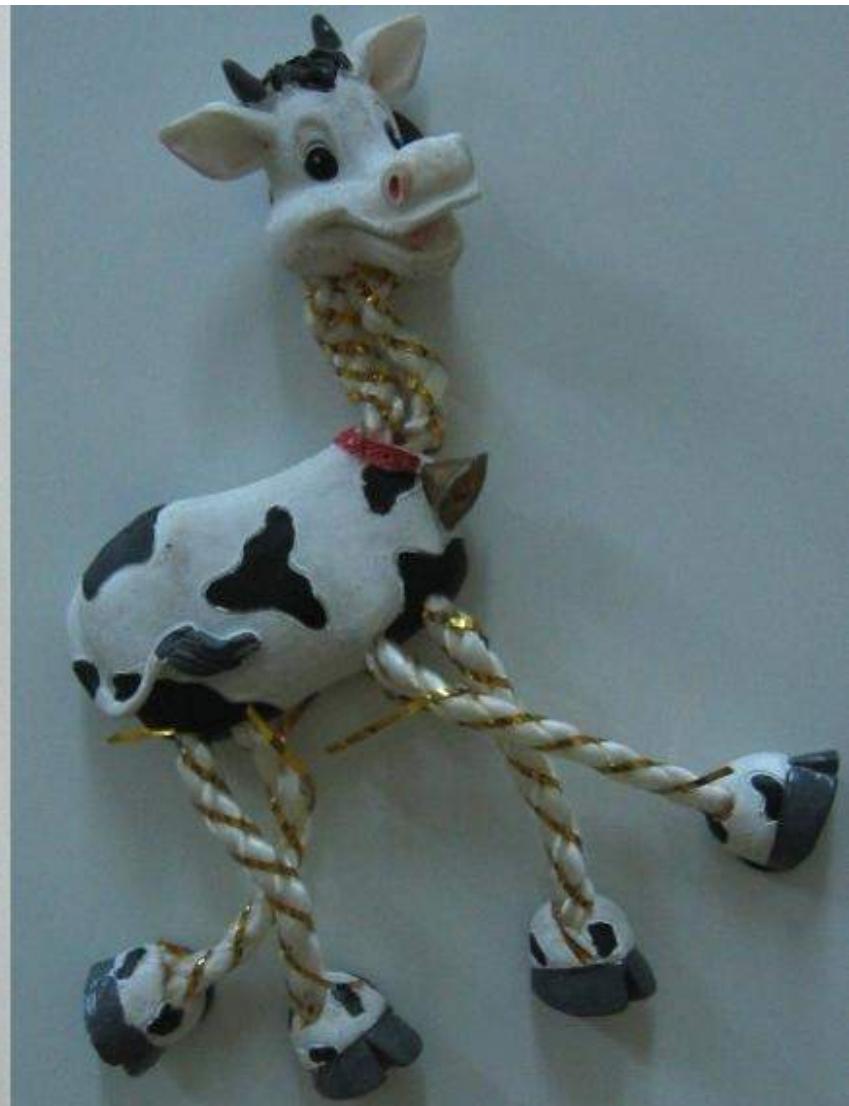
Harris  
operator



$\lambda_-$

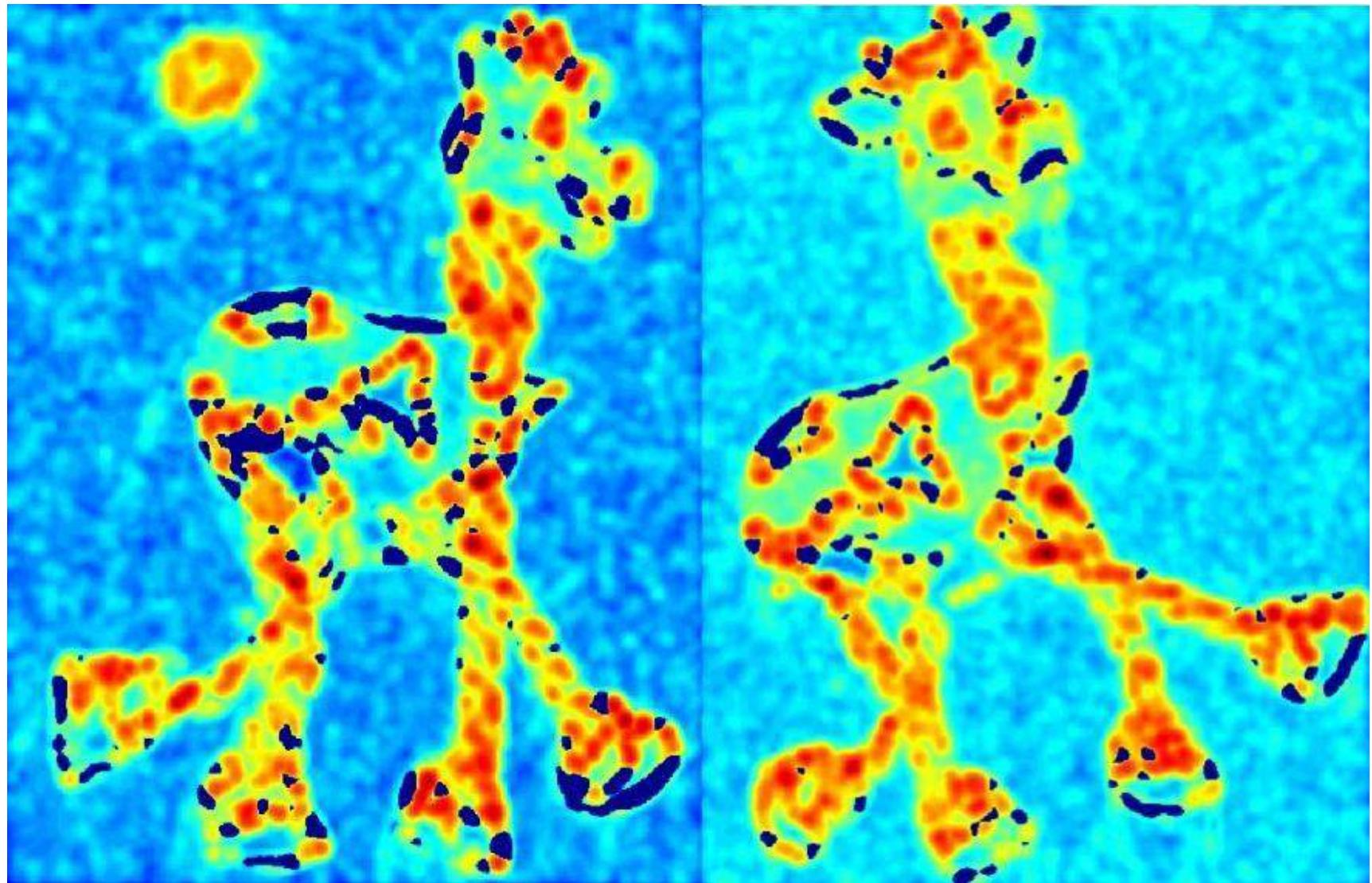
# Harris detector example

---



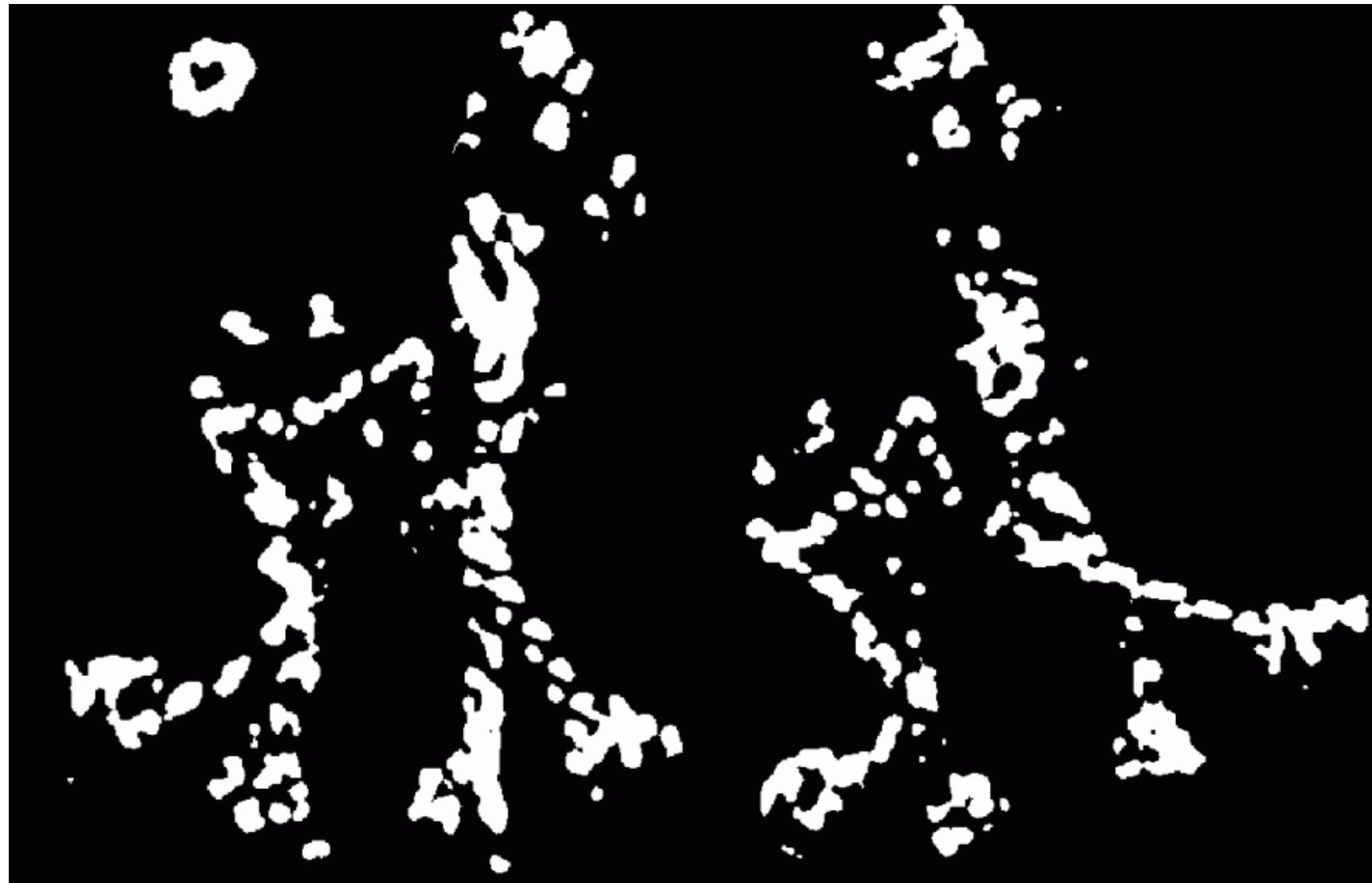
$f$  value (red high, blue low)

---



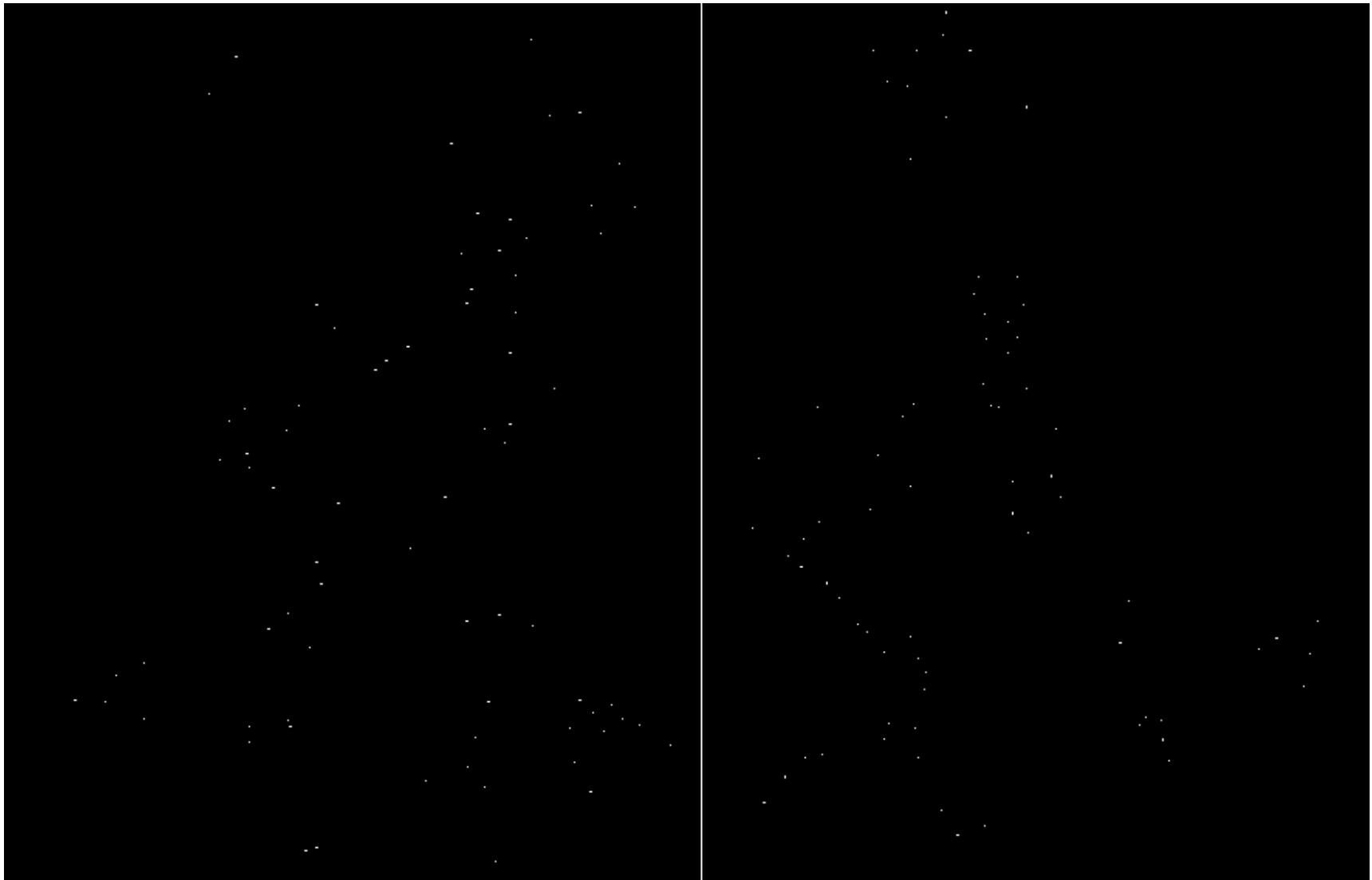
## Threshold ( $f > \text{value}$ )

---



# Find local maxima of $f$

---



# Harris features (in red)

---



# Invariance

---

Suppose you **rotate** the image by some angle

- Will you still pick up the same features?

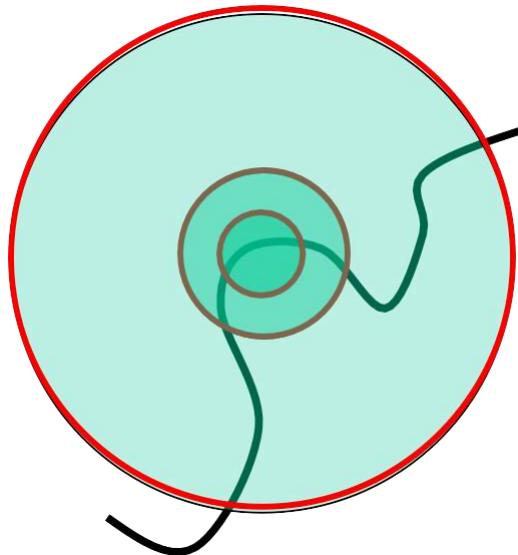
What if you change the brightness?

Scale?

# Scale invariant detection

---

Suppose you're looking for corners

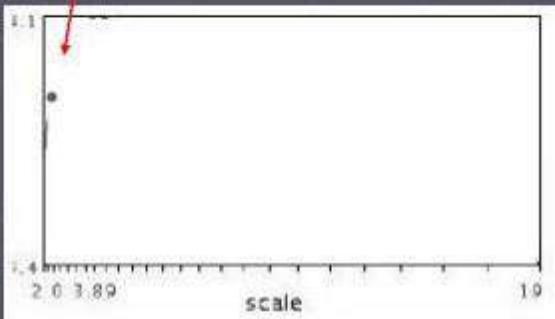


Key idea: find scale that gives local maximum of  $f$

- $f$  is a local maximum in both position and scale
- Common definition of  $f$ : Laplacian  
(or difference between two Gaussian filtered images with different sigmas)

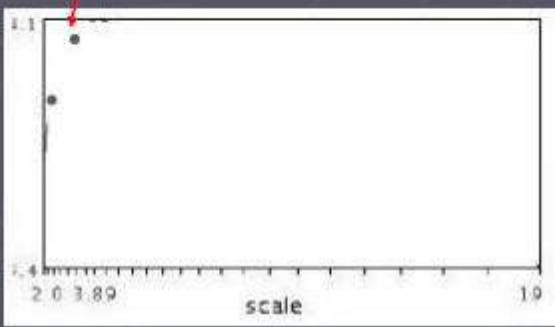
# Automatic scale selection

Lindeberg et al., 1996



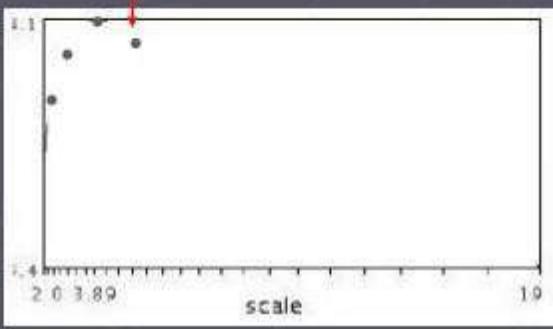
$$f(I_{i_1, i_m}(x, \sigma))$$

# Automatic scale selection



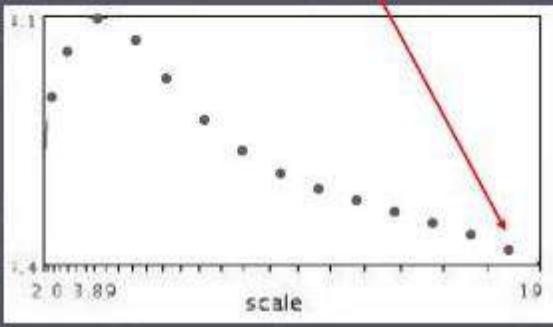
$f(I_{i_1 \dots i_m}(x, \sigma))$

# Automatic scale selection



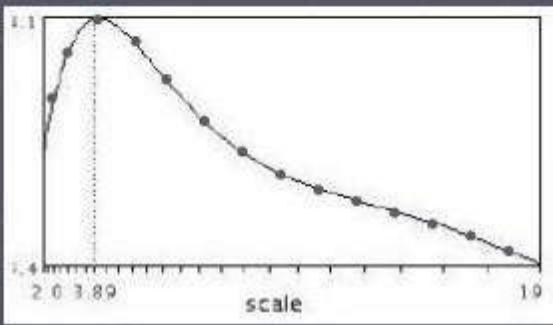
$f(I_{i_c, i_m}(x, \sigma))$

# Automatic scale selection



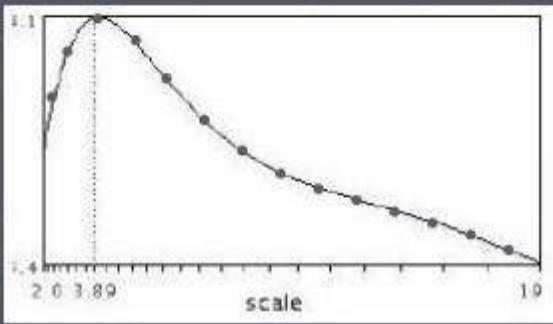
$$f(I_{i_1...i_m}(x, \sigma))$$

# Automatic scale selection

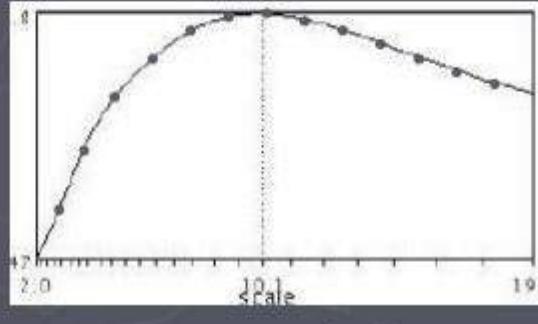


$$f(I_{i_*, i_m}(x, \sigma))$$

# Automatic scale selection



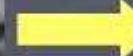
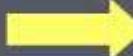
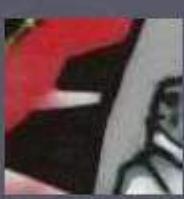
$f(I_{i_1...i_m}(x, \sigma))$



$f(I_{i_1...i_m}(x', \sigma'))$

# Automatic scale selection

Normalize: rescale to fixed size

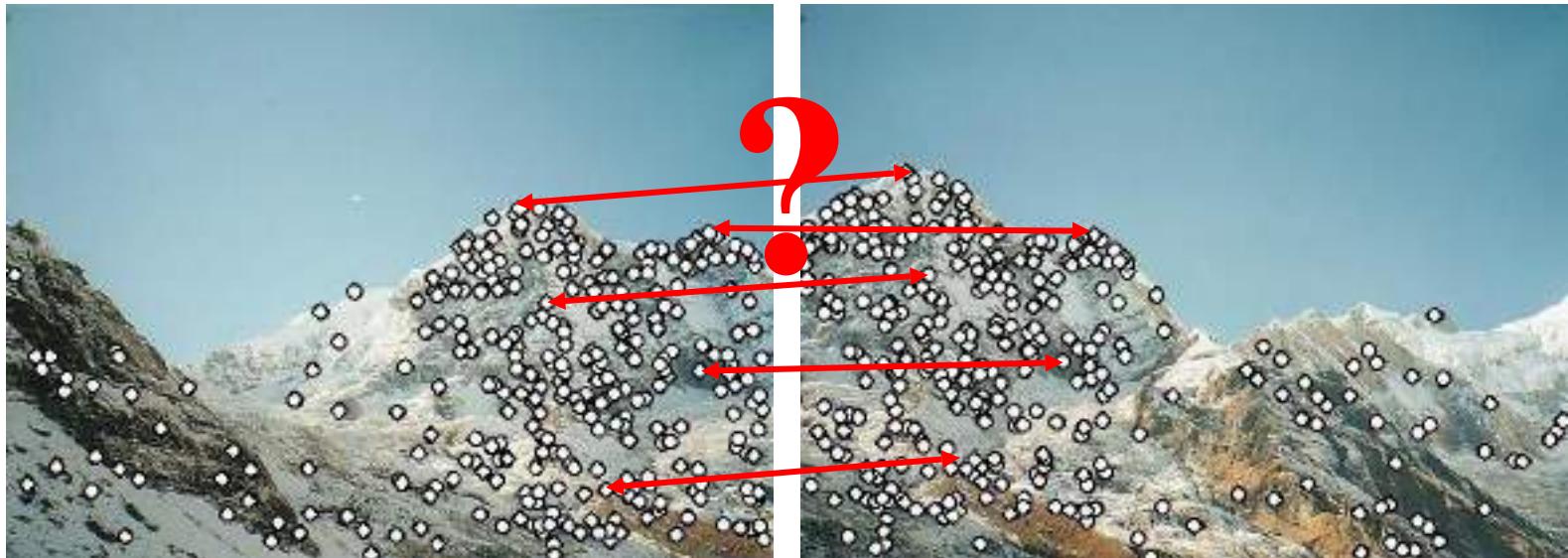


# Feature descriptors

---

We know how to detect good points

Next question: **How to match them?**



Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC <http://www.cs.ubc.ca/~lowe/keypoints/>

# Invariance

---

Suppose we are comparing two images  $I_1$  and  $I_2$

- $I_2$  may be a transformed version of  $I_1$
- What kinds of transformations are we likely to encounter in practice?

We'd like to find the same features regardless of the transformation

- This is called transformational ***invariance***
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about **60** degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

# How to achieve invariance

---

Need both of the following:

1. Make sure your detector is invariant
  - Harris is invariant to translation and rotation
  - Scale is trickier
    - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
    - More sophisticated methods find “the best scale” to represent each feature (e.g., SIFT)
2. Design an invariant feature *descriptor*
  - A descriptor captures the information in a region around the detected feature point
  - The simplest descriptor: a square window of pixels
    - What's this invariant to?
  - Let's look at some better approaches...

# Rotation invariance for feature descriptors

---

Find dominant orientation of the image patch

- This is given by  $\mathbf{x}_+$ , the eigenvector of  $\mathbf{H}$  corresponding to  $\lambda_+$ 
  - $\lambda_+$  is the *larger* eigenvalue
- Rotate the patch according to this angle

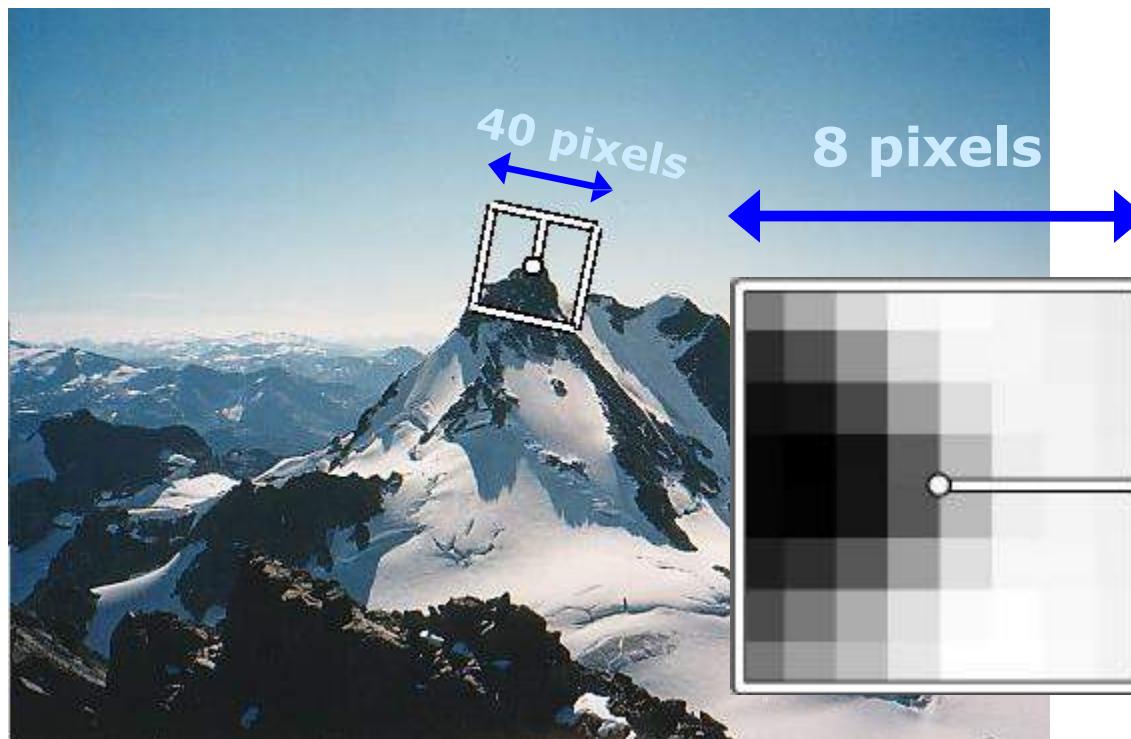


Figure by Matthew Brown

# Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

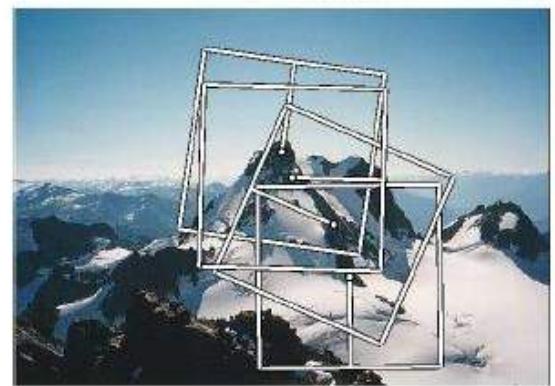
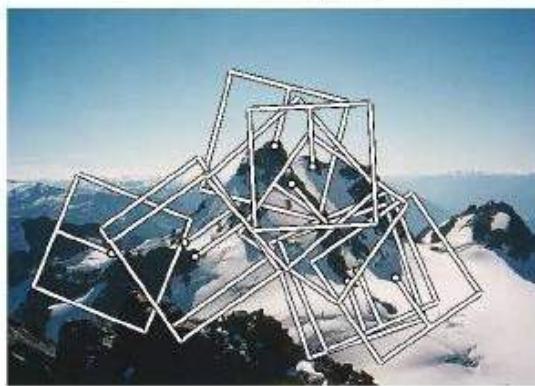
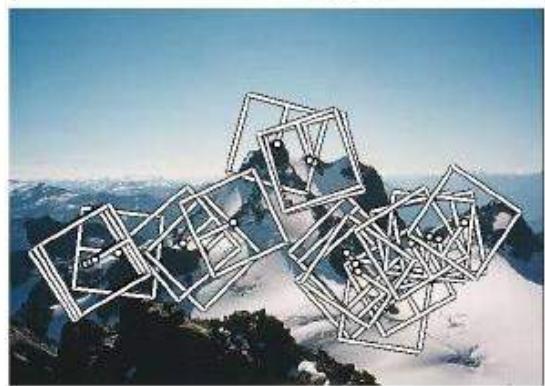
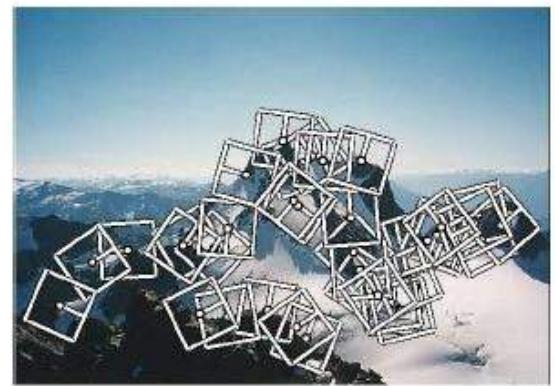
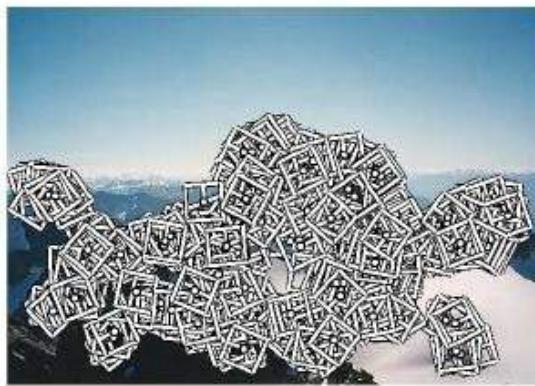
- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window



Adapted from slide by Matthew Brown

# Detections at multiple scales

---

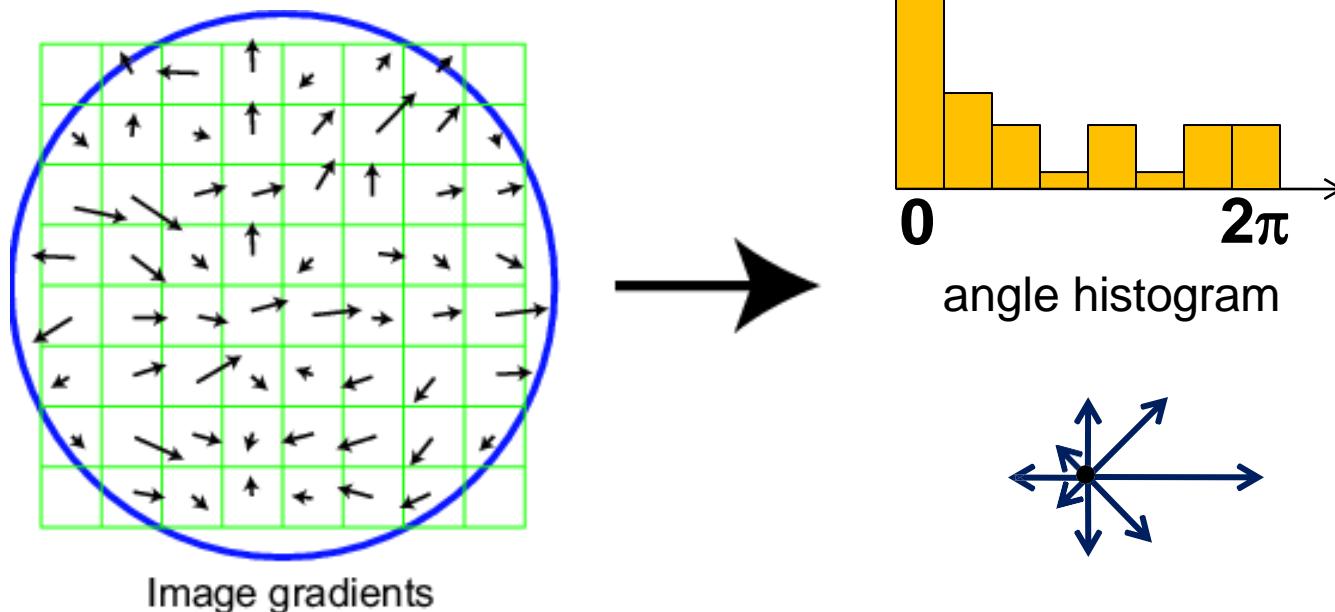


*Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.*

# Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations



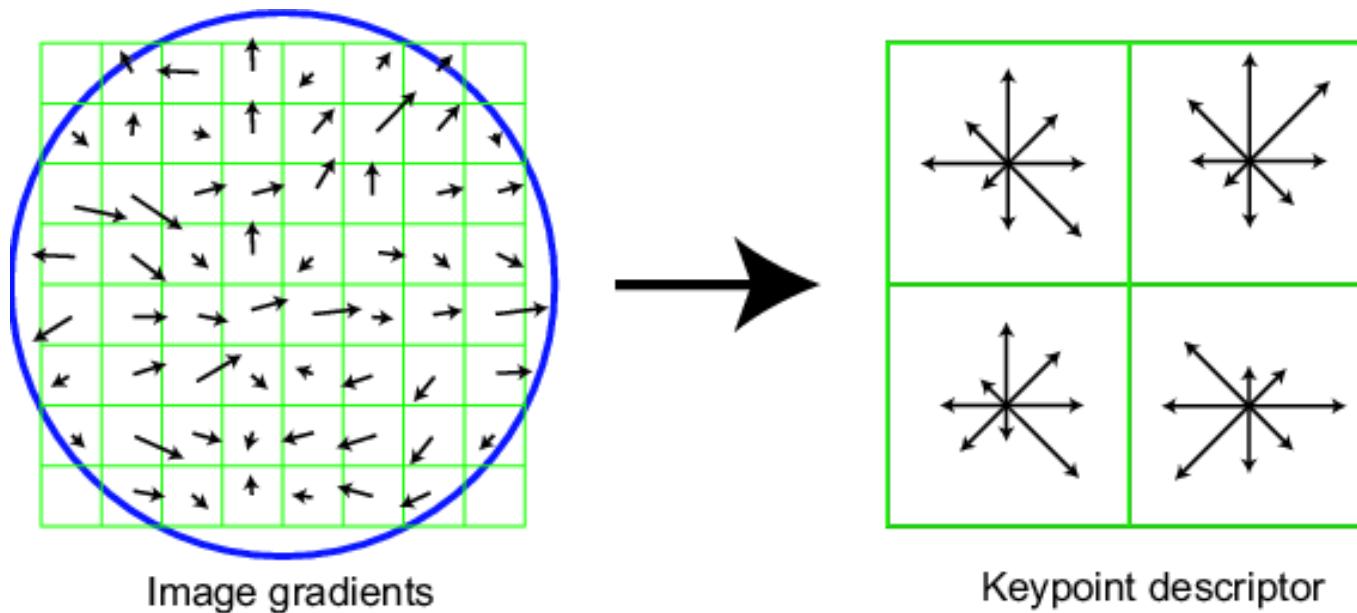
Adapted from slide by David Lowe

# SIFT descriptor

---

## Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells \* 8 orientations = 128 dimensional descriptor



Adapted from slide by David Lowe

# Properties of SIFT

---

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - [http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known\\_implementations\\_of\\_SIFT](http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT)



# Feature matching

---

Given a feature in  $I_1$ , how to find the best match in  $I_2$ ?

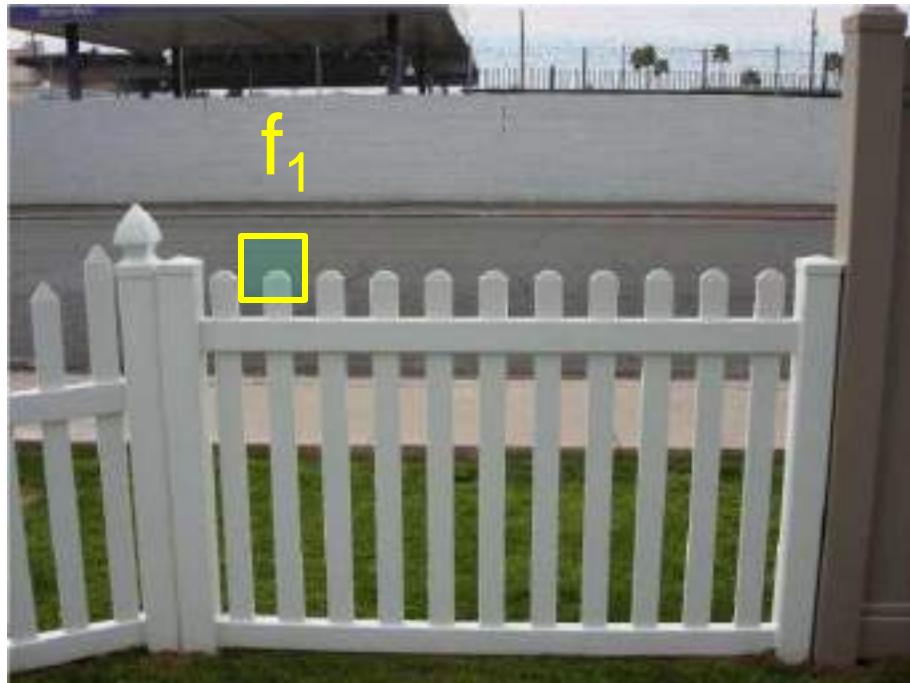
1. Define distance function that compares two descriptors
2. Test all the features in  $I_2$ , find the one with min distance

# Feature distance

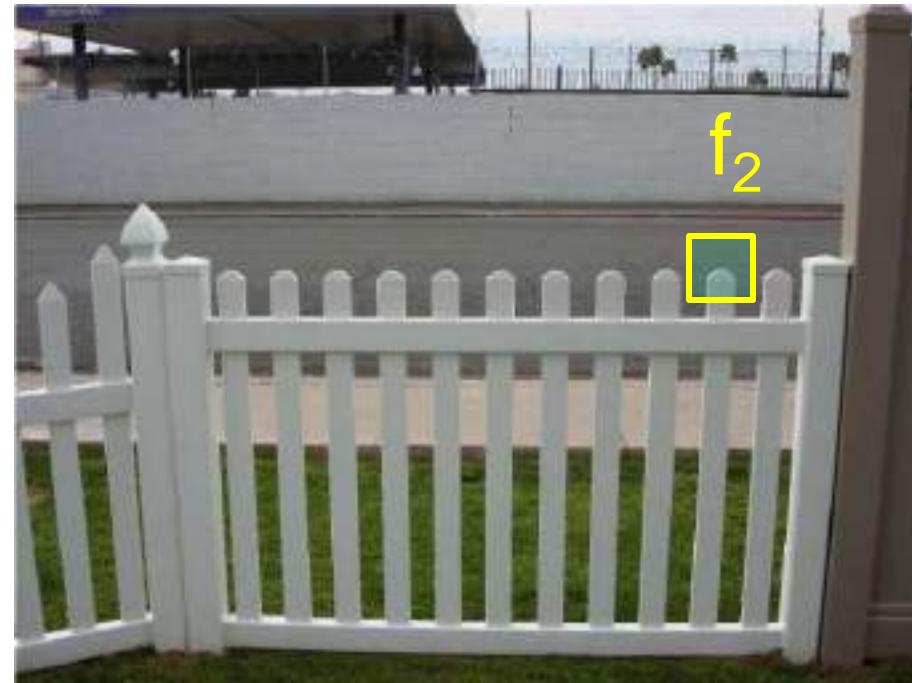
---

How to define the difference between two features  $f_1, f_2$ ?

- Simple approach is  $\text{SSD}(f_1, f_2)$ 
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches



$I_1$



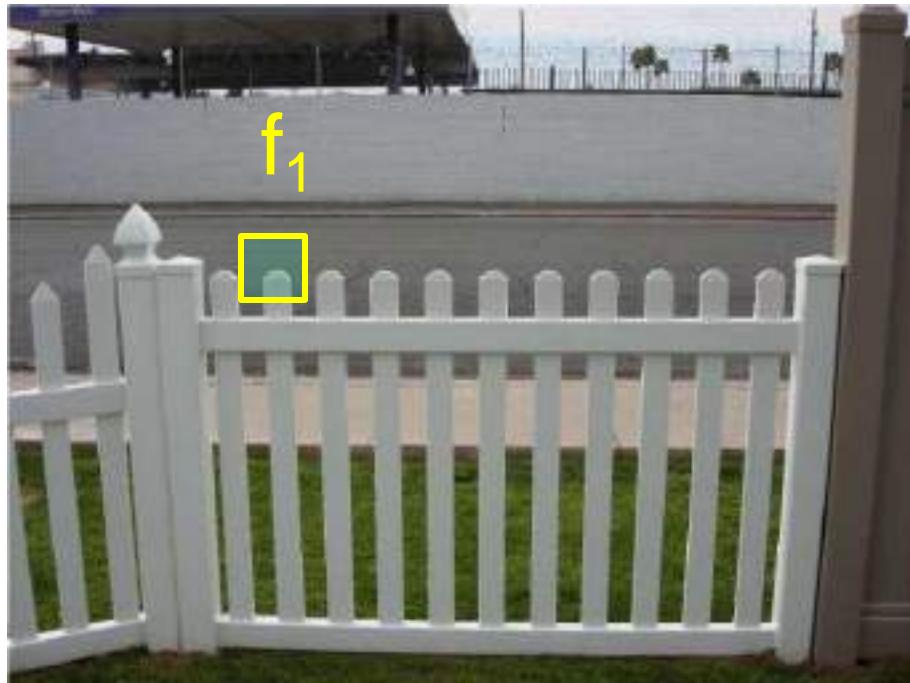
$I_2$

# Feature distance

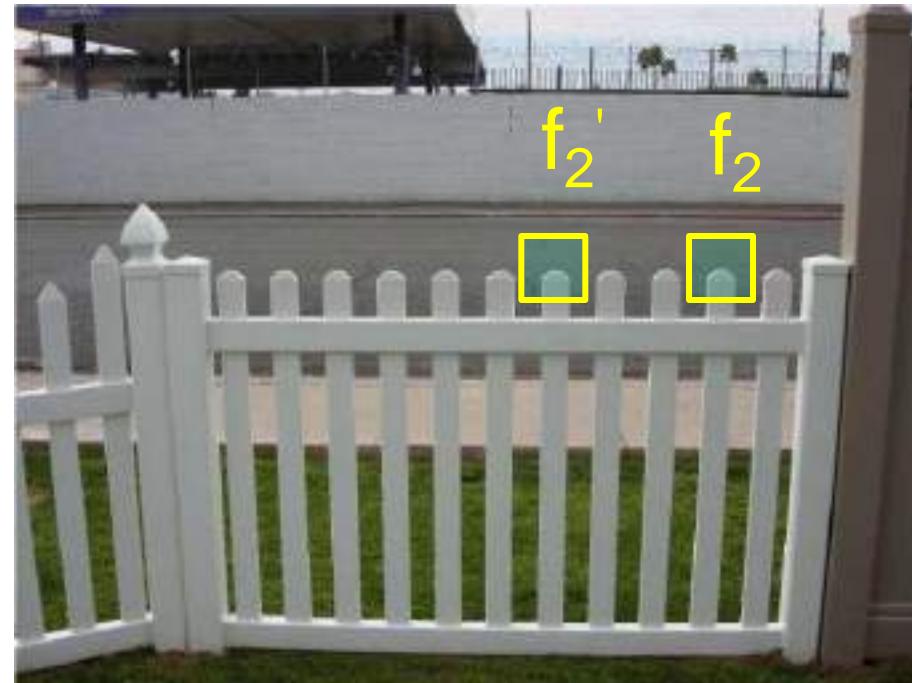
---

How to define the difference between two features  $f_1, f_2$ ?

- Better approach: ratio distance =  $\text{SSD}(f_1, f_2) / \text{SSD}(f_1, f_2')$ 
  - $f_2$  is best SSD match to  $f_1$  in  $I_2$
  - $f_2'$  is 2<sup>nd</sup> best SSD match to  $f_1$  in  $I_2$
  - gives small values for ambiguous matches



$I_1$

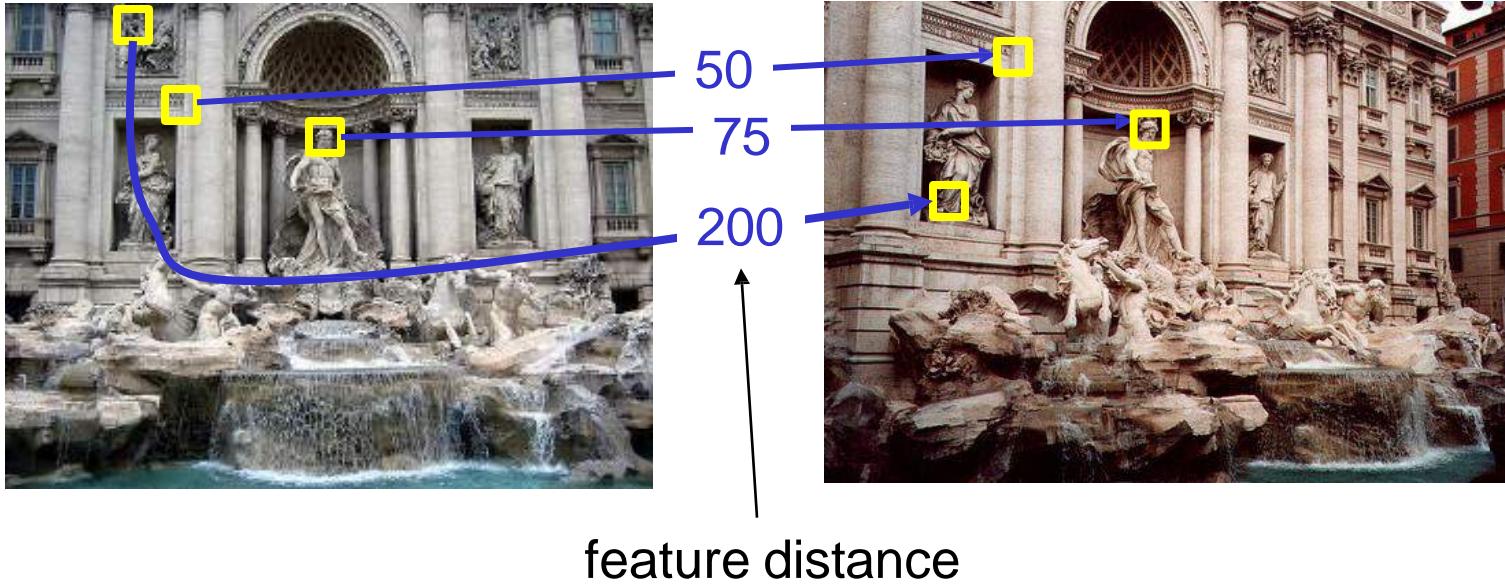


$I_2$

# Evaluating the results

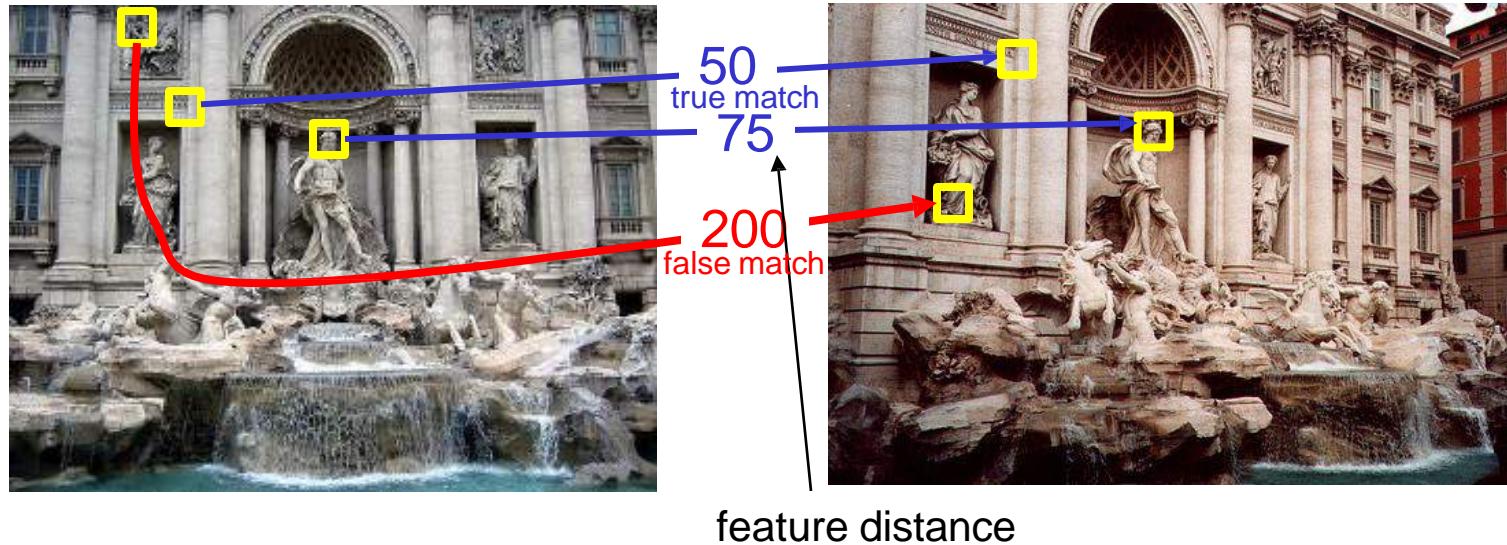
---

How can we measure the performance of a feature matcher?



# True/false positives

---



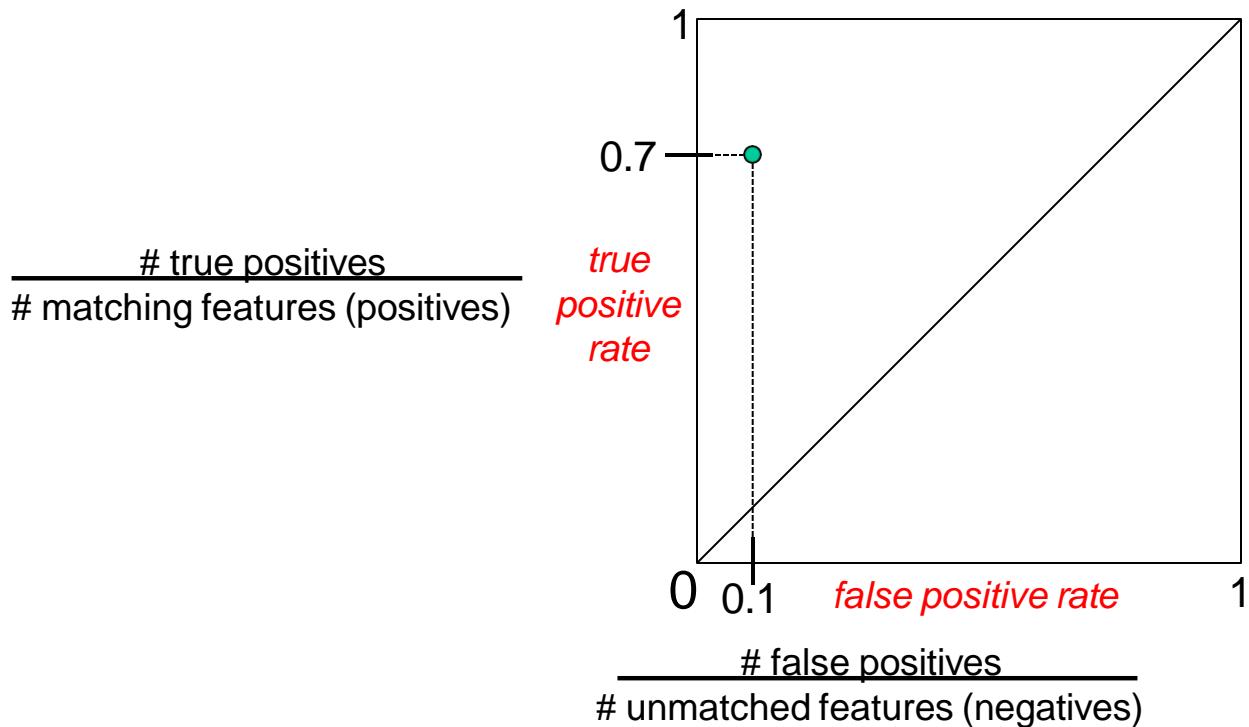
The distance threshold affects performance

- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

# Evaluating the results

---

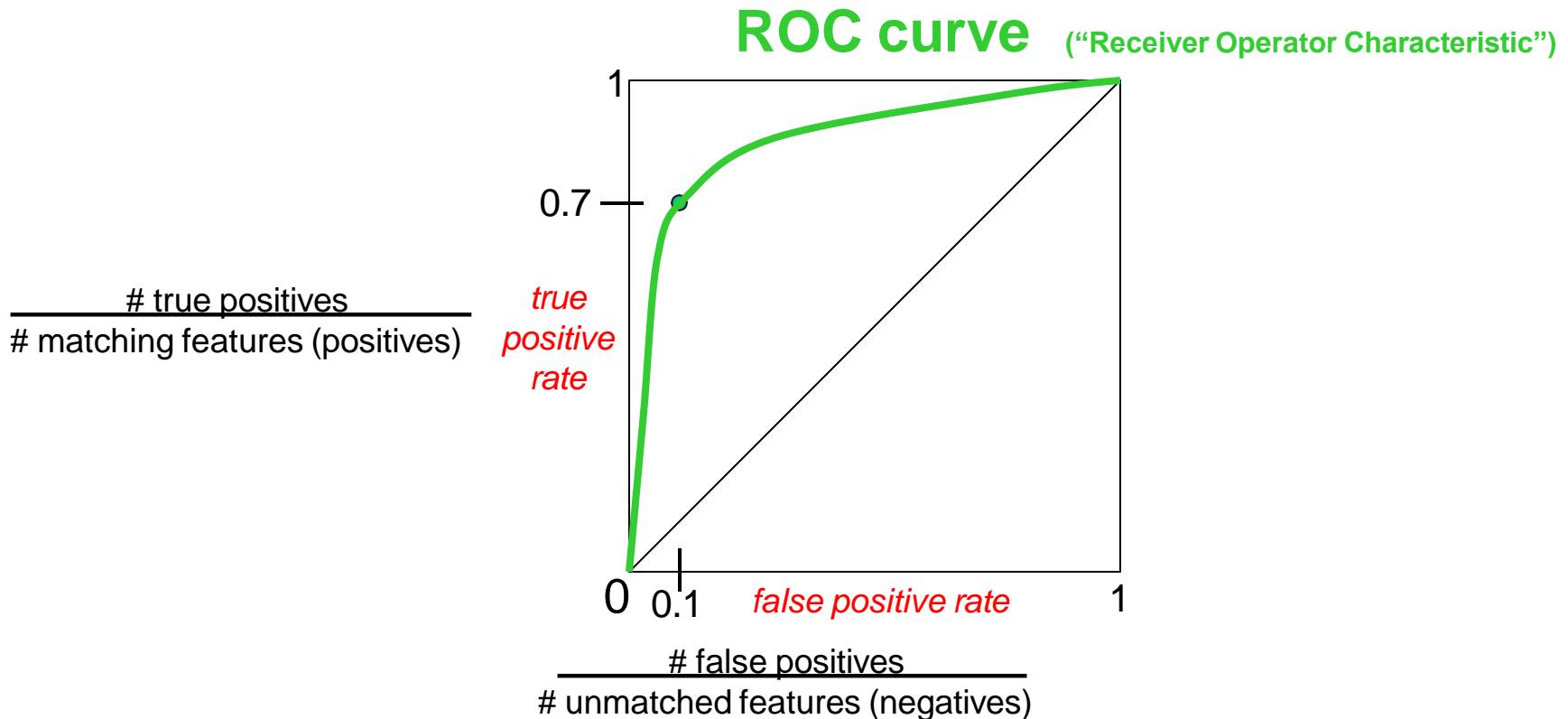
How can we measure the performance of a feature matcher?



# Evaluating the results

---

How can we measure the performance of a feature matcher?



## ROC Curves

- Generated by counting # current/incorrect matches, for different thresholds
- Want to maximize area under the curve (AUC)
- Useful for comparing different feature matching methods
- For more info: [http://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](http://en.wikipedia.org/wiki/Receiver_operating_characteristic)

# More on feature detection/description

---



## Publications

### Region detectors

- *Harris-Affine & Hessian Affine*: [K. Mikolajczyk](#) and [C. Schmid](#), Scale and Affine invariant interest point detectors. In IJCV 1(60):63-86, 2004. [PDF](#)
- *MSER*: [J. Matas](#), [O. Chum](#), [M. Urban](#), and [T. Pajdla](#), Robust wide baseline stereo from maximally stable extremal regions. In BMVC p. 384-393, 2002. [PDF](#)
- *IBR & EBR*: [T. Tuytelaars](#) and [L. Van Gool](#), Matching widely separated views based on affine invariant regions. In IJCV 1(59):61-85, 2004. [PDF](#)
- *Salient regions*: [T. Kadir](#), [A. Zisserman](#), and [M. Brady](#), An affine invariant salient region detector. In ECCV p. 404-416, 2004. [PDF](#)

### Region descriptors

- *SIFT*: [D. Lowe](#), Distinctive image features from scale invariant keypoints. In IJCV 2(60):91-110, 2004. [PDF](#)

### Performance evaluation

- [K. Mikolajczyk](#), [T. Tuytelaars](#), [C. Schmid](#), [A. Zisserman](#), [J. Matas](#), [F. Schaffalitzky](#), [T. Kadir](#) and [L. Van Gool](#), A comparison of affine region detectors. Technical Report, accepted to IJCV. [PDF](#)
- [K. Mikolajczyk](#), [C. Schmid](#), A performance evaluation of local descriptors. Technical Report, accepted to PAMI. [PDF](#)

# Lots of applications

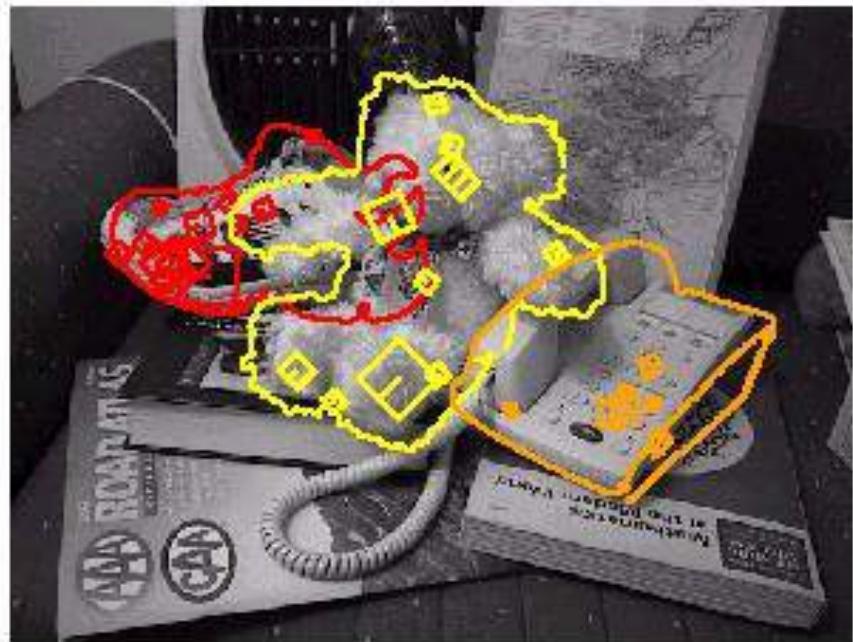
---

Features are used for:

- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- ... other

# Object recognition (David Lowe)

---



# Sony Aibo

## AIBO® Entertainment Robot

Official U.S. Resources and Online Destinations

### SIFT usage:

- Recognize charging station
- Communicate with visual cards
- Teach object recognition



# Computer Vision

---

## **UNIT II**

**HIGH Level: Instance Recognition, Object Detection, Image Classification, Semantic Segmentation, Video Understanding, Vision and Language.(Chap 6)**

## 6.1 Instance Recognition

---



**Figure 6.2** *Recognizing objects in a cluttered scene (Lowe 2004) © 2004 Springer. Two of the training images in the database are shown on the left. They are matched to the cluttered scene in the middle using SIFT features, shown as small squares in the right image. The affine warp of each recognized database image onto the scene is shown as a larger parallelogram in the right image.*

# 6.1 Instance Recognition

---

## 1. Definition

• General object recognition has two categories:

- **Instance Recognition:**
  - Recognizing a **specific known 2D or 3D rigid object**.
  - Challenges: new viewpoints, cluttered background, partial occlusions.
  - Example: Recognizing objects in cluttered scenes (Figure 6.2).
- **Class Recognition (Category-level recognition):**
  - Recognizing any instance of a **general class** (e.g., *cat*, *car*, *bicycle*).
  - Much more difficult.

# Early Approaches

---

**Geometric-based methods:** Extracting lines, contours, 3D surfaces → match to 3D object models.

- **Appearance-based methods:** Capturing many images under various viewpoints & illuminations → eigenspace decomposition (Murase & Nayar, 1995).

# Modern Approaches

---

- Use viewpoint-invariant 2D features (e.g., SIFT, affine regions).
- Pipeline:
  - Extract sparse, informative 2D features from image & database.
  - Match features using feature matching strategies.
  - Verify matches via geometric transformation alignment.
- Example: SIFT-based recognition and affine warping of objects onto cluttered scenes (Figure 6.2).

# Modern Approaches

---

- **Steps in recognition:**

- Extract **interest points** & store descriptors in a search structure (e.g., tree).
- Extract features from new image.
- Find  $\geq 3$  matches for an object.
- Perform **match verification** to check spatial arrangement consistency.

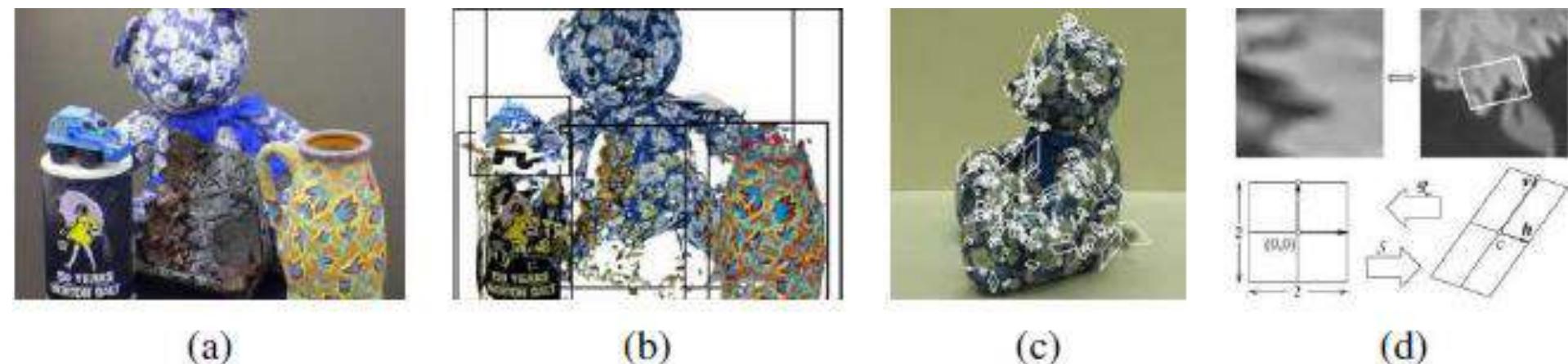
- **Challenges: Outliers due to clutter & shared features.**

- **Solutions:**

- **Lowe (2004):**
  - Uses **Hough transform** to accumulate votes for **affine transformations**.
  - Works for planar / quasi-planar objects.
  - Shown in Figure 6.2 (affine warp shown as parallelogram).
- **Rothganger, Lazebnik et al. (2006):**
  - Detect affine regions  $\rightarrow$  rectify patches.
  - Compute **SIFT + color histogram** descriptors.
  - Build **3D affine models** (factorization  $\rightarrow$  Euclidean reconstruction).
  - Use **local geometric constraints** for verification.
  - Example results: **5 recognized objects in cluttered scene** (Figure 6.3a–b).
  - Affine regions and rectified patches shown in Figure 6.3c–d.

# Modern Approaches

---



**Figure 6.3** 3D object recognition with affine regions (Rothganger, Lazebnik et al. 2006) © 2006 Springer: (a) sample input image; (b) five of the recognized (reprojected) objects along with their bounding boxes; (c) a few of the local affine regions; (d) local affine region (patch) reprojected into a canonical (square) frame, along with its geometric affine transformations.

# Modern Approaches

---

- **Pixel-level segmentation approaches:**

- Ferrari, Tuytelaars, Van Gool (2006): Recognition + segmentation.
- Kannala, Rahtu et al. (2008): Extended to **non-rigid deformations**.

- **Shift in research focus:**

- Early-mid 2000s → locating known 3D objects
- Later → **Instance retrieval (CBIR)** = searching in large image databases.
- Related areas:
  - **Visual similarity search**
  - **3D pose estimation**

# Key Techniques & Tools

- Feature descriptors: SIFT, affine regions.
- Verification methods: Hough transform, geometric constraints.
- 3D reconstruction: Factorization → Euclidean upgrade.
- Applications:
  - Recognition in cluttered scenes (Figures 6.2, 6.3).
  - Large-scale **image retrieval**.
  - **Recognition + segmentation**.

# Image Classification (6.2)

---

## 1. Introduction

- **Goal:** Assign a label to an image based on the main object or scene.
- **Difference from Instance Recognition**
  - Instance recognition → recognizes a *specific known object*.
  - Classification → recognizes *any member of a class* (e.g., “cat,” “car”).
- **Challenges:** High variation in pose, scale, lighting, background, occlusion.
- **Applications:** Large-scale datasets and benchmarks have fueled progress.



(a)



**Figure 6.4** Challenges in image recognition: (a) sample images from the Xerox 10 class dataset (Csurka, Dance et al. 2006) © 2007 Springer; (b) axes of difficulty and variation from the ImageNet dataset (Russakovsky, Deng et al. 2015) © 2015 Springer.

# Feature-based Methods

---

- Early classification systems used handcrafted features.
- Bag-of-Features (BoF) pipeline:
  - Extract local descriptors (e.g., SIFT, HOG).
  - Quantize into **visual words** (clustering).
  - Represent image as a **histogram of word counts**.
  - Classify using SVM or other classifiers.
- Spatial Pyramid Matching (Lazebnik et al., 2006): Adds spatial layout info → improves accuracy (Figure 6.5).
- Attribute-based representations: Images described by semantic attributes like “striped,” “furry” (Figure 6.6).
- Large-scale benchmarks:
  - Caltech-101/256, Pascal VOC (Figure 6.7).
  - SUN, ImageNet → drove progress.

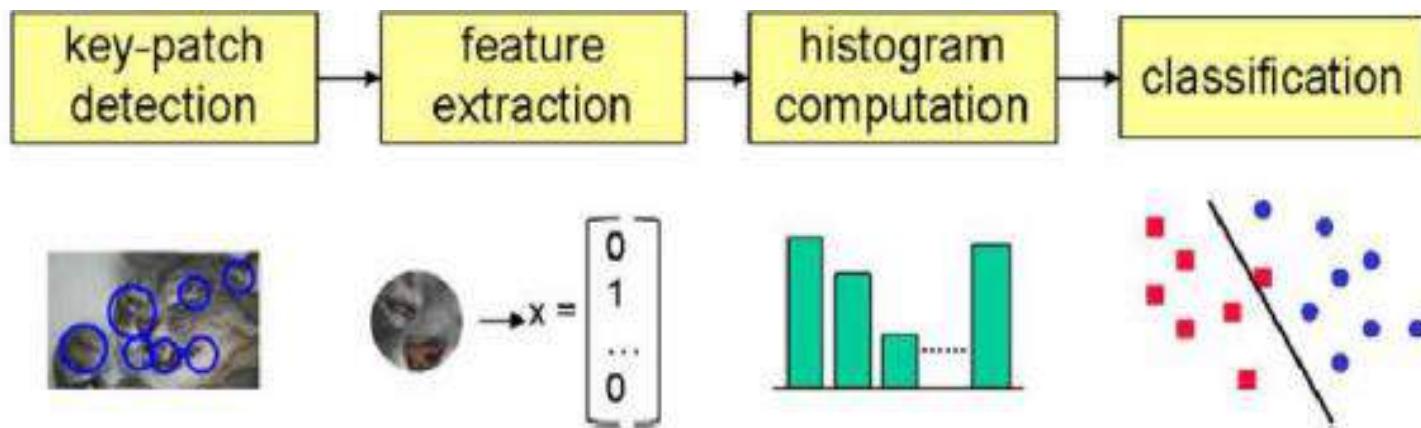


(a)

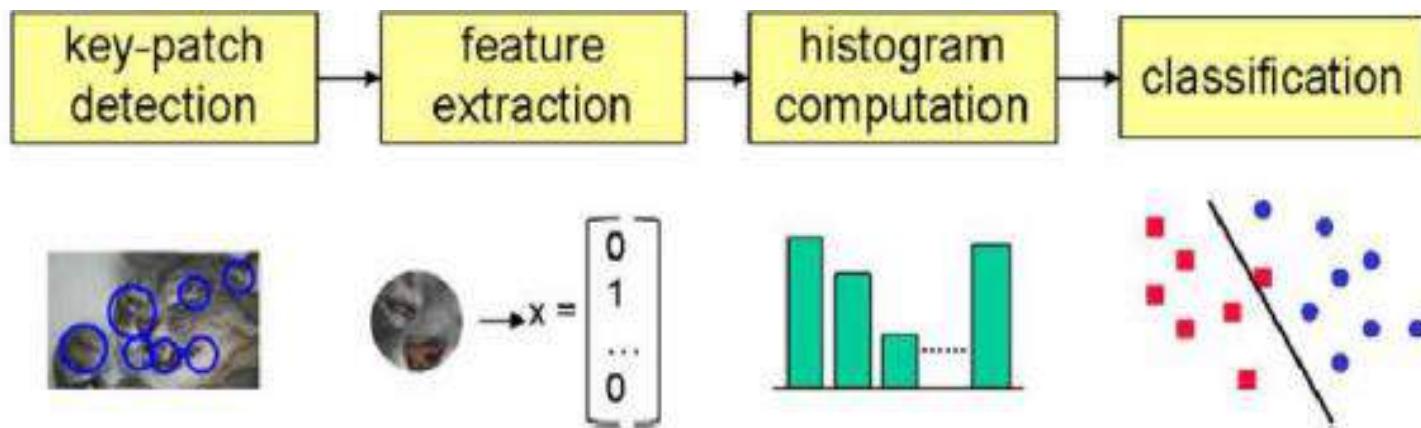


(b)

**Figure 6.5** Sample images from two widely used image classification datasets: (a) PASCAL Visual Object Categories (VOC) (Everingham, Eslami et al. 2015) © 2015 Springer; (b) ImageNet (Russakovsky, Deng et al. 2015) © 2015 Springer.



**Figure 6.6** A typical processing pipeline for a bag-of-words category recognition system (Csurka, Dance et al. 2006) © 2007 Springer. Features are first extracted at keypoints and then quantized to get a distribution (histogram) over the learned visual words (feature cluster centers). The feature distribution histogram is used to learn a decision surface using a classification algorithm, such as a support vector machine.



**Figure 6.6** A typical processing pipeline for a bag-of-words category recognition system (Csurka, Dance et al. 2006) © 2007 Springer. Features are first extracted at keypoints and then quantized to get a distribution (histogram) over the learned visual words (feature cluster centers). The feature distribution histogram is used to learn a decision surface using a classification algorithm, such as a support vector machine.

# Deep Networks

---

- Revolution: Deep Convolutional Neural Networks (CNNs).
- AlexNet (2012): Won ImageNet competition, massive jump in accuracy (Figure 6.11).
- CNN pipeline:
  - Convolution + pooling layers extract hierarchical features.
  - Fully connected layers → classification.
- Improvements:
  - Deeper networks: VGG , ResNet
  - Efficient training with GPUs + large datasets.
  - Transfer learning: pretrained models used for many tasks.
- Visualization: Filters and feature maps show how CNNs learn shapes & patterns (Figure 6.14).
- Modern networks:
  - GoogLeNet (Inception)
  - ResNet with skip connections
  - DenseNet, EfficientNet
- Performance trend: Accuracy steadily improved across ImageNet challenges (Figure 6.18).



Two-spotted ladybug  
*Adalia bipunctata*



Seven-spotted ladybug  
*Coccinella septempunctata*

(a)

<u>otter</u>	
black:	yes
white:	no
brown:	yes
stripes:	no
water:	yes
eats fish:	yes



<u>polar bear</u>	
black:	no
white:	yes
brown:	no
stripes:	no
water:	yes
eats fish:	yes



<u>zebra</u>	
black:	yes
white:	yes
brown:	no
stripes:	yes
water:	no
eats fish:	no

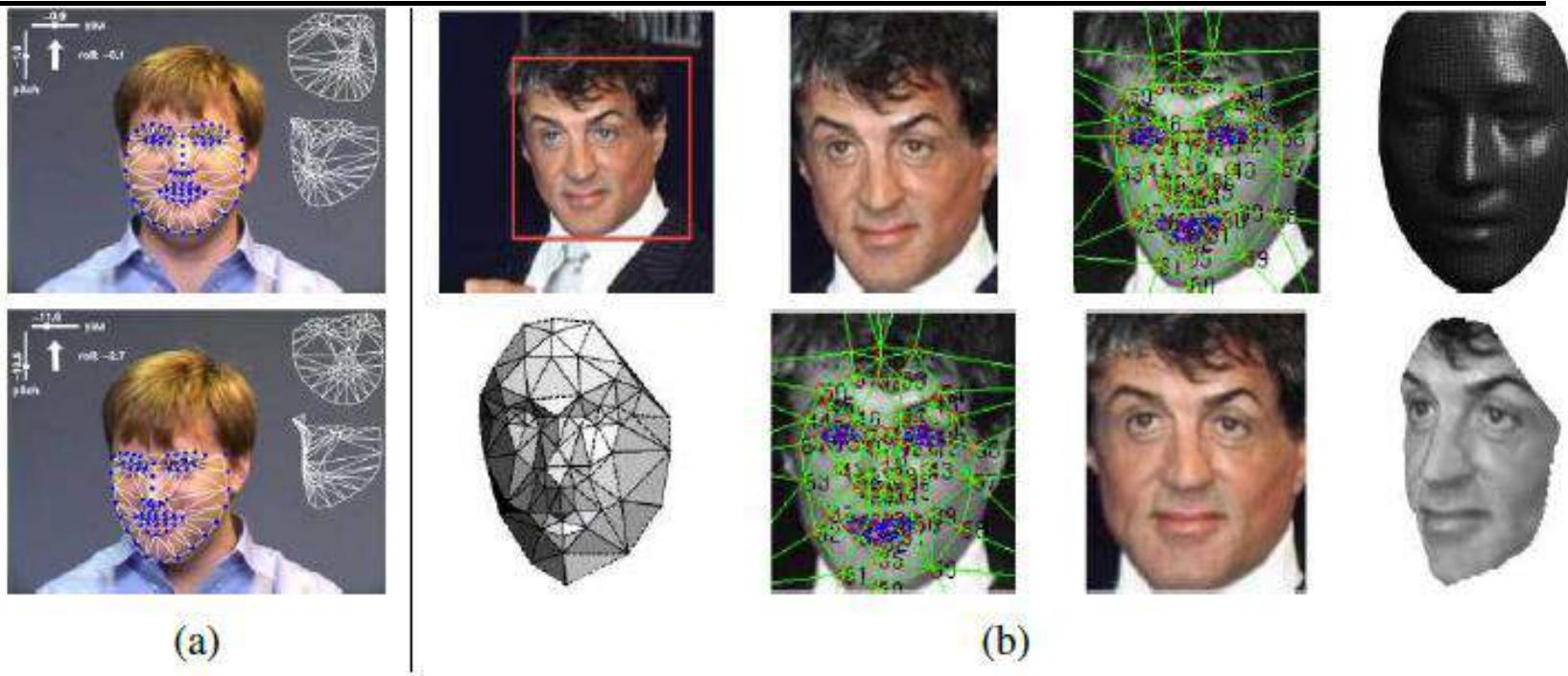


(b)

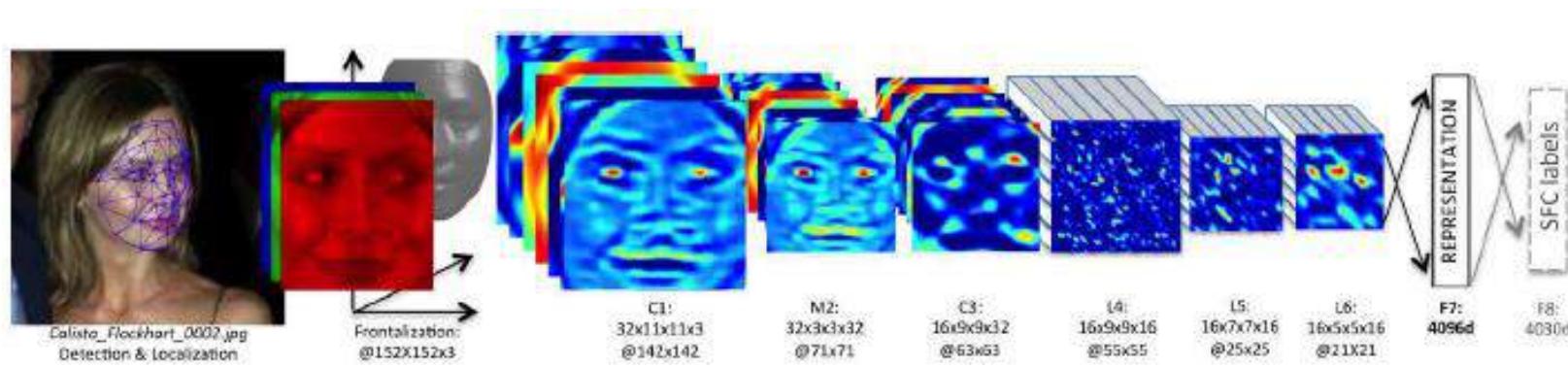
**Figure 6.10** Fine-grained category recognition. (a) The iNaturalist website and app allows citizen scientists to collect and classify images on their phones (Van Horn, Mac Aodha et al. 2018) © 2018 IEEE. (b) Attributes can be used for fine-grained categorization and zero-shot learning (Lampert, Nickisch, and Harmeling 2014) © 2014 Springer. These images are part of the Animals with Attributes dataset.



**Figure 6.13** Humans can recognize low-resolution faces of familiar people (Sinha, Balas et al. 2006) © 2006 IEEE.



**Figure 6.16** Head tracking and frontalization: (a) using 3D active appearance models (AAMs) (Matthews, Xiao, and Baker 2007) © 2007 Springer, showing video frames along with the estimated yaw, pitch, and roll parameters and the fitted 3D deformable mesh; (b) using six and then 67 fiducial points in the DeepFace system (Taigman, Yang et al. 2014) © 2014 IEEE, used to frontalize the face image (bottom row).



**Figure 6.17** *The DeepFace architecture (Taigman, Yang et al. 2014) © 2014 IEEE, starts with a frontalization stage, followed by several locally connected (non-convolutional) layers, and then two fully connected layers with a K-class softmax.*

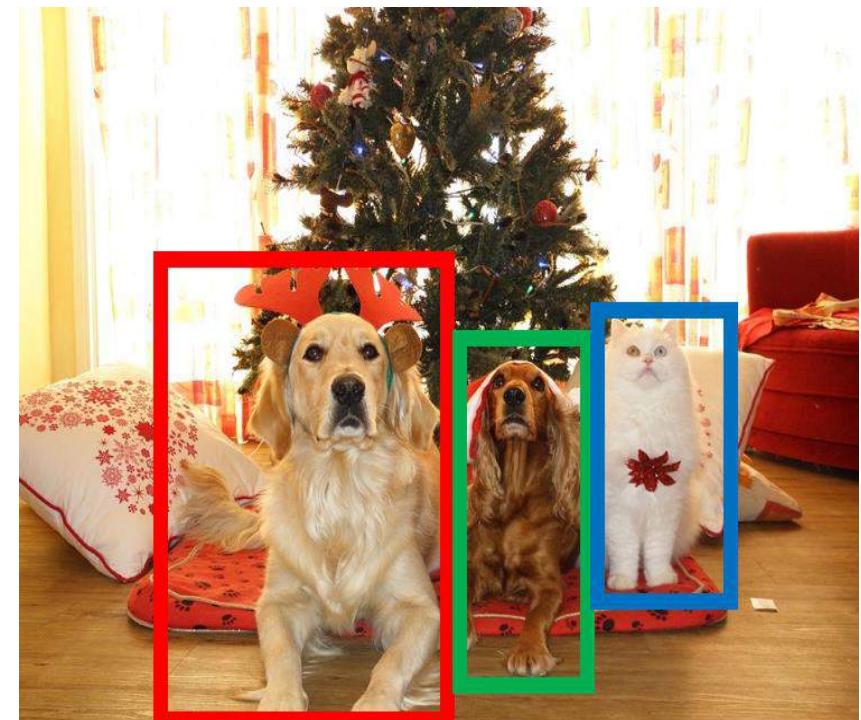
# Object Detection: Task Definition

---

**Input:** Single RGB Image

**Output:** A set of detected objects;  
For each object predict:

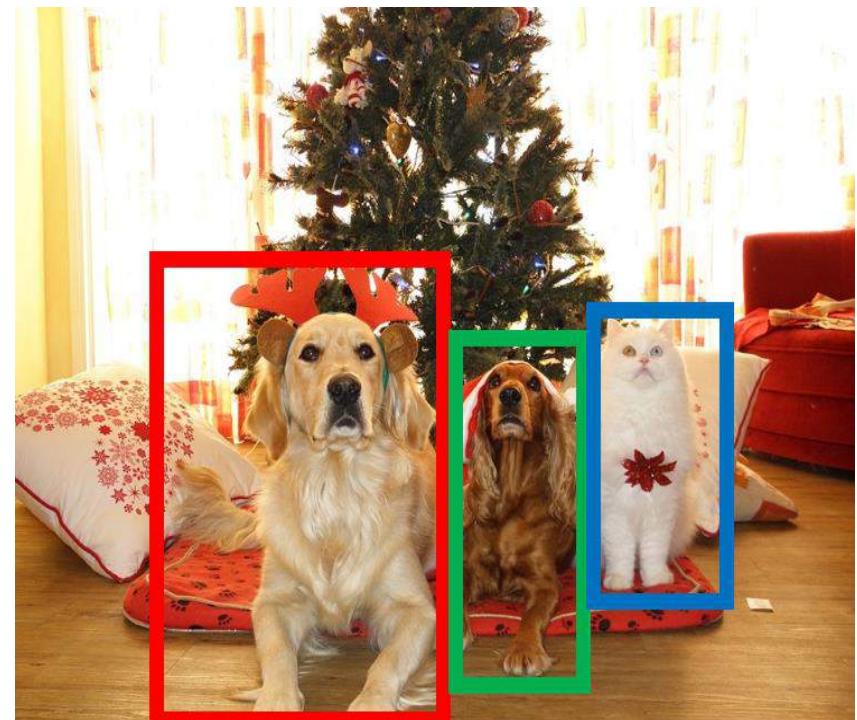
1. Category label (from fixed, known set of categories)
2. Bounding box (four numbers: x, y, width, height)



# Object Detection: Challenges

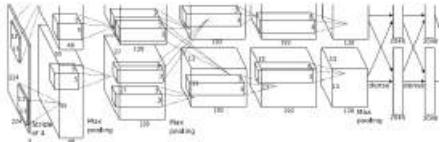
---

- **Multiple outputs:** Need to output variable numbers of objects per image
- **Multiple types of output:** Need to predict "what" (category label) as well as "where" (bounding box)
- **Large images:** Classification works at 224x224; need higher resolution for detection, often ~800x600



---

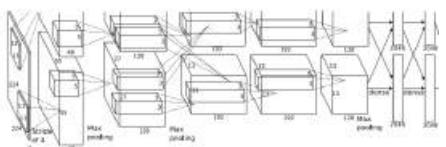
# Detecting a single object



This image is CC0 public domain

**Vector:**  
4096

# Detecting a single object “What”



This image is CC0 public domain

**Vector:**  
4096

Fully  
Connected:  
4096 to 1000

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

**Correct label:**  
Cat

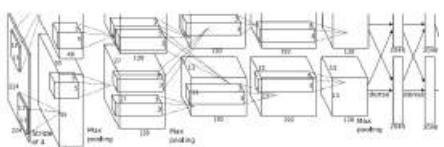
**Softmax**  
**Loss**

# Detecting a single object “What”



This image is CC0 public domain

Treat localization as a  
regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
( $x$ ,  $y$ ,  $w$ ,  $h$ )

**Correct label:**  
Cat

**Softmax  
Loss**

**L2 Loss**

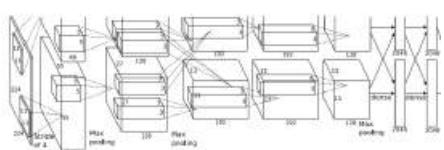
**Correct box:**  
( $x'$ ,  $y'$ ,  $w'$ ,  $h'$ )

# Detecting a single object “What”



This image is CC0 public domain

Treat localization as a regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

Fully  
Connected:  
4096 to 4

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

**Box  
Coordinates**  
( $x$ ,  $y$ ,  $w$ ,  $h$ )

**Correct label:**  
Cat

**Softmax  
Loss**

**Weighted  
Sum** → **Loss**

**L2 Loss**

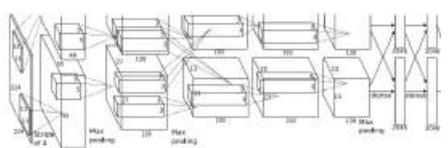
**Correct box:**  
( $x'$ ,  $y'$ ,  $w'$ ,  $h'$ )

# Detecting a single object “What”



This image is CC0 public domain

Treat localization as a  
regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

Fully  
Connected:  
4096 to 4

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

**Box  
Coordinates**  
( $x$ ,  $y$ ,  $w$ ,  $h$ )

**Correct label:**  
Cat

**Softmax**

**Loss**

**Multitask  
Loss**

**Weighted  
Sum**

**Loss**

**L2 Loss**

**Correct box:**  
( $x'$ ,  $y'$ ,  $w'$ ,  $h'$ )

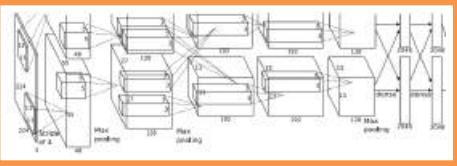
# Detecting a single object “What”

Often pretrained  
on ImageNet  
(Transfer learning)



This image is CC0 public domain

Treat localization as a  
regression problem!



Vector:  
4096

“Where”

Fully  
Connected:  
4096 to 4

Box  
Coordinates  
 $(x, y, w, h)$

Fully  
Connected:  
4096 to 1000

Class Scores  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

Correct label:  
Cat

Softmax

Loss

Multitask  
Loss

Weighted  
Sum

Loss

L2 Loss

Correct box:  
 $(x', y', w', h')$

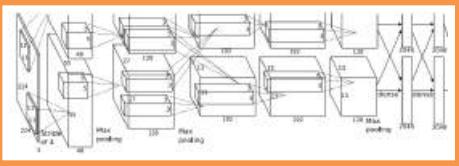
# Detecting a single object “What”

Often pretrained  
on ImageNet  
(Transfer learning)



This image is CC0 public domain

Treat localization as a  
regression problem!



Vector:  
4096

“Where”

Fully  
Connected:  
4096 to 1000

Fully  
Connected:  
4096 to 4

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...

**Box  
Coordinates**  
( $x$ ,  $y$ ,  $w$ ,  $h$ )

**Correct label:**  
Cat

**Softmax**

**Loss**

Multitask  
Loss

**Weighted  
Sum**

**Loss**

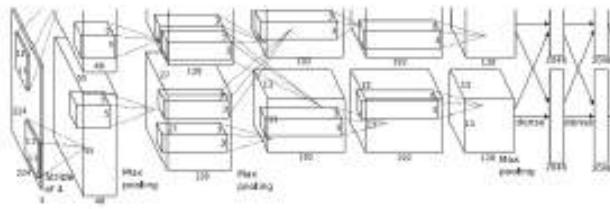
**L2 Loss**

**Correct box:**  
( $x'$ ,  $y'$ ,  $w'$ ,  $h'$ )

**Problem:** Images can have  
more than one object!

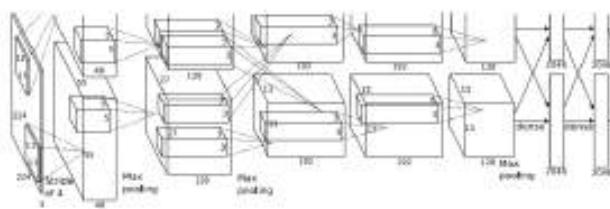
# Detecting Multiple Objects

---

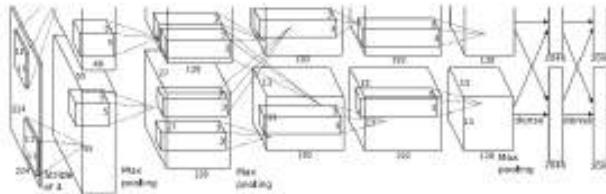


Need different numbers  
of outputs per image

CAT: (x, y, w, h)      4 numbers



DOG: (x, y, w, h)  
DOG: (x, y, w, h)      16 numbers  
CAT: (x, y, w, h)

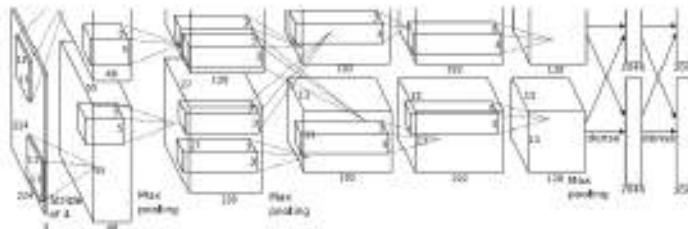


DUCK: (x, y, w, h)  
DUCK: (x, y, w, h)      Many  
numbers!  
....

# Detecting Multiple Objects: Sliding Window

---

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

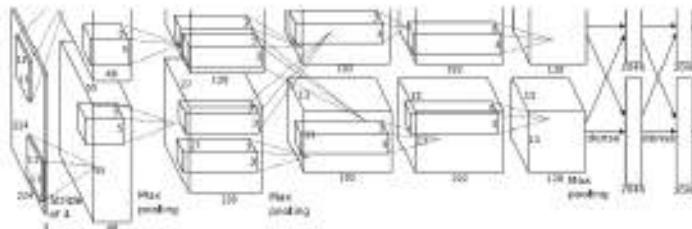


Dog? NO  
Cat? NO  
Background? YES

# Detecting Multiple Objects: Sliding Window

---

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

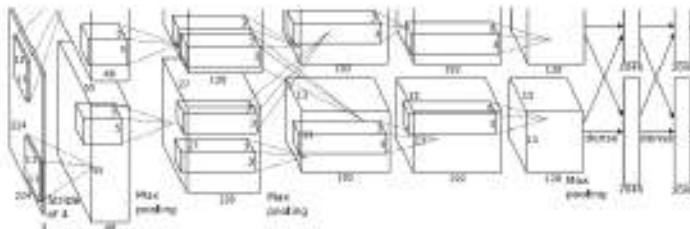


Dog? YES  
Cat? NO  
Background? NO

# Detecting Multiple Objects: Sliding Window

---

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



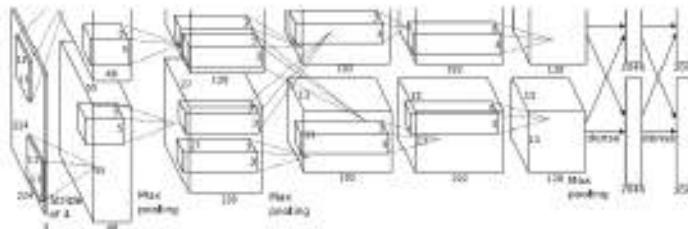
Dog? YES  
Cat? NO  
Background? NO

# Detecting Multiple Objects: Sliding Window

---



Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO  
Cat? YES  
Background? NO

# Detecting Multiple Objects: Sliding Window

---

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

**Question:** How many possible boxes are there in an image of size  $H \times W$ ?



# Detecting Multiple Objects: Sliding Window

---



Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

**Question:** How many possible boxes are there in an image of size  $H \times W$ ?

Consider a box of size  $h \times w$ :

Possible x positions:  $W - w + 1$

Possible y positions:  $H - h + 1$

Possible positions:

$$(W - w + 1) * (H - h + 1)$$

800 x 600 image  
has ~58M boxes!  
No way we can  
evaluate them all

Total possible boxes:

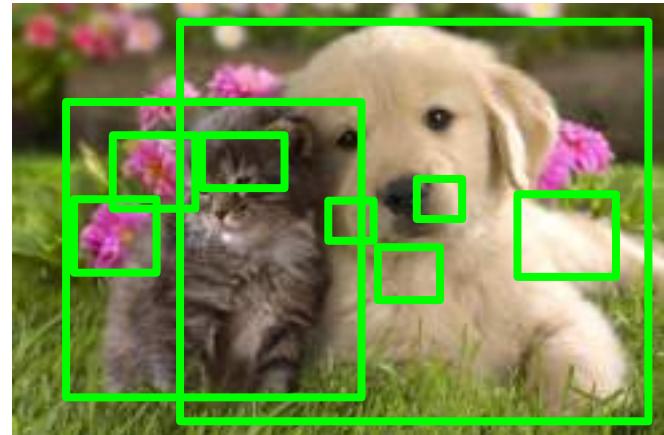
$$\sum_{h=1}^H \sum_{w=1}^W (W - w + 1)(H - h + 1)$$

$$= \frac{H(H + 1)}{2} \frac{W(W + 1)}{2}$$

# Region Proposals

---

- Find a small set of boxes that are likely to cover all objects
- Often based on heuristics: e.g. look for “blob-like” image regions
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012

Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013

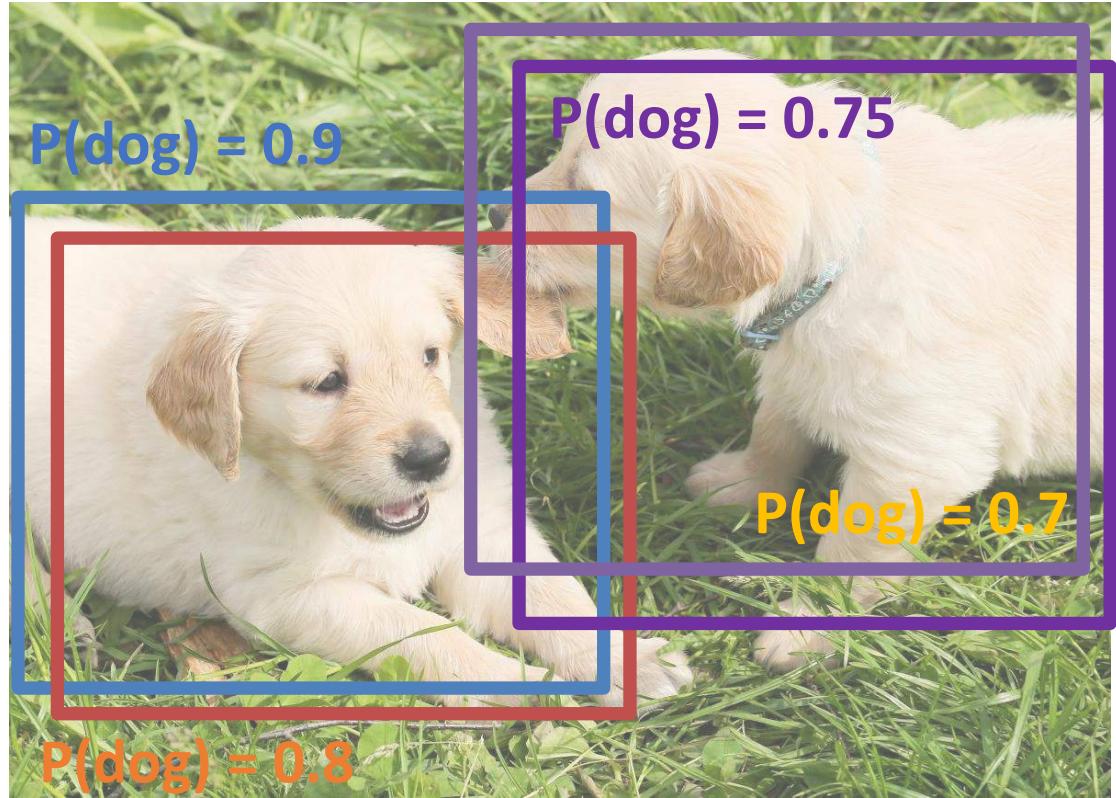
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014

Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

# Overlapping Boxes

---

**Problem:** Object detectors often output many overlapping detections:



[Puppy image is CC0 Public Domain](#)

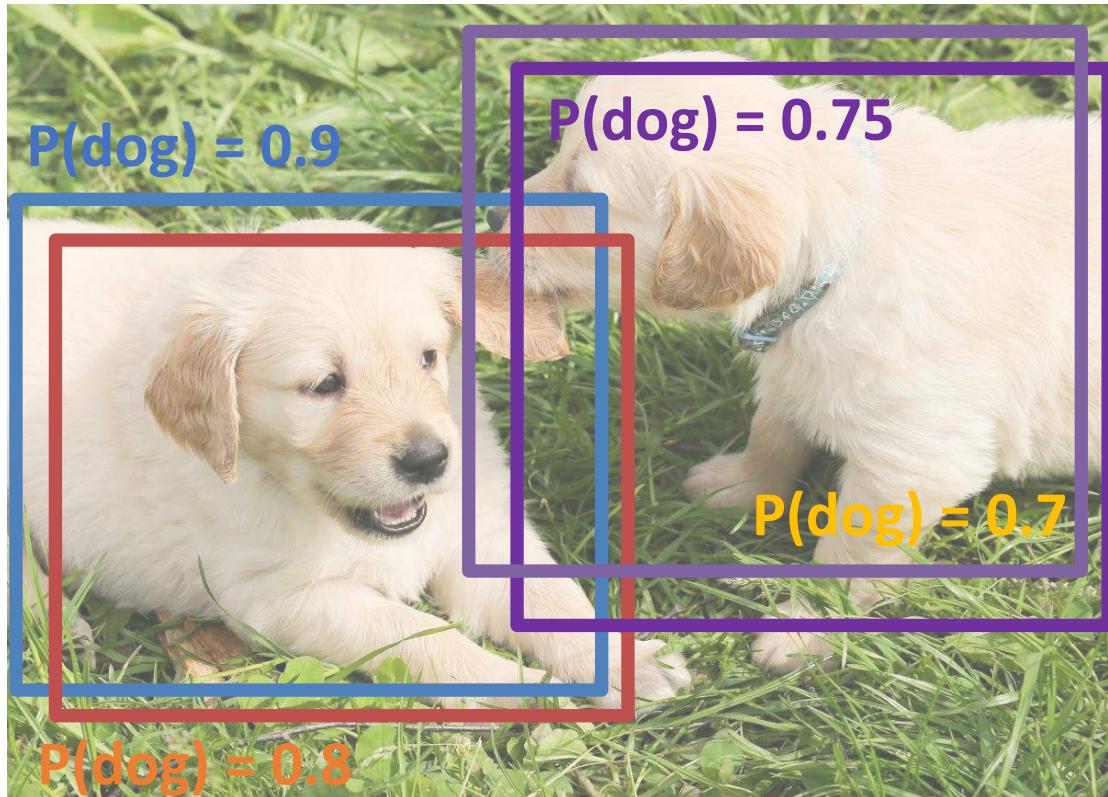
# Overlapping Boxes: Non-Max Suppression ~~(NMS)~~

---

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1



Puppy image is CCO Public Domain

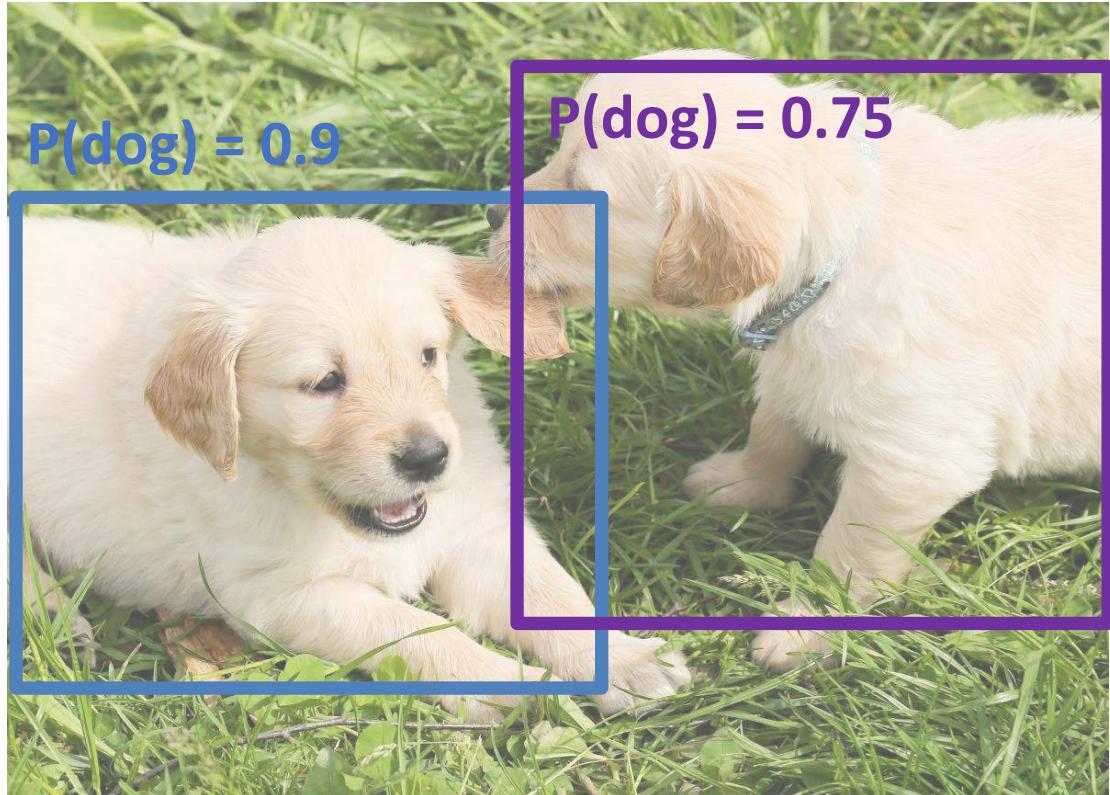
# Overlapping Boxes: Non-Max Suppression ~~(NMS)~~

---

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1



Puppy image is CCO Public Domain

# Overlapping Boxes: Non-Max Suppression ~~(NMS)~~

---

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1

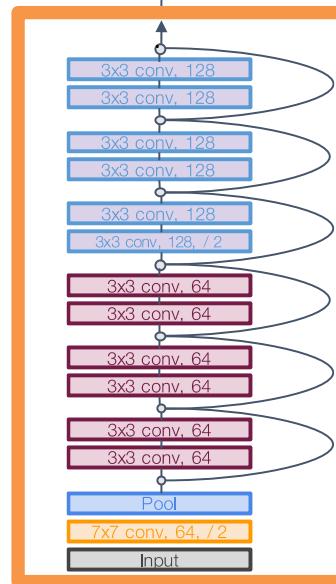
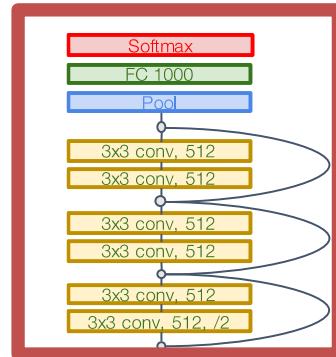
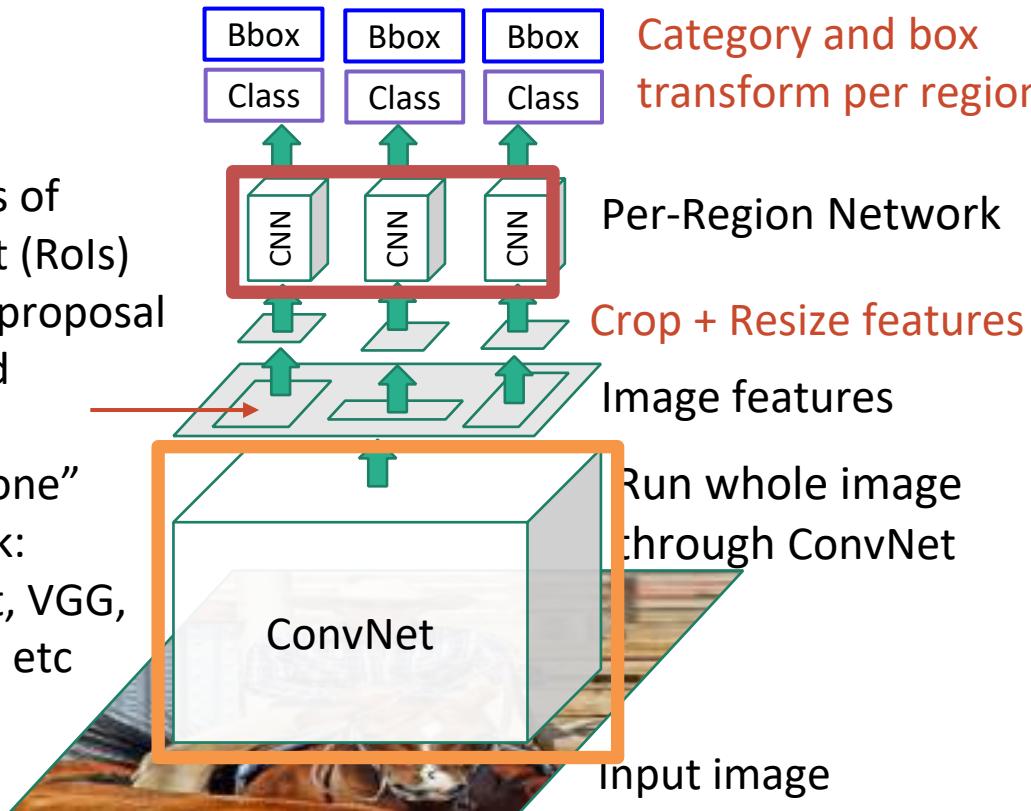
**Problem:** NMS may eliminate "good" boxes when objects are highly overlapping... no good solution =(



Crowd image is free for commercial use under the [Pixabay license](#)

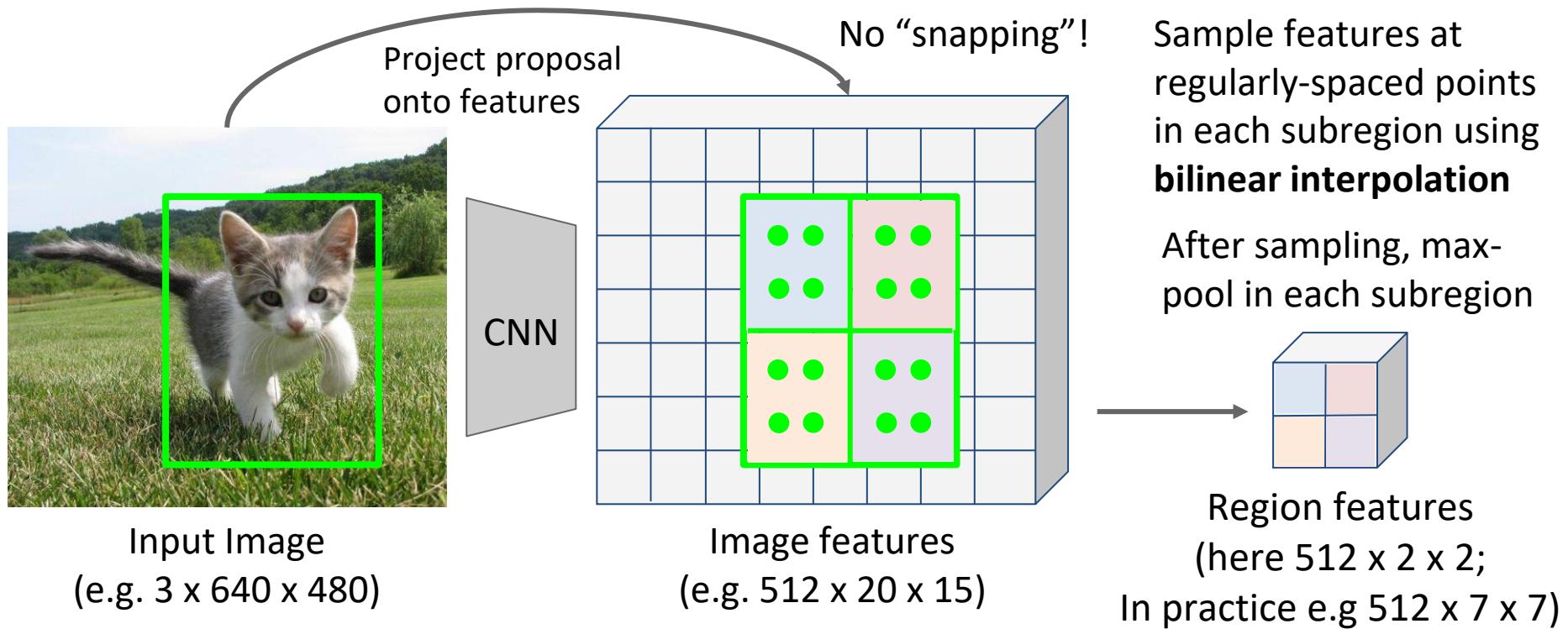
# Fast R-CNN

Regions of Interest (Rois) from a proposal method  
“Backbone” network: AlexNet, VGG, ResNet, etc



Example:  
For ResNet, last stage is used as per-region network; the rest of the network is used as backbone

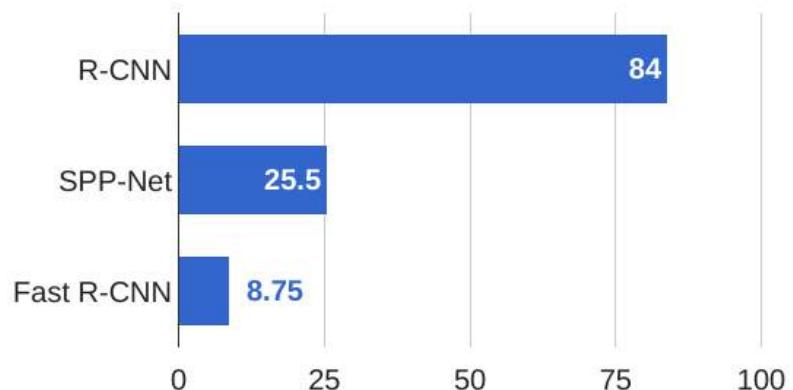
# Cropping Features: RoI Align



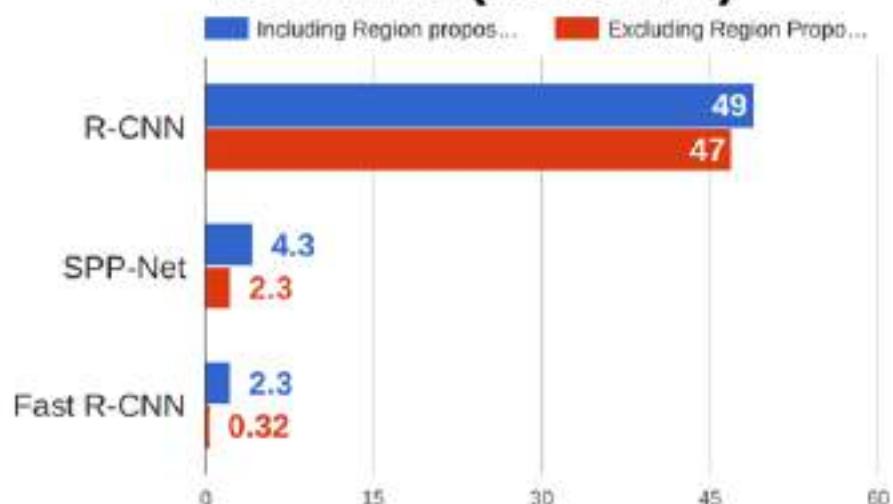
# Fast R-CNN vs “Slow” R-CNN

---

Training time (Hours)



Test time (seconds)

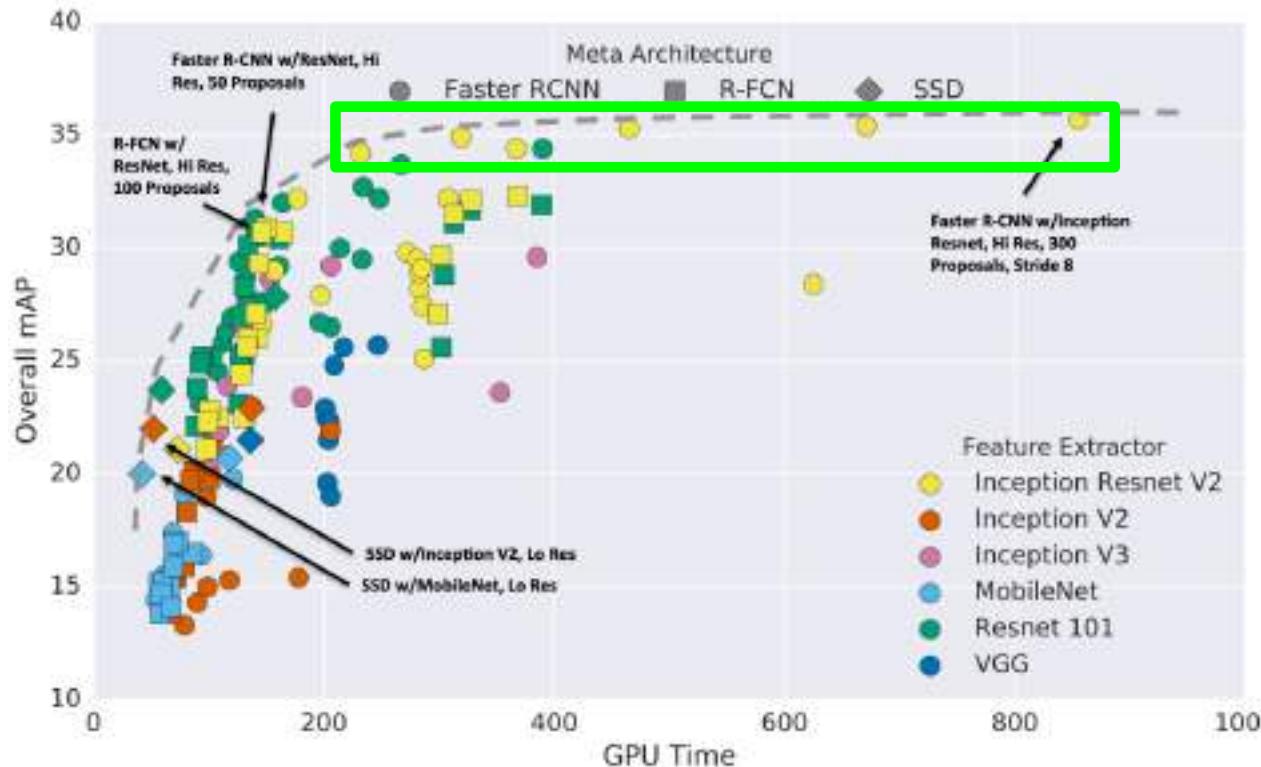


Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014

Girshick, “Fast R-CNN”, ICCV 2015

# Object Detection: Lots of variables!



## Takeaways:

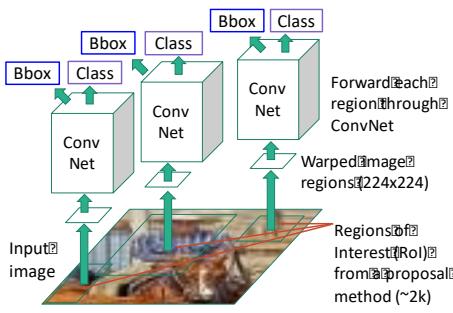
- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well
- Bigger backbones improve performance, but are slower
- Diminishing returns for slower methods

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

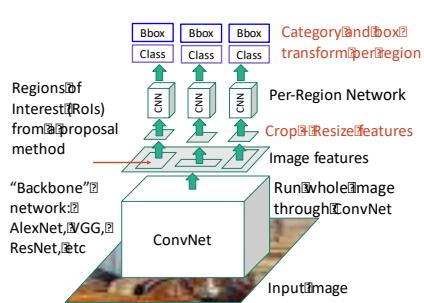
# Summary

---

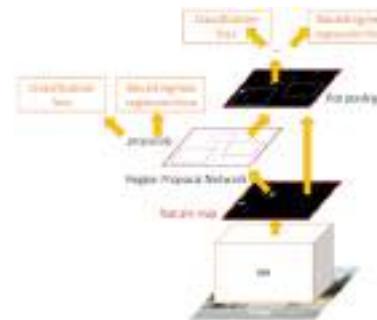
**“Slow” R-CNN:** Run CNN independently for each region



**Fast R-CNN:** Apply differentiable cropping to shared image features



**Faster R-CNN:** Compute proposals with CNN



**Single-Stage:** Fully convolutional detector



# Segmentation and detection

---

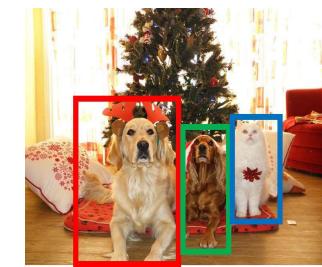
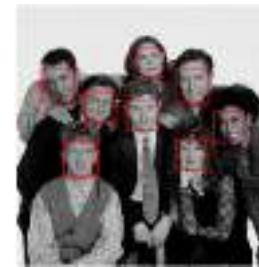
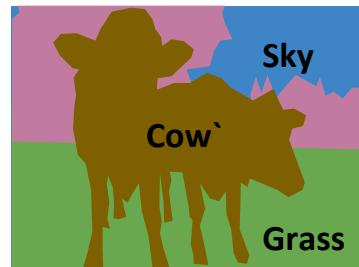
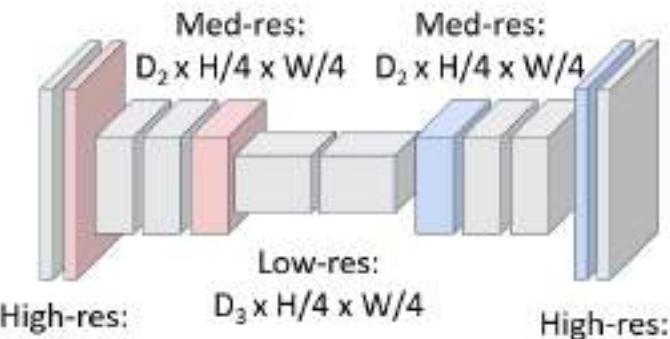
Adversarial examples

Bottlenecks and U-Nets

Segmentation

Face detection

Object detection



# So far: Image Classification

---



[This image](#) is CC0 public domain

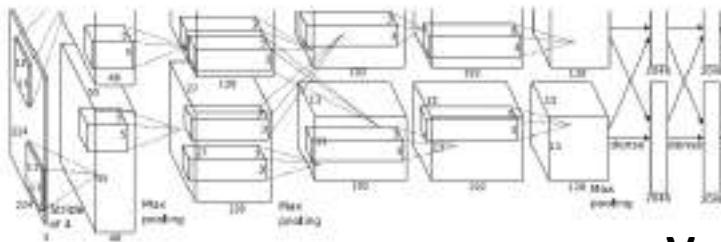


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**  
4096

**Class Scores**  
Cat: 0.9  
Dog: 0.05  
Car: 0.01  
...  
**Fully-Connected:**  
4096 to 1000

# Computer Vision Tasks: Semantic Segmentation

Classification



CAT

No spatial extent

Semantic  
Segmentation



GRASS, CAT, TREE,  
SKY

No objects, just pixels

Object  
Detection



DOG, DOG, CAT

Instance  
Segmentation



DOG, DOG, CAT

Multiple Objects

# Semantic Segmentation

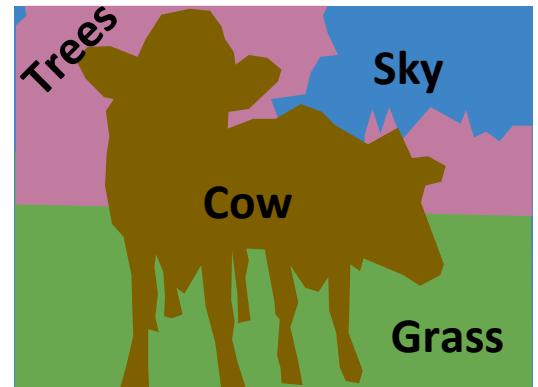
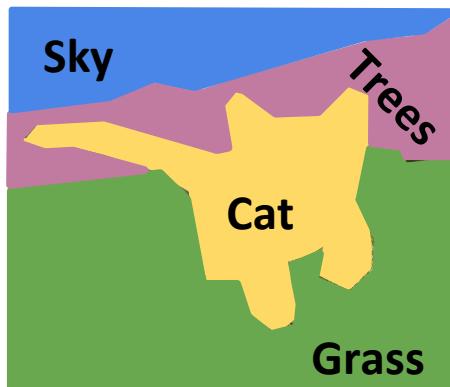
---

Label each pixel in the image with a category label

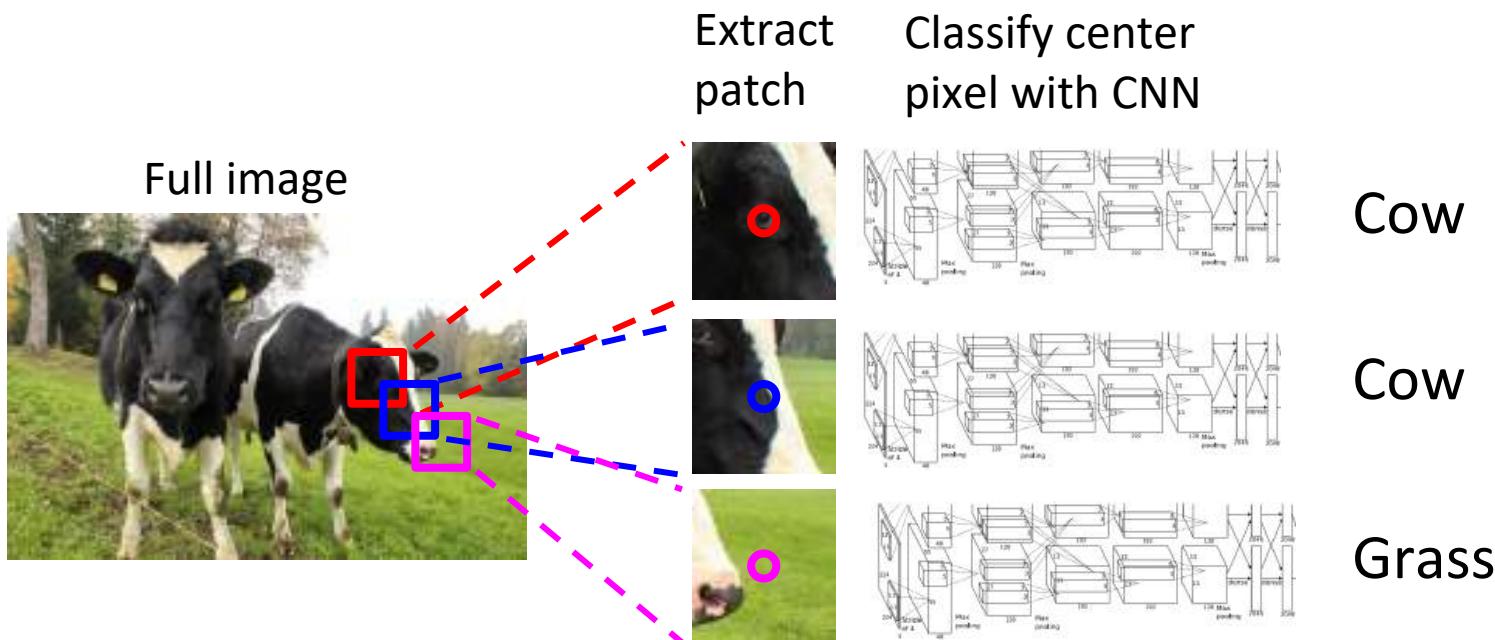
Don't differentiate instances, only care about pixels



[This image is CC0 public domain](#)



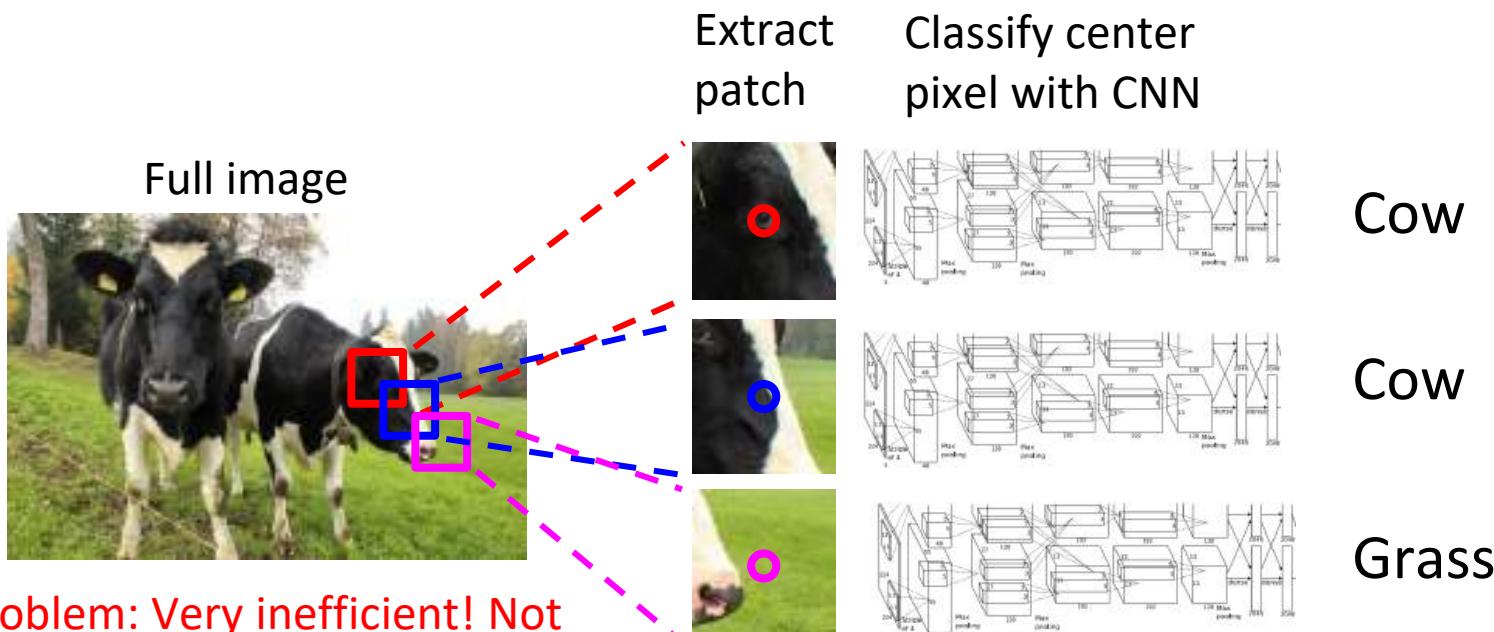
# Semantic Segmentation Idea: Sliding Window



Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

# Semantic Segmentation Idea: Sliding Window

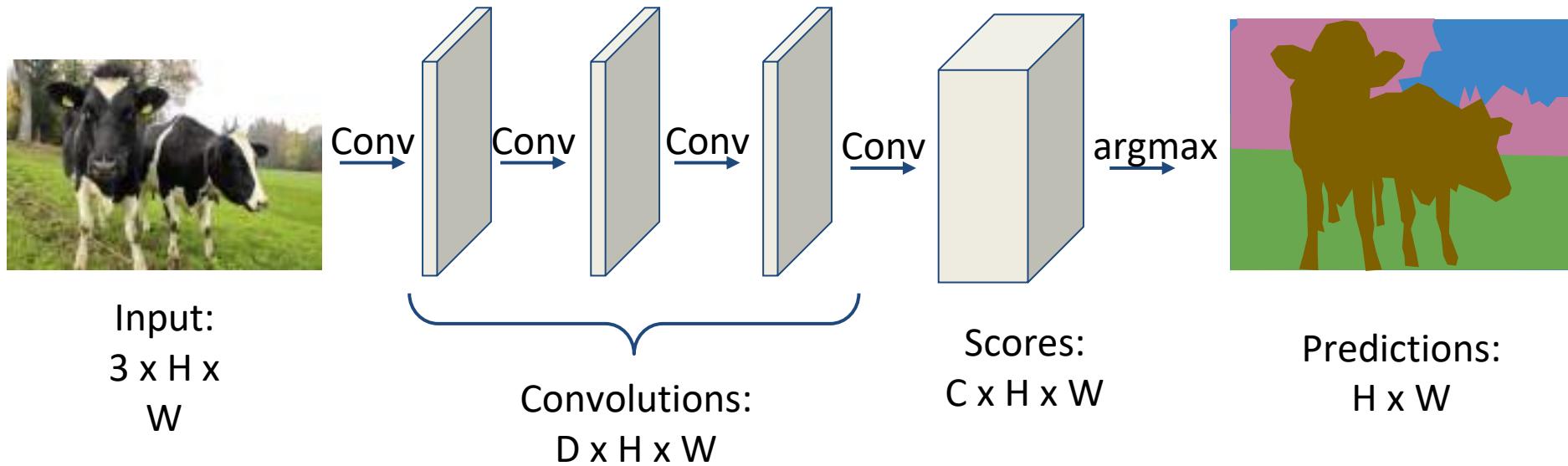


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013  
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

# Semantic Segmentation: Fully Convolutional Network

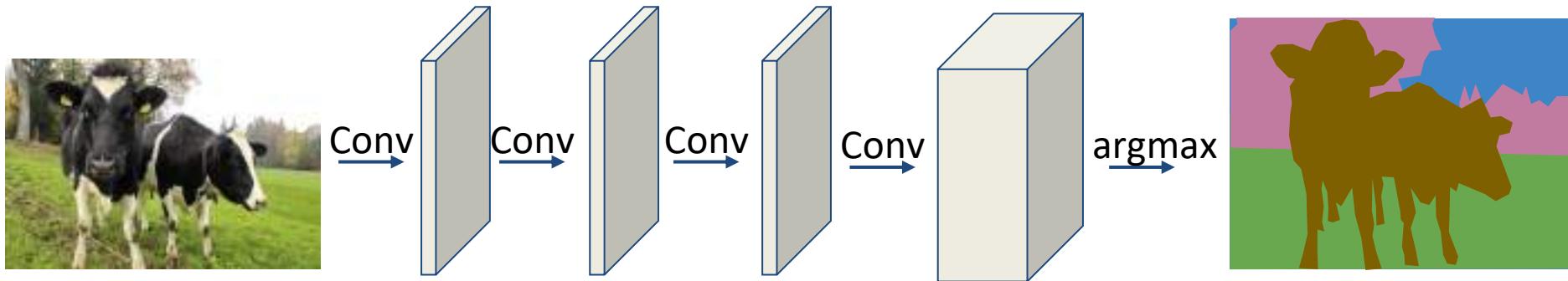
Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

# Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



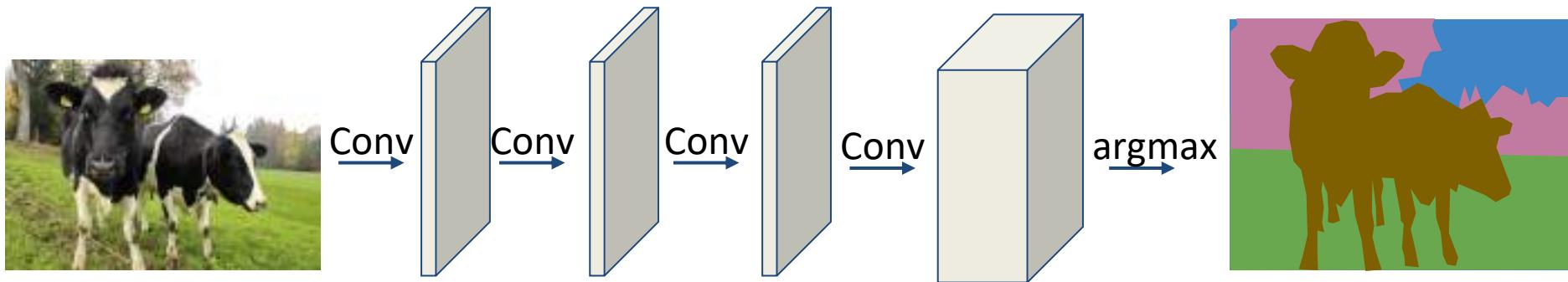
Input:  $3 \times H \times W$

**Problem #1:** Effective receptive field size is linear in number of conv layers: With  $L$   $3 \times 3$  conv layers, receptive field is  $1+2L$

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

# Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:  
3 x H x  
W

**Problem #1:** Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is  $1+2L$

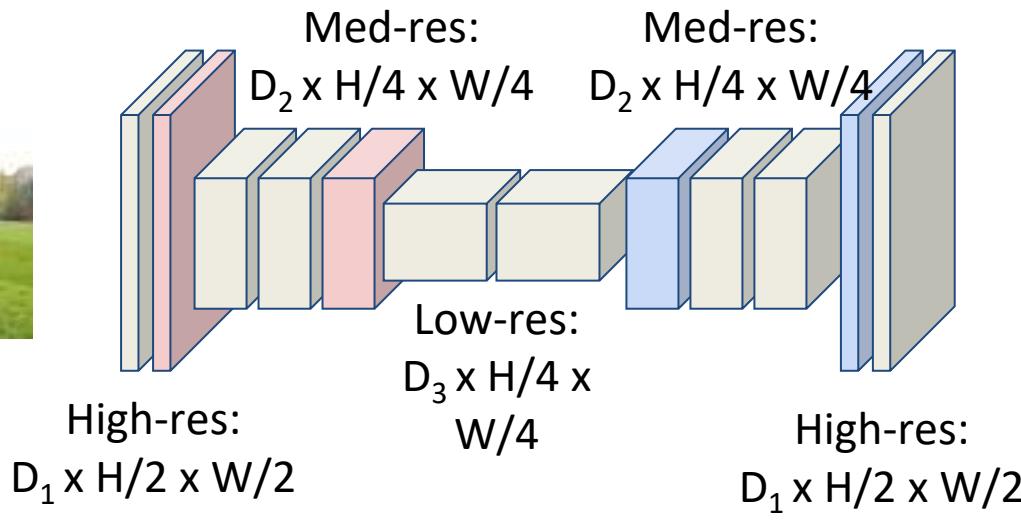
**Problem #2:** Convolution on high res images is expensive!  
Recall ResNet stem aggressively downsamples

# Semantic Segmentation: Fully Convolutional Network

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



Input:  
 $3 \times H \times W$



Predictions:  
 $H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015  
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

# Semantic Segmentation: Fully Convolutional Network

**Downsampling:**  
Pooling, strided convolution

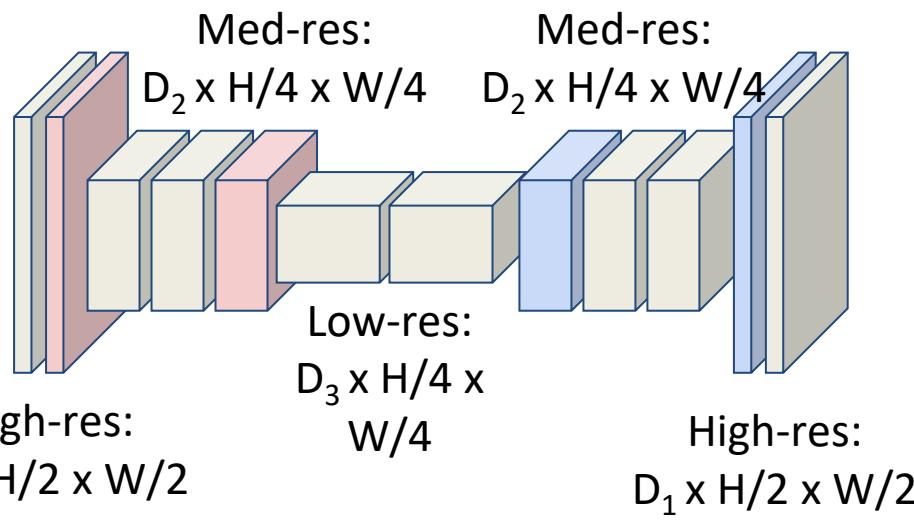


Input:  
 $3 \times H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015  
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Upsampling:**  
???

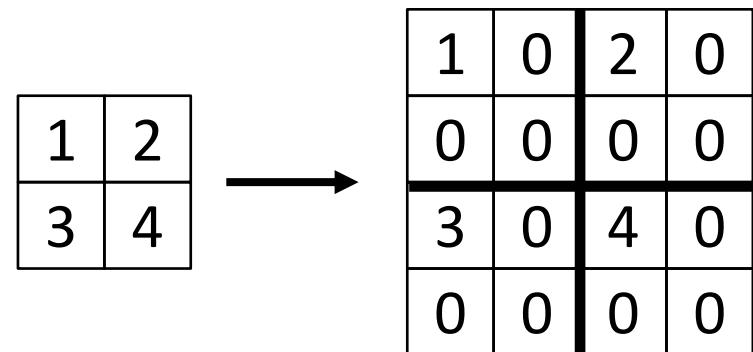


Predictions:  
 $H \times W$

# In-Network Upsampling: “Unpooling”

---

**Bed of Nails**



Input  
 $C \times 2 \times 2$

Output  
 $C \times 4 \times 4$

# In-Network Upsampling: “Unpooling”

---

**Bed of Nails**

1	2
3	4



1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input  
 $C \times 2 \times 2$

Output  
 $C \times 4 \times 4$

**Nearest Neighbor**

1	2
3	4



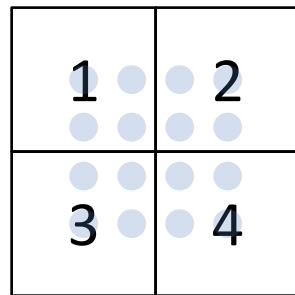
1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input  
 $C \times 2 \times 2$

Output  
 $C \times 4 \times 4$

# In-Network Upsampling: Bilinear ~~Interpolation~~

---



Input:  $C \times 2 \times 2$



1.0	1.2	1.7	2.0
0	5	5	0
1.5	1.7	2.2	2.5
0	5	5	0
2.5	2.7	3.2	3.5
0	5	5	0
3.0	3.2	3.7	4.0
0	5	5	0

Output:  $C \times 4 \times 4$

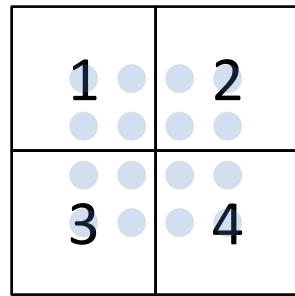
$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|) \quad i \in \{\lfloor x \rfloor - 1, \dots, \lceil x \rceil + 1\}$$

Use two closest neighbors in x and  
y to construct linear  
approximations

$$j \in \{\lfloor y \rfloor - 1, \dots, \lceil y \rceil + 1\}$$

# In-Network Upsampling: Bicubic ~~Interpolation~~

---



0.6	1.0	1.5	1.8
8	2	6	9
1.3	1.6	2.2	2.5
5	8	3	6
2.4	2.7	3.3	3.6
4	7	2	5
3.1	3.4	3.9	4.3
1	4	8	2

Input:  $C \times 2 \times 2$

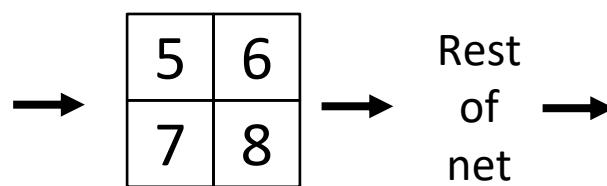
Output:  $C \times 4 \times 4$

Use **three** closest neighbors in x and y to  
construct **cubic** approximations  
(This is how we normally resize images!)

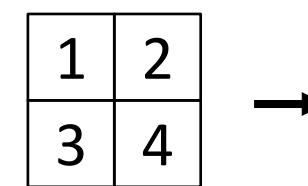
# In-Network Upsampling: “Max Unpooling”

**Max Pooling:** Remember which position had the max

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

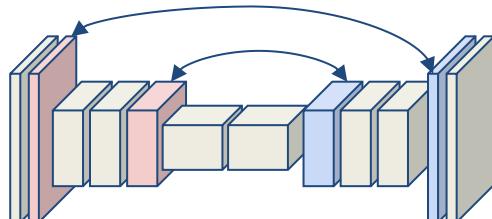


Rest  
of  
net



**Max Unpooling:** Place into remembered positions

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4



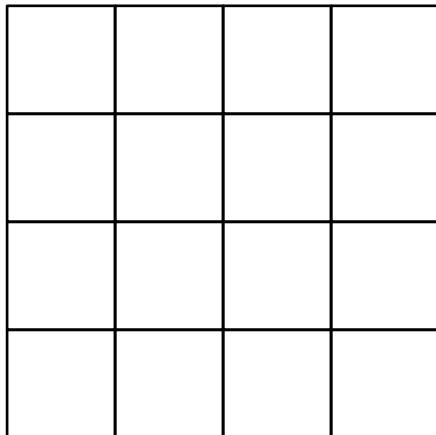
Pair each downsampling layer with an upsampling layer

Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

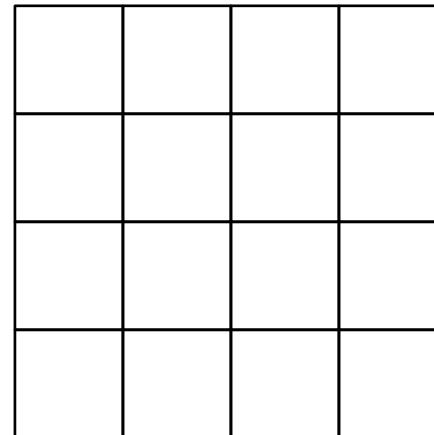
# Learnable Upsampling: Transposed Convolution

---

**Recall:** Normal  $3 \times 3$  convolution, stride 1, pad 1



Input:  $4 \times 4$

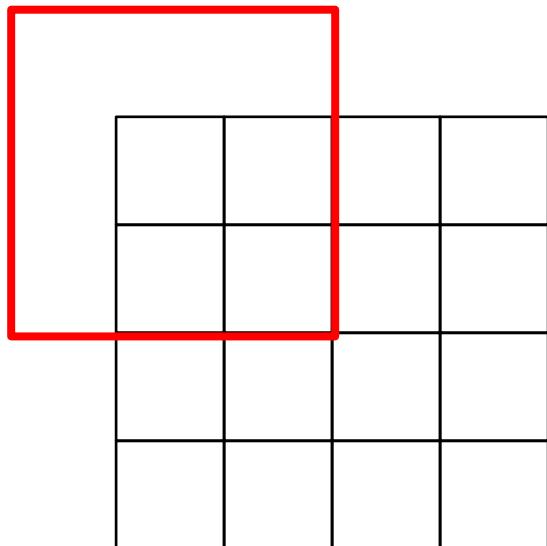


Output:  $4 \times 4$

# Learnable Upsampling: Transposed Convolution

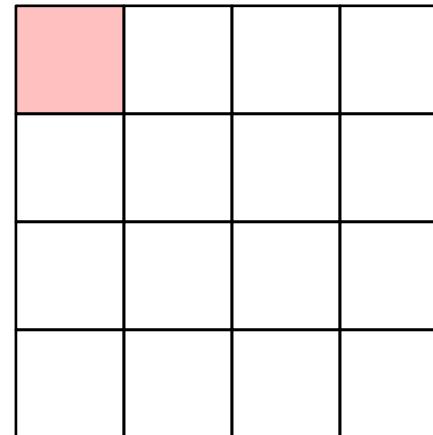
---

**Recall:** Normal  $3 \times 3$  convolution, stride 1, pad 1



Input:  $4 \times 4$

Dot product  
between input  
and filter

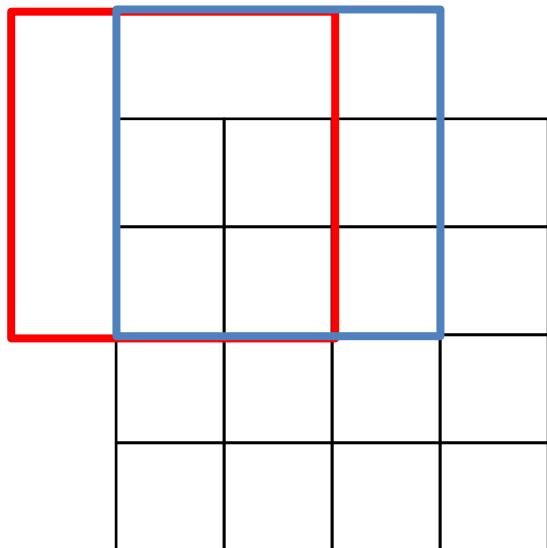


Output:  $4 \times 4$

# Learnable Upsampling: Transposed Convolution

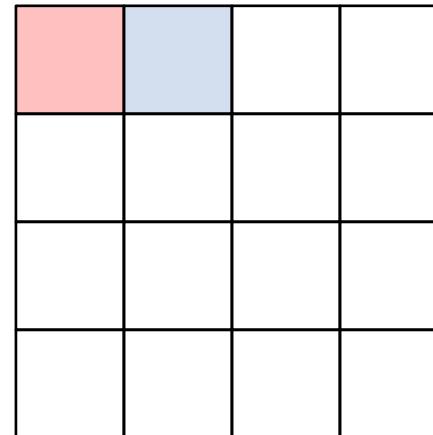
---

**Recall:** Normal  $3 \times 3$  convolution, stride 1, pad 1



Input:  $4 \times 4$

Dot product  
between input  
and filter

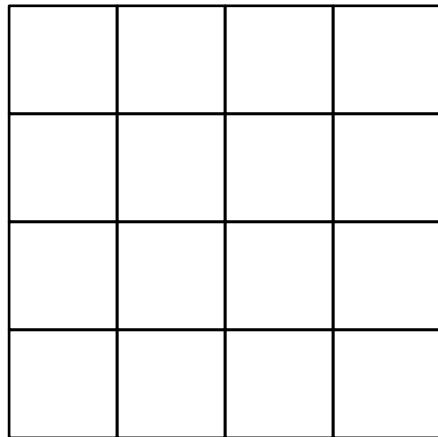


Output:  $4 \times 4$

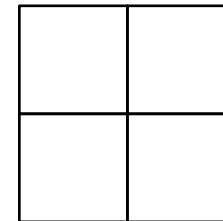
# Learnable Upsampling: Transposed ~~Convolution~~

---

**Recall:** Normal  $3 \times 3$  convolution, stride 2, pad 1



Input:  $4 \times 4$

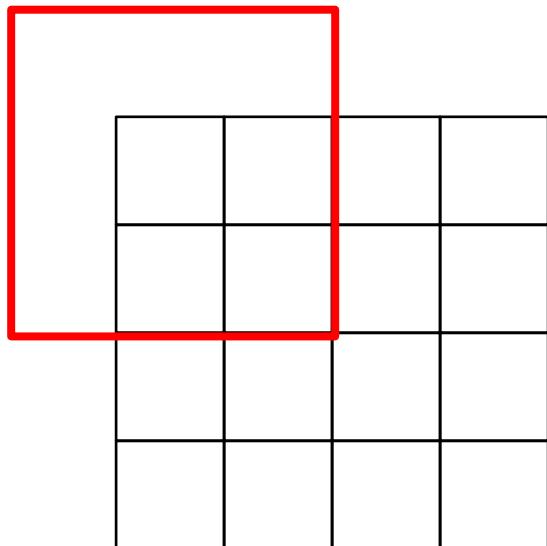


Output:  $2 \times 2$

# Learnable Upsampling: Transposed Convolution

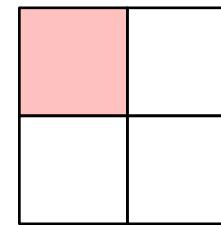
---

**Recall:** Normal  $3 \times 3$  convolution, stride 2, pad 1



Input:  $4 \times 4$

Dot product  
between input  
and filter

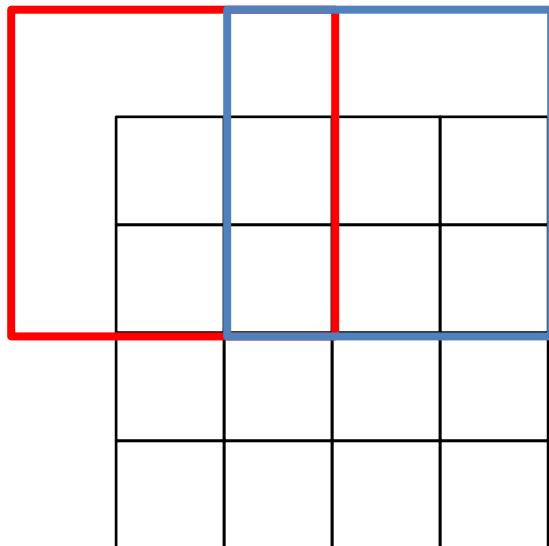


Output:  $2 \times 2$

# Learnable Upsampling: Transposed Convolution

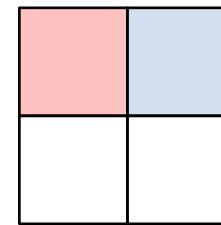
---

**Recall:** Normal  $3 \times 3$  convolution, stride 2, pad 1



Input:  $4 \times 4$

Dot product  
between input  
and filter

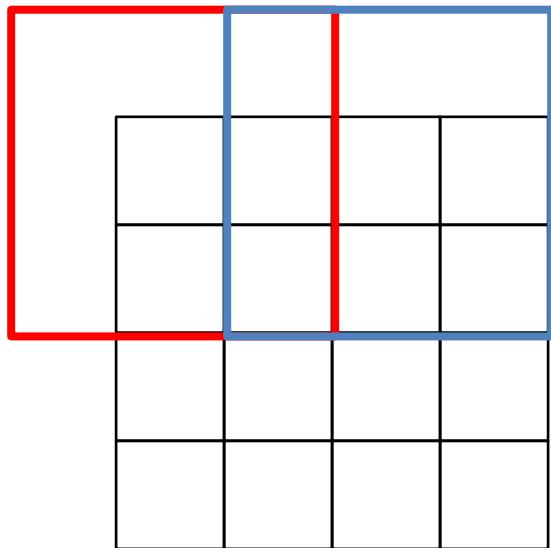


Output:  $2 \times 2$

# Learnable Upsampling: Transposed Convolution

---

**Recall:** Normal  $3 \times 3$  convolution, stride 2, pad 1

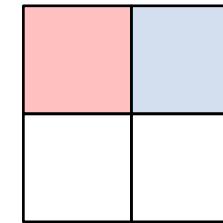


Input:  $4 \times 4$

Convolution with stride  $> 1$  is “Learnable Downsampling”  
Can we use stride  $< 1$  for “Learnable Upsampling”?



Dot product  
between input  
and filter



Output:  $2 \times 2$

# Learnable Upsampling: Transposed Convolution

---

**3 x 3 convolution transpose, stride 2**





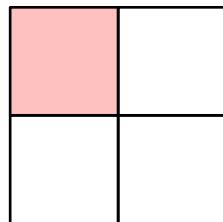
Input: 2 x 2

Output: 4 x 4

# Learnable Upsampling: Transposed Convolution

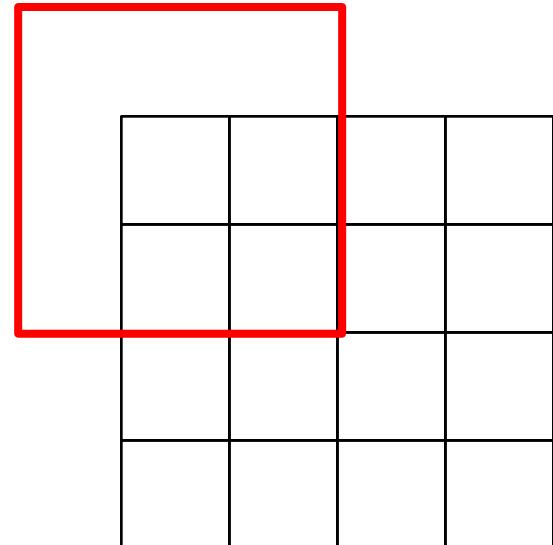
---

**3 x 3 convolution transpose, stride 2**



Input: 2 x 2

Weight filter by  
input value and  
copy to output



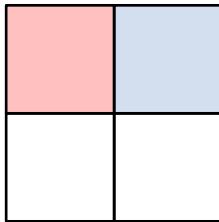
Output: 4 x 4

# Learnable Upsampling: Transposed Convolution

---

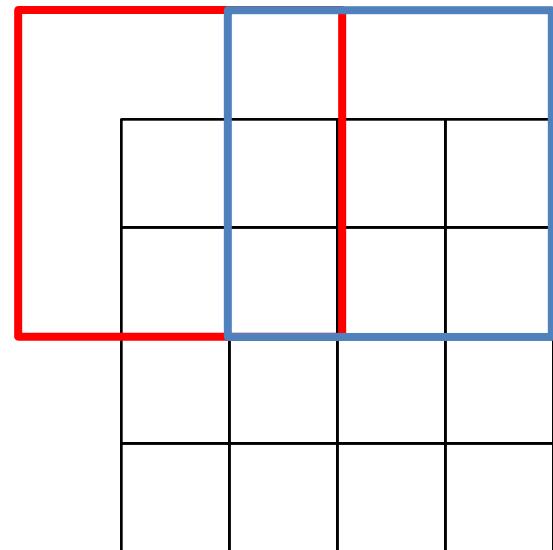
**3 x 3 convolution transpose, stride 2**

Filter moves 2 pixels in output  
for every 1 pixel in input



Input: 2 x 2

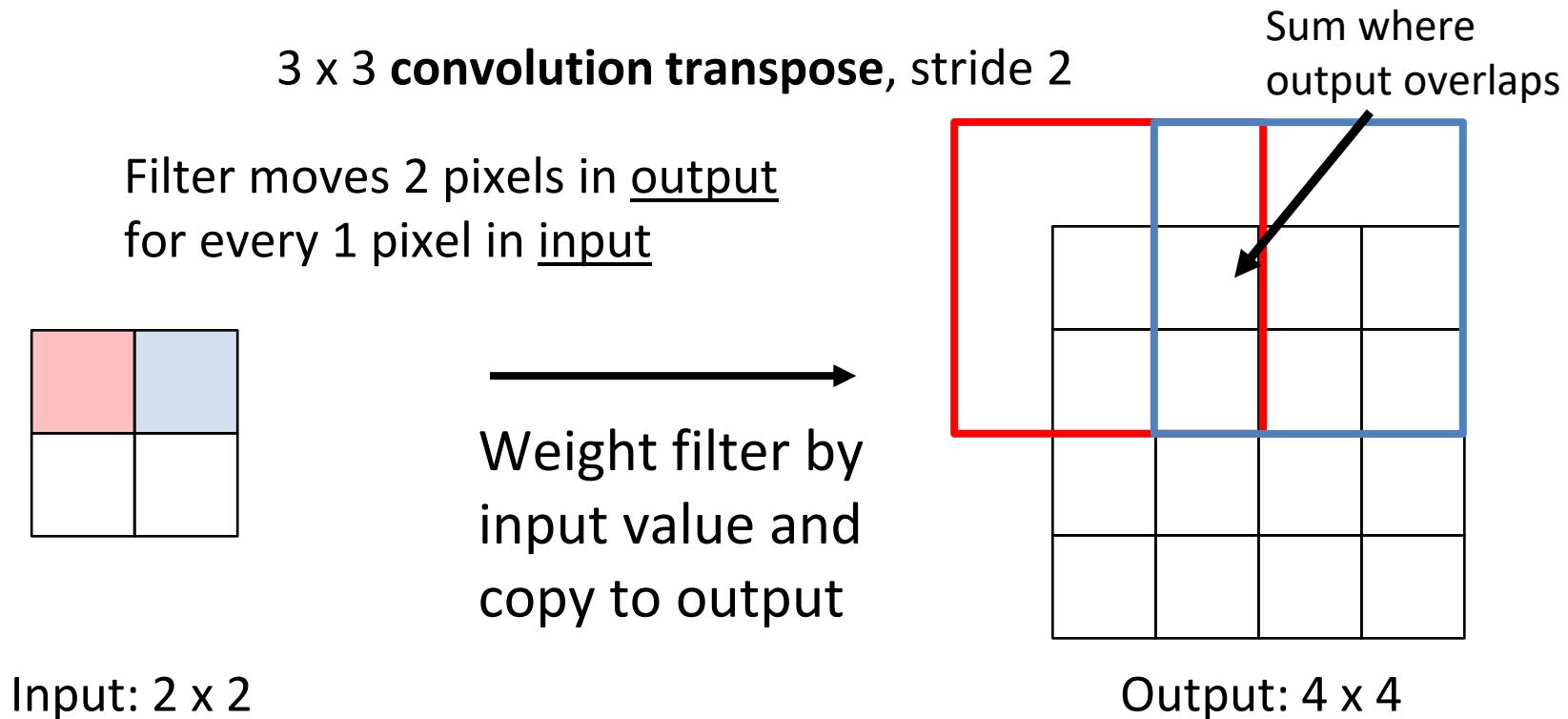
Weight filter by  
input value and  
copy to output



Output: 4 x 4

# Learnable Upsampling: Transposed Convolution

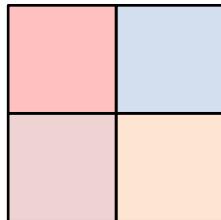
---



# Learnable Upsampling: Transposed Convolution

---

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output

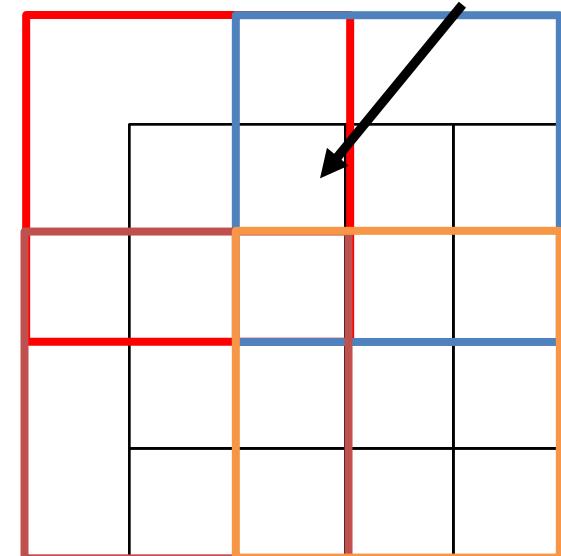


Input: 2 x 2

Weight filter by  
input value and  
copy to output

3 x 3 convolution transpose, stride 2

Sum where  
output overlaps



# Transposed Convolution: 1D example

---

**Input**

a
b

**Filter**

x
y
z

**Output**

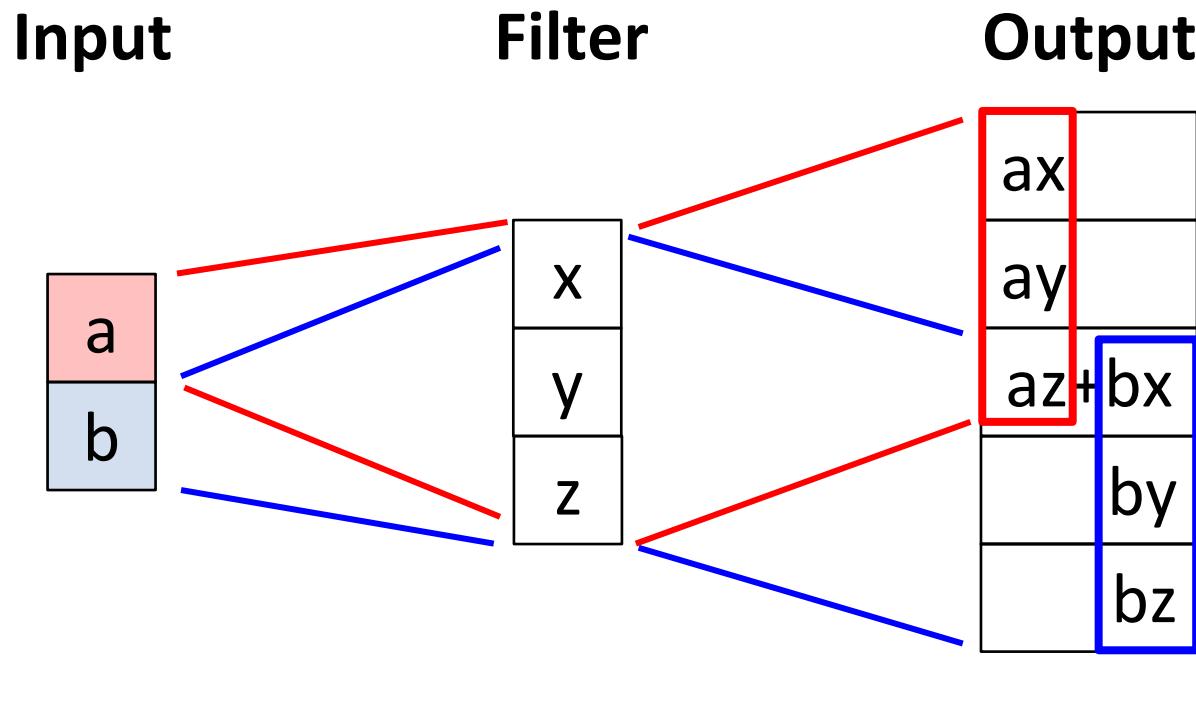
ax
ay
az+bx
by
bz

Output has copies of filter weighted by input

Stride 2: Move 2 pixels output for each pixel in input

Sum at overlaps

# Transposed Convolution: 1D example



This has many names:

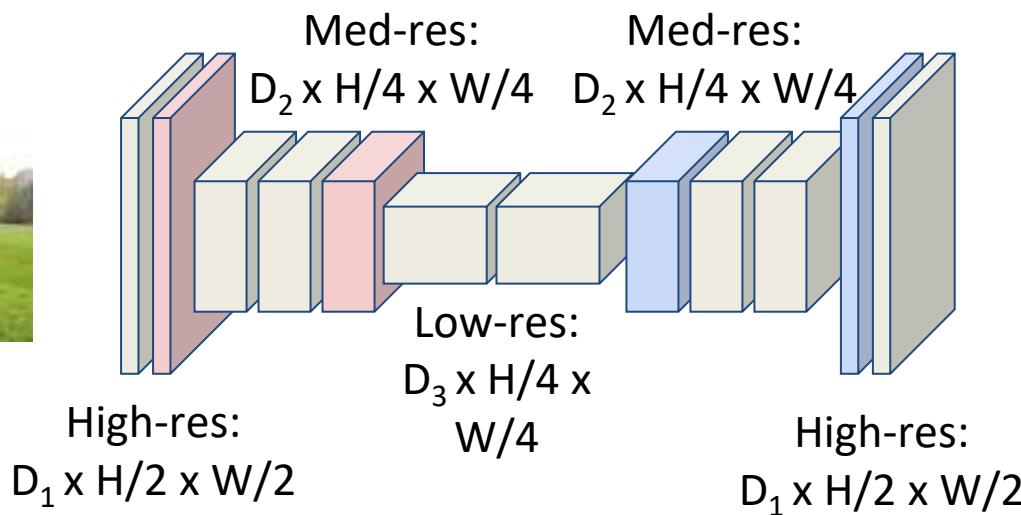
- Deconvolution (bad)!
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution
- Transposed Convolution (best name)

# Semantic Segmentation: Fully Convolutional Network

**Downsampling:**  
Pooling, strided convolution



Input:  
 $3 \times H \times W$



Design network as a bunch of convolutional layers, with  
**downsampling** and **upsampling** inside the network!

**Upsampling:**  
interpolation,  
transposed conv



Loss function: Per-Pixel cross-entropy

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015  
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

# U-Nets

---

Skip connections between  
encoding and decoding stages  
Widely used in image denoising,  
inpainting, flow, stereo, ...

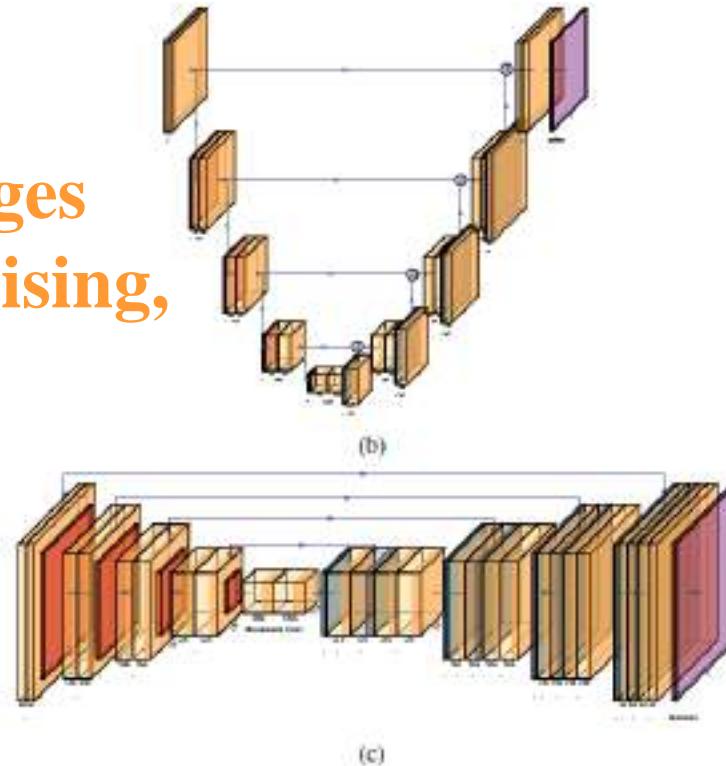
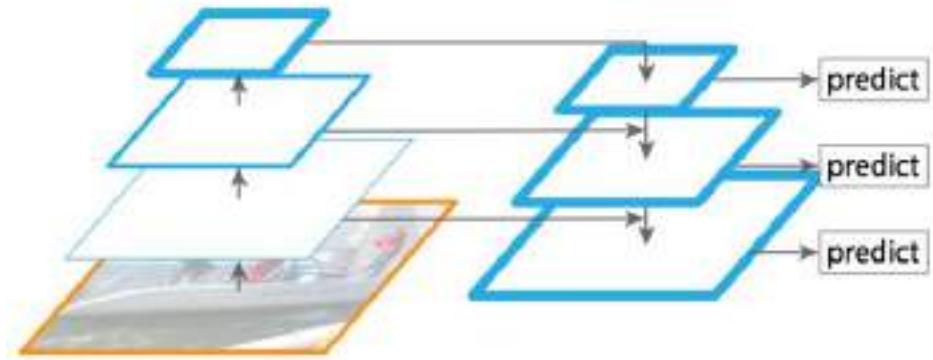
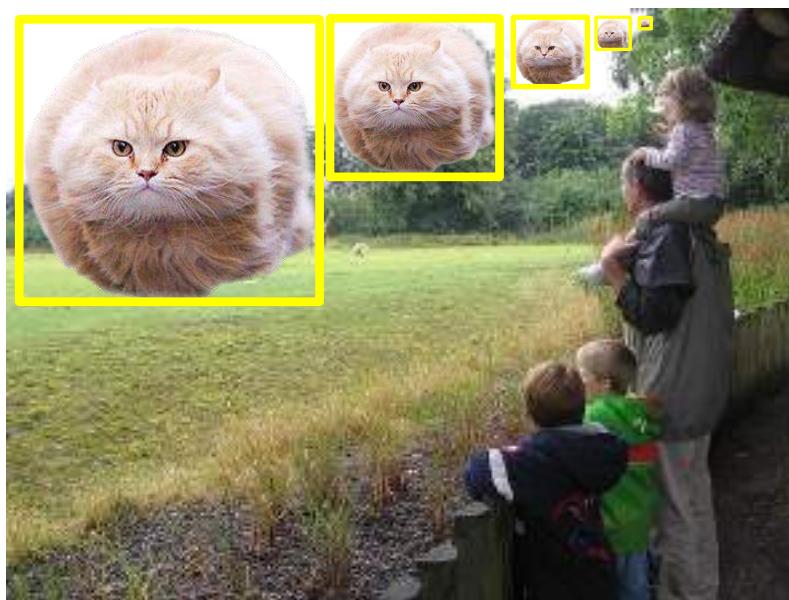


Figure 5.41 (a) The deconvolution network of Noh, Hong, and Han (2015) © 2015 IEEE and (b–c) the U-Net of Ronneberger, Fischer, and Brox (2015), redrawn using the PlotNeuralNet LaTeX package by Marc Dietke. In addition to the fine-to-coarse-to-fine bottleneck used in (a), the U-Net also has skip connections between encoding and decoding layers at the same resolution.

# Feature Pyramid Network

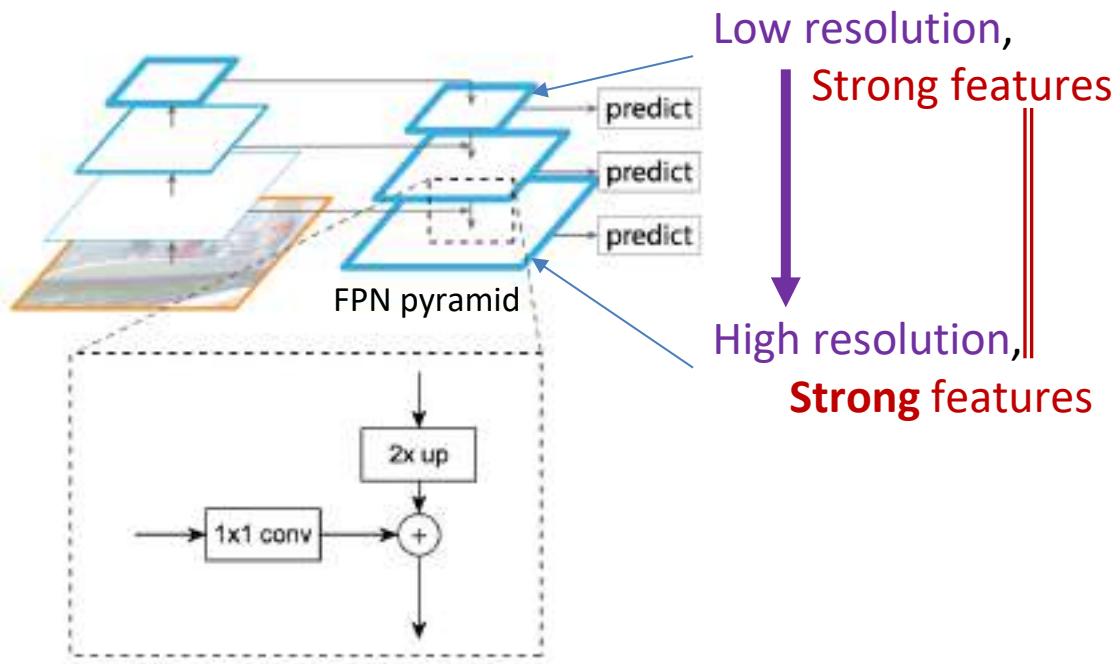
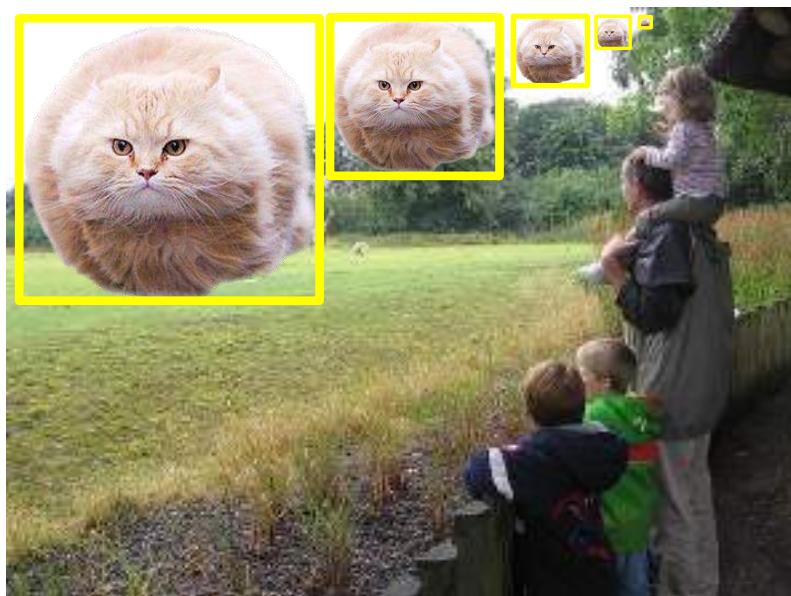
---



(d) Feature Pyramid Network

Top-down enrichment of high-res features –  
*fast, less suboptimal*

# No Compromise on Feature Quality, Still Fast



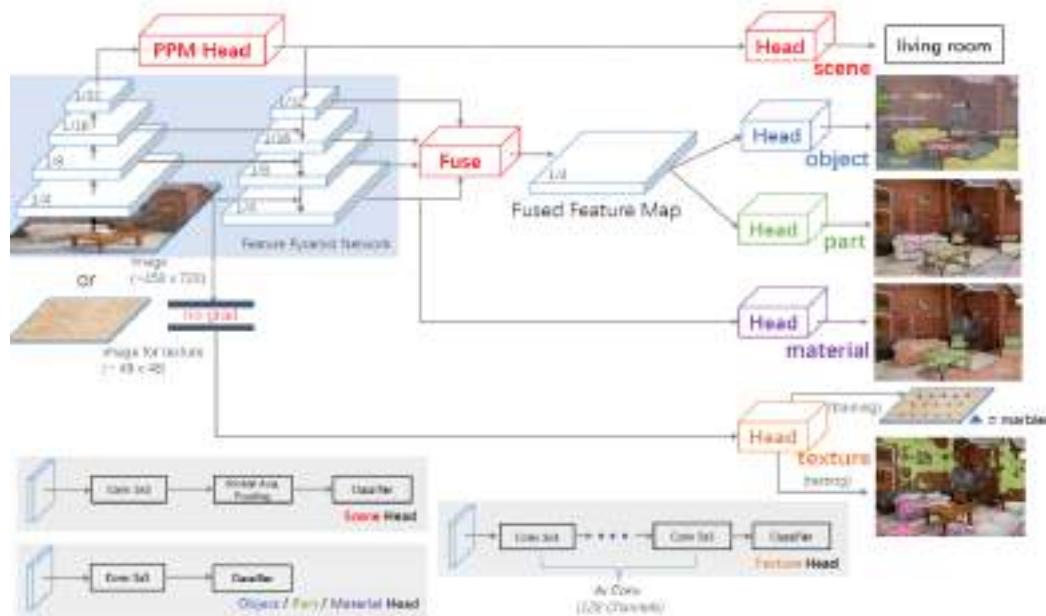
Lin et al. Feature Pyramid Networks for Object Detection. CVPR 2017. See also: Shrivastava's TDM.

# UPerNet: FPN + fusion

---

Unified Perceptual Parsing for Scene Understanding

7



# Application: monocular depth estimation

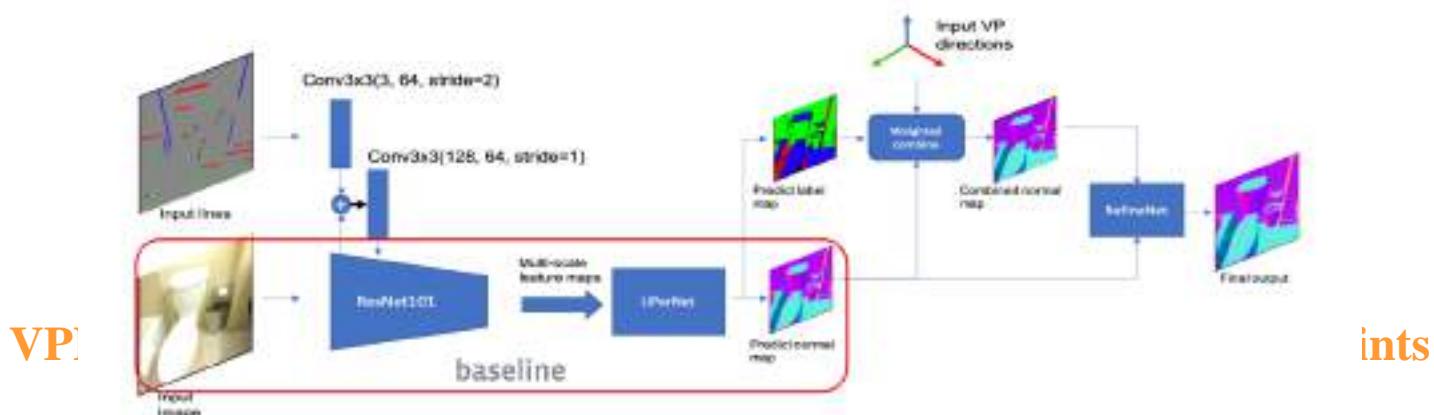


Figure 2. The full pipeline of our proposed model. The network takes an RGB image and a Manhattan line map as input, and produces a Manhattan label map and a raw normal prediction as intermediate output. These intermediate outputs are then combined with the analytically computed dominant vanishing points to generate a “combined normal map”. This operation is differentiable. Finally, the combined and raw normal maps are fused through a refinement network to produce the final normal prediction.

## **6.5 Video Understanding - Definition**

---

**Extension of image understanding to video**

**Detecting and describing human actions**

**Actions form sequences of activity**

# Early Work (1990s)

**Human activity recognition + motion tracking**  
**Techniques: Point & mesh tracking, spatio-temporal signatures**  
**Survey: Aggarwal & Cai (1999)**

# Early Work (1990s)

---



DensePose-RCNN Results



DensePose COCO Dataset



**Figure 6.43** *Dense pose estimation aims at mapping all human pixels of an RGB image to the 3D surface of the human body (Güler, Neverova, and Kokkinos 2018) © 2018 IEEE. The paper describes DensePose-COCO, a large-scale ground-truth dataset containing manually annotated image-to-surface correspondences for 50K persons and a DensePose-RCNN trained to densely regress UV coordinates at multiple frames per second.*

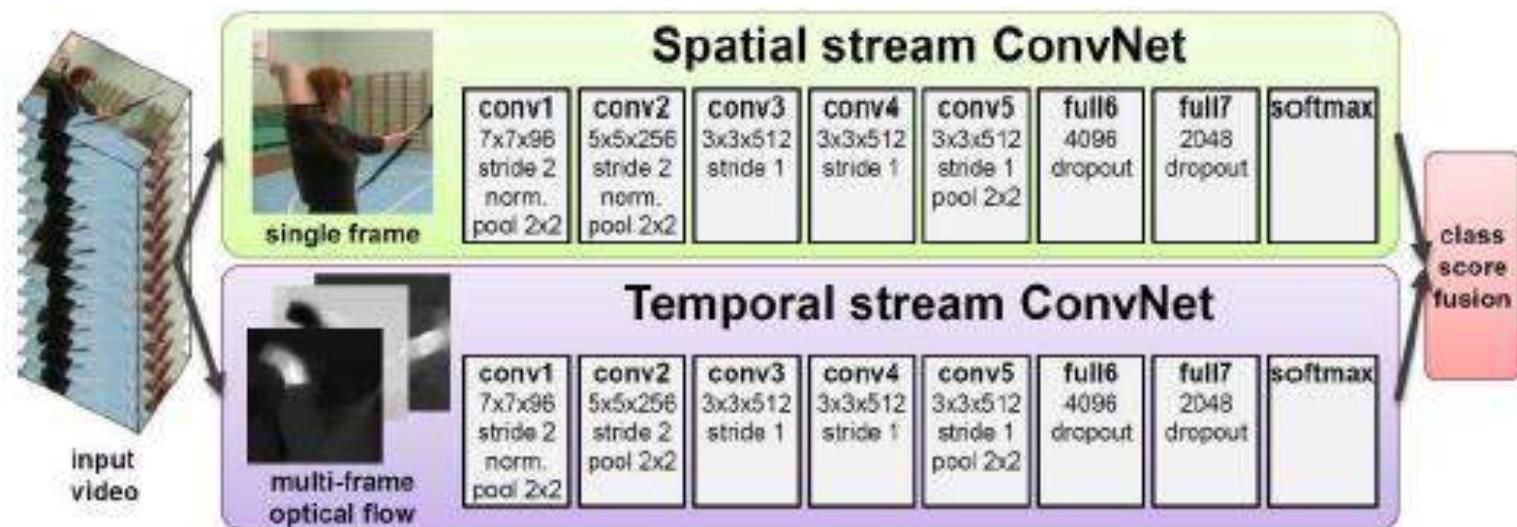
# 2000s – Spatio-temporal Features

---

Optical flow patches (Efros et al. 2003)

Spatio-temporal feature detectors (Laptev et al. 2008)

Enhancements: image context, tracked trajectories



# Dataset for Video Understanding and Action Recog.

---

Name/URL	Metadata	Contents/Reference
Charades <a href="https://prior.allenai.org/projects/charades">https://prior.allenai.org/projects/charades</a>	Actions, objects, descriptions	9.8k videos Sigurdsson, Varol <i>et al.</i> (2016)
YouTube8M <a href="https://research.google.com/youtube8m">https://research.google.com/youtube8m</a>	Entities	4.8k visual entities, 8M videos Abu-El-Haija, Kothari <i>et al.</i> (2016)
Kinetics <a href="https://deepmind.com/research/open-source/kinetics">https://deepmind.com/research/open-source/kinetics</a>	Action classes	700 action classes, 650k videos Carreira and Zisserman (2017)
“Something-something” <a href="https://20bn.com/datasets/something-something">https://20bn.com/datasets/something-something</a>	Actions with objects	174 actions, 220k videos Goyal, Kahou <i>et al.</i> (2017)
AVA <a href="https://research.google.com/ava">https://research.google.com/ava</a>	Actions	80 actions in 430 15-minute videos Gu, Sun <i>et al.</i> (2018)
EPIC-KITCHENS <a href="https://epic-kitchens.github.io">https://epic-kitchens.github.io</a>	Actions and objects	100 hours of egocentric videos Damen, Doughty <i>et al.</i> (2018)

**Table 6.3** *Datasets for video understanding and action recognition.*

# 2010s – Deep Learning for Video

---

**3D ConvNets (Ji et al. 2013, Karpathy et al. 2014, Tran et al. 2015)**

**Two-stream CNNs (Simonyan & Zisserman, 2014)**

**Hybrid CNN + LSTM models**

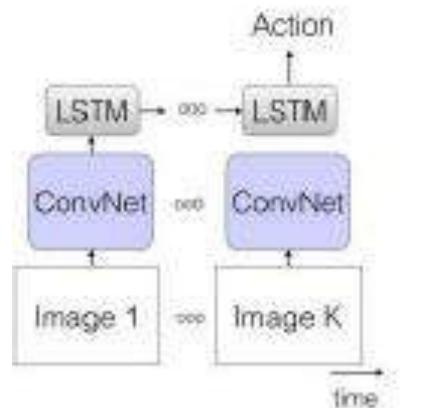
**SlowFast networks (Feichtenhofer et al. 2019)**

**Long-term feature bank (Wu et al. 2019)**

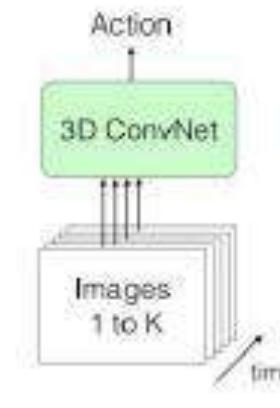
# 2010s – Deep Learning for Video

---

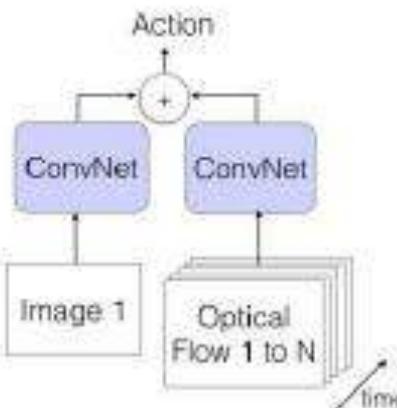
a) LSTM



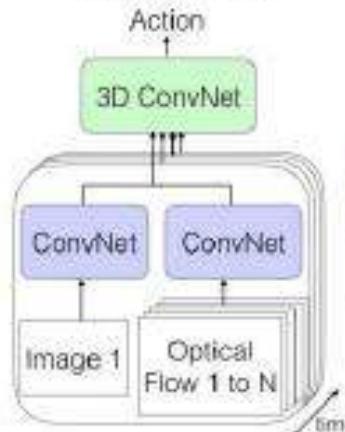
b) 3D-ConvNet



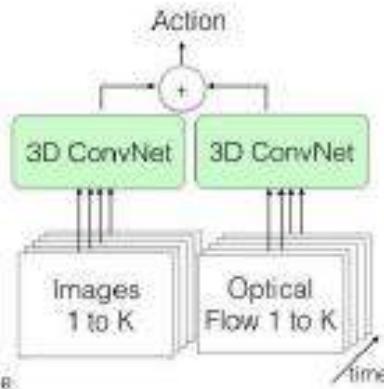
c) Two-Stream



d) 3D-Fused  
Two-Stream

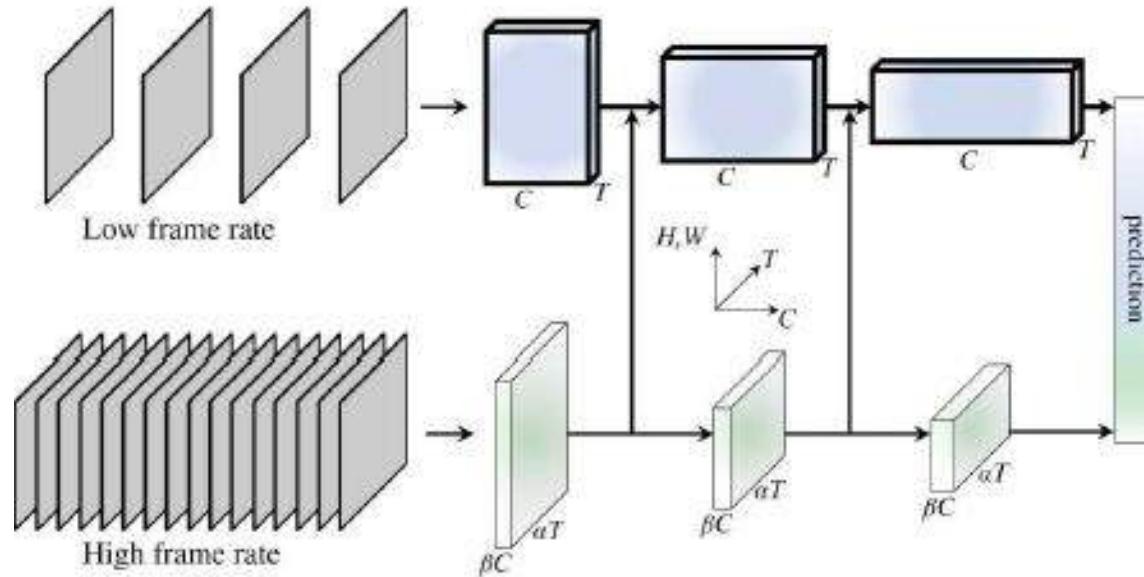


e) Two-Stream  
3D-ConvNet



# 2010s – Deep Learning for Video

---



a SlowFast network

with a low frame rate, low temporal resolution Slow pathway and a high frame rate, higher temporal resolution Fast pathway (Feichtenhofer, Fan et al. 2019) © 2019 IEEE.

# Recent Advances

---

**Self-supervised learning for video**

**Multi-modal input: video + audio**

**Beyond actions: Dynamic scene recognition**

## 6.6 Vision & Language - Goal

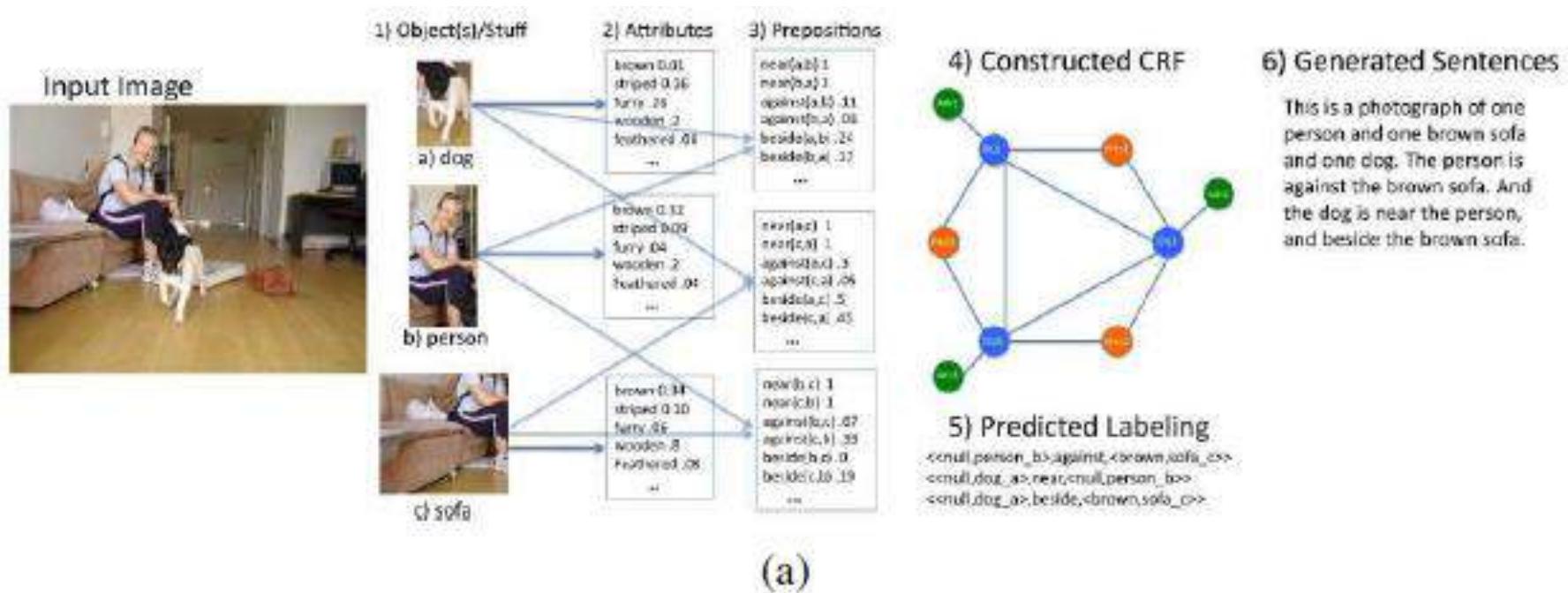
---

**Toward Artificial General Intelligence (AGI)**  
**Integration with speech, language, reasoning**  
**& knowledge**

# Early Work

## Image captioning & labeling

BabyTalk system (Kulkarni et al. 2013): detect objects, infer labels, generate captions



# Image Captioning Systems

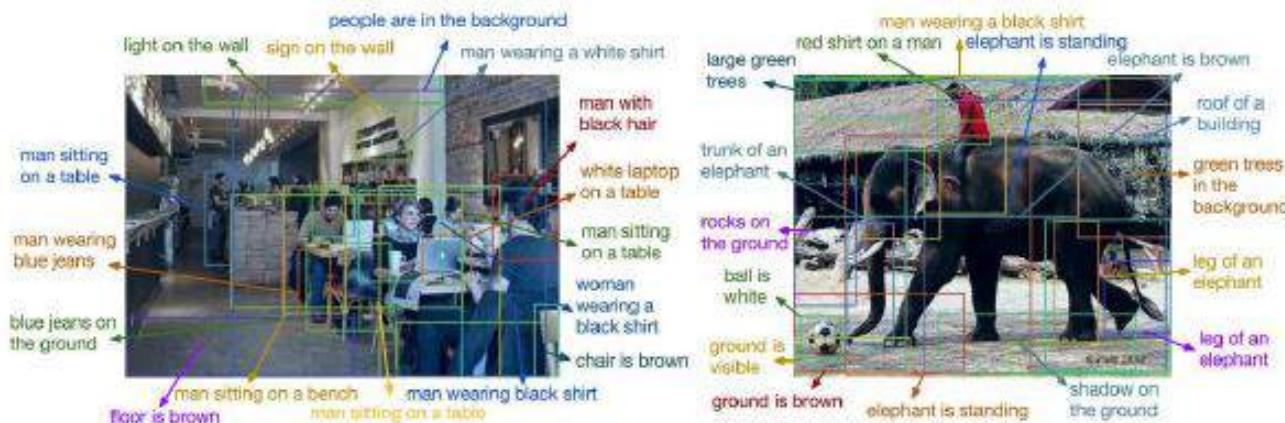
---

**CNN + RNN/LSTM pipelines**

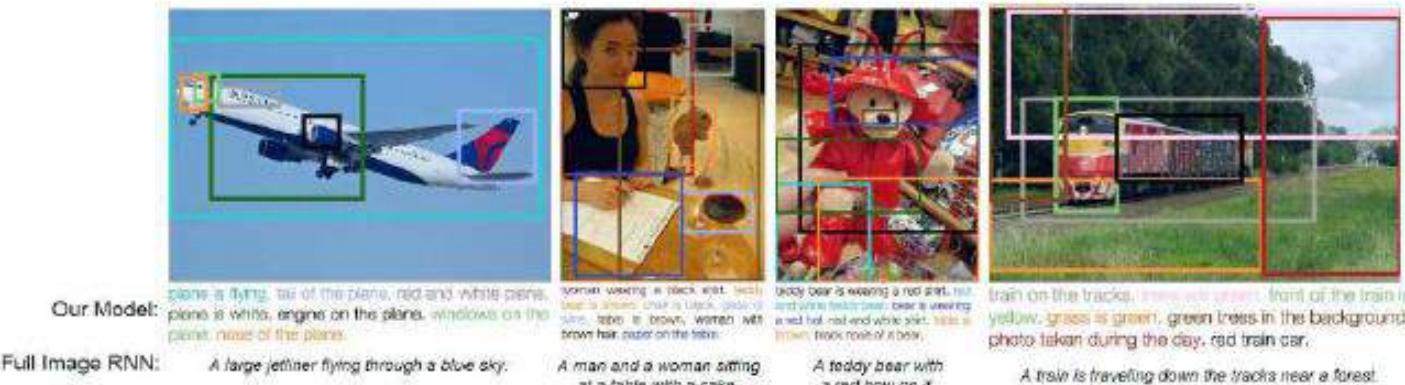
**Techniques: MIL, maximum entropy,  
attention**

**Nearest-neighbor captioning (Devlin et al.  
2015)**

**Transformers for captioning**



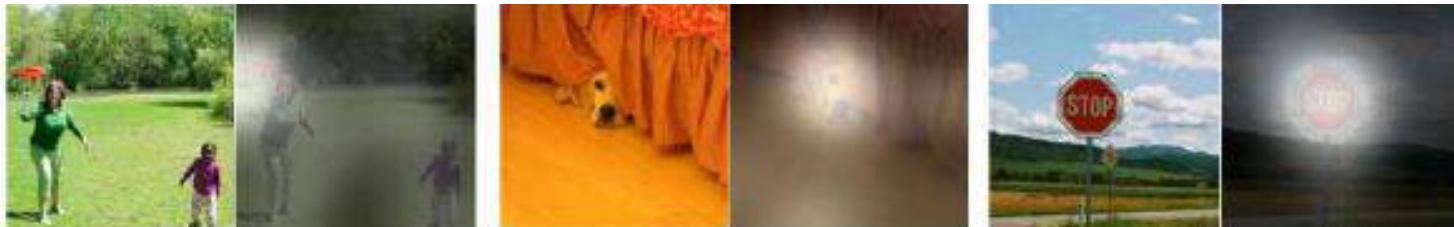
(b)



(c)

**Figure 6.45** *Image captioning systems: (a) BabyTalk detects objects, attributes, and positional relationships and composes these into image captions (Kulkarni, Premraj et al. 2013) © 2013 IEEE; (b–c) DenseCap associates word phrases with regions and then uses an RNN to construct plausible sentences (Johnson, Karpathy, and Fei-Fei 2016) © 2016 IEEE.*

# Image Captioning Systems

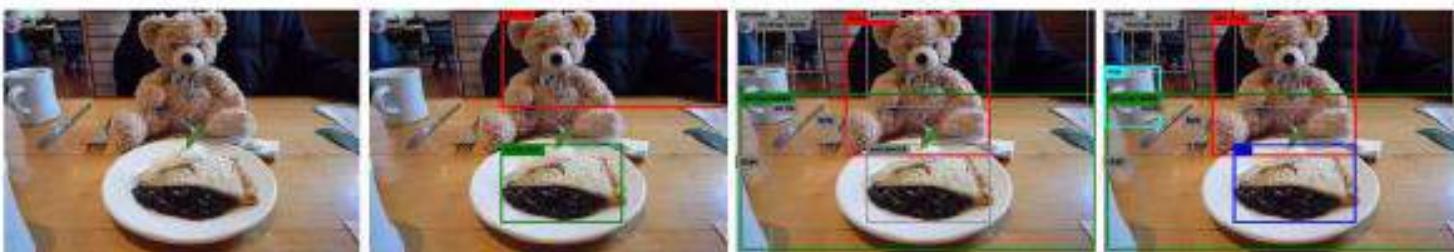


A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background,

(a)



A close up of a stuffed animal on a plate.

A person is sitting at a table with a sandwich.

A teddy bear sitting on a table with a plate of food.

A Mr. Ted sitting at a table with a pie and a cup of coffee.

(b)

**Figure 6.46** Image captioning with attention: (a) The “Show, Attend, and Tell” system, which uses hard attention to align generated words with image regions © Xu, Ba et al. (2015); (b) Neural Baby Talk captions generated using different detectors, showing the association between words and grounding regions (Lu, Yang et al. 2018) © 2018 IEEE.

# Zero-shot Models

---

**CLIP (Radford et al. 2021)**

**Trained on 400M text–image pairs**

**Contrastive learning for matching**

**Enables zero-shot classification**

**Robust but vulnerable to adversarial attacks**



**Figure 6.47** An adversarial typographic attack used against CLIP (Radford, Kim et al. 2021) discovered by ©Goh, Cammarata et al. (2021). Instead of predicting the object that exists in the scene, CLIP predicts the output based on the adversarial handwritten label.

# Datasets for Vision + Language

---

Name/URL	Metadata	Contents/Reference
Flickr30k (Entities) <a href="https://shannon.cs.illinois.edu/DenotationGraph">https://shannon.cs.illinois.edu/DenotationGraph</a> <a href="http://bryanplummer.com/Flickr30kEntities">http://bryanplummer.com/Flickr30kEntities</a>	Image captions (grounded)	30k images (+ bounding boxes) Young, Lai <i>et al.</i> (2014) Plummer, Wang <i>et al.</i> (2017)
COCO Captions <a href="https://cocodataset.org/#captions-2015">https://cocodataset.org/#captions-2015</a>	Whole image captions	1.5M captions, 330k images Chen, Fang <i>et al.</i> (2015)
Conceptual Captions <a href="https://ai.google.com/research/ConceptualCaptions">https://ai.google.com/research/ConceptualCaptions</a>	Whole image captions	3.3M image caption pairs Sharma, Ding <i>et al.</i> (2018)
YFCC100M <a href="http://projects.dfki.uni-kl.de/yfcc100m">http://projects.dfki.uni-kl.de/yfcc100m</a>	Flickr metadata	100M images with metadata Thomee, Shamma <i>et al.</i> (2016)
Visual Genome <a href="https://visualgenome.org">https://visualgenome.org</a>	Dense annotations	108k images with region graphs Krishna, Zhu <i>et al.</i> (2017)
VQA v2.0 <a href="https://visualqa.org">https://visualqa.org</a>	Question/answer pairs	265k images Goyal, Khot <i>et al.</i> (2017)
VCR <a href="https://visualcommonsense.com">https://visualcommonsense.com</a>	Multiple choice questions	110k movie clips, 290k QAs Zellers, Bisk <i>et al.</i> (2019)
GQA <a href="https://visualreasoning.net">https://visualreasoning.net</a>	Compositional QA	22M questions on Visual Genome Hudson and Manning (2019)
VisDial <a href="https://visualdialog.org">https://visualdialog.org</a>	Dialogs for chatbot	120k COCO images + dialogs Das, Kottur <i>et al.</i> (2017)

# Evaluation Metrics

---

BLEU, ROUGE, METEOR

CIDEr (consensus-based), SPICE (semantic-based)

# Text-to-Image Generation

---

**Early: RNNs (Mansimov 2016), GANs (Reed 2016)**

**Recent: DALL·E (Ramesh 2021)**

**VQ-VAE-2 + Transformer decoder**

**High-resolution creative images & image completion**

# Visual Question Answering (VQA)

---

**Beyond captioning: reasoning-based QA**

**Datasets: VQA, VCR, GQA**

**Methods: attention, fusion models,  
transformers**

# Visual Dialog

---

**Conversational Q&A about images**

**Dataset: VisDial (Das et al. 2017)**

**Applications: Image-grounded chatbots**

# Vision-Language Pre-training

---

**Large-scale multimodal pre-training  
Examples: ViLBERT, Oscar, other  
transformer-based models**

## **Class Test Questions**

Q1. What is the main difference between nearest-neighbor interpolation and radial basis function (RBF) interpolation when fitting scattered data?

Q2. Given 15 random sample points from a smooth 2D function, which interpolation technique would you choose to get a smooth continuous surface and why?

Q3. You fit a function using interpolation and get the following MSEs: Method A: training=0.001, validation=0.200; Method B: training=0.020, validation=0.025. Which method shows signs of overfitting and why?

Q4. Why might the L1 norm be preferred over the L2 norm when fitting data that contains outliers?

Q5. What is the general form of an energy minimization problem in variational methods, and what do the data term and regularization term represent?

Q6. What property does total variation regularization preserve better than simple L2 smoothing, especially in image denoising?

Q7. What is the key advantage of a bilateral solver compared to a standard smoothing filter?

Q8. In an image labeling MRF, what do the unary and pairwise potentials represent?

Q9. How does a Conditional Random Field (CRF) differ from a standard MRF in terms of probability modeling?

Q10. In interactive segmentation using an MRF, how can user-provided scribbles be incorporated into the energy minimization formulation?

## Answer Sheet with Explanations

**Q1 Answer:** Nearest-neighbor assigns the value of the closest sample point; RBF uses weighted sums of radial basis functions for smooth interpolation.

**Explanation:** RBF interpolation considers all sample points for smoothness, while nearest neighbor is discontinuous.

**Q2 Answer:** RBF interpolation or Kriging, because they handle irregularly spaced data and produce smooth continuous surfaces.

**Explanation:** Global smooth methods like RBF work well without grid constraints.

**Q3 Answer:** Method A shows overfitting: very low training error but high validation error.

**Explanation:** Overfitting = learns noise in training data, poor generalization.

**Q4 Answer:** L1 norm is less sensitive to outliers than L2 norm.

**Explanation:** L2 squares errors, amplifying large deviations; L1 treats all proportionally.

**Q5 Answer:**  $E(f) = E_{\text{data}}(f) + \lambda E_{\text{reg}}(f)$ , where data term fits observed data and regularization imposes smoothness or priors.

**Explanation:** Variational methods balance data fidelity with smoothness or prior knowledge.

**Q6 Answer:** Preserves sharp edges while removing noise.

**Explanation:** TV penalizes total gradient magnitude, allowing edges but removing small noise gradients.

**Q7 Answer:** Smooths while preserving strong edges by considering both spatial proximity and intensity similarity.

**Explanation:** Bilateral filters adapt smoothing strength to avoid blurring across edges.

**Q8 Answer:** Unary: cost of assigning a label to a pixel based on data. Pairwise: penalty for label differences between neighbors.

**Explanation:** Unary = pixel data; Pairwise = spatial coherence.

**Q9 Answer:** CRF models  $P(x|y)$  directly given observations, MRF models joint  $P(x, y)$ .

**Explanation:** CRFs avoid modeling data distribution explicitly.

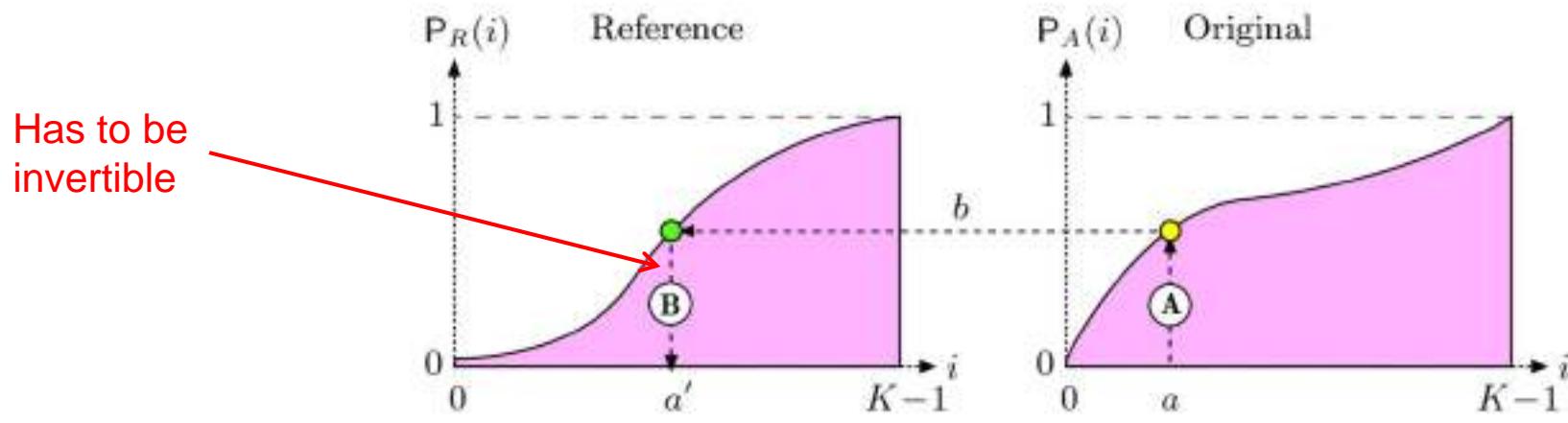
**Q10 Answer:** Add scribbles as hard constraints or high-weight unary terms forcing pixels to take user-specified labels.

**Explanation:** Guidance is integrated so optimization respects user input.



# Histogram Matching

- Prior method needed reference distribution to be invertible



$$f_{hs}(a) = a' = P_R^{-1}(P_A(a))$$

- What if reference histogram is not invertible?
- For example not invertible if histogram has some intensities that occur with probability 0? i.e.  $p(k) = 0$
- Use different method called **histogram matching**

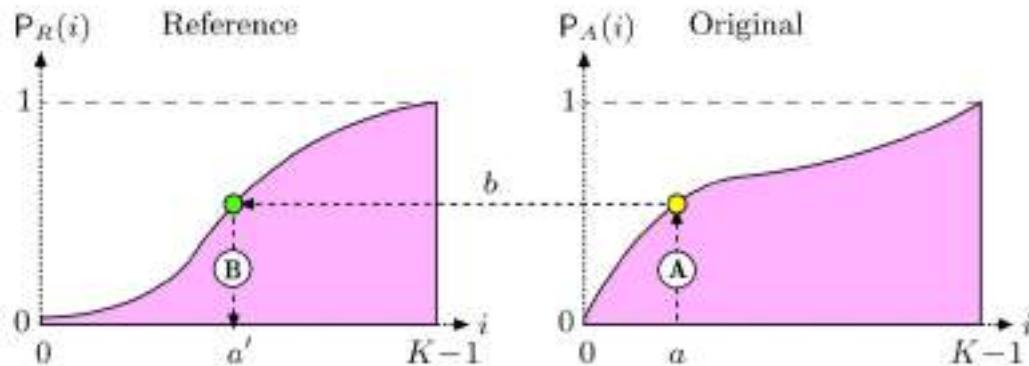


# Histogram Matching

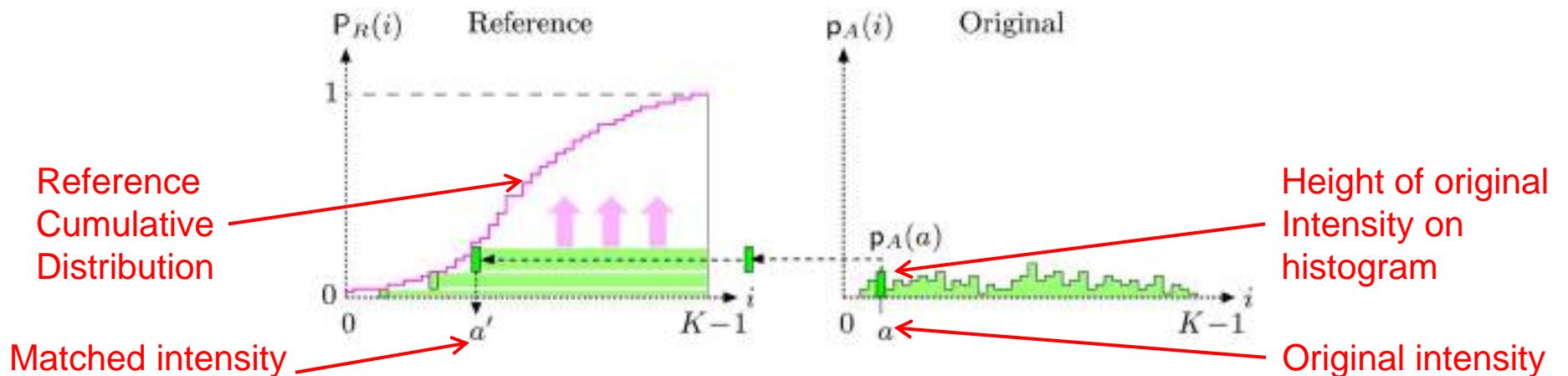
- Given two images  $I_A$  and  $I_B$ , we want to make their intensity profiles look as similar as possible.
- How?
  - “Match” their cumulative histograms  $H_A$  and  $H_B$
- Works well for images with similar content.
- Looks bad for images with different content.



# Adjusting to a Given Histogram

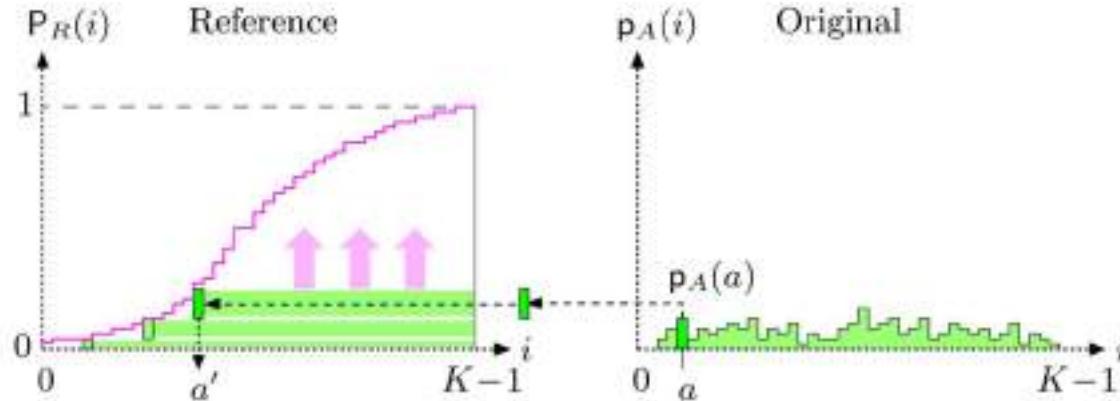


$$f_{hs}(a) = a' = \min \{ j \mid (0 \leq j < K) \wedge (P_A(a) \leq P_R(j)) \}$$





# Adjusting to a Given Histogram



1: MATCHHISTOGRAMS( $\mathbf{h}_A, \mathbf{h}_R$ )

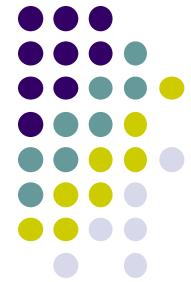
$\mathbf{h}_A$ : histogram of the target image

$\mathbf{h}_R$ : reference histogram (of same size as  $\mathbf{h}_A$ )

```

2: Let  $K \leftarrow \text{Size}(\mathbf{h}_A)$ 
3: Let  $\mathbf{P}_A \leftarrow \text{CDF}(\mathbf{h}_A)$                                  $\triangleright$  cdf for  $\mathbf{h}_A$  (Alg. 5.1)
4: Let  $\mathbf{P}_R \leftarrow \text{CDF}(\mathbf{h}_R)$                                  $\triangleright$  cdf for  $\mathbf{h}_R$  (Alg. 5.1)
5: Create a table  $f_{\text{hs}}[ ]$  of size  $K$                        $\triangleright$  pixel mapping function  $f_{\text{hs}}$ 
6: for  $a \leftarrow 0 \dots (K-1)$  do
7:    $j \leftarrow K-1$ 
8:   repeat
9:      $f_{\text{hs}}[a] \leftarrow j$ 
10:     $j \leftarrow j - 1$ 
11:   while ( $j \geq 0$ )  $\wedge (\mathbf{P}_A(a) \leq \mathbf{P}_R(j))$ 
12: return  $f_{\text{hs}}$ .
```

# Adjusting to a Given Histogram

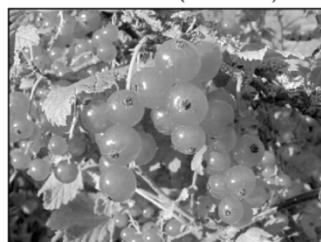


### Original Image



(a)  $I_A$

Gaussian ( $\sigma = 50$ )



(b)  $I_{G50}$

Gaussian ( $\sigma = 100$ )



(c)  $I_{G100}$

## Reference Histogram



$$\mathbf{h}_R(i)$$

## Cumulative Reference Histogram

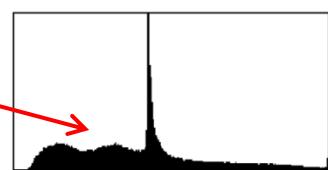


$$\mathsf{H}_R(i)$$

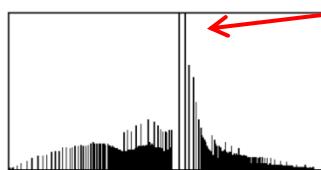
(d)

(e)

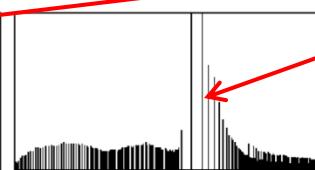
# original histogram



(f)  $h_A$

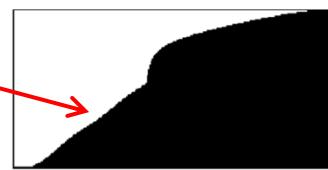


(g)  $h_{G50}$



(h)  $h_{G10e}$

## CDF of original histogram



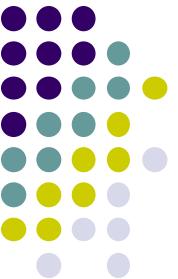
(i)  $\mathsf{H}_A$



(j)  $H_{G50}$

## Original histogram after matching

- CDF of original
- histogram after matching



# Adjusting to a Given Histogram

Target Image



(a)  $I_A$

Reference Image

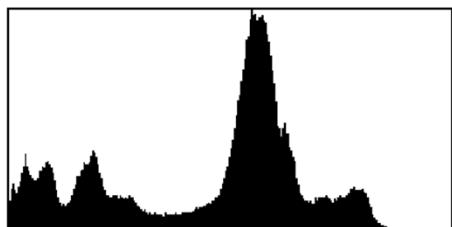


(b)  $I_R$

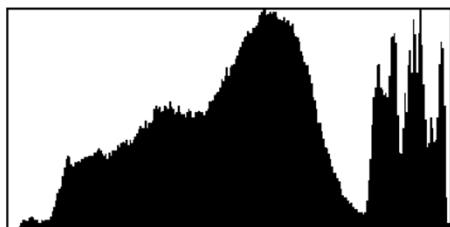
Modified Image



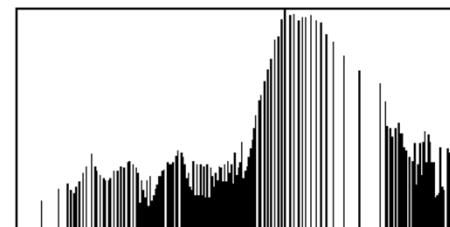
(c)  $I_{A'}$



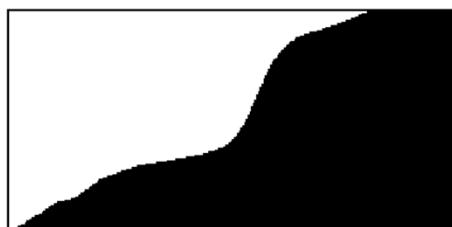
(d)  $h_A$



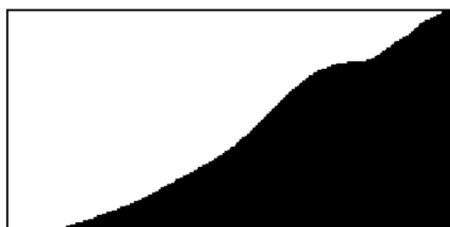
(e)  $h_R$



(f)  $h_{A'}$



(g)  $H_A$

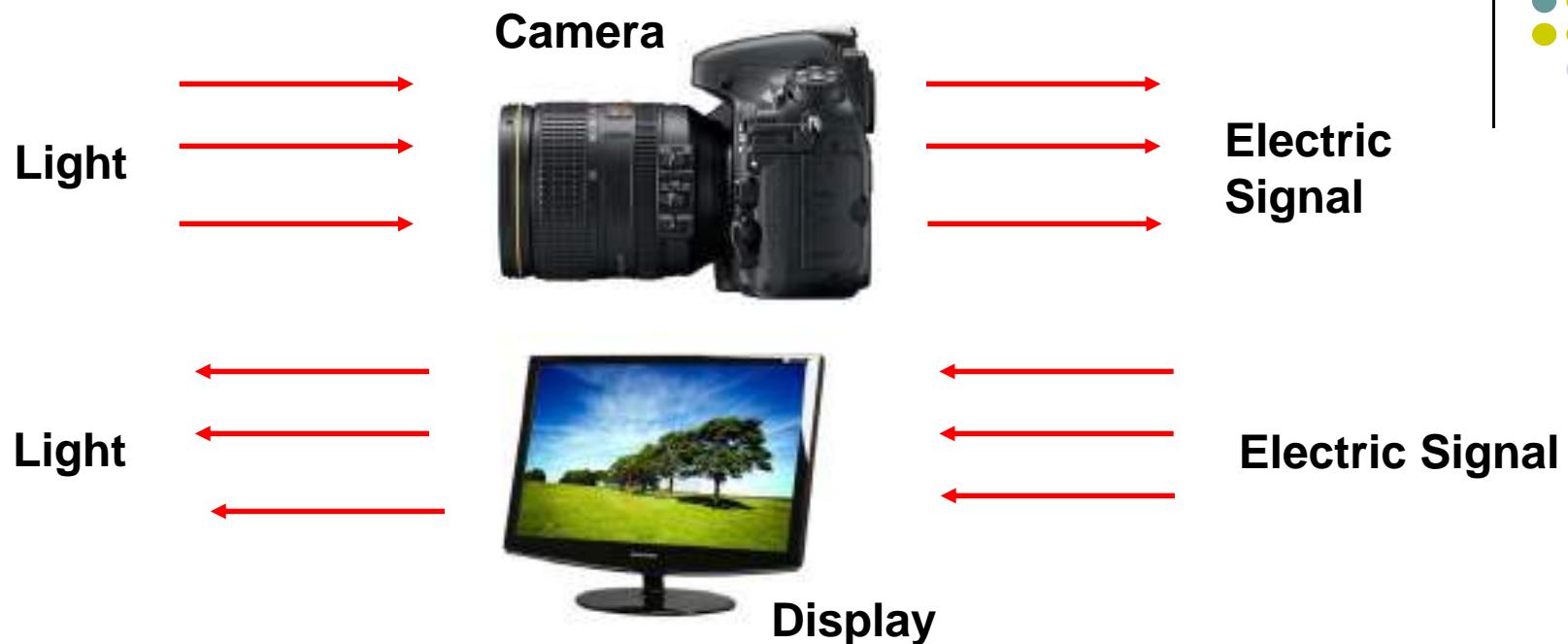
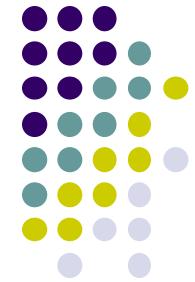


(h)  $H_R$



(i)  $H_{A'}$

# Gamma Correction



- Different camera sensors
  - Have different responses to light intensity
  - Produce different electrical signals for same input
- How do we ensure there is consistency in:
  - a) Images recorded by different cameras for given light input
  - b) Light emitted by different display devices for same image?



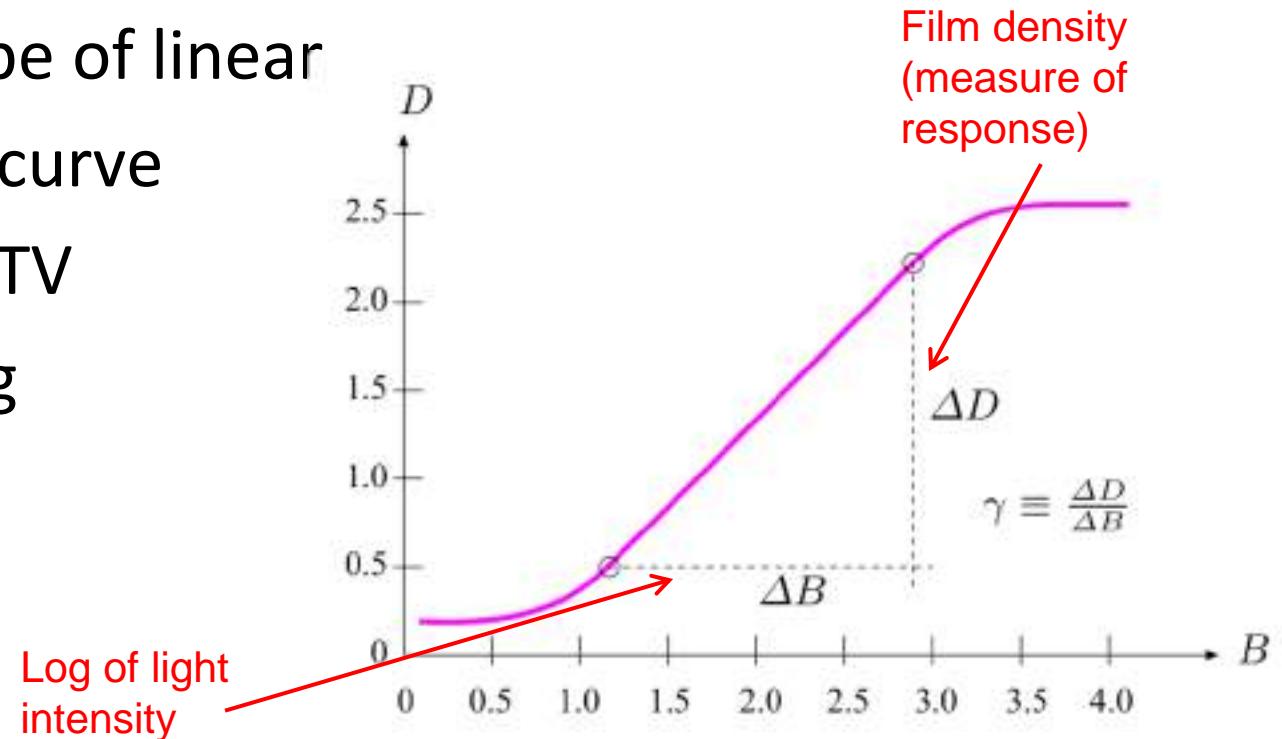
# Gamma Correction

- What is the relation between:
  - **Camera:** Light on sensor vs. “**intensity**” of corresponding pixel
  - **Display:** Pixel intensity vs. light from that pixel
- Relation between pixel value and corresponding physical quantity is usually complex, nonlinear
- An approximation ?

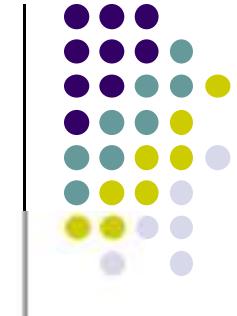


# What is Gamma?

- Originates from analog photography
- **Exposure function:** relationship between:
  - logarithmic light intensity vs. resulting film density.
- **Gamma:** slope of linear range of the curve
- The same in TV broadcasting



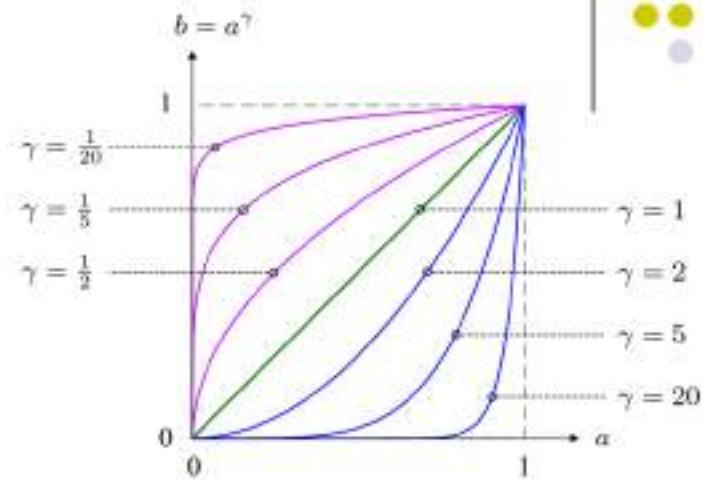
# What is Gamma?



- **Gamma function:** a good approximation of exposure curve
- Inverse of a Gamma function is another gamma function with

$$\bar{\gamma} = 1/\gamma$$

- Gamma of CRT and LCD monitors:
- 1.8-2.8 (typically 2.4)



$$b = f_\gamma(a) = a^\gamma \quad \text{for } a \in \mathbb{R}, \gamma > 0$$

$$a = f_\gamma^{-1}(b) = b^{1/\gamma}$$

$$f_\gamma^{-1}(b) = f_{\bar{\gamma}}(b)$$

$$s = B^{\gamma_c}$$

Output signal  
Raised by gamma

$$b = s^{1/\gamma_c} = (B^{\gamma_c})^{1/\gamma_c} = B^{(\gamma_c \frac{1}{\gamma_c})} = B^1$$

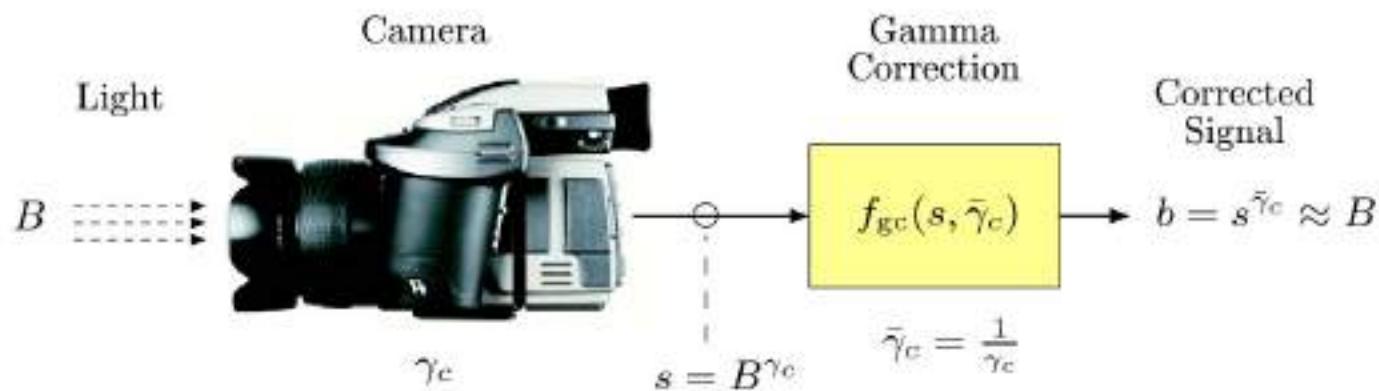
Correct output signal  
By dividing by 1/ gamma  
(called Gamma correction)

$$\bar{\gamma} = 1/\gamma$$



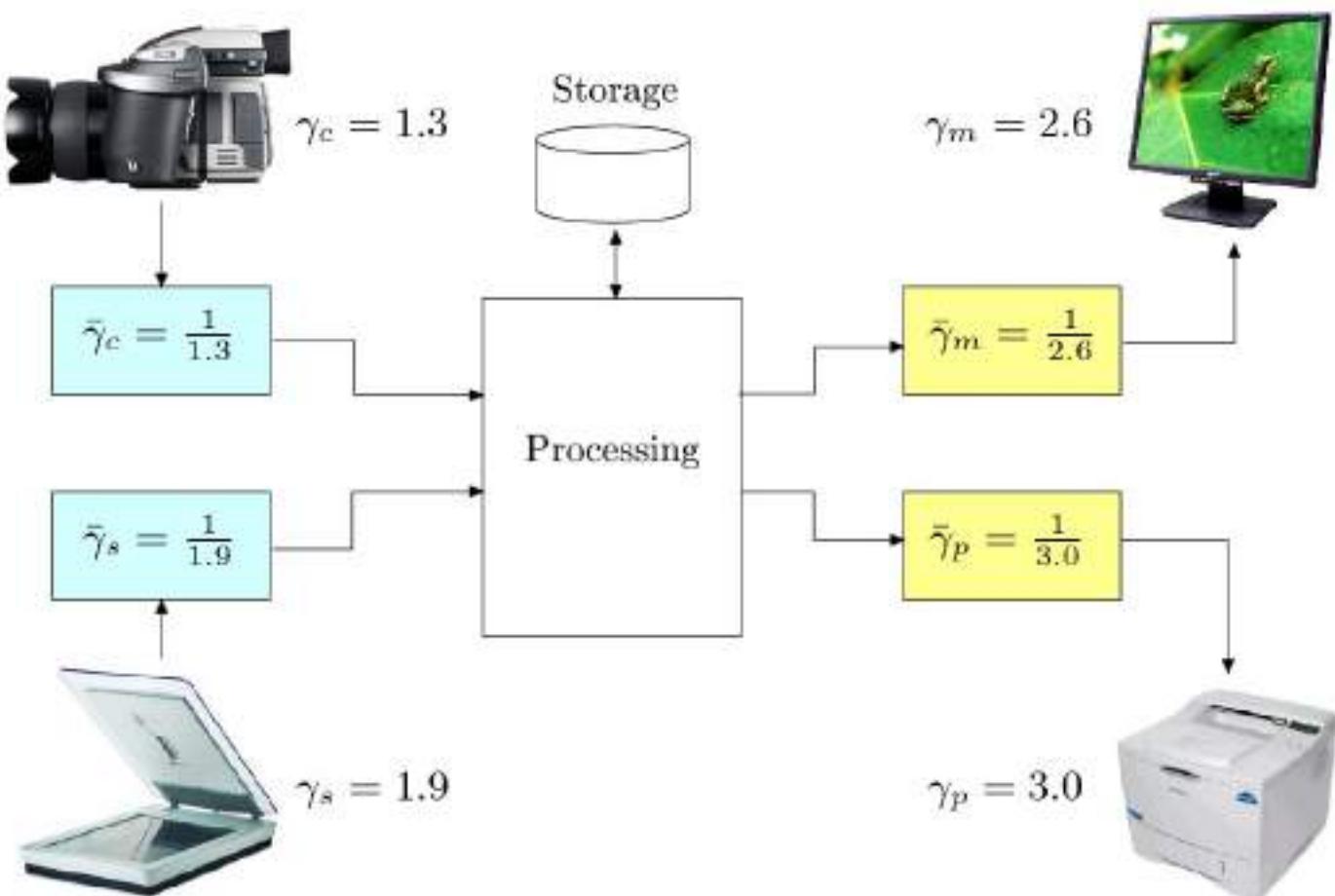
# Gamma Correction

- Obtain a measurement **b** proportional to original light intensity **B** by applying inverse gamma function
- Gamma correction is important to achieve a device independent representation





# Gamma Correction





# Gamma Correction Code

```
1  public void run(ImageProcessor ip) {  
2      // works for 8-bit images only  
3      int K = 256;  
4      int aMax = K - 1;  
5      double GAMMA = 2.8;  
6  
7      // create a lookup table for the mapping function  
8      int[] Fgc = new int[K];  
9  
10     for (int a = 0; a < K; a++) {  
11         double aa = (double) a / aMax;    // scale to [0, 1]  
12         double bb = Math.pow(aa,GAMMA); // gamma function  
13         // scale back to [0, 255]:  
14         int b = (int) Math.round(bb * aMax);  
15         Fgc[a] = b;                  ← Compute corrected intensity and  
16     }                                store in lookup table fgc  
17  
18     ip.applyTable(Fgc);   // modify the image ip  
19 }
```



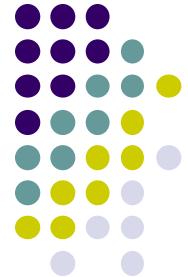
# Point Operations in ImageJ

void abs()	$I'(u, v) \leftarrow  I(u, v) $
void add(int <i>p</i> )	$I'(u, v) \leftarrow I(u, v) + p$
void gamma(double <i>g</i> )	$I'(u, v) \leftarrow (I(u, v)/255)^g \cdot 255$
void invert(int <i>p</i> )	$I'(u, v) \leftarrow 255 - I(u, v)$
void log()	$I'(u, v) \leftarrow \log_{10}(I(u, v))$
void max(double <i>s</i> )	$I'(u, v) \leftarrow \max(I(u, v), s)$
void min(double <i>s</i> )	$I'(u, v) \leftarrow \min(I(u, v), s)$
void multiply(double <i>s</i> )	$I'(u, v) \leftarrow \text{round}(I(u, v) \cdot s)$
void sqr()	$I'(u, v) \leftarrow I(u, v)^2$
void sqrt()	$I'(u, v) \leftarrow \sqrt{I(u, v)}$



# ImageJ Operations involving 2 images

ADD	$ip1 \leftarrow ip1 + ip2$
AVERAGE	$ip1 \leftarrow (ip1 + ip2) / 2$
DIFFERENCE	$ip1 \leftarrow  ip1 - ip2 $
DIVIDE	$ip1 \leftarrow ip1 / ip2$
MAX	$ip1 \leftarrow \max(ip1, ip2)$
MIN	$ip1 \leftarrow \min(ip1, ip2)$
MULTIPLY	$ip1 \leftarrow ip1 \cdot ip2$
SUBTRACT	$ip1 \leftarrow ip1 - ip2$



# Example: Alpha Blending

$$I'(u, v) \leftarrow \alpha \cdot I_{\text{BG}}(u, v) + (1 - \alpha) \cdot I_{\text{FG}}(u, v)$$



$I_{\text{BG}}$



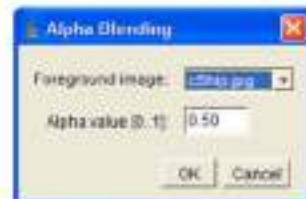
$\alpha = 0.25$



$I_{\text{FG}}$



$\alpha = 0.50$



GenericDialog

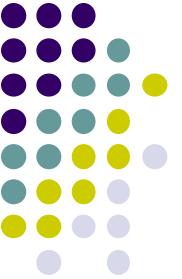


$\alpha = 0.75$



# Alpha Blending PlugIn

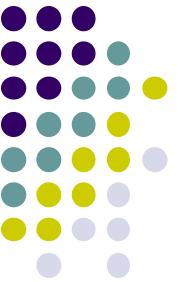
```
1 import ij.IJ;
2 import ij.ImagePlus;
3 import ij.WindowManager;
4 import ij.gui.GenericDialog;
5 import ij.plugin.filter.PlugInFilter;
6 import ij.process.*;
7
8 public class Alpha_Blending implements PlugInFilter {
9
10    static double alpha = 0.5; // transparency of foreground image
11    ImagePlus fgIm = null;    // foreground image
12
13    public int setup(String arg, ImagePlus imp) {
14        return DOES_8G;
15    }
16
17    public void run(ImageProcessor bgIp) { // background image
18        if(runDialog()) {
19            ImageProcessor fgIp
20                = fgIm.getProcessor().convertToByte(false);
21            fgIp = fgIp.duplicate();
22            fgIp.multiply(1-alpha);
23            bgIp.multiply(alpha);
24            bgIp.copyBits(fgIp, 0, 0, Blitter.ADD);
25        }
26    }
27
28    // continued ...
```



# What Is Image Enhancement?

Image enhancement makes images more useful by:

- Highlighting interesting detail in images
- Removing noise from images
- Making images more visually appealing



# Image Enhancement Examples

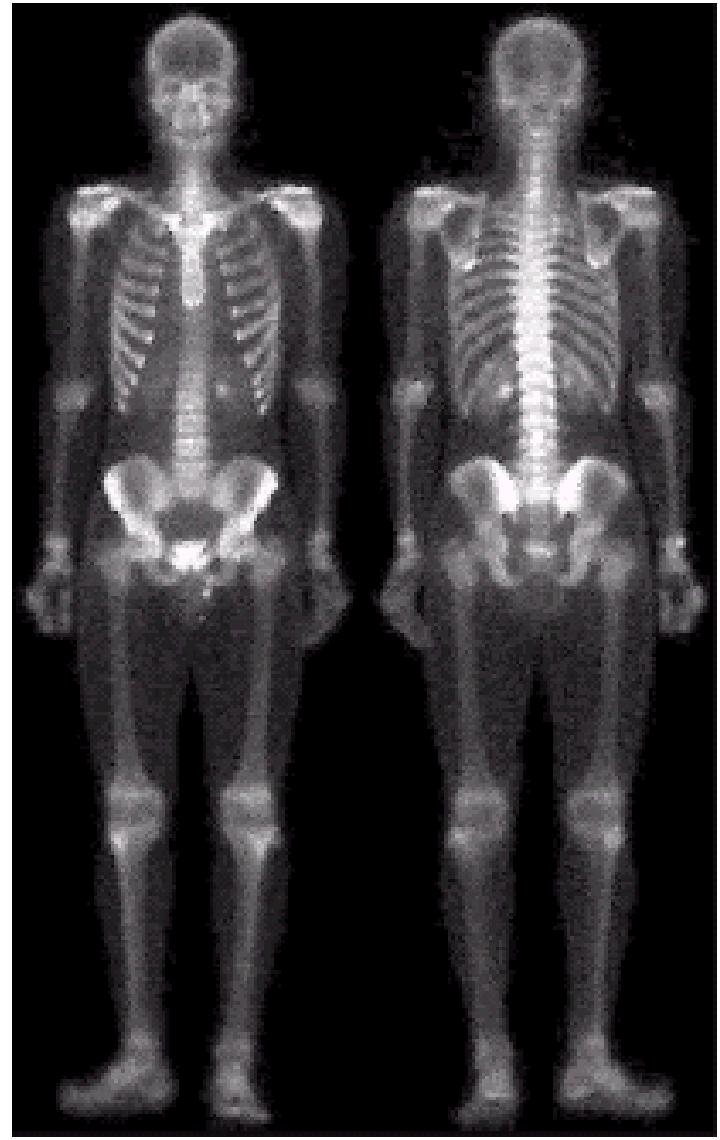
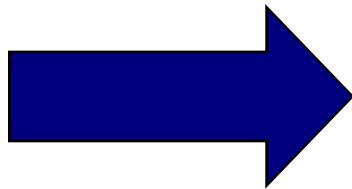
Images taken from Gonzalez & Woods, Digital Image Processing (2002)





## Image Enhancement Examples (cont...)

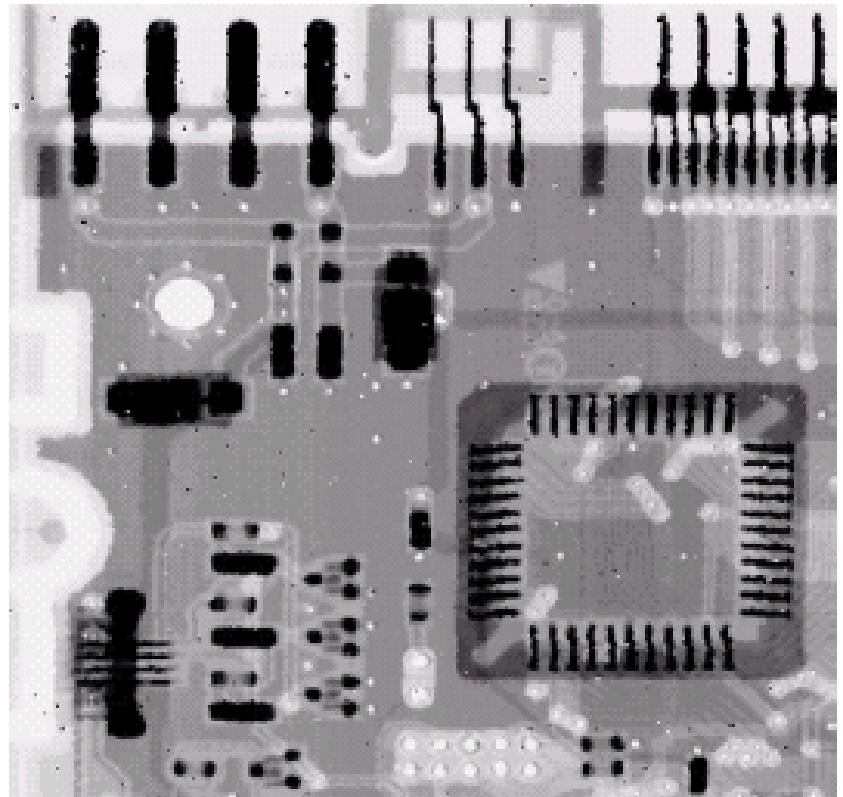
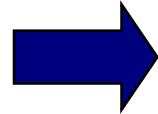
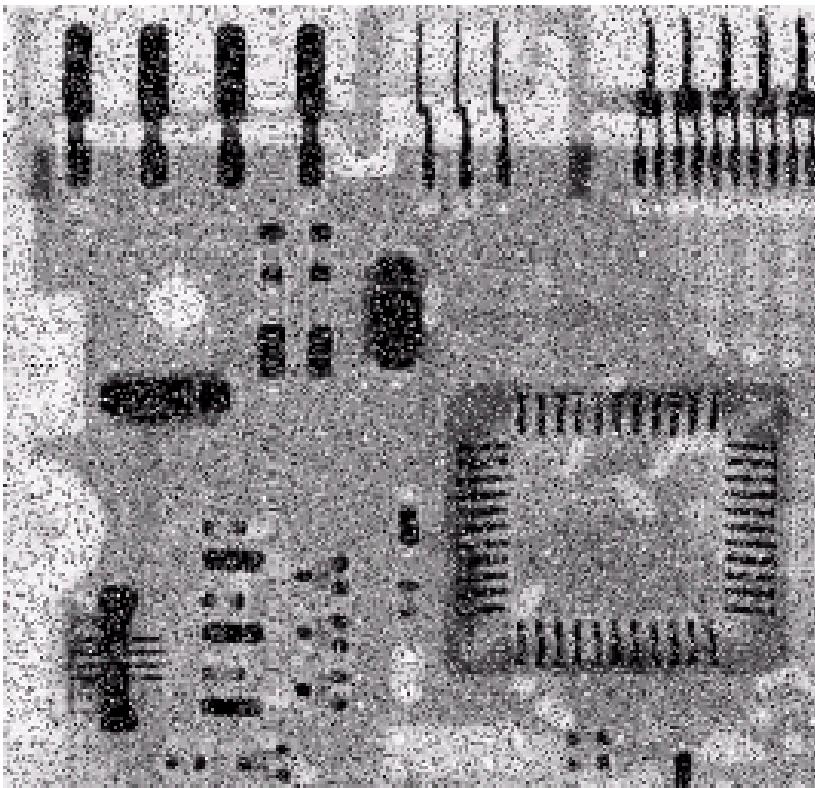
Images taken from Gonzalez & Woods, Digital Image Processing (2002)





## Image Enhancement Examples (cont...)

Images taken from Gonzalez & Woods, Digital Image Processing (2002)





## Image Enhancement Examples (cont...)

Images taken from Gonzalez & Woods, Digital Image Processing (2002)





# Spatial & Frequency Domains

There are two broad categories of image enhancement techniques

- Spatial domain techniques
  - Direct manipulation of image pixels (intensity values)
- Frequency domain techniques
  - Manipulation of Fourier transform or wavelet transform of an image

First spatial domain techniques

Later: frequency domain techniques



# What is a Filter?

- Capabilities of point operations are limited
- **Filters:** combine **pixel's value + values of neighbors**
- **E.g blurring:** Compute average intensity of block of pixels



- Combining multiple pixels needed for certain operations:
  - Blurring, Smoothing
  - Sharpening



# What Point Operations Can't Do

- Example: sharpening

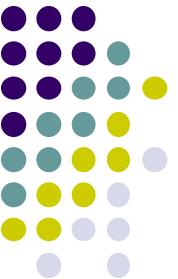




## What Point Operations Can't Do

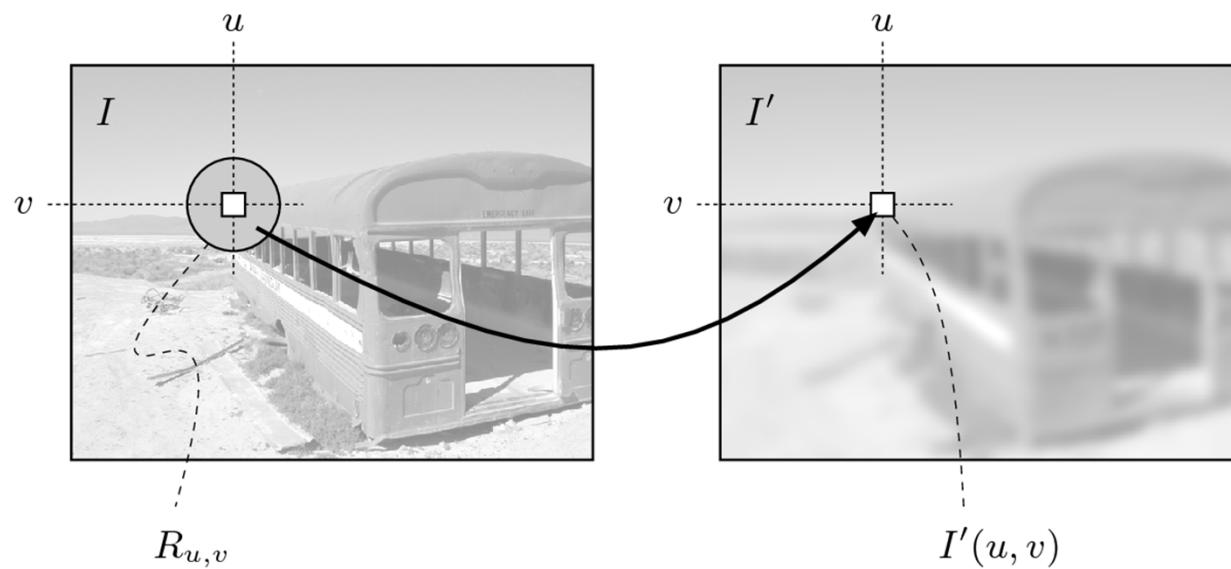
- Other cool artistic patterns by combining pixels



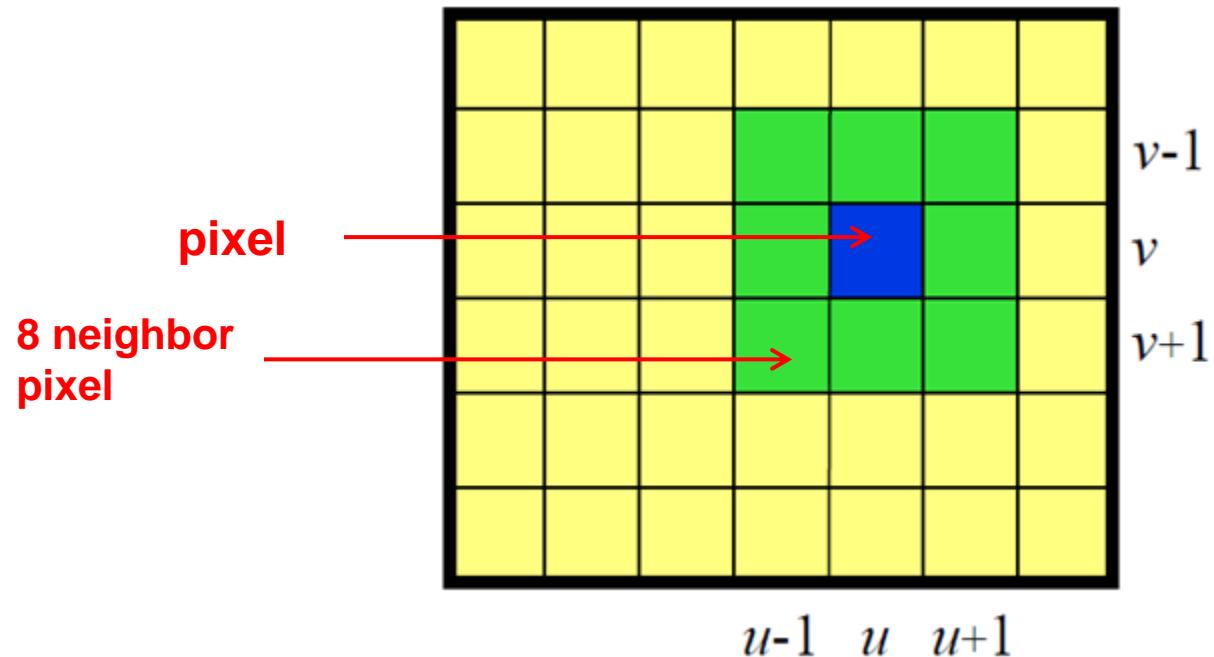
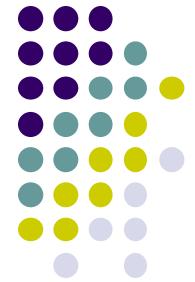


# Definition: Spatial Filter

- An image operation that combines each pixel's intensity  $I(u, v)$  with that of neighboring pixels
- E.g: average/weighted average of group of pixels



# Example: Average (Mean) of 3x3 Neighborhood



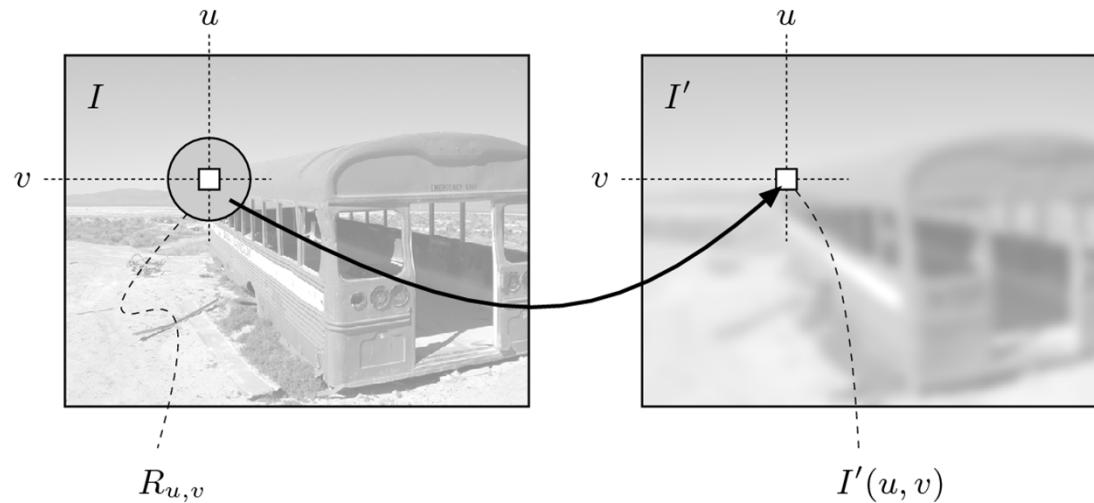
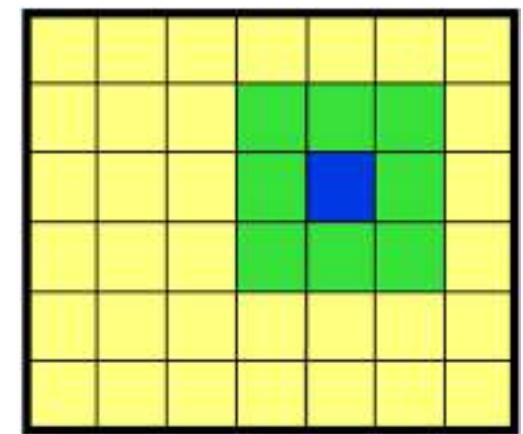
Blurring: Replace each pixel with AVERAGE Intensity of pixel + neighbors



# Smoothing an Image by Averaging

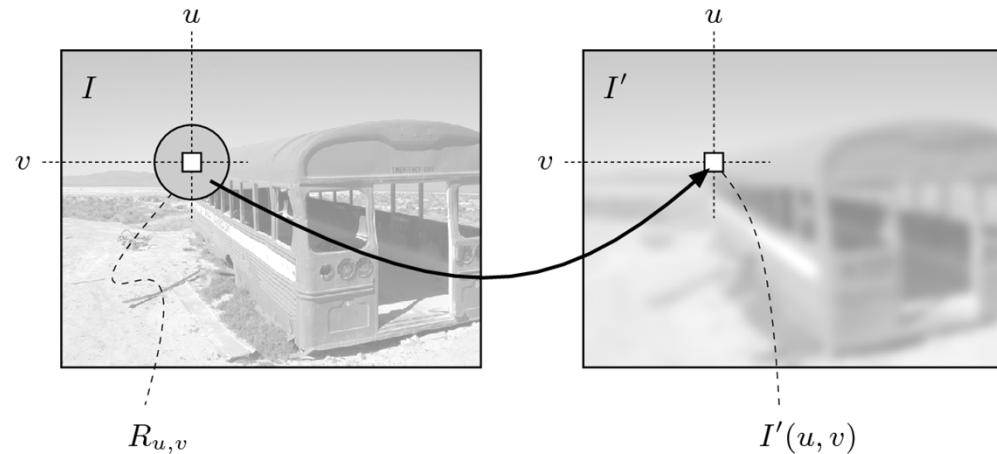
- Replace each pixel by average of pixel + neighbors
- For 3x3 neighborhood:

$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$





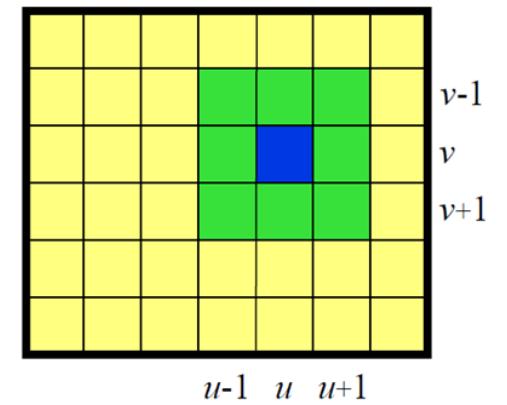
# Smoothing an Image by Averaging



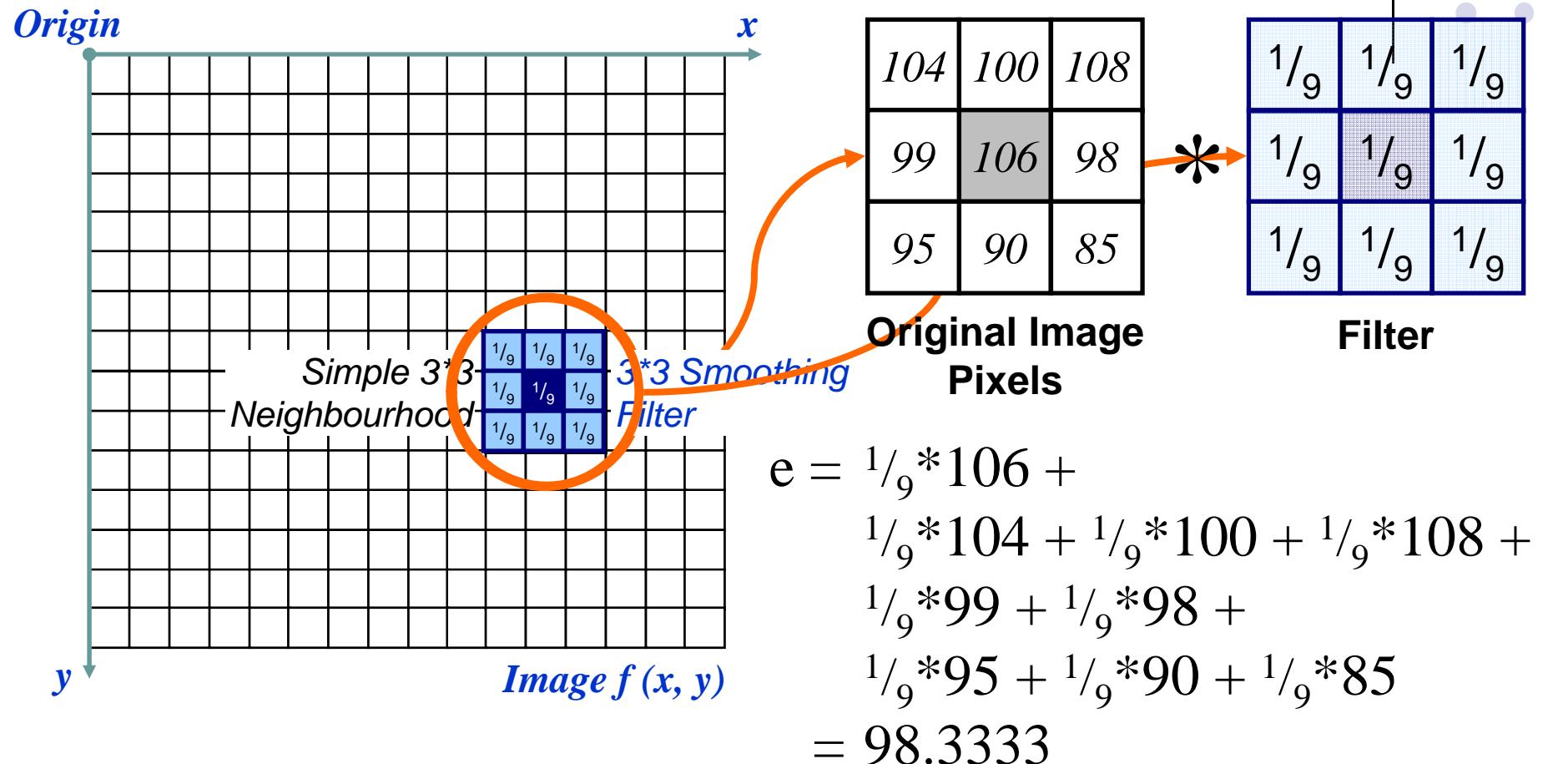
$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot [ I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + I(u-1, v) + I(u, v) + I(u+1, v) + I(u-1, v+1) + I(u, v+1) + I(u+1, v+1) ]$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^1 \sum_{i=-1}^1 I(u+i, v+j)$$

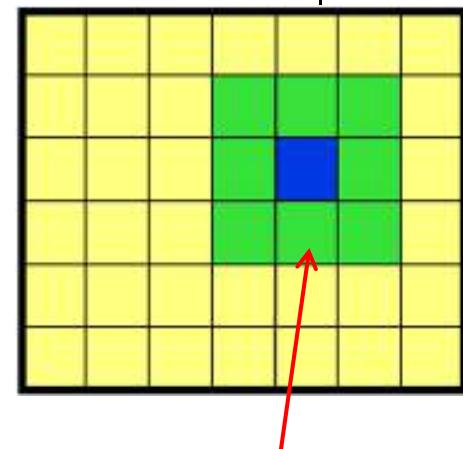
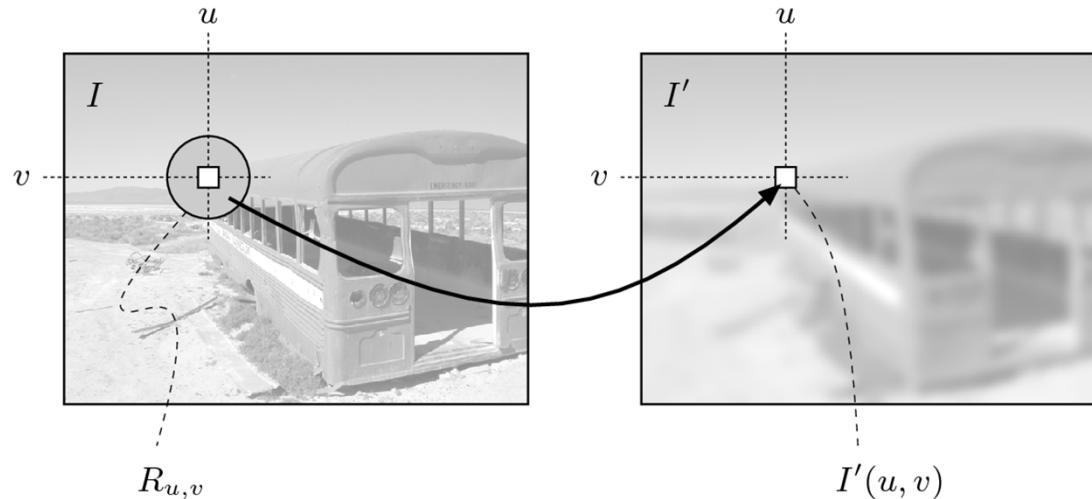
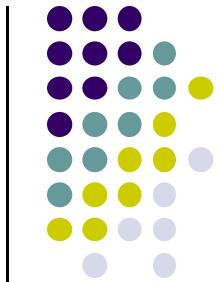


# Example: Smoothing Spatial Filtering



The above is repeated for every pixel in the original image to generate the smoothed image

# Smoothing an Image by Averaging



Previous example:  
Filter size: 3x3

- Many possible filter parameters (size, weights, function, etc)
- **Filter size (size of neighborhood):** 3x3, 5x5, 7x7, ..., 21x21,..
- **Filter shape:** not necessarily square. Can be rectangle, circle, etc
- **Filter weights:** May apply unequal weighting to different pixels
- **Filters function:** can be linear (a weighted summation) or nonlinear



# The Filter Matrix

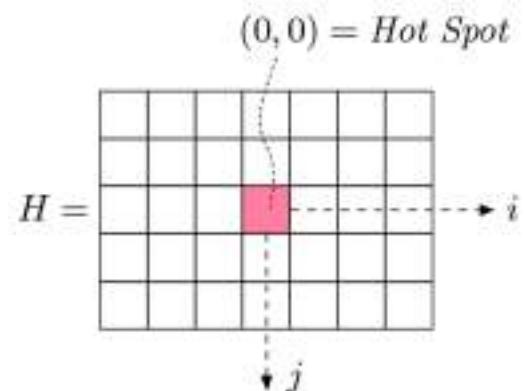
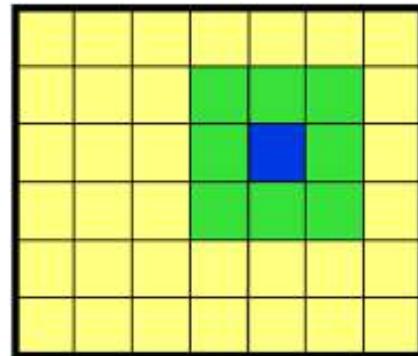
$$I'(u, v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$

$$I'(u, v) \leftarrow \frac{1}{9} \cdot [ I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + \\ I(u-1, v) + I(u, v) + I(u+1, v) + \\ I(u-1, v+1) + I(u, v+1) + I(u+1, v+1) ]$$

$$H(i, j) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filter operation can be expressed as a matrix  
Example: averaging filter

Filter matrix also called filter mask  $H(i,j)$





## Example: What does this Filter Do?



$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$



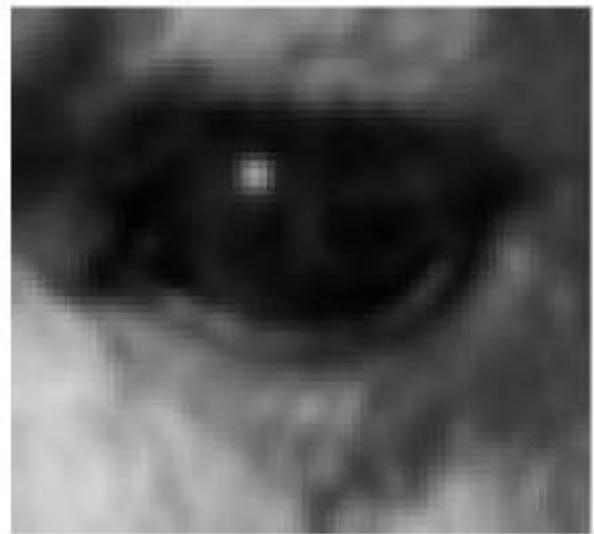
Identity function (leaves image alone)



# What Does this Filter Do?



$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$



Mean (averages neighborhood)



## Mean Filters: Effect of Filter Size



Original



$7 \times 7$



$15 \times 15$



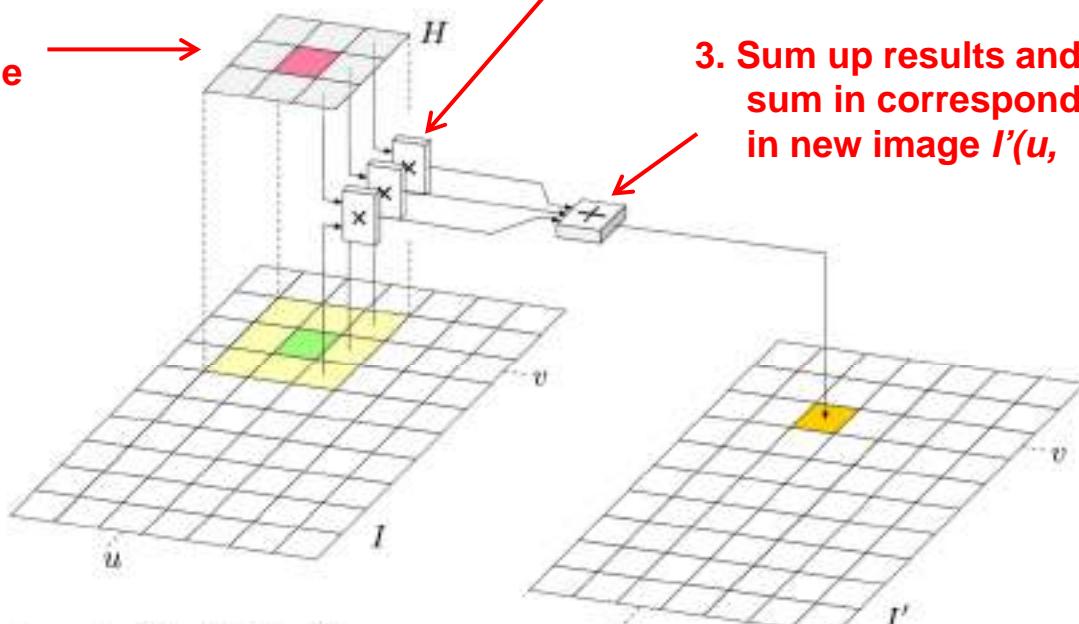
$41 \times 41$



# Applying Linear Filters: Convolution

For each image position  $I(u, v)$ :

1. Move filter matrix  $H$  over image such that  $H(0,0)$  coincides with current image position  $(u, v)$



2. Multiply all filter coefficients  $H(i, j)$  with corresponding pixel  $I(u + i, v + j)$

3. Sum up results and store sum in corresponding position in new image  $I'(u, v)$

Stated formally:

$$I'(u, v) \leftarrow \sum_{(i,j) \in R_H} I(u + i, v + j) \cdot H(i, j)$$

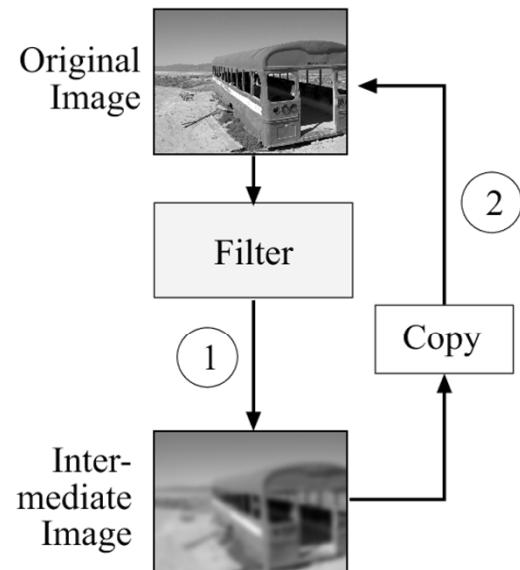
$R_H$  is set of all pixels Covered by filter.  
For 3x3 filter, this is:

$$I'(u, v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot H(i, j)$$



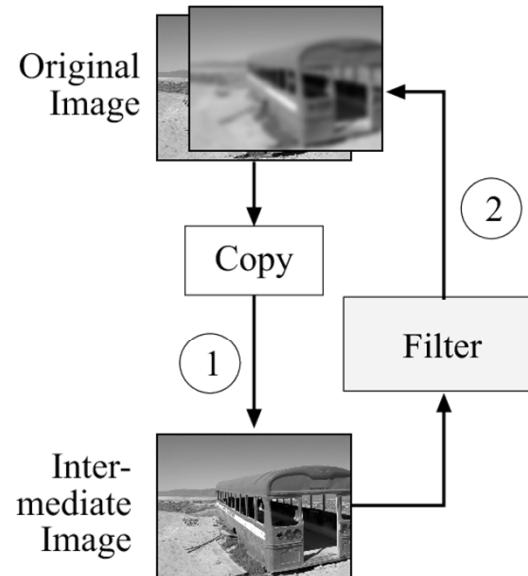
# Computing Filter Operation

- Filter matrix  $H$  moves over each pixel in original image  $I$  to compute corresponding pixel in new image  $I'$
- Cannot overwrite new pixel value in original image  $I$  Why?



Version A

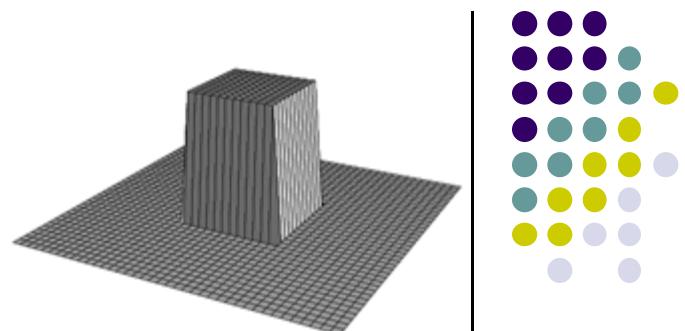
Store results  $I'$  in intermediate image, then copy back to replace  $I$



Version B

Copy original image  $I$  to intermediate image, use it as source, then store results  $I'$  to replace original image

# Simple 3x3 Averaging Filter (“Box” Filter)



```
1 import ij.*;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.*;
4
5 public class Filter_Average3x3 implements PlugInFilter {
6     ...
7     public void run(ImageProcessor orig) {
8         int w = orig.getWidth();
9         int h = orig.getHeight();
10        ImageProcessor copy = orig.duplicate(); ←
11
12        for (int v = 1; v <= h-2; v++) {
13            for (int u = 1; u <= w-2; u++) { ←
14                //compute filter result for position (u,v)
15                int sum = 0;
16                for (int j = -1; j <= 1; j++) {
17                    for (int i = -1; i <= 1; i++) {
18                        int p = copy.getPixel(u+i, v+j);
19                        sum = sum + p;
20                    }
21                }
22                int q = (int) Math.round(sum/9.0);
23                orig.putPixel(u, v, q); ←
24            }
25        }
26    }
27 } // end of class Filter_Average3x3
```

No explicit filter matrix since all coefficients are the same (1/9)

No clamping required

Make copy of original image to use as source

Loop over all pixels in image

Filter computation by adding current pixel's neighbors

Store result back in original image



# Weighted Smoothing Filters

- More effective smoothing filters can be generated by allowing different pixels in the neighbourhood different weights in the averaging function
  - Pixels closer to central pixel more important
  - Often referred to as a *weighted averaging*

$1/_{16}$	$2/_{16}$	$1/_{16}$
$2/_{16}$	$4/_{16}$	$2/_{16}$
$1/_{16}$	$2/_{16}$	$1/_{16}$

Weighted  
averaging filter



# Another Smoothing Filter

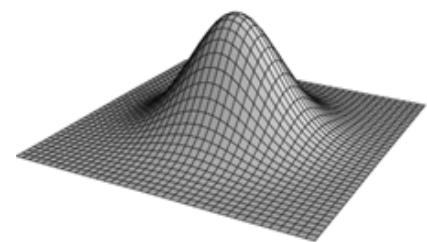
```

1  public void run(ImageProcessor orig) {
2      int w = orig.getWidth();
3      int h = orig.getHeight();
4      // 3 × 3 filter matrix
5      double[][] filter = {
6          {0.075, 0.125, 0.075},
7          {0.125, 0.200, 0.125},
8          {0.075, 0.125, 0.075}
9      };
10     ImageProcessor copy = orig.duplicate();
11
12     for (int v = 1; v <= h-2; v++) {
13         for (int u = 1; u <= w-2; u++) {
14             // compute filter result for position (u, v)
15             double sum = 0;
16             for (int j = -1; j <= 1; j++) {
17                 for (int i = -1; i <= 1; i++) {
18                     int p = copy.getPixel(u+i, v+j);
19                     // get the corresponding filter coefficient:
20                     double c = filter[j+1][i+1];
21                     sum = sum + c * p;
22                 }
23             }
24             int q = (int) Math.round(sum);
25             orig.putPixel(u, v, q);
26         }
27     }
28 }
```

Use real filter matrix with coefficients  
Apply bell-shaped function  $H(i,j)$

$$H(i,j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \textbf{0.2} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

Bell-shaped function  $H(i,j)$ ?  
- More weight applied to center



Apply filter

Store result back in original



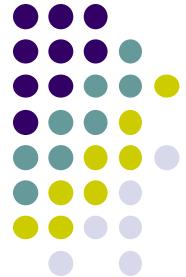
# Integer Coefficients

- Instead of floating point coefficients, more efficient, simpler to use:

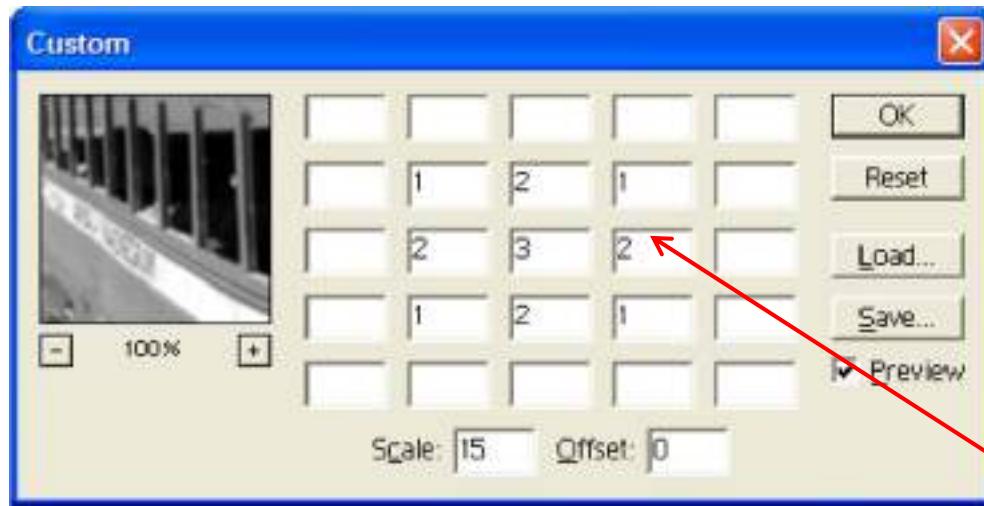
scaling factor + integer coefficients

$$H(i, j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} 3 & 5 & 3 \\ 5 & \underline{8} & 5 \\ 3 & 5 & 3 \end{bmatrix}$$

$$H(i, j) = s \cdot H'(i, j)$$



# Example: 5x5 Filter in Adobe Photoshop



Integer filter  
coefficients

$$I'(u, v) \leftarrow \text{Offset} + \frac{1}{\text{Scale}} \sum_{j=-2}^{j=2} \sum_{i=-2}^{i=2} I(u+i, v+j) \cdot H(i, j)$$

If resulting pixel value is  
negative, offset shifts it  
into visible range

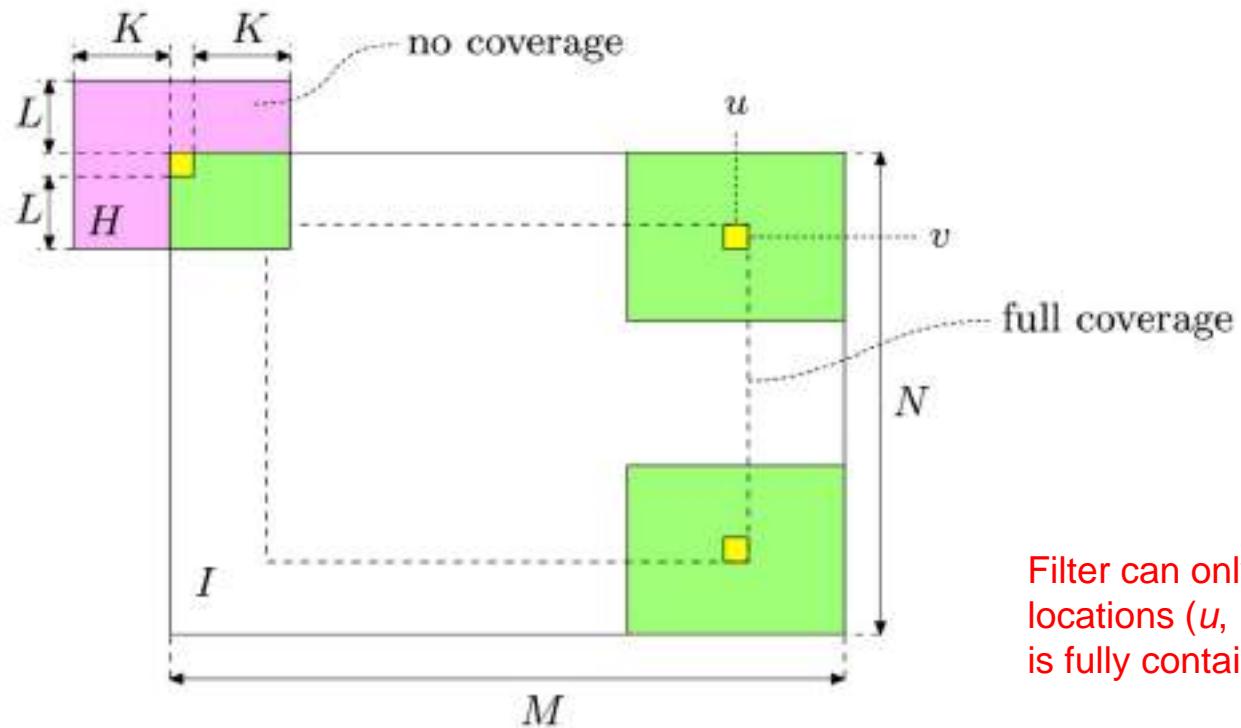
Scaling factor for  
coefficients



# Computation Range

- For a filter of size  $(2K+1) \times (2L+1)$ , if image size is  $M \times N$ , filter is computed over the range:

$$K \leq u' \leq (M-K-1) \quad \text{and} \quad L \leq v' \leq (N-L-1)$$



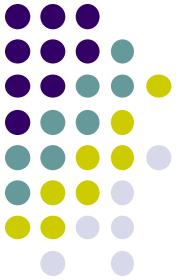


# Filter of Arbitrary Size

```
1  public void run(ImageProcessor orig) {  
2      int M = orig.getWidth();  
3      int N = orig.getHeight();  
4  
5      // filter matrix of size  $(2K + 1) \times (2L + 1)$   
6      int[][] filter = {  
7          {0,0,1,1,1,0,0},  
8          {0,1,1,1,1,1,0},  
9          {1,1,1,1,1,1,1},  
10         {0,1,1,1,1,1,0},  
11         {0,0,1,1,1,0,0}  
12     };  
13     double s = 1.0/23; // sum of filter coefficients is 23  
14  
15     int K = filter[0].length/2;  
16     int L = filter.length/2;  
17  
18     ImageProcessor copy = orig.duplicate();  
19  
20     for (int v = L; v <= N-L-1; v++) {  
21         for (int u = K; u <= M-K-1; u++) {  
22             // compute filter result for position (u,v)  
23             int sum = 0;  
24             for (int j = -L; j <= L; j++) {  
25                 for (int i = -K; i <= K; i++) {  
26                     int p = copy.getPixel(u+i, v+j);  
27                     int c = filter[j+L][i+K];  
28                     sum = sum + c * p;  
29                 }  
30             }  
31             int q = (int) Math.round(s * sum);  
32             if (q < 0) q = 0; ←  
33             if (q > 255) q = 255;  
34             orig.putPixel(u, v, q);  
35         }  
36     }  
37 }
```

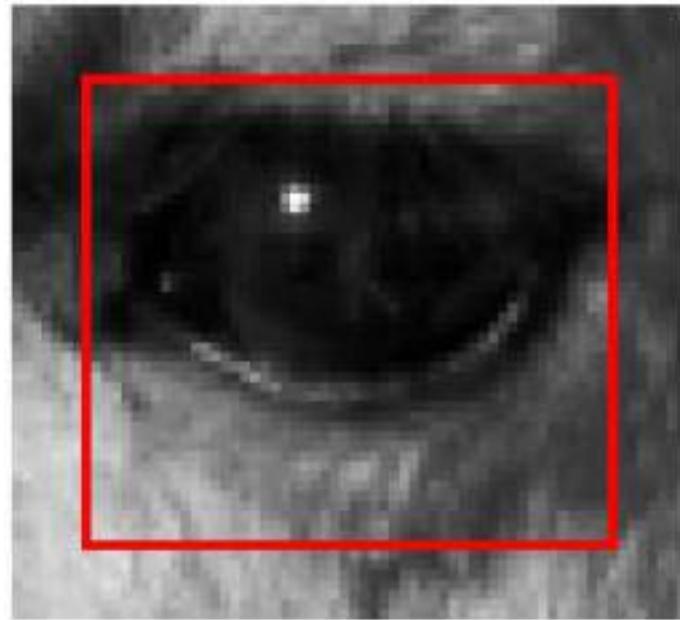
Declare filter

Clamp result as a precaution



# What to do at image boundaries?

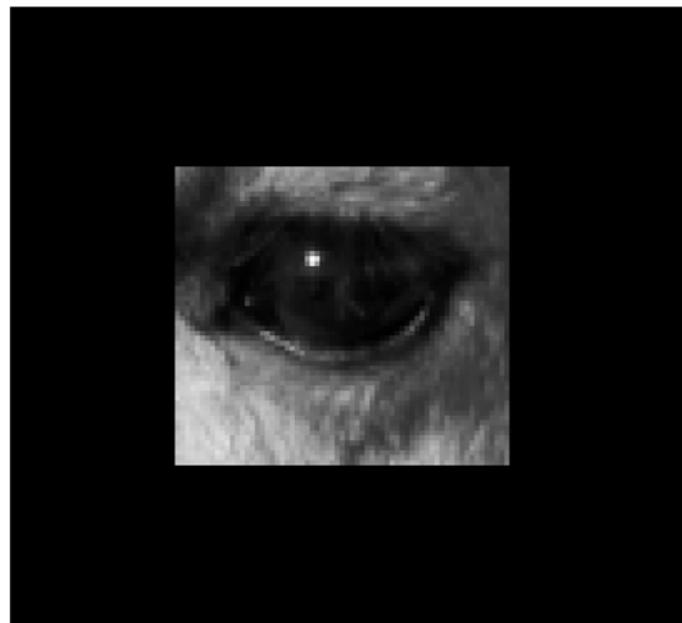
a) Crop





# What to do at image boundaries?

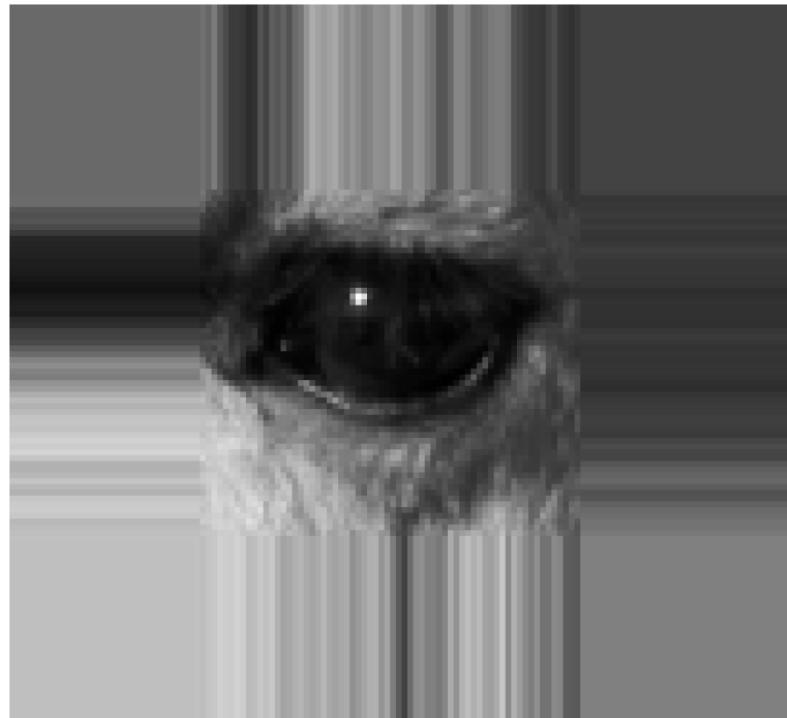
- a) Crop
- b) Pad

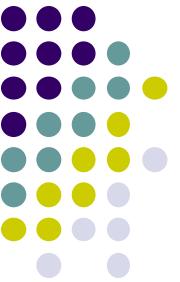




# What to do at image boundaries?

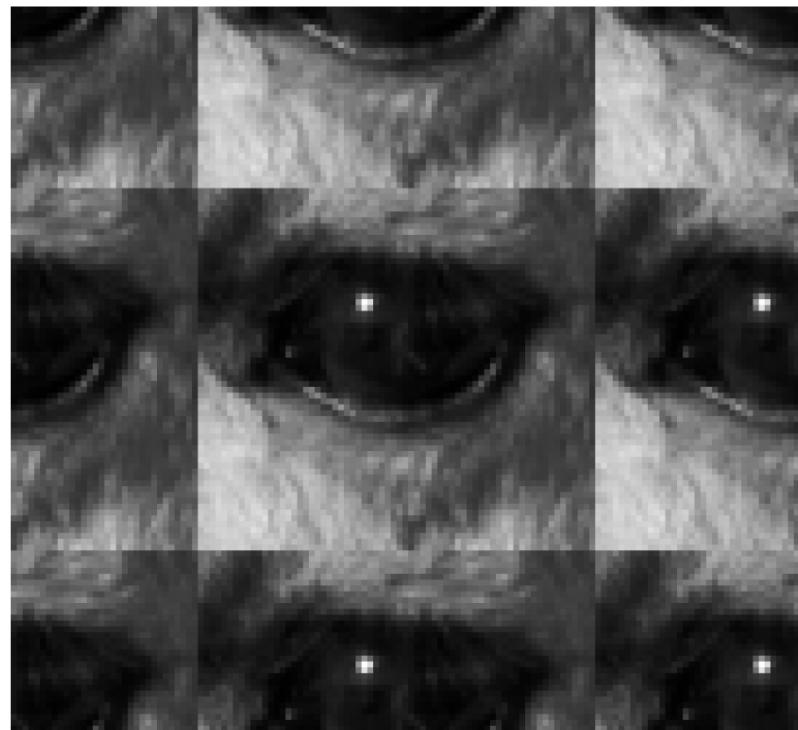
- a) Crop
- b) Pad
- c) Extend





# What to do at image boundaries?

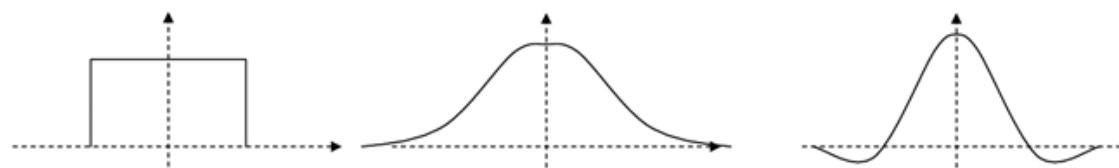
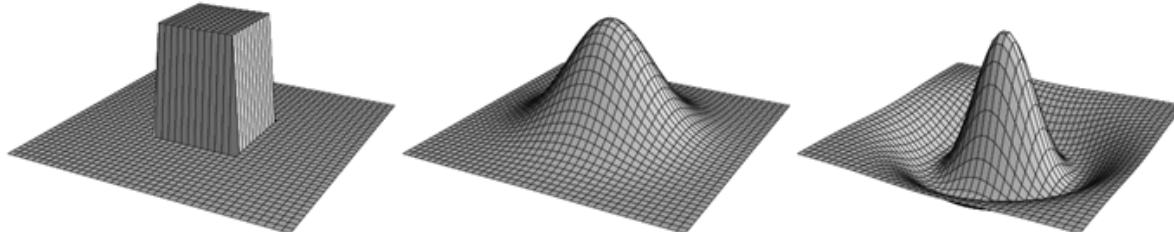
- a) Crop
- b) Pad
- c) Extend
- d) Wrap





# Linear Filters: Smoothing Filters

- 2 main classes of linear filters:
  - **Smoothing:** +ve coefficients (weighted average). E.g box, gaussian
  - **Difference** filters: +ve and -ve weights. E.g. Laplacian



0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

(a)

**Box**

0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

(b)

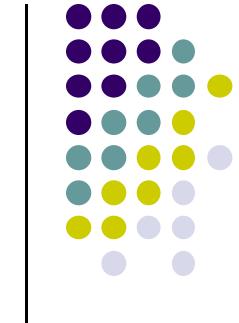
**Gaussian**

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

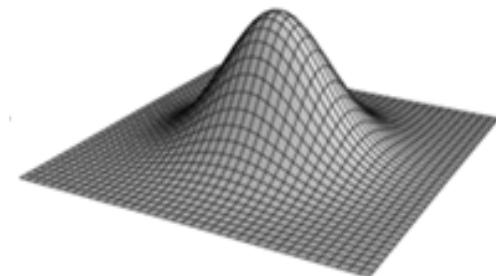
(c)

**Laplacian**

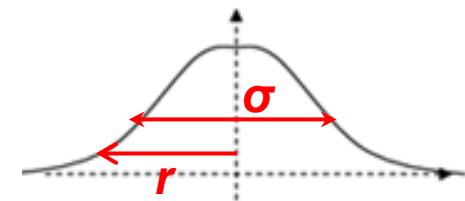
# Gaussian Filter



$$G_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \text{or} \quad G_\sigma(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$



- where
  - $\sigma$  is width (standard deviation)
  - $r$  is distance from center



0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

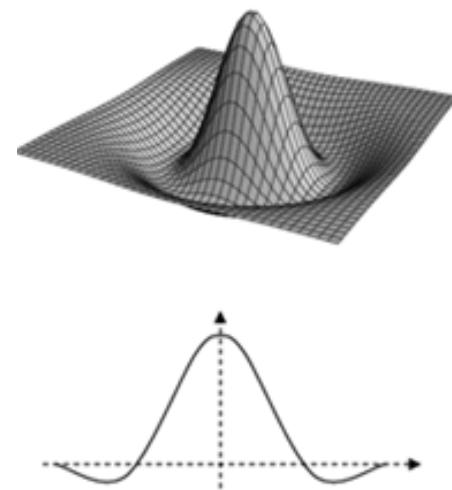
Gaussian  
filter



# Difference Filters

- **Coefficients:** some +ve, some negative
- Example: Laplacian filter
- Computation is difference

$$\sum (+ve \text{ coefficients}) - \sum (-ve \text{ coefficients})$$



$$I'(u, v) = \sum_{(i,j) \in R_H^+} I(u+i, v+j) \cdot |H(i, j)| - \sum_{(i,j) \in R_H^-} I(u+i, v+j) \cdot |H(i, j)|$$

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Laplacian  
filter



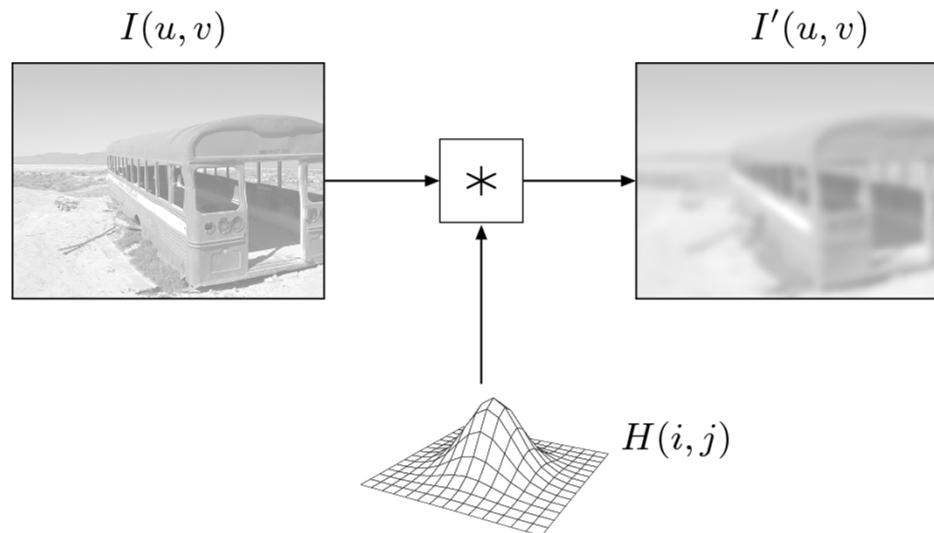
# Mathematical Properties of Convolution

- Applying a filter as described called ***linear convolution***
- For discrete 2D signal, convolution defined as:

$$I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j)$$

Formal definition:  
Sum to  $\pm \infty$

$$I' = I * H$$





# Properties of Convolution

- Commutativity

$$I * H = H * I$$

Same result if we convolve  
image with filter or vice versa

- Linearity

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

If image multiplied by scalar  
Result multiplied by same scalar

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

(notice)

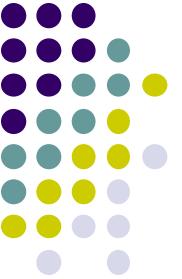
$$(b + I) * H \neq b + (I * H)$$

If 2 images added and convolve  
result with a kernel  $H$ ,  
Same result if each image  
is convolved individually + added

- Associativity

$$A * (B * C) = (A * B) * C$$

Order of filter application irrelevant  
Any order, same result



# References

- Wilhelm Burger and Mark J. Burge, Digital Image Processing, Springer, 2008
  - Histograms (Ch 4)
  - Point operations (Ch 5)
  - Filters (Ch 6)
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012

# What will we learn today?

- Why Geometric Vision Matters
- Geometric Primitives in 2D & 3D
- 2D & 3D Transformations

**Images are  
2D projections of  
the 3D world**

# Simplified Image Formation

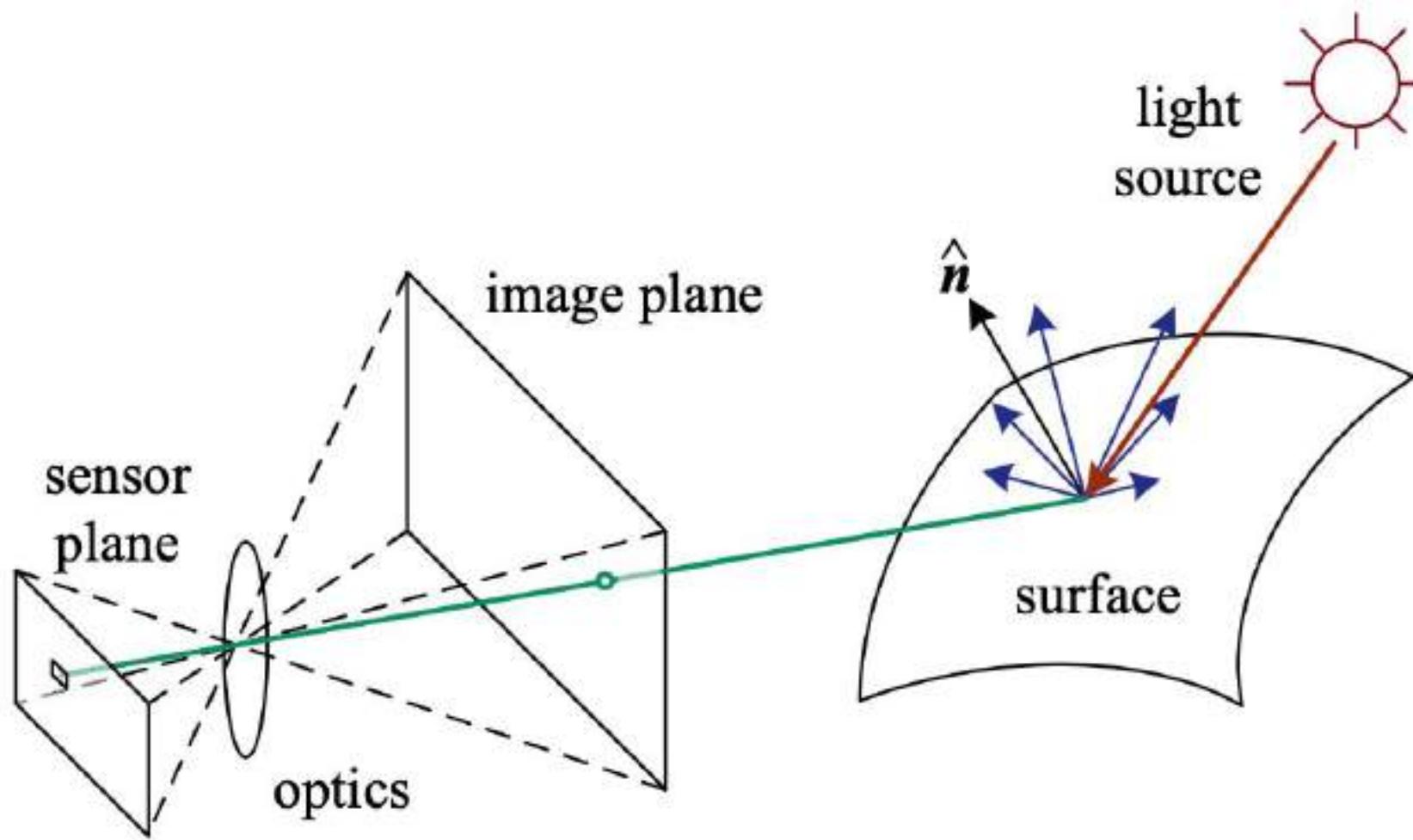


Figure: R. Szeliski

# Perspective Projection

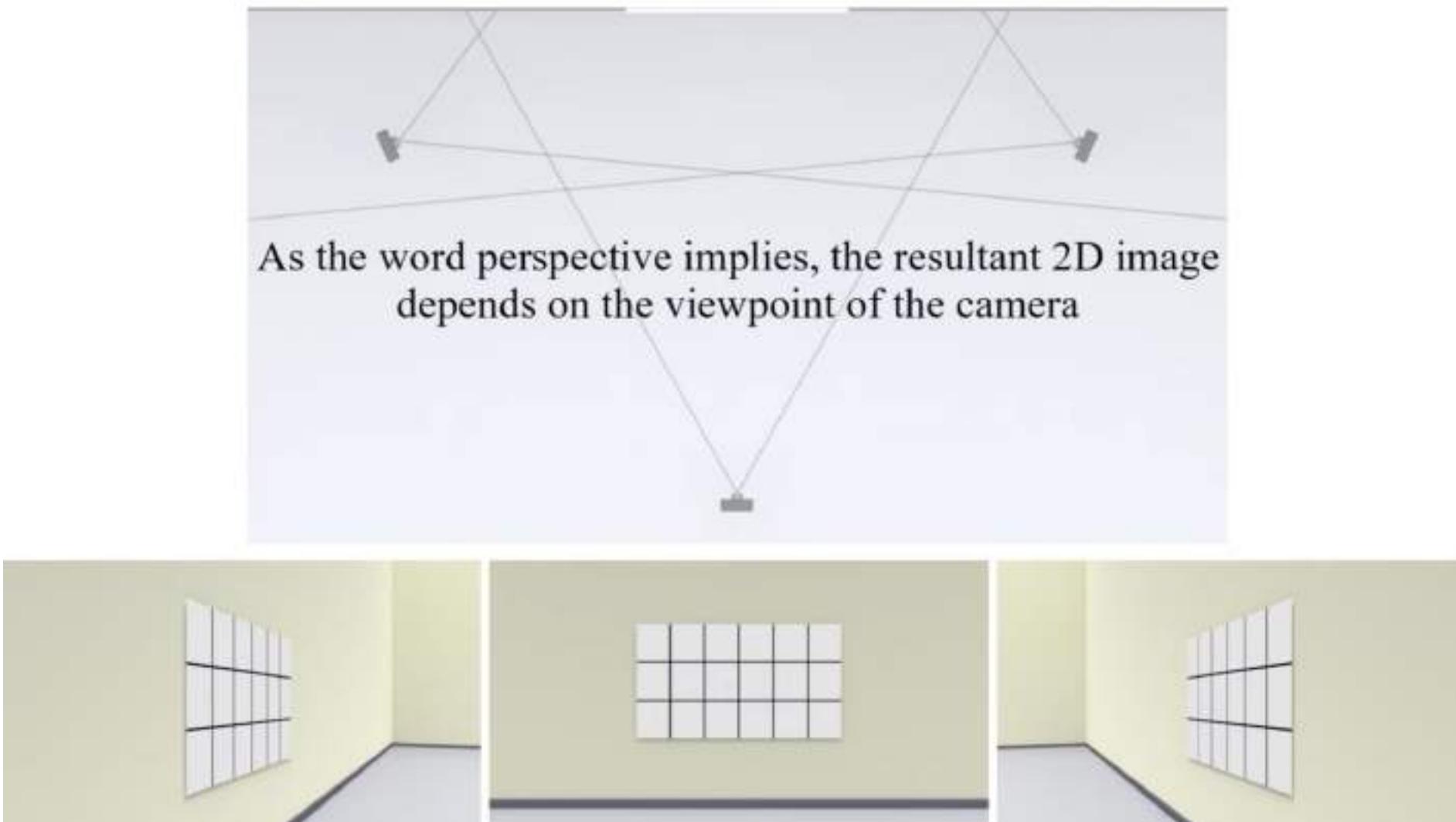
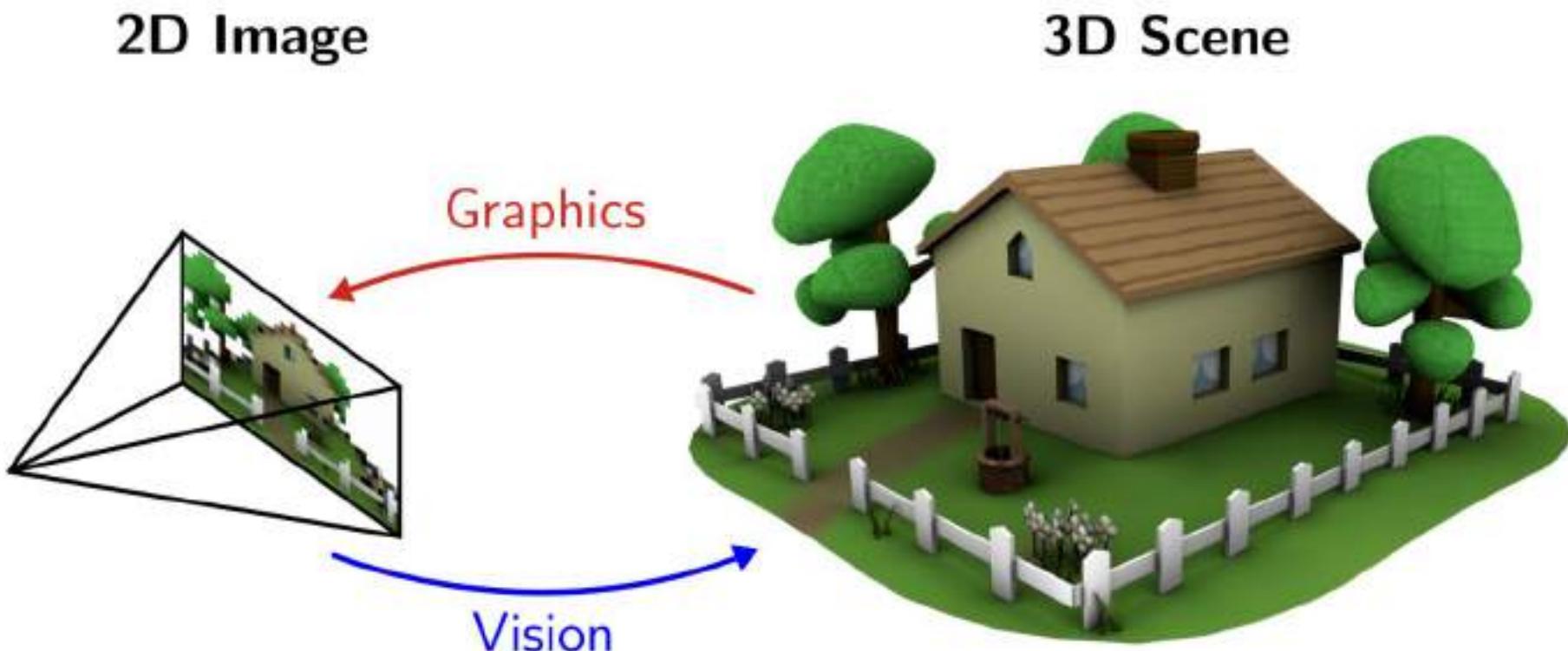


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

**Can we understand  
the 3D world  
from 2D images?**

# CV is an **ill-posed** inverse problem



Pixel Matrix

217	191	252	255	239
102	80	200	146	138
159	94	91	121	138
179	106	136	85	41
115	129	83	112	67
94	114	105	111	89

Objects

Shape/Geometry

Semantics

Material

Motion

3D Pose

# **What will we learn today?**

Why Geometric Vision Matters

Geometric Primitives in 2D & 3D

2D & 3D Transformations

# Points in Cartesian and Homogeneous Coordinates

2D points:  $\mathbf{x} = (x, y) \in \mathcal{R}^2$  or column vector  $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

3D points:  $\mathbf{x} = (x, y, z) \in \mathcal{R}^3$  (often noted  $\mathbf{X}$  or  $\mathbf{P}$ )

Homogeneous coordinates: append a 1

$$\bar{\mathbf{x}} = (x, y, 1) \quad \bar{\mathbf{x}} = (x, y, z, 1)$$

Why?

# Homogeneous coordinates in 2D

2D Projective Space:  $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$  (same story in 3D with  $\mathcal{P}^3$ )

- heterogeneous  $\rightarrow$  homogeneous

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

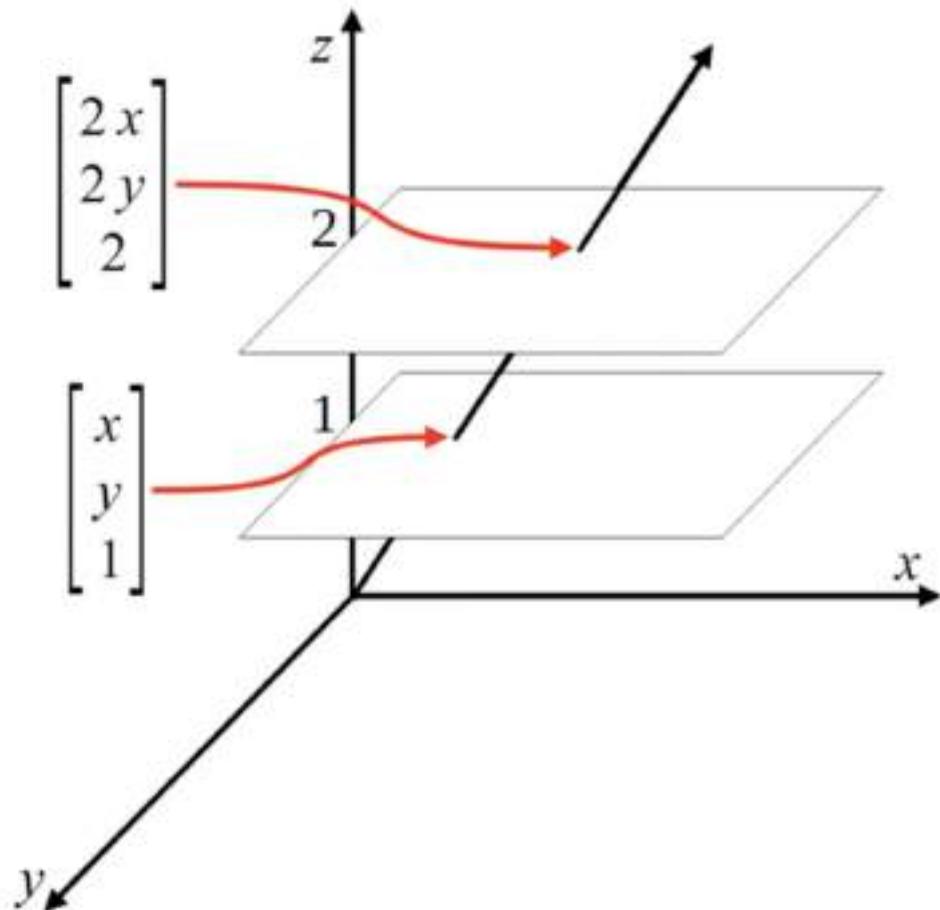
- homogeneous  $\rightarrow$  heterogeneous

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

- points differing only by scale are *equivalent*:  $(x, y, w) \sim \lambda (x, y, w)$

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\mathbf{x}}$$

# Homogeneous coordinates in 2D



In homogeneous coordinates, a point and its scaled versions are same

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \quad w \neq 0$$

Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Everything is easier in Projective Space

2D Lines:

Representation:  $l = (a, b, c)$

Equation:  $ax + by + c = 0$

In homogeneous coordinates:  $\bar{x}^T l = 0$

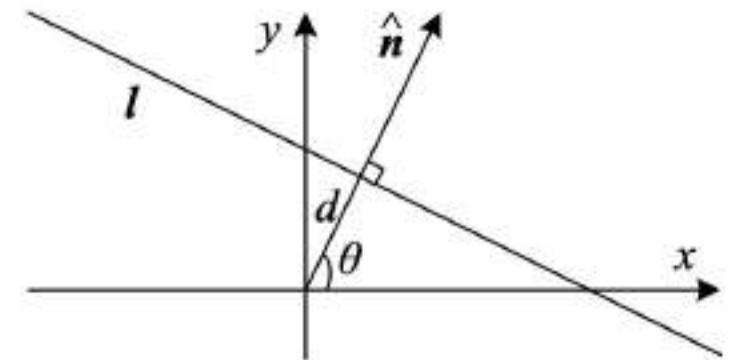
General idea: homogenous coordinates unlock the full power of linear algebra!

# Everything is easier in Projective Space

2D Lines:

$$\tilde{\mathbf{x}}^T \mathbf{l} = 0, \forall \tilde{\mathbf{x}} = (x, y, w) \in P^2$$

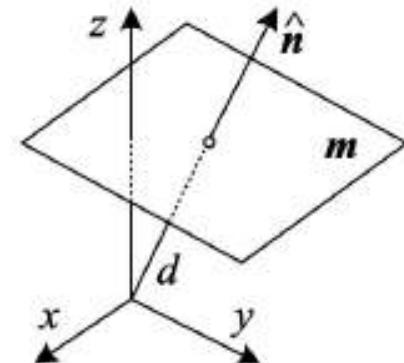
$$\mathbf{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\mathbf{n}}, d) \text{ with } \|\hat{\mathbf{n}}\| = 1$$



3D planes: same!

$$\tilde{\mathbf{x}}^T \mathbf{m} = 0, \forall \tilde{\mathbf{x}} = (x, y, z, w) \in P^3$$

$$\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\mathbf{n}}, d) \text{ with } \|\hat{\mathbf{n}}\| = 1$$



# Lines in 3D

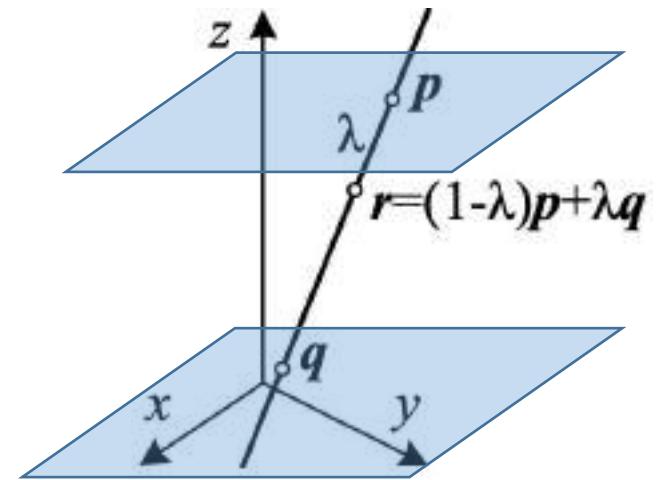
Two-point parametrization:

$$\mathbf{r} = (1 - \lambda)\mathbf{p} + \lambda\mathbf{q} \quad \tilde{\mathbf{r}} = \mu\tilde{\mathbf{p}} + \lambda\tilde{\mathbf{q}}$$

Two-plane parametrization:

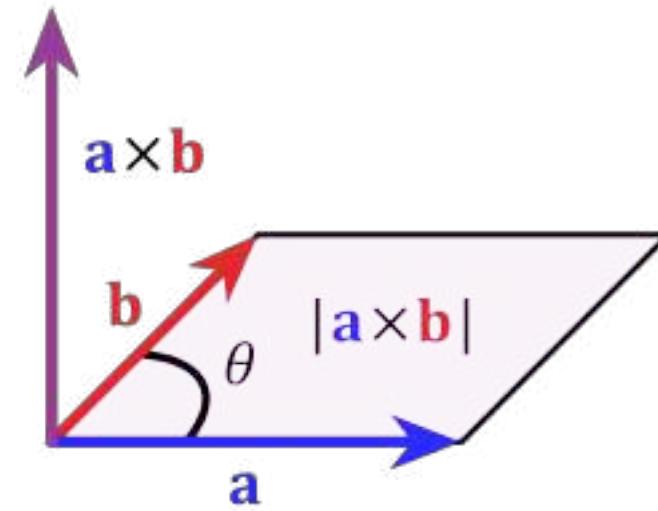
coordinates  $(x_0, y_0)$  &  $(x_1, y_1)$  of intersection

with planes at  $z = 0, 1$  (or other planes)



# Cross-product quick reminder

$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \sin(\theta) \mathbf{n}$$



$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_{\times} \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

# Benefits of Homogeneous Coordinates

- Line – Point duality:
  - line between two 2D points:  $\tilde{l} = \tilde{x}_1 \times \tilde{x}_2$
  - intersection of two 2D lines:  $\tilde{x} = \tilde{l}_1 \times \tilde{l}_2$
- Representation of Infinity:
  - points at infinity:  $(x, y, 0)$ ; line at infinity:  $(0, 0, 1)$
- Parallel & vertical lines are easy (take-home: intersect //)
- Makes 2D & 3D transformations linear!

# Questions?

# **What will we learn today?**

Why Geometric Vision Matters

Geometric Primitives in 2D & 3D

2D & 3D Transformations

# The camera as a coordinate transformation

A camera is a mapping

from:

the 3D world

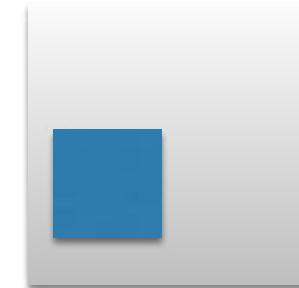
to:

a 2D image

3D object



3D to 2D transform  
(camera)



2D image

# The camera as a coordinate transformation

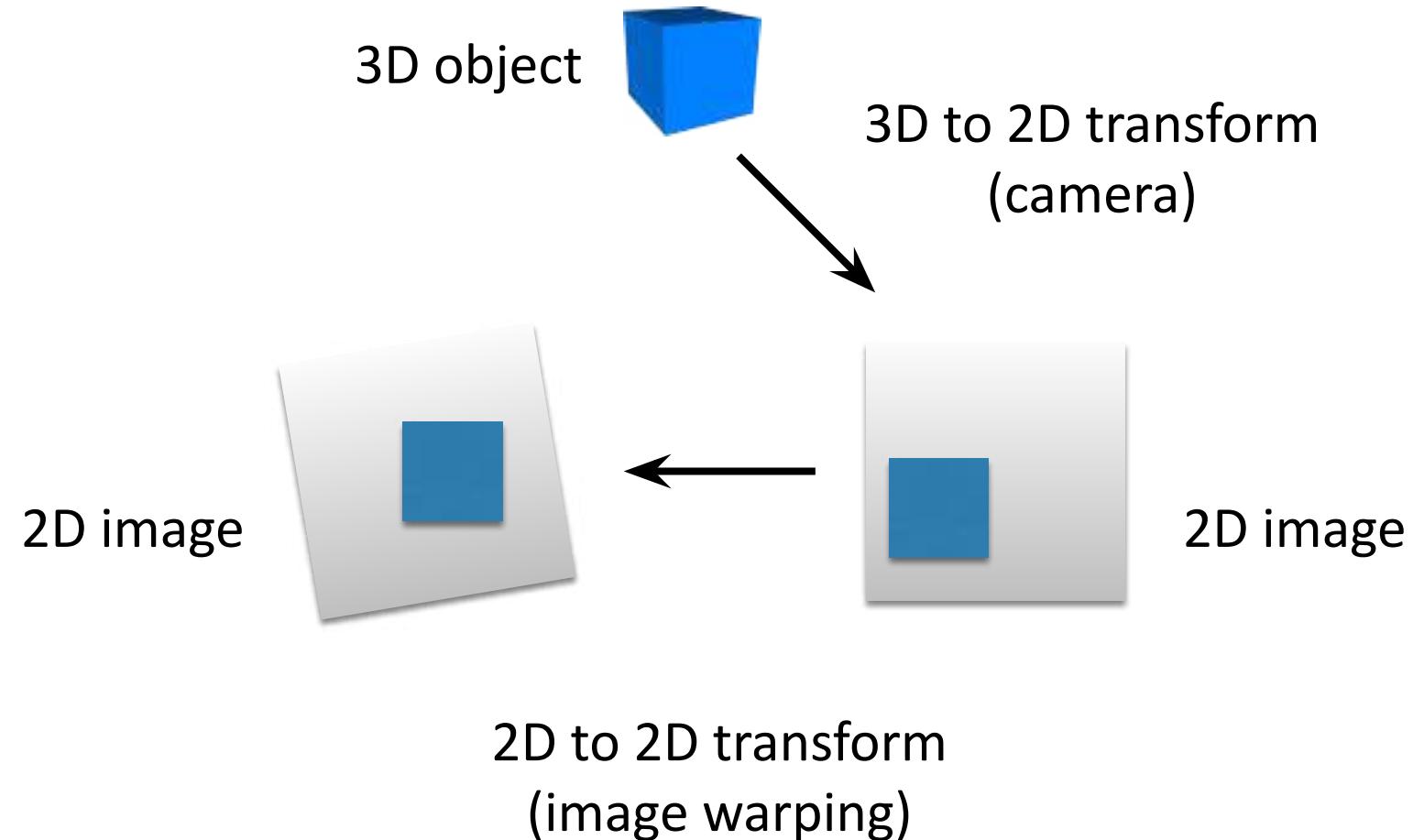
A camera is a mapping

from:

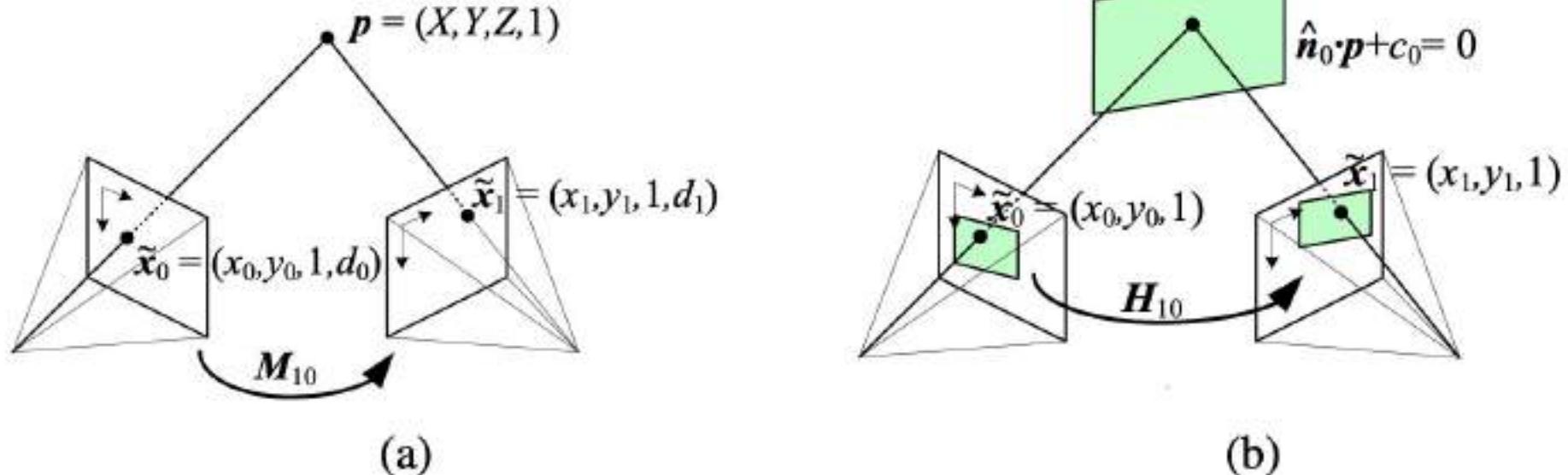
the 3D world

to:

a 2D image



# Cameras and objects can move!



**Figure 2.12** A point is projected into two images: (a) relationship between the 3D point coordinate  $(X, Y, Z, 1)$  and the 2D projected point  $(x, y, 1, d)$ ; (b) planar homography induced by points all lying on a common plane  $\hat{\mathbf{n}}_0 \cdot \mathbf{p} + c_0 = 0$ .

# 2D Transformations Zoo

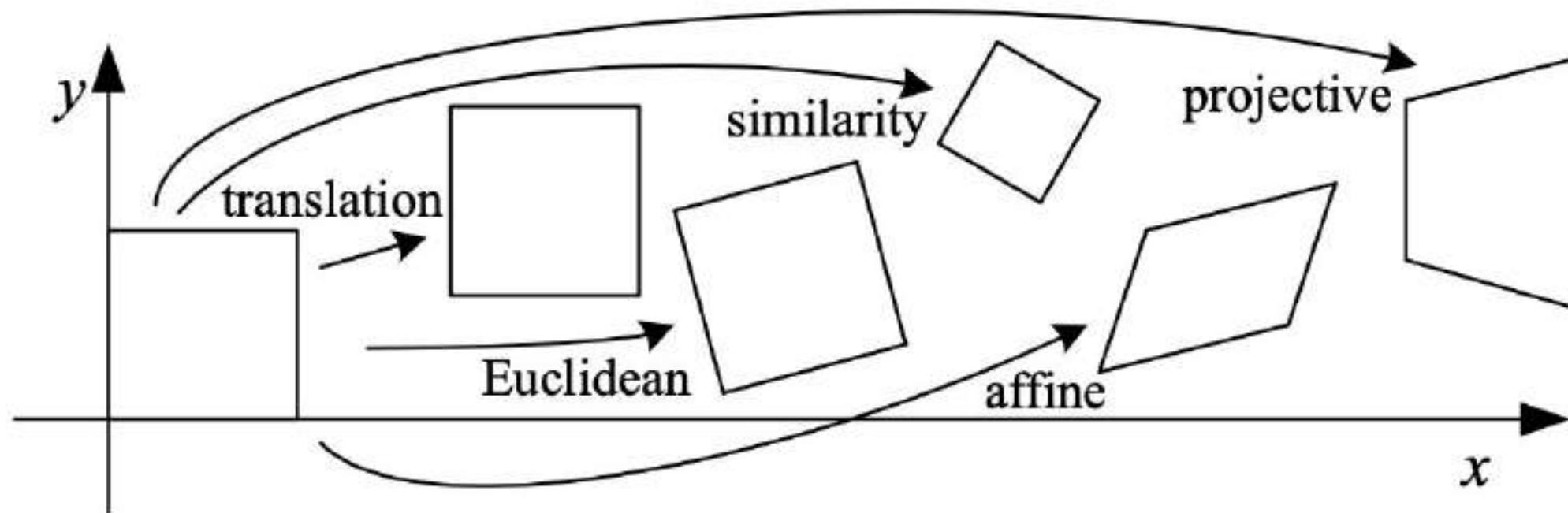


Figure: R. Szeliski

# Transformation = Matrix Multiplication

Scale

$$\mathbf{M} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Flip across y

$$\mathbf{M} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Rotate

$$\mathbf{M} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Flip across origin

$$\mathbf{M} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Shear

$$\mathbf{M} = \begin{bmatrix} 1 & s_x \\ s_y & 1 \end{bmatrix}$$

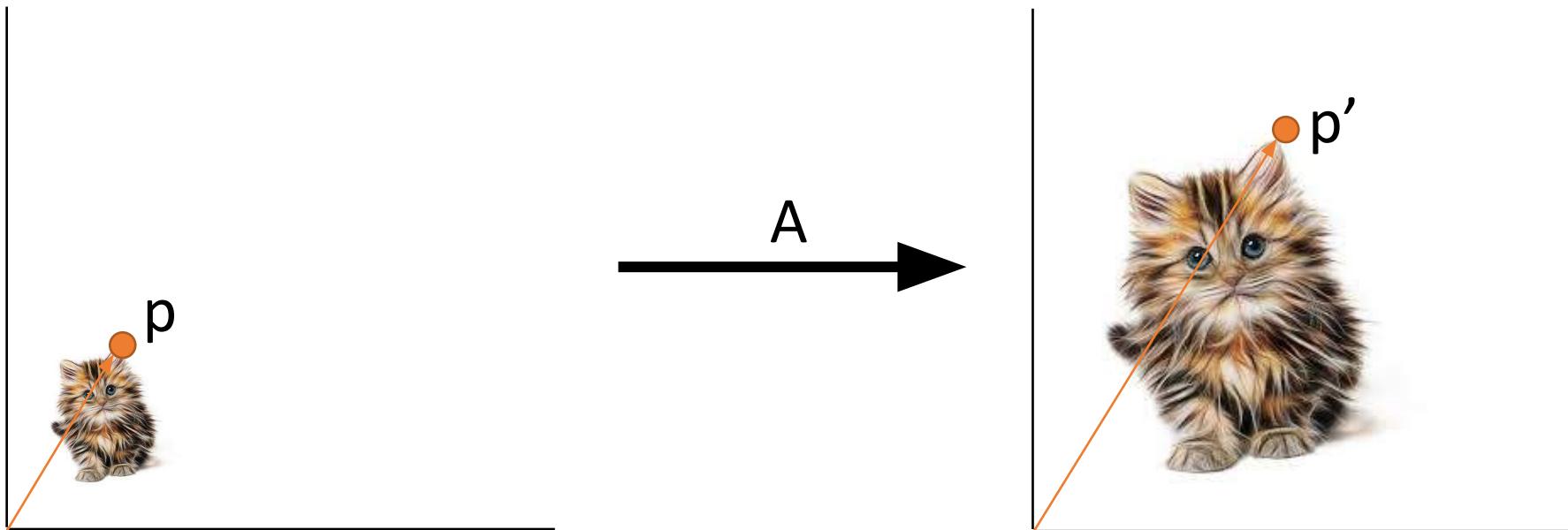
Identity

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

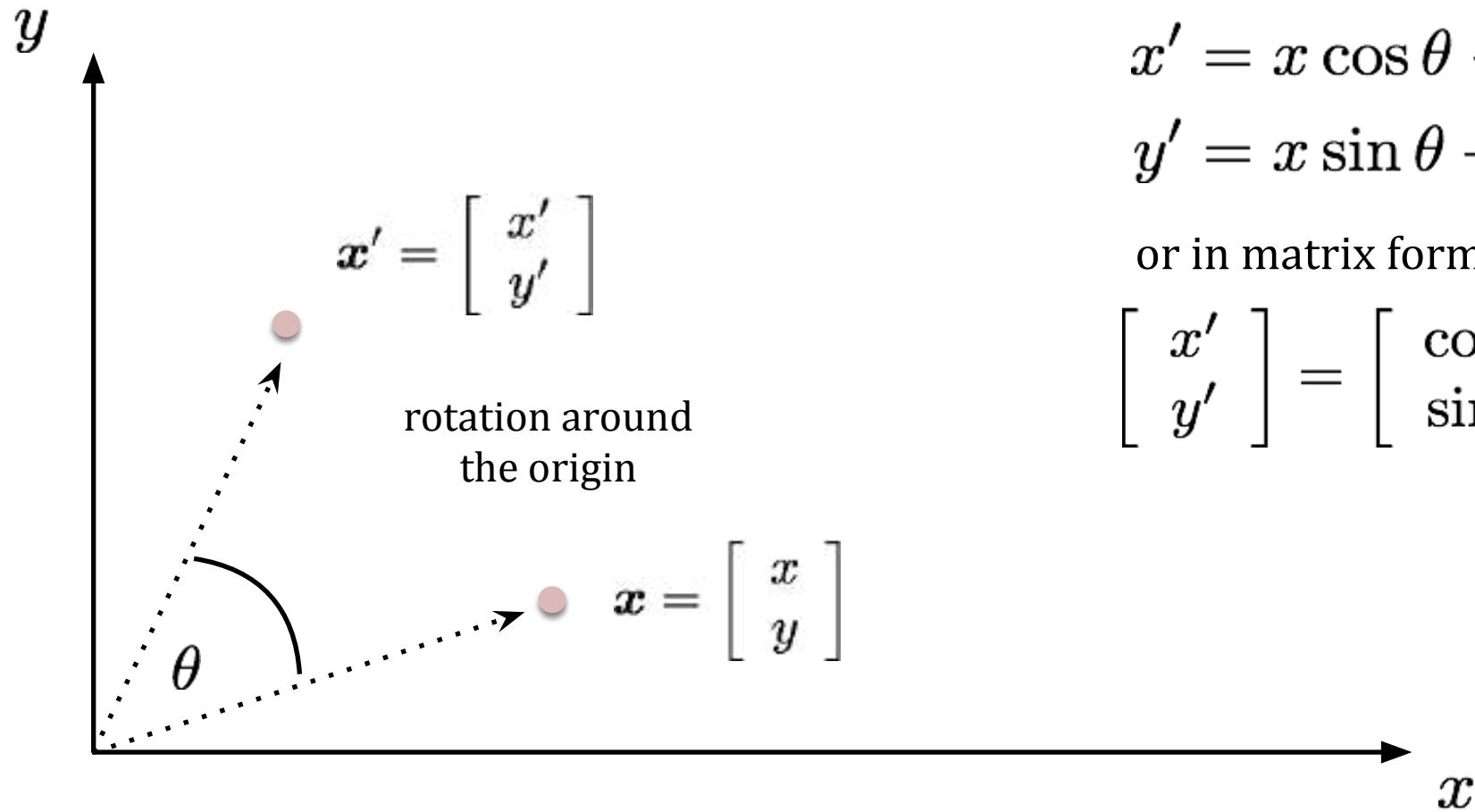
# Scaling

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

A                    p                    p'



# Rotation



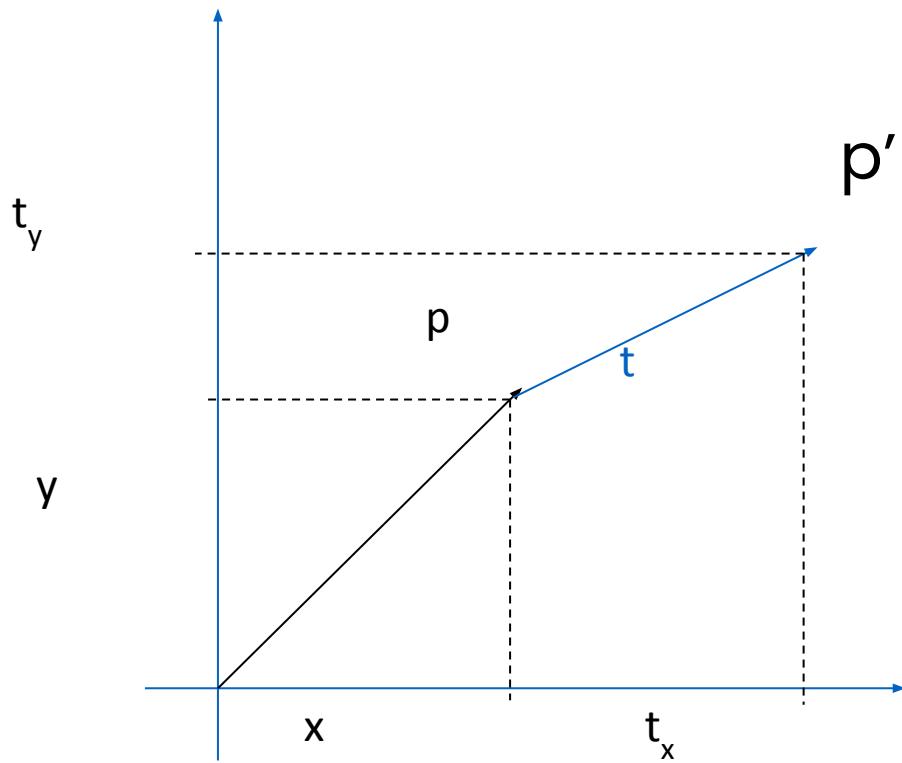
$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

or in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

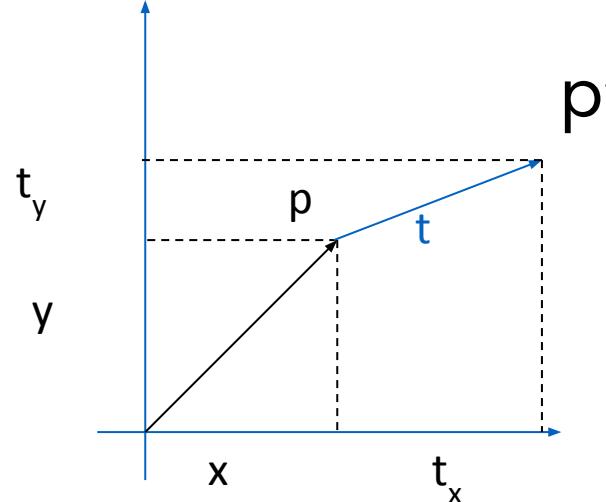
# Translation



$$x' = x + t_x$$
$$y' = y + t_y$$

As a matrix?

# Translation with homogeneous coordinates



$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$t = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix}$$

$$p' = Tp$$

$$p' \rightarrow \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} I & t \\ 0 & 1 \end{bmatrix} p = Tp$$

# 2D Transformations with homogeneous coordinates

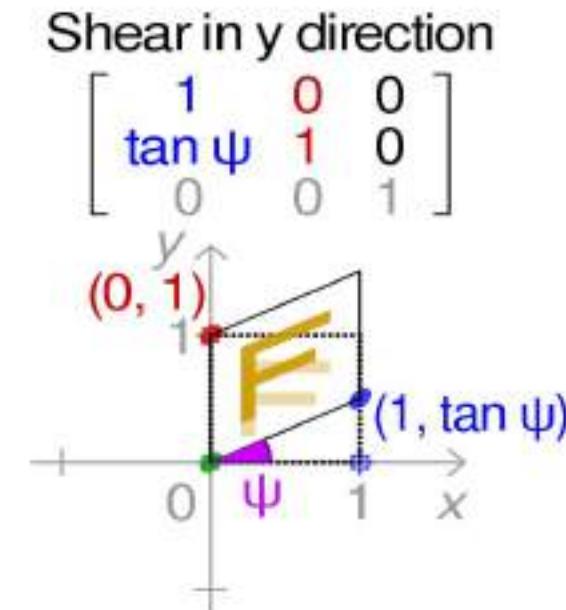
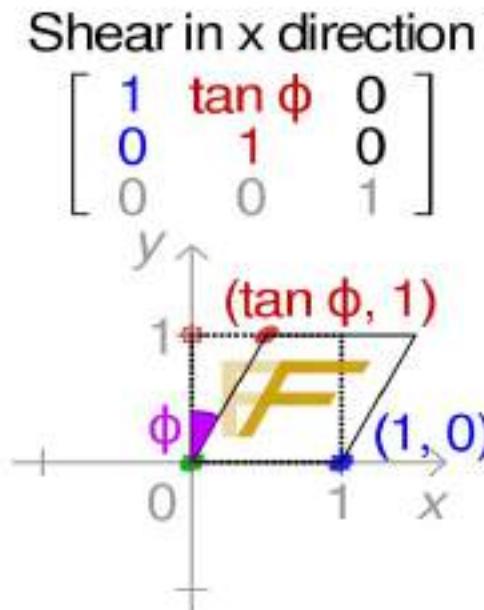
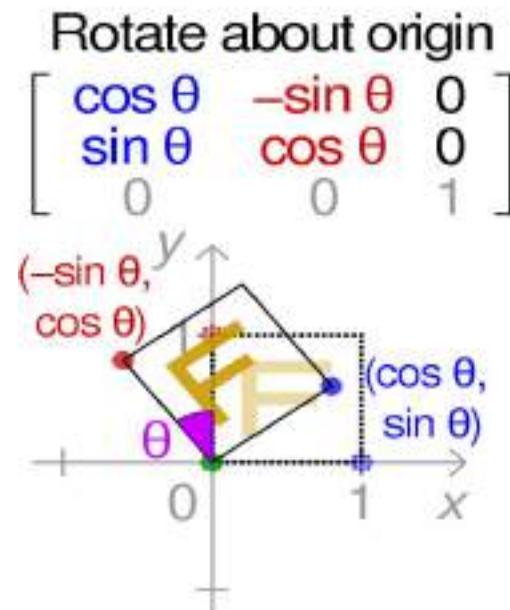
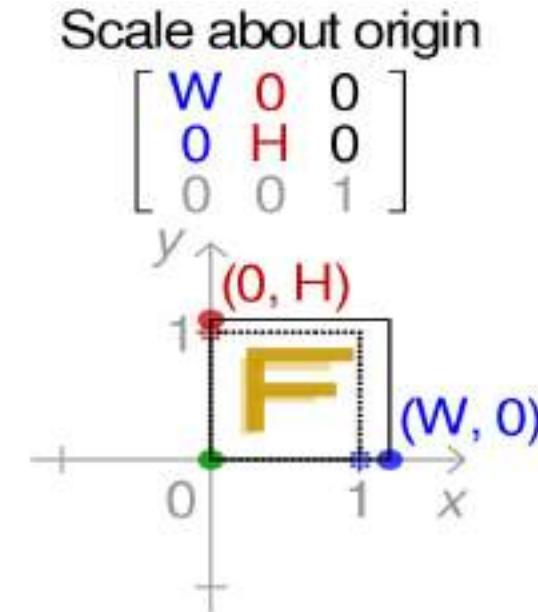
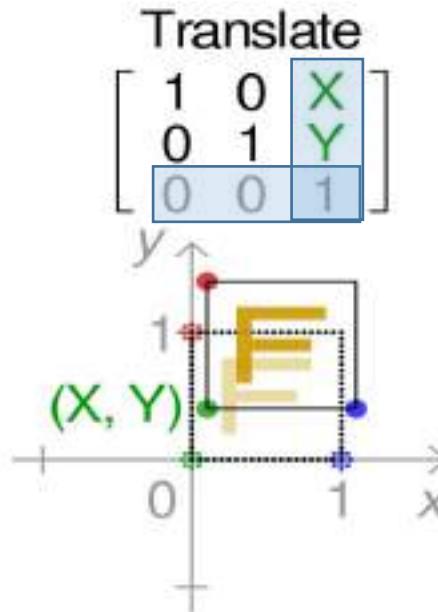
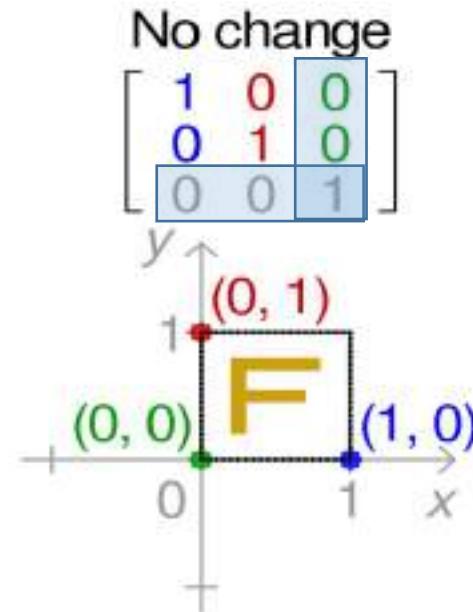


Figure: Wikipedia

# Questions?

# 2D Transformations Zoo

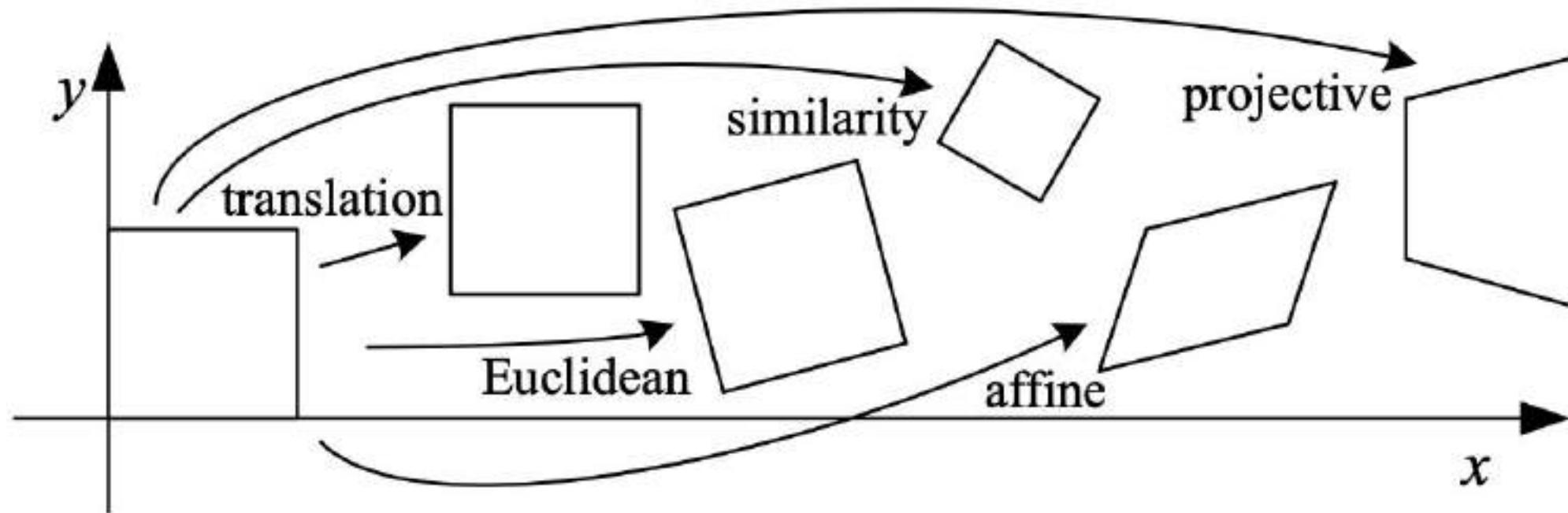


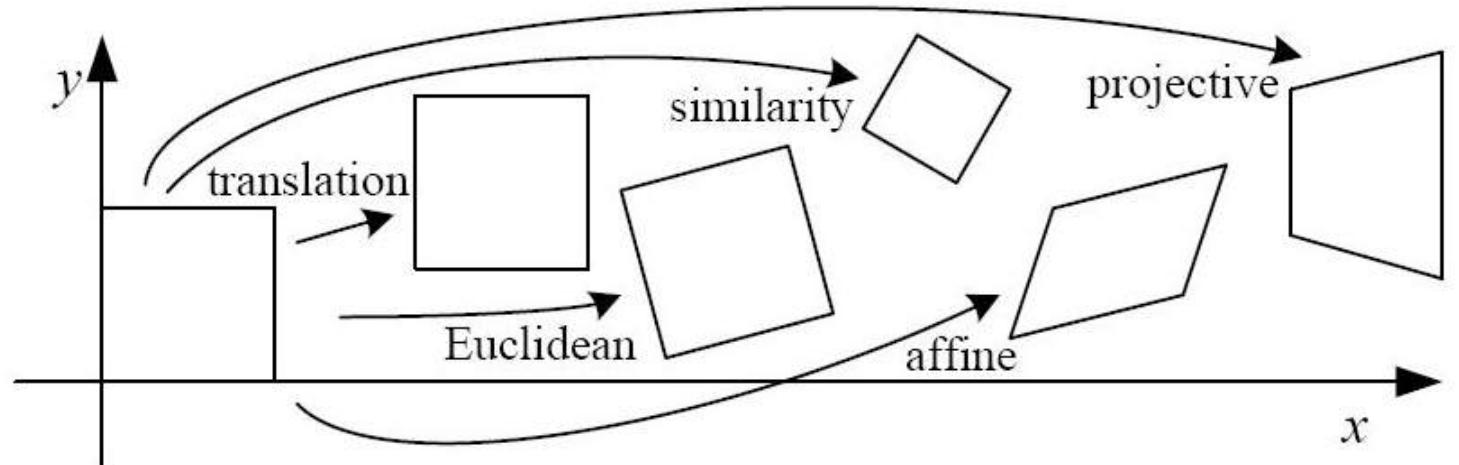
Figure: R. Szeliski

# Euclidean / Rigid Transformation

Euclidean (rigid): rotation + translation

$$\begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

How many degrees of freedom?

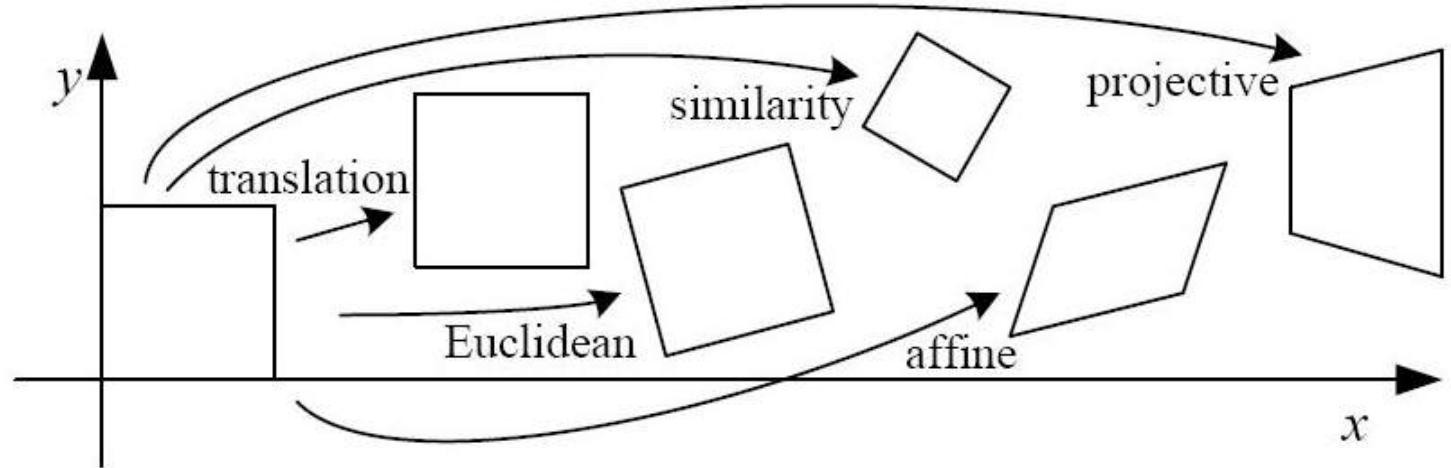


# Similarity Transformation

Similarity: Scaling + rotation + translation

$$\begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

How many degrees of freedom?

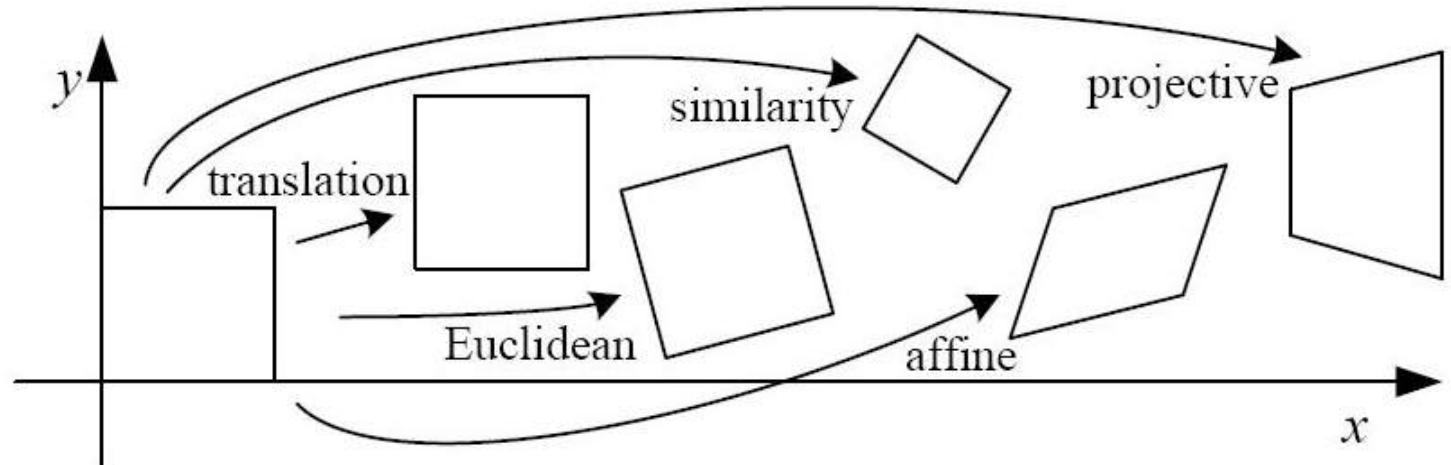


# Similarity Transformation

Similarity: Scaling + rotation + translation

$$\begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha \cos \theta & -\alpha \sin \theta & b_0 \\ \alpha \sin \theta & \alpha \cos \theta & b_1 \\ 0 & 0 & 1 \end{bmatrix}$$

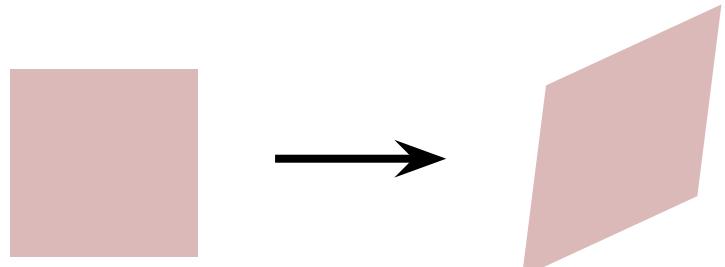
How many degrees of freedom?



# Affine Transformation

Affine transformations are combinations of

- Arbitrary (4-DOF) linear transformations + translations



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \qquad \longrightarrow$$

Cartesian  
coordinates

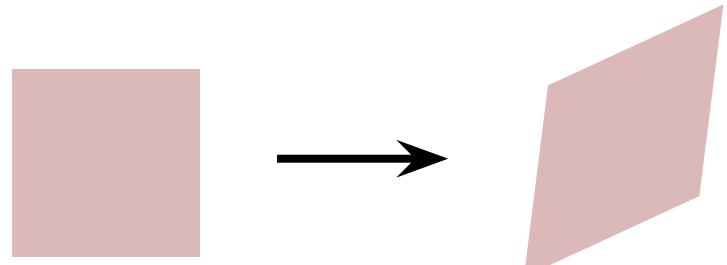
Homogeneous  
coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Affine Transformation

Affine transformations are combinations of

- Arbitrary (4-DOF) linear transformations + translations



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \qquad \longrightarrow$$

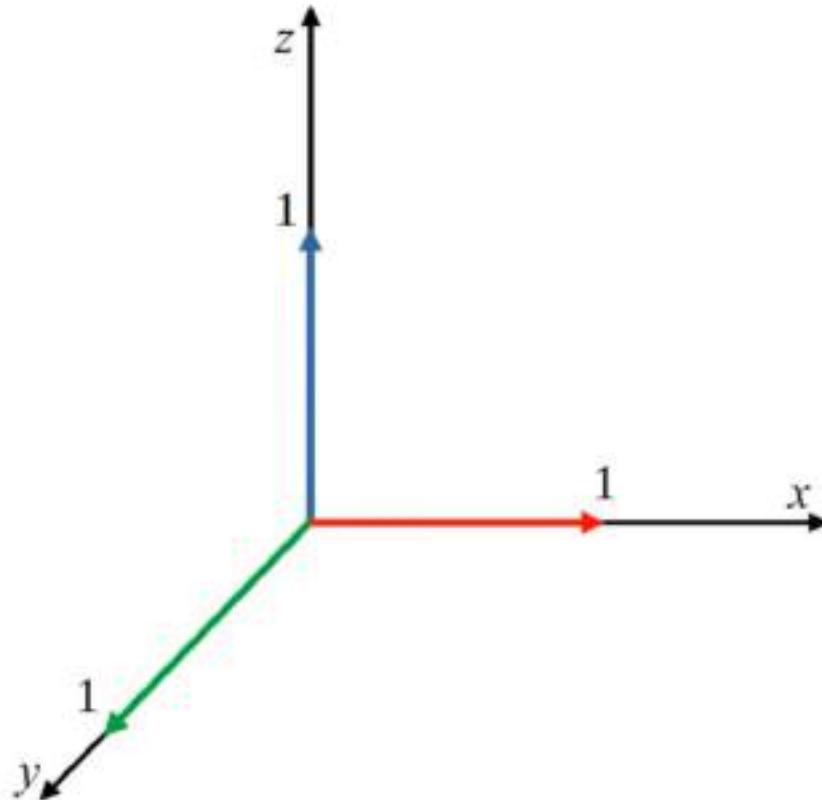
Cartesian  
coordinates

Homogeneous  
coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How many degrees of freedom?

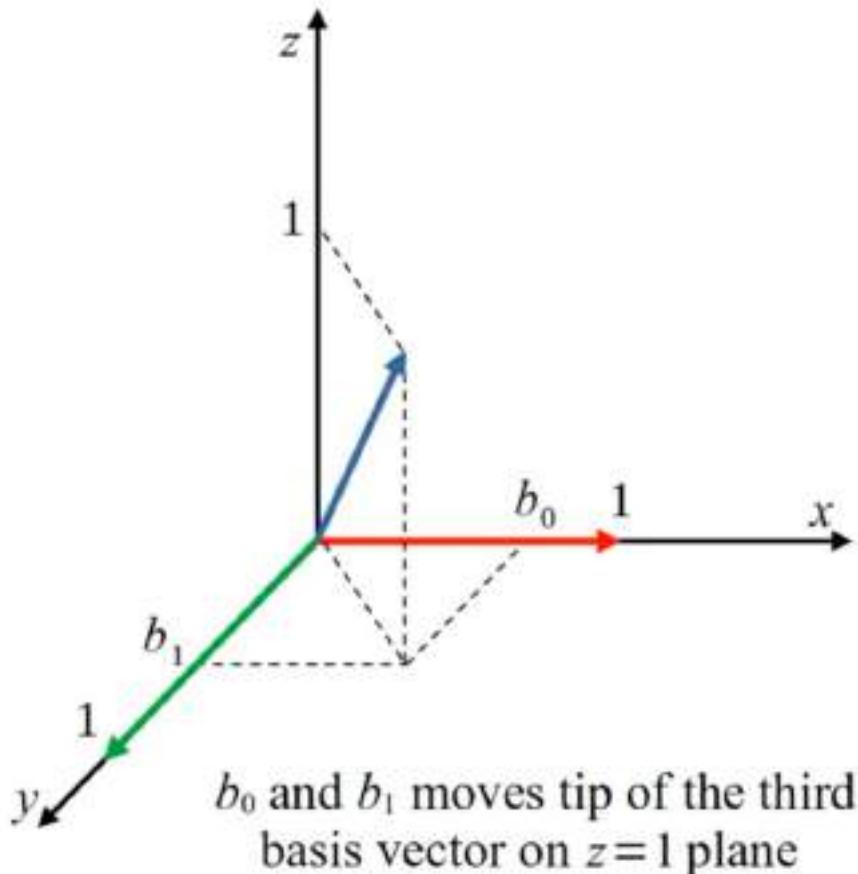
# Affine Transformation



This matrix is a linear transformation matrix in 3D

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Affine Transformation



This matrix is a linear transformation matrix in 3D

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Then this column is the third basis vector of transformed vector space

And what  $b_0$  and  $b_1$  do is to change the orientation of that basis vector

# Affine Transformation

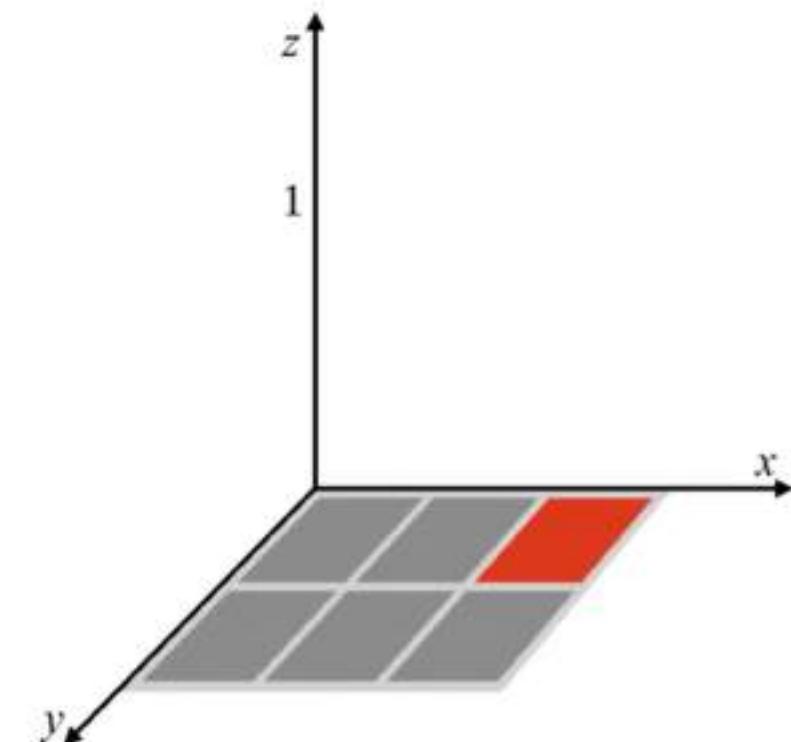


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Affine Transformation

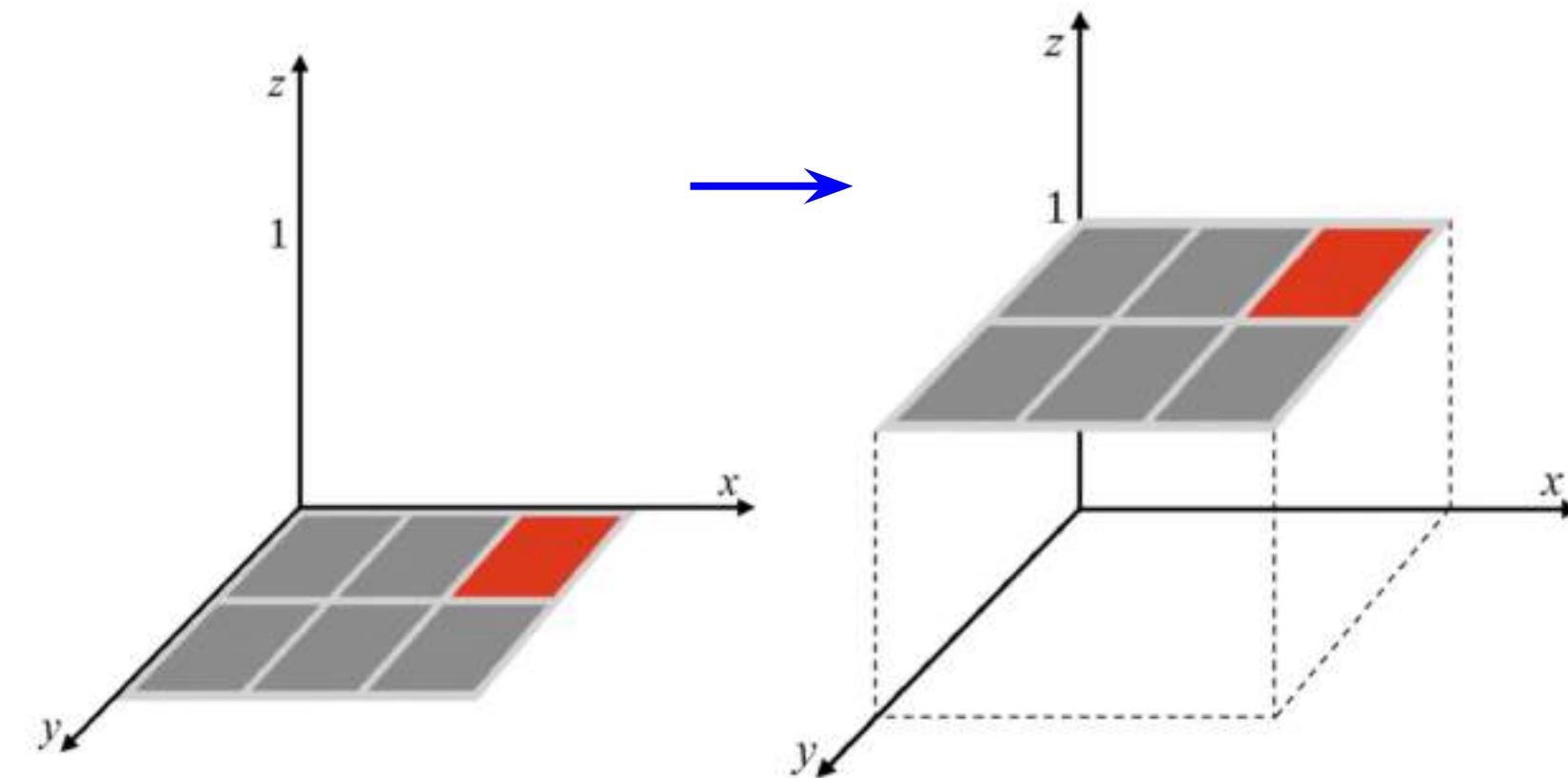


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Affine Transformation

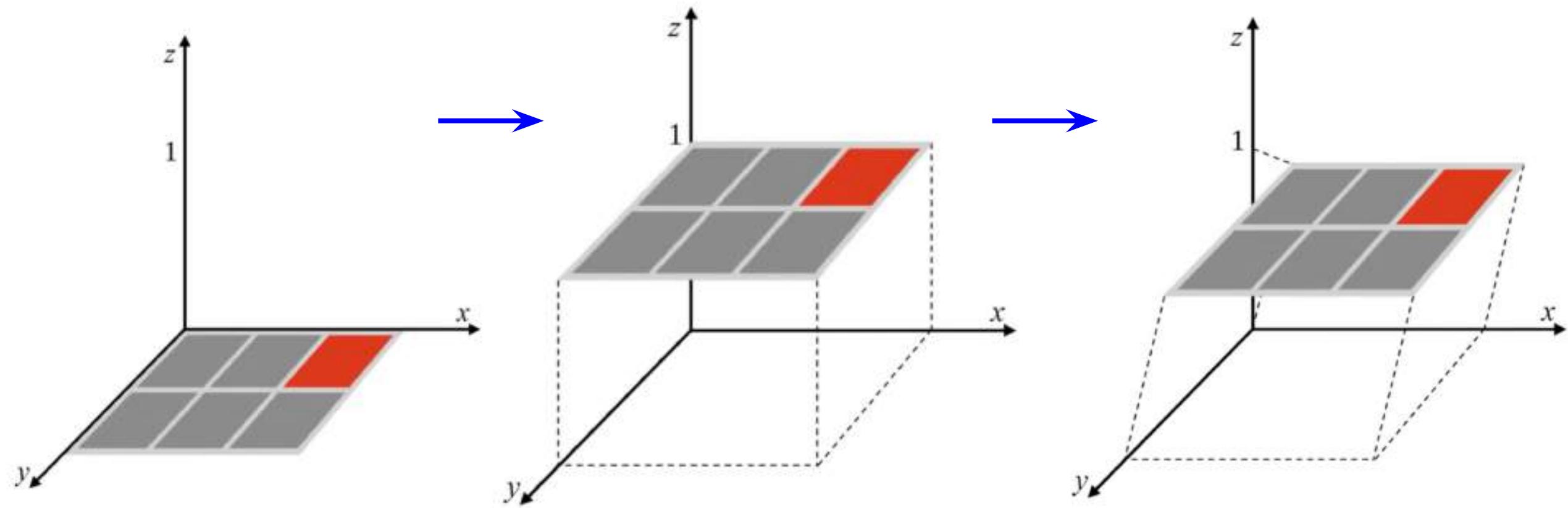


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Affine Transformation

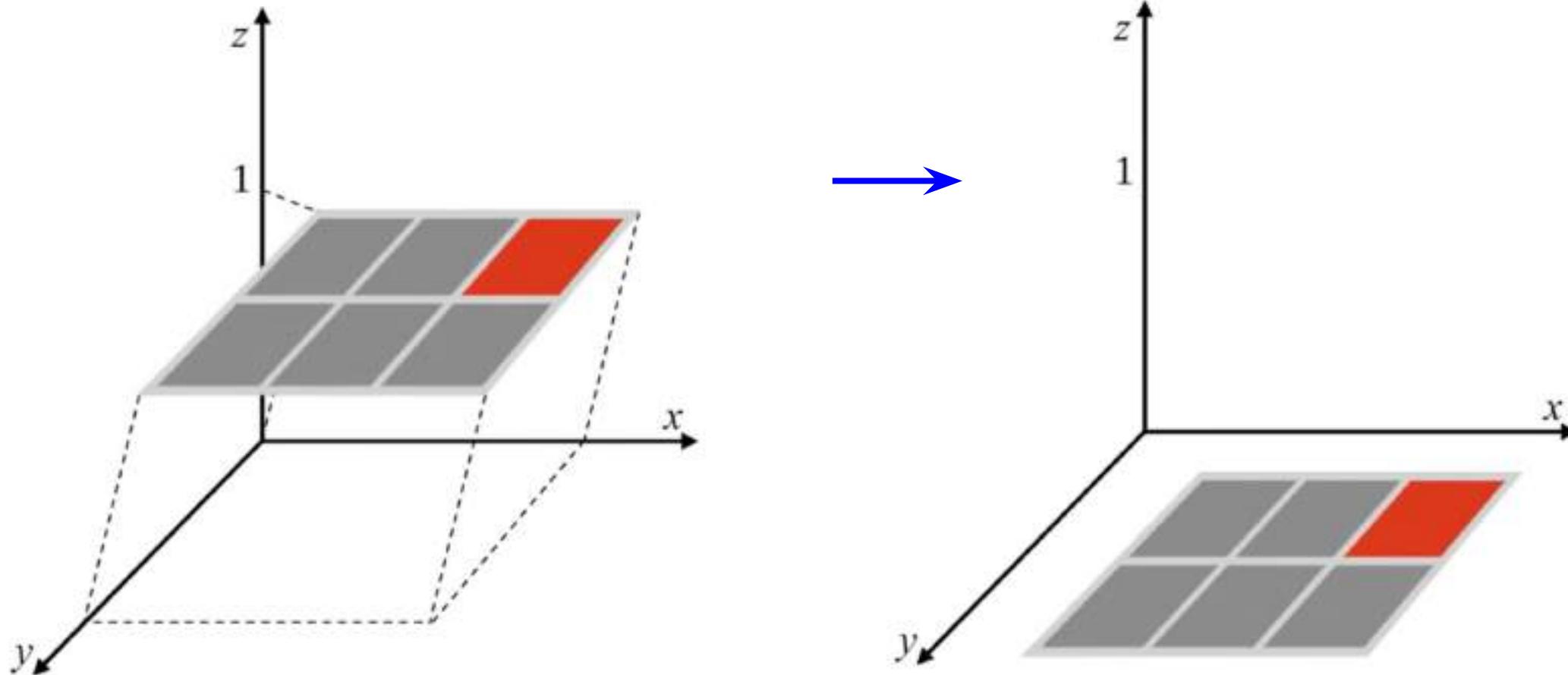
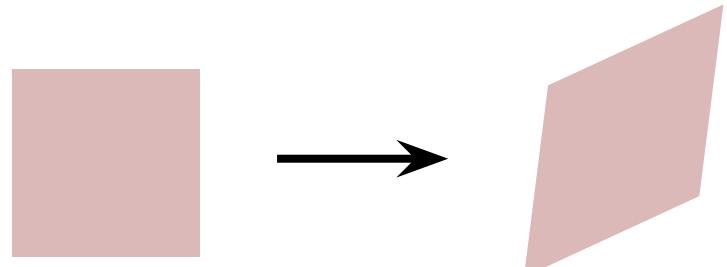


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Affine Transformation

Affine transformations are combinations of

- Arbitrary (4-DOF) linear transformations + translations



Properties of affine transformations:

- origin does not necessarily map to origin
- lines map to lines
- parallel lines map to parallel lines
- ratios are preserved

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

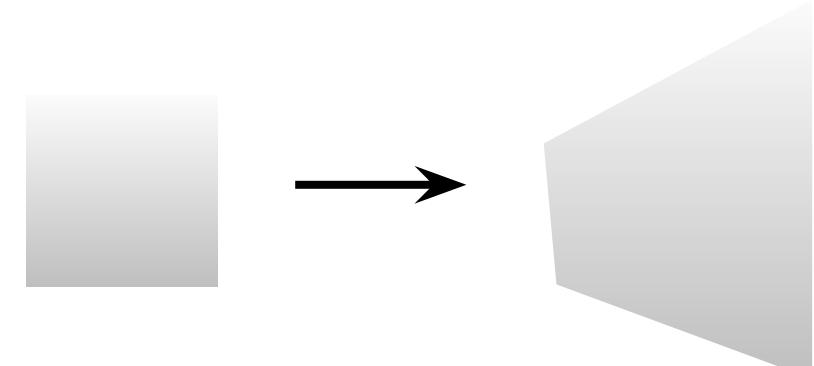
# Projective Transformation (homography)

Projective transformations are combinations of

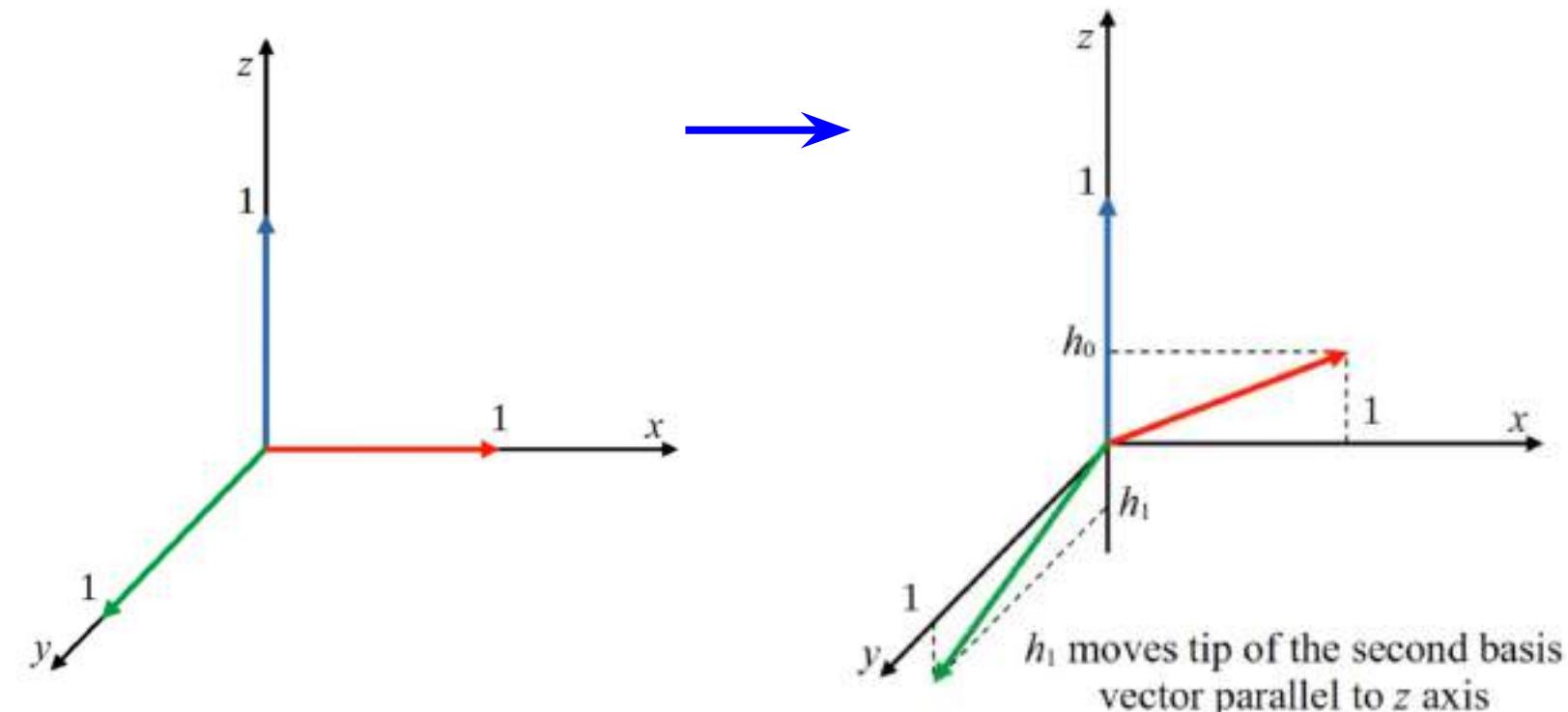
- Affine transformations + projective warps

$$w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ h_0 & h_1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How many degrees of freedom?



# Projective Transformation (homography)



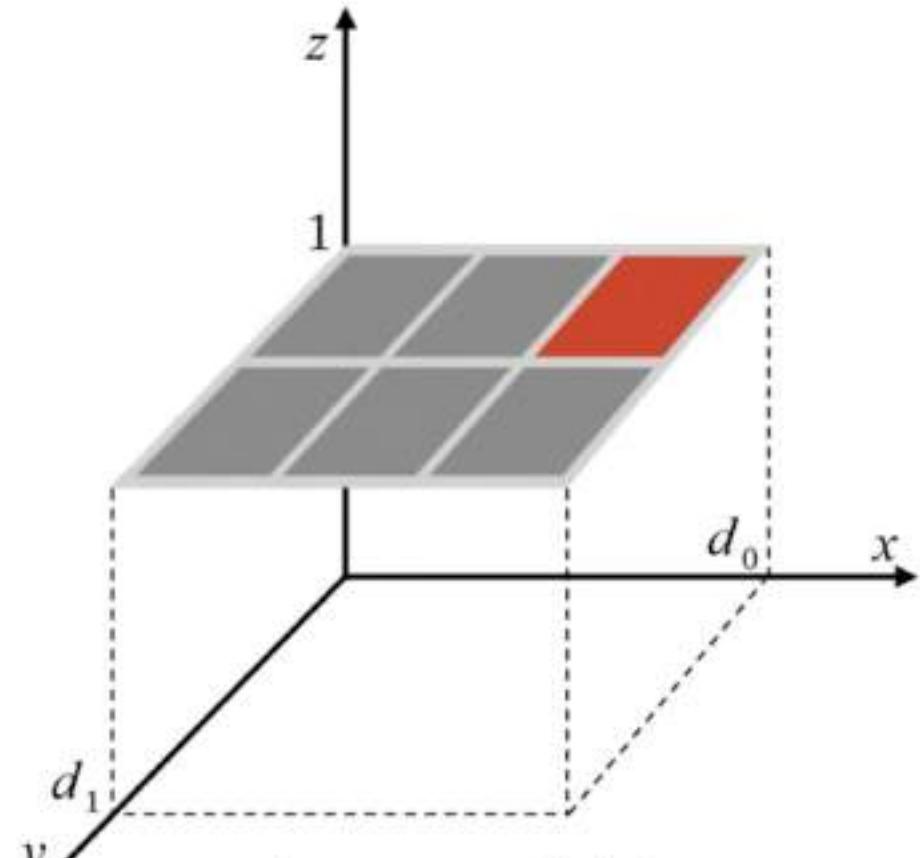
This matrix is a linear transformation matrix in 3D

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h_0 & h_1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Then these two columns are the first and the second basis vectors of transformed vector space

And what  $h_0$  and  $h_1$  do is to change the orientation of those basis vectors

# Projective Transformation (homography)



As an example let  
 $h_0 > 0$ ,  $h_1 < 0$  and  $|h_0| > |h_1|$

Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Projective Transformation (homography)

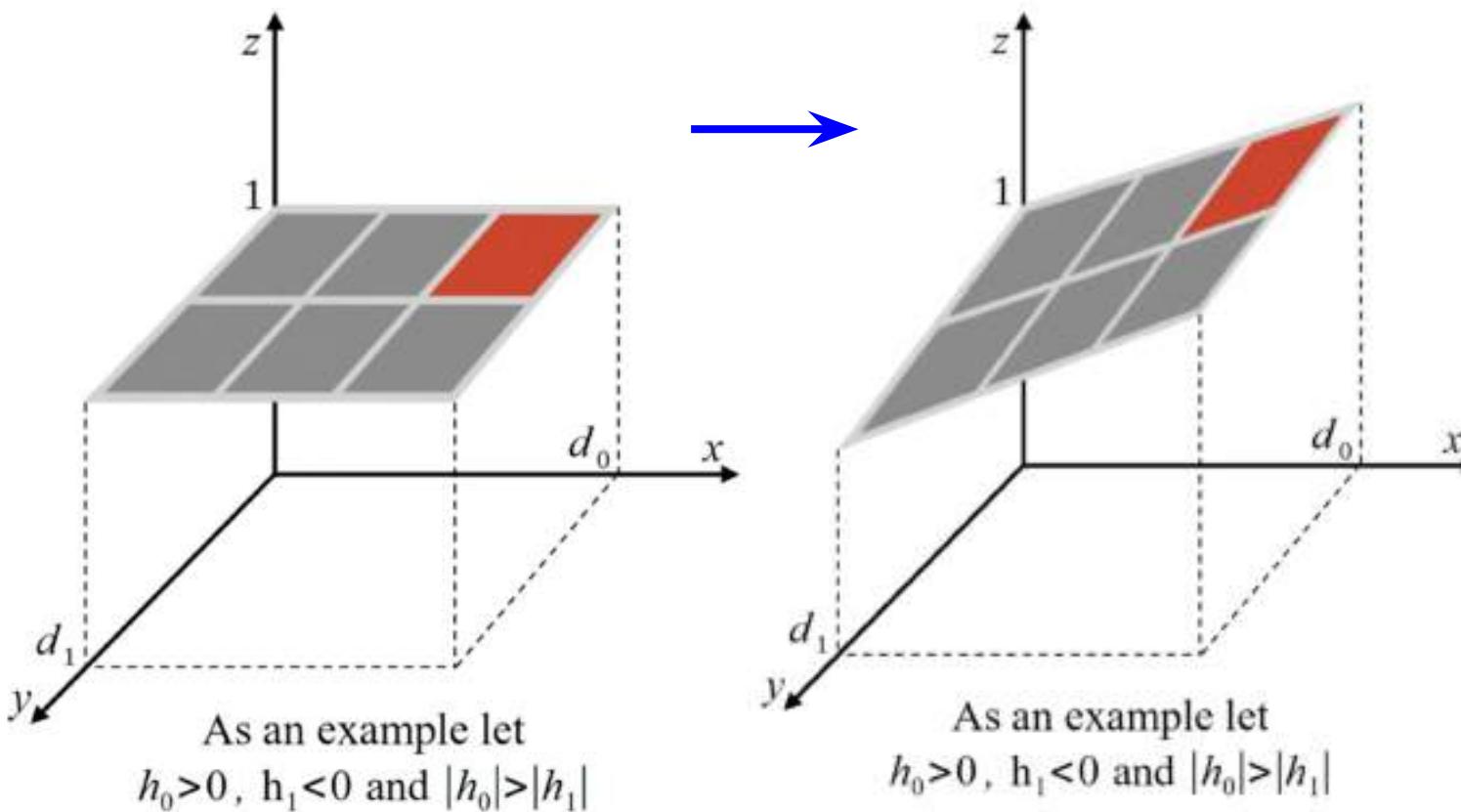


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Projective Transformation (homography)

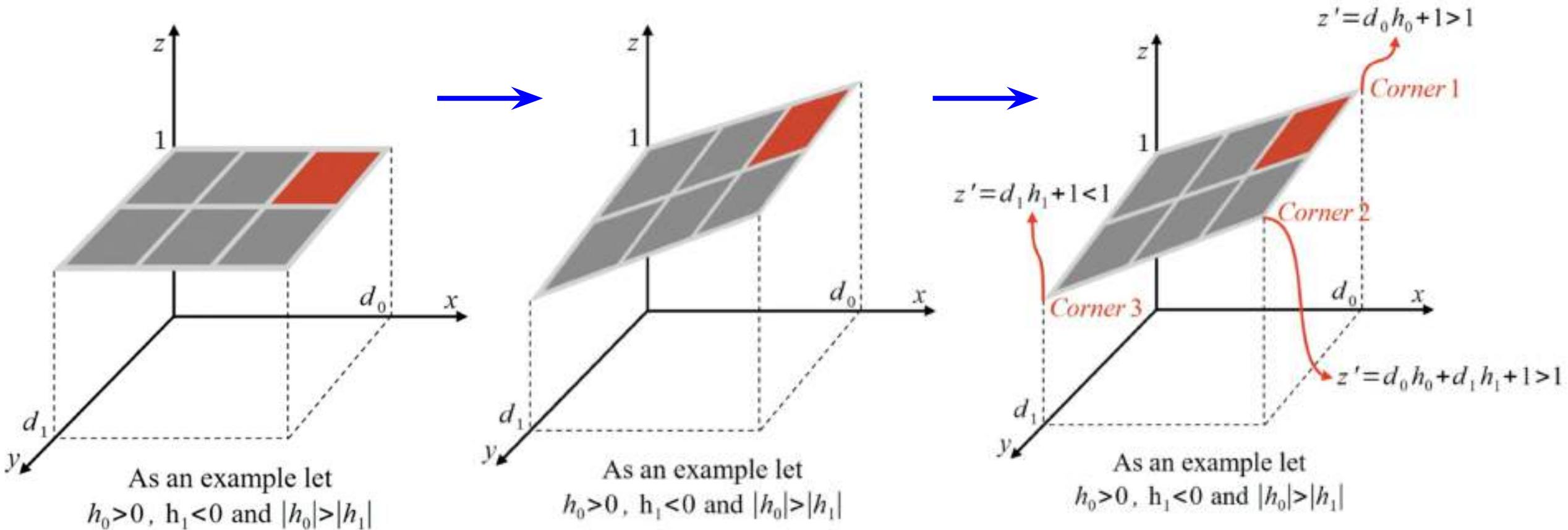


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Projective Transformation (homography)

When going back to  
Cartesian coordinates

$$\frac{x'}{z'}$$

$$\frac{y'}{z'}$$

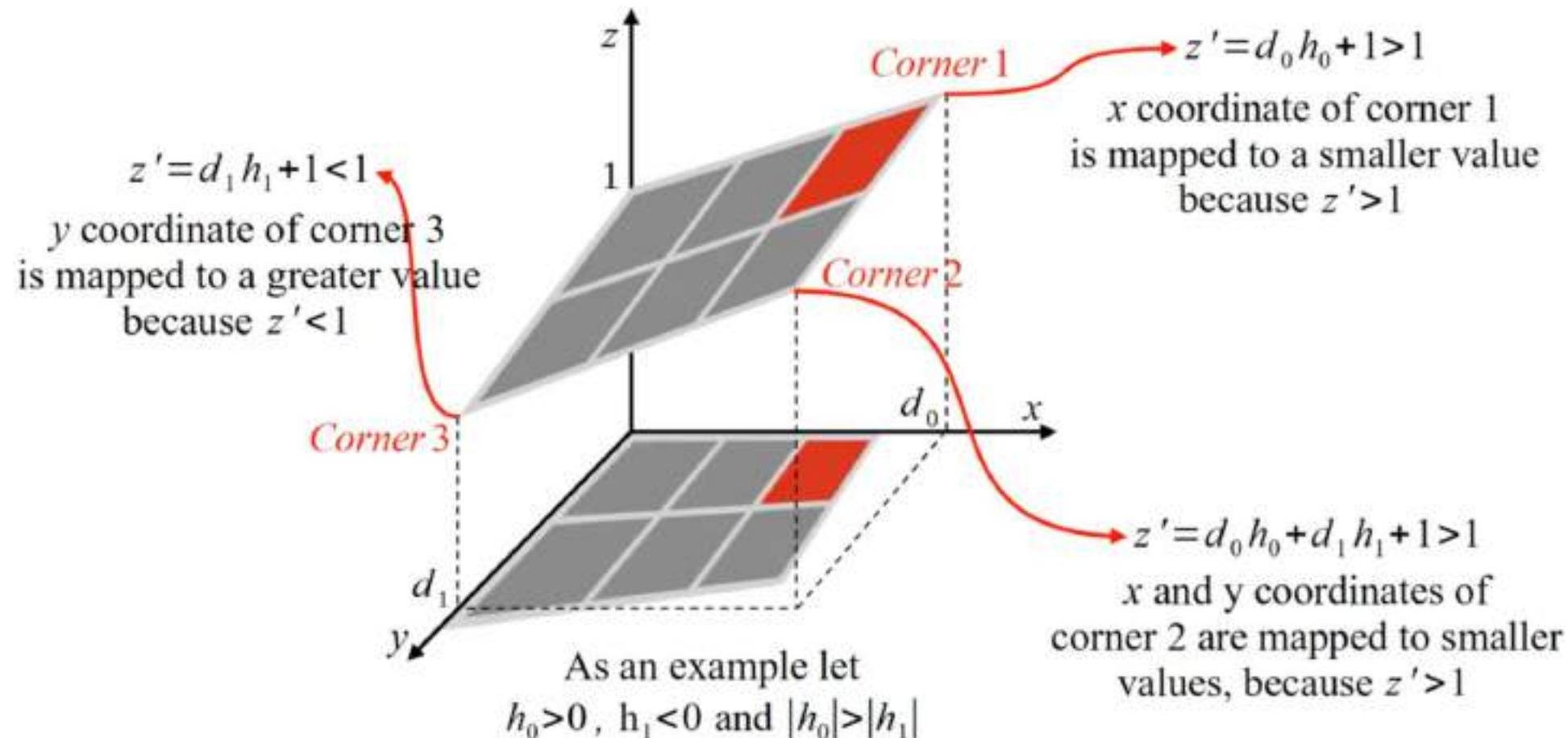


Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

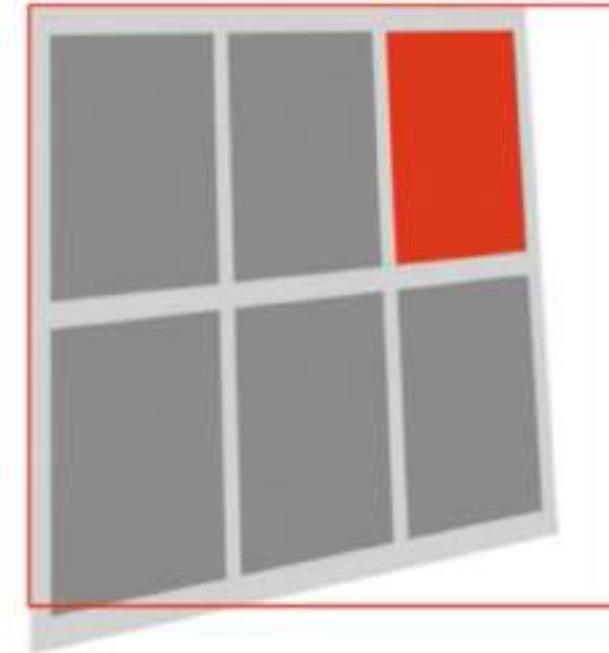
# Projective Transformation (homography)



Original Image

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h_0 & h_1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$h_0 > 0, h_1 < 0 \text{ and } |h_0| > |h_1|$$



Warped Image

Figure: [https://www.youtube.com/@huseyin\\_ozde...](https://www.youtube.com/@huseyin_ozde...)

# Projective Transformation (homography)

Projective transformations are combinations of

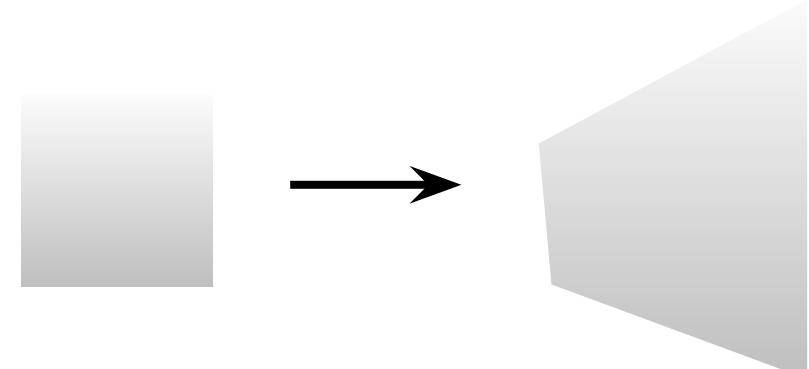
- Affine transformations + projective warps

$$w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} & b_0 \\ A_{10} & A_{11} & b_1 \\ h_0 & h_1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How many degrees of freedom?

Properties of projective transformations:

- origin does not necessarily map to origin
- lines map to lines
- parallel lines do not necessarily map to parallel lines
- ratios are not necessarily preserved



# Questions?

# Composing Transformations

Transformations = Matrices => Composition by Multiplication!

$$p' = R_2 R_1 S p$$

In the example above, the result is equivalent to

$$p' = R_2(R_1(Sp))$$

Equivalent to multiply the matrices into single transformation matrix:

$$p' = (R_2 R_1 S)p$$

Order Matters! Transformations from *right to left*.

# Scaling & Translating != Translating & Scaling

$$p'' = TSp = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix}$$

$$p''' = STp = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + s_x t_x \\ s_y y + s_y t_y \\ 1 \end{bmatrix}$$

# Similarity: Translation + Rotation + Scaling

$$p' = (T R S) p$$

$$p' = TRSp = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= [R \quad t] [S \quad 0] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boxed{[RS \quad t]} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This is the form of the general-purpose transformation matrix

# 2D Transforms = Matrix Multiplication

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

**Table 2.1** *Hierarchy of 2D coordinate transformations, listing the transformation name, its matrix form, the number of degrees of freedom, what geometric properties it preserves, and a mnemonic icon. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The  $2 \times 3$  matrices are extended with a third  $[0^T \ 1]$  row to form a full  $3 \times 3$  matrix for homogeneous coordinate transformations.*

Figure: R. Szeliski

# 3D Transforms = Matrix Multiplication

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	6	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{3 \times 4}$	7	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{3 \times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{4 \times 4}$	15	straight lines	

**Table 2.2** Hierarchy of 3D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The  $3 \times 4$  matrices are extended with a fourth  $[0^T \ 1]$  row to form a full  $4 \times 4$  matrix for homogeneous coordinate transformations. The mnemonic icons are drawn in 2D but are meant to suggest transformations occurring in a full 3D cube.

Figure: R. Szeliski

•  
•  
•  
•  
•  
•  
•

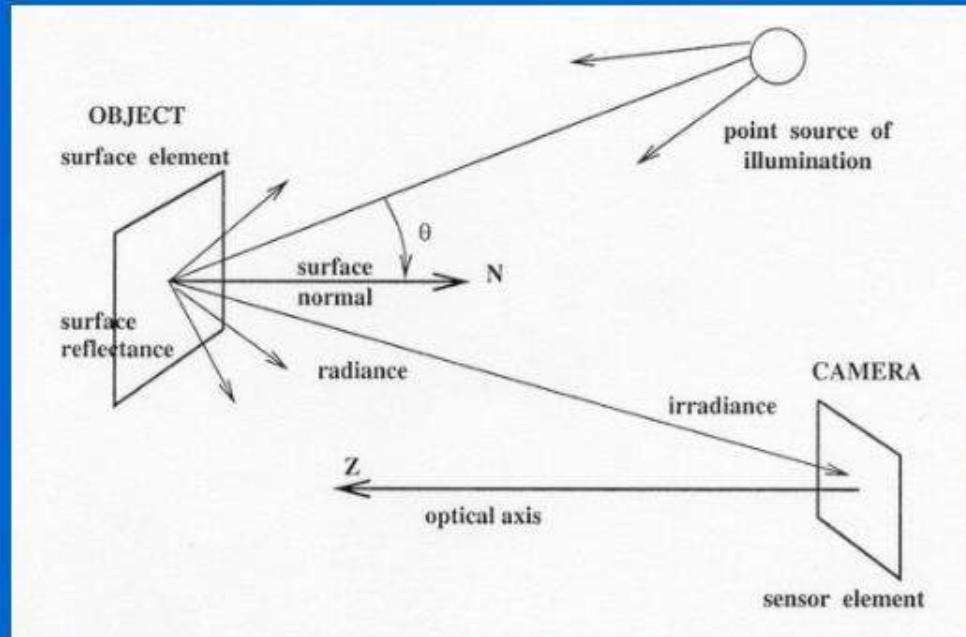
# Image Formation and Representation



**CS485/685 Computer Vision**  
Dr. George Bebis

# A Simple model of image formation

- The scene is illuminated by a single source.
- The scene reflects radiation towards the camera.
- The camera senses it via solid state cells (CCD cameras)



## Image formation (cont'd)

- There are two parts to the image formation process:
  - (1) The **geometry**, which determines where in the image plane the projection of a point in the scene will be located.
  - (2) The **physics of light**, which determines the brightness of a point in the image plane.

Simple model:  $f(x,y) = i(x,y) r(x,y)$

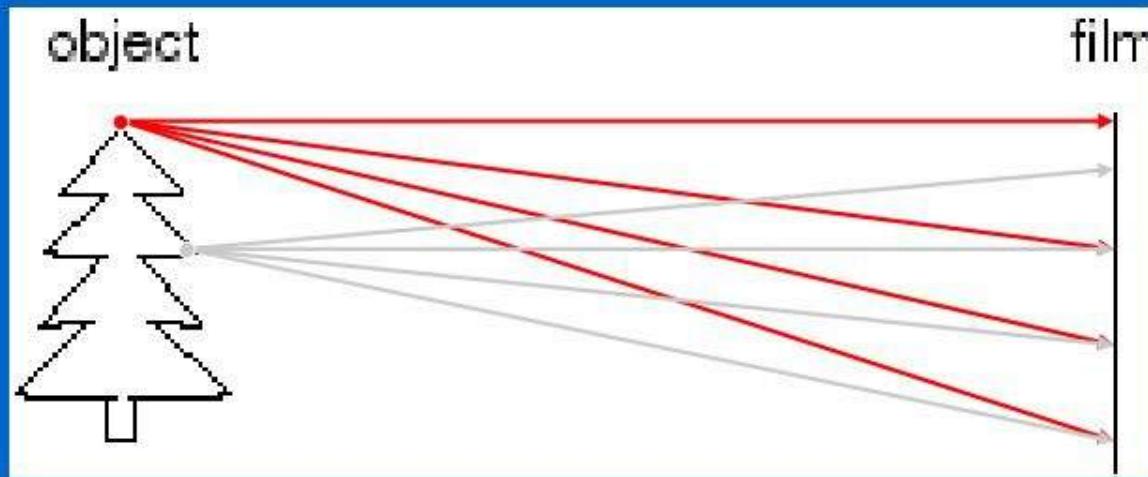
i: illumination, r: reflectance

•

•

•

# Let's design a camera



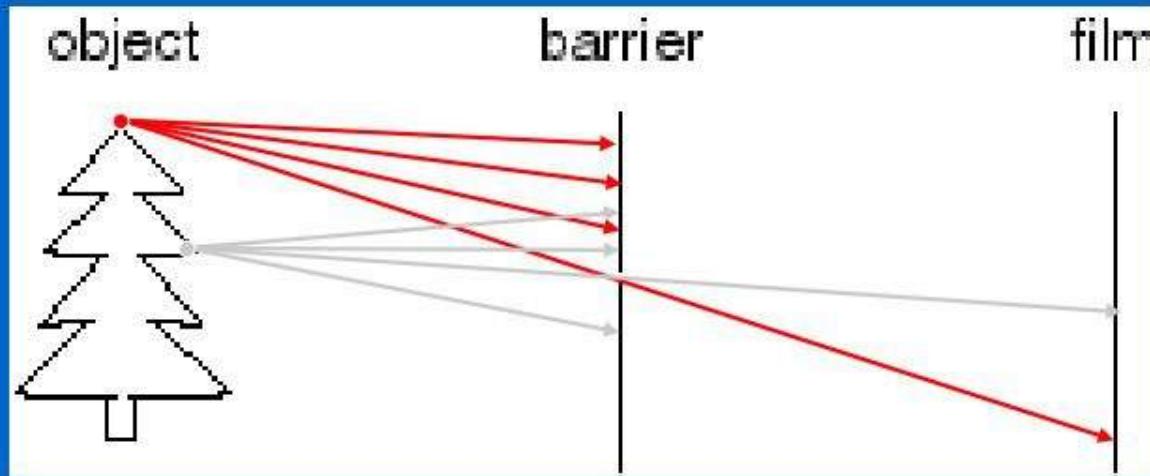
- Put a piece of film in front of an object - do we get a reasonable image?
  - Blurring - need to be more selective!

•

•

•

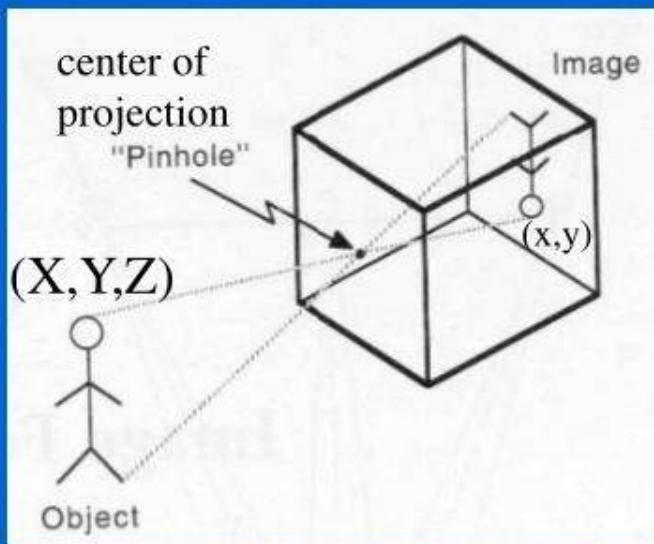
## Let's design a camera (cont'd)



- Add a barrier with a small opening (i.e. **aperture**) to block off most of the rays
  - Reduces blurring

# “Pinhole” camera model

- The simplest device to form an image of a 3D scene on a 2D surface.
- Rays of light pass through a "pinhole" and form an inverted image of the object on the image plane.



**perspective projection:**

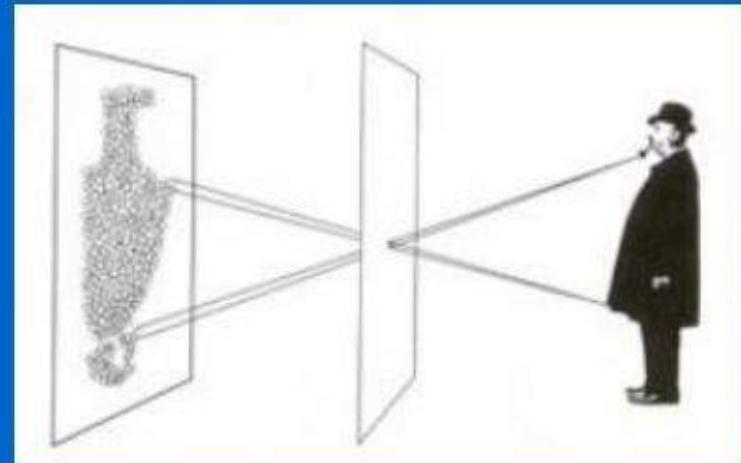
$$x = \frac{fX}{Z} \quad y = \frac{fY}{Z}$$

f: focal length

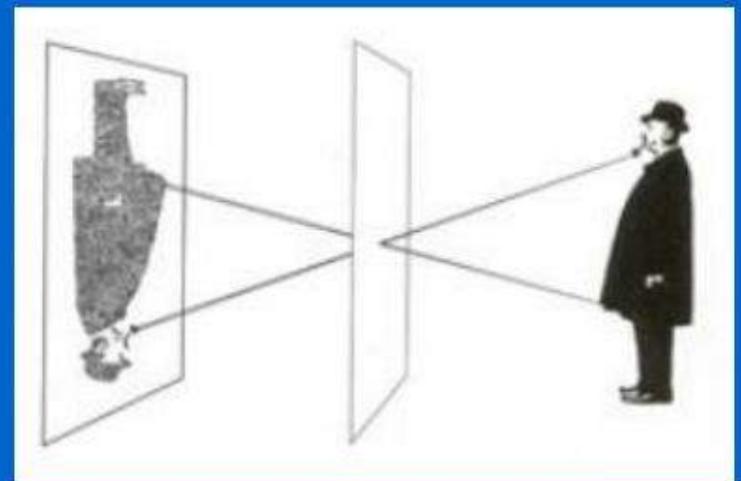
•  
•  
•

# What is the effect of aperture size?

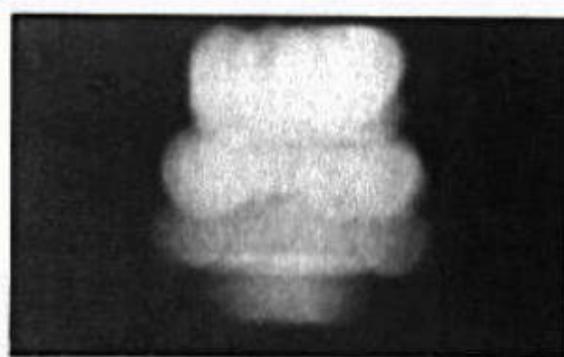
Large aperture: light from the source spreads across the image (i.e., not properly focused), making it **blurry**!



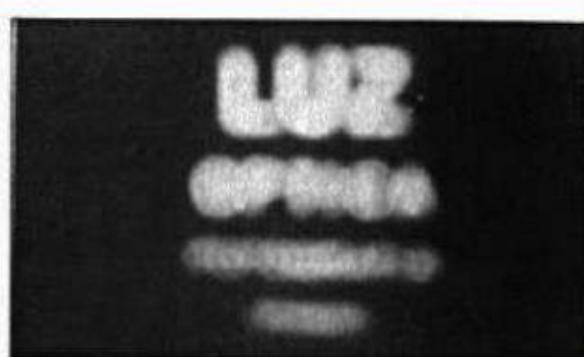
Small aperture: reduces blurring but (i) it limits the amount of light entering the camera and (ii) causes light diffraction.



## Example: varying aperture size



2 mm



1 mm



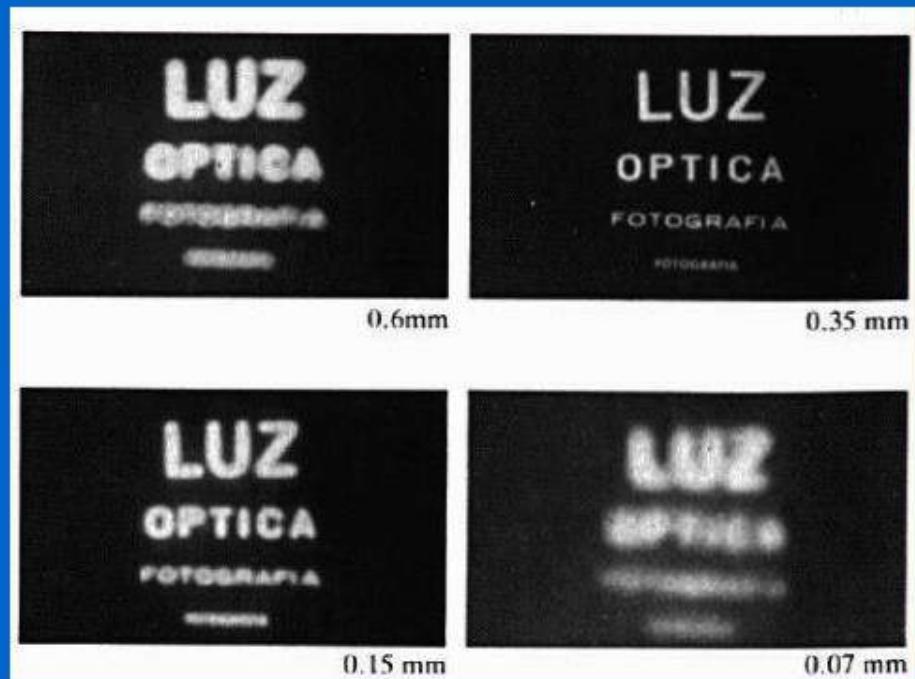
0.6mm



0.35 mm

## Example: varying aperture size (cont'd)

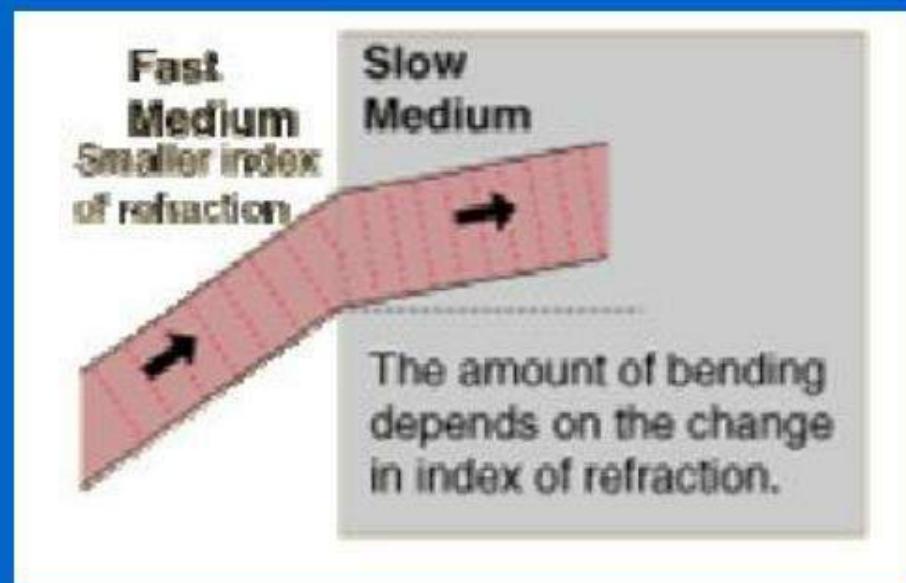
- What happens if we keep decreasing aperture size?
- When light passes through a small hole, it does not travel in a straight line and is scattered in many directions (i.e., **diffraction**)



SOLUTION: **refraction**

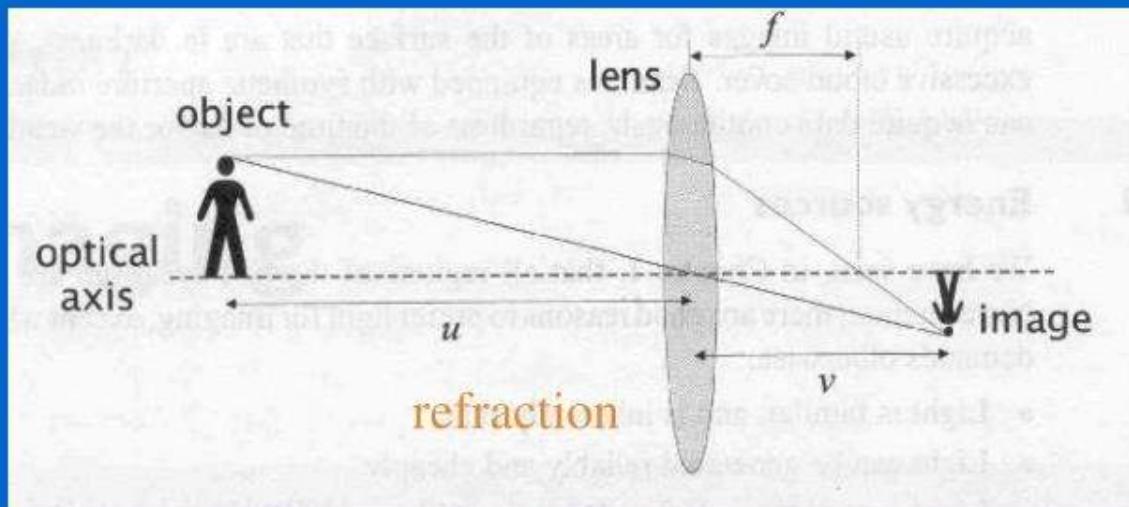
# Refraction

- Bending of wave when it enters a medium where its speed is different.



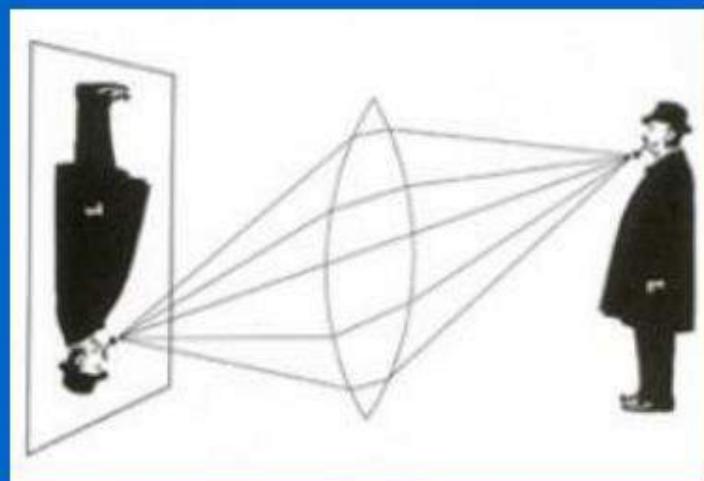
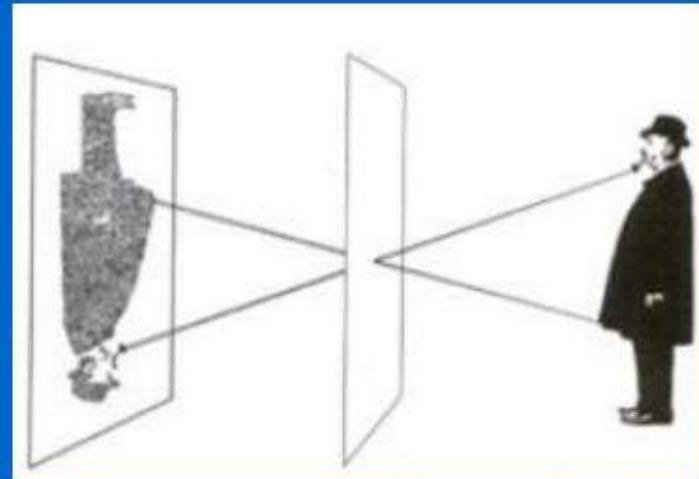
# Lens

- Lens **duplicate** pinhole geometry without resorting to undesirably small apertures.
  - Gather all the light radiating from an object point towards the lens's finite aperture .
  - Bring light into **focus** at a single distinct image point.

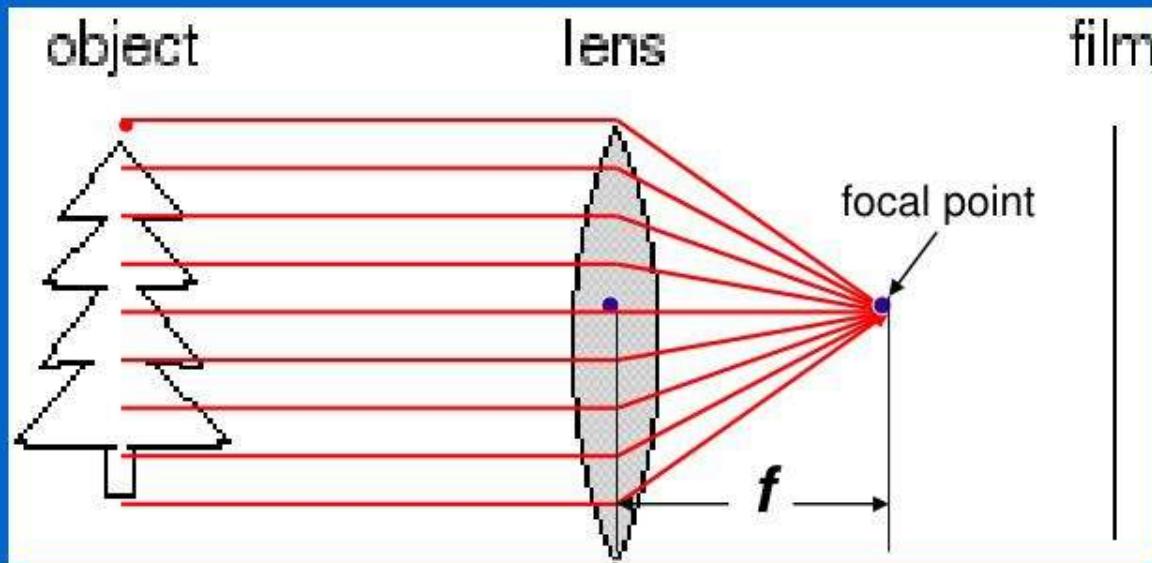


## Lens (cont'd)

- Lens improve image quality, leading to sharper images.



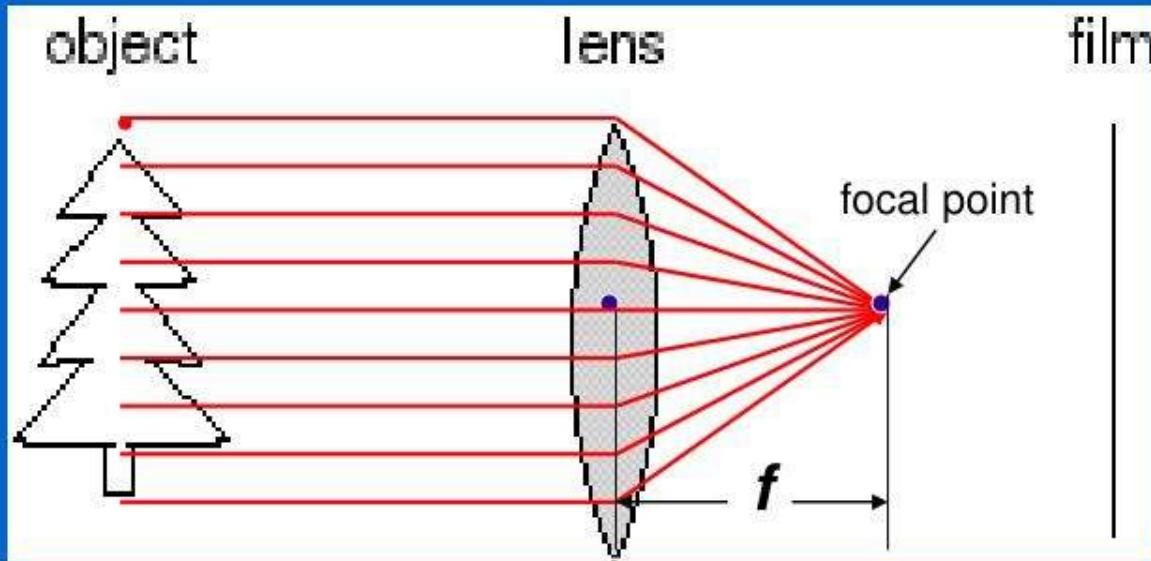
# Properties of “thin” lens (i.e., ideal lens)



- Light rays passing through the center are not deviated.
- Light rays passing through a point far away from the center are deviated more.

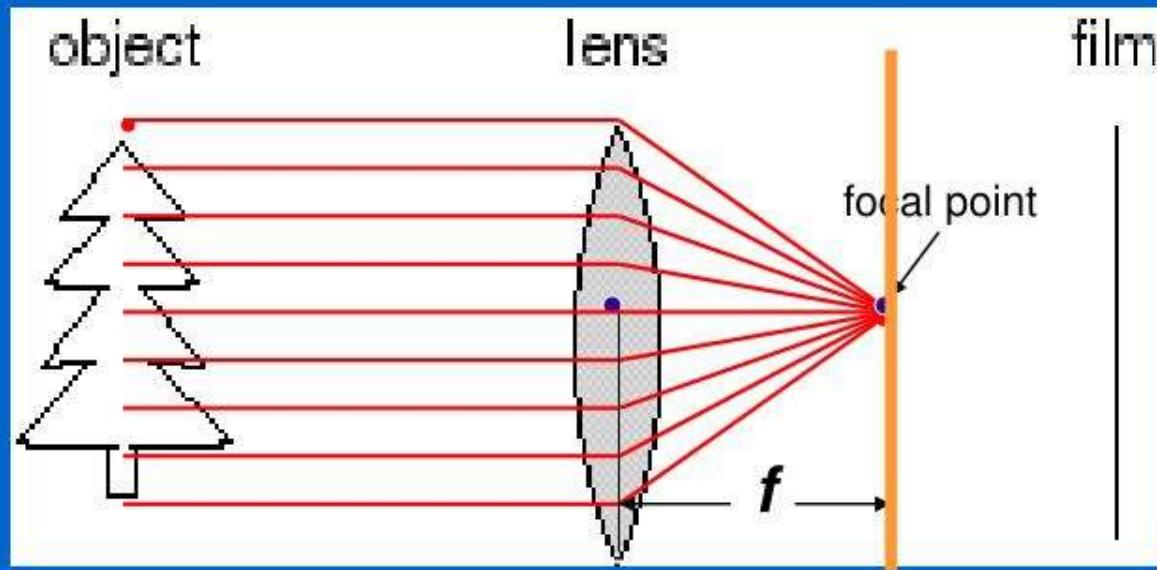
- 
- 
- 

## Properties of “thin” lens (i.e., ideal lens)



- All parallel rays converge to a single point.
- When rays are perpendicular to the lens, it is called focal point.

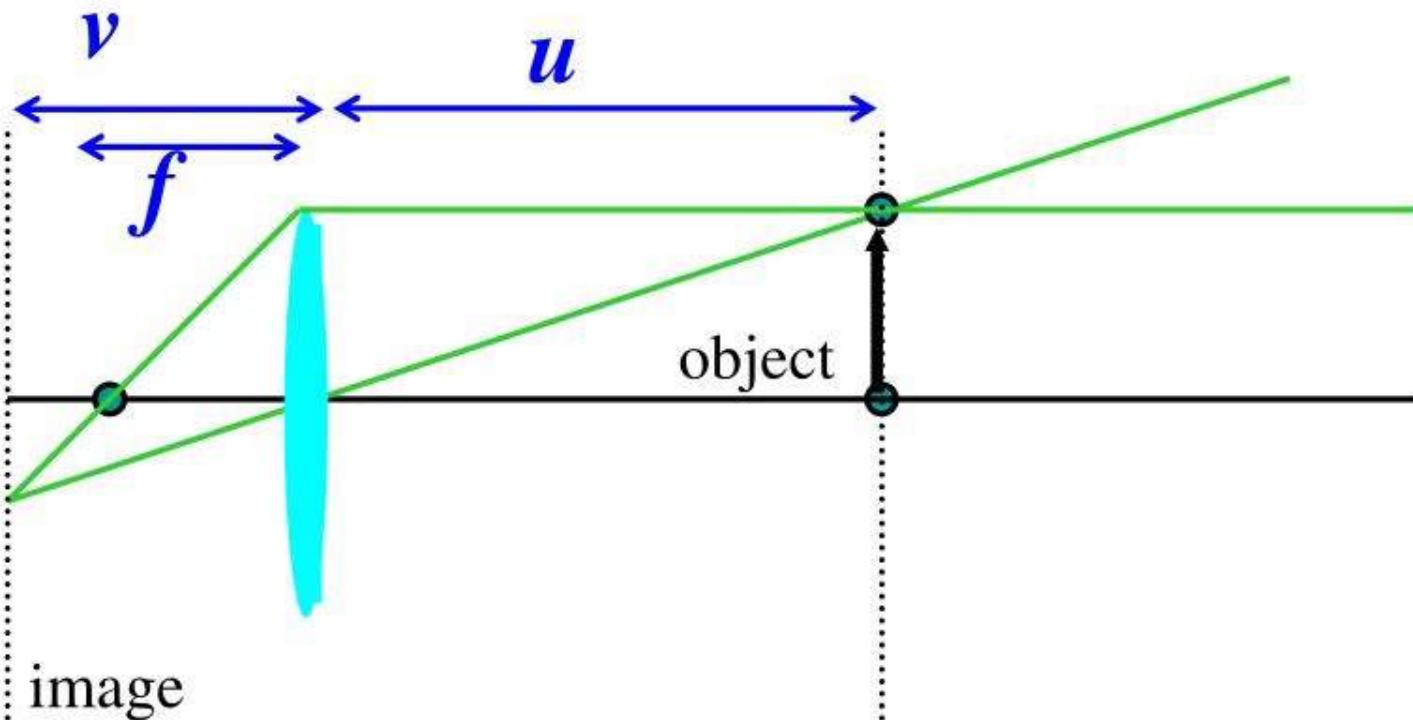
# Properties of “thin” lens



- The plane parallel to the lens at the focal point is called the **focal plane**.
- The distance between the lens and the focal plane is called the **focal length** (i.e.,  $f$ ) of the lens.

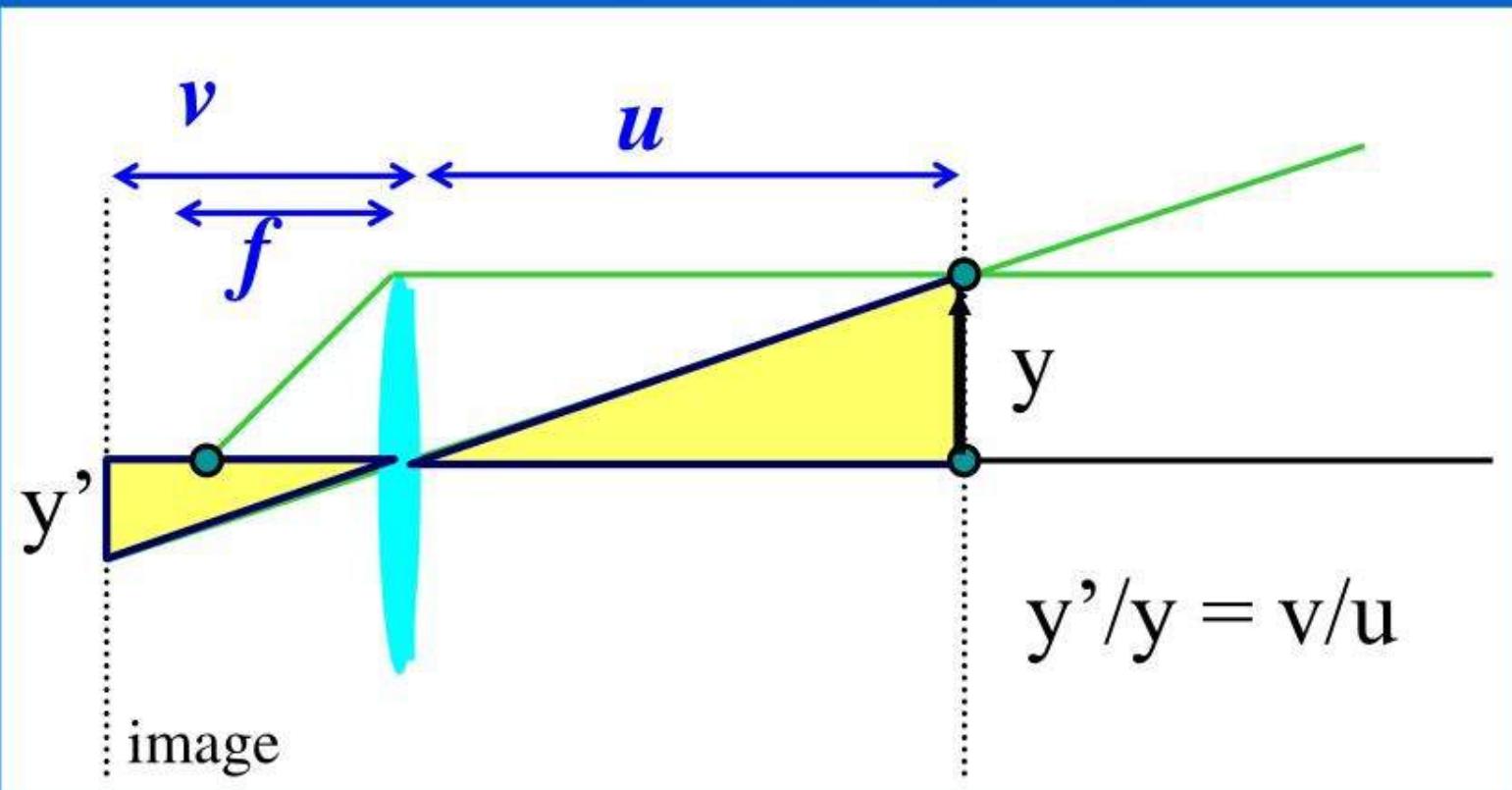
# Thin lens equation

Assume an object at distance  $u$  from the lens plane:



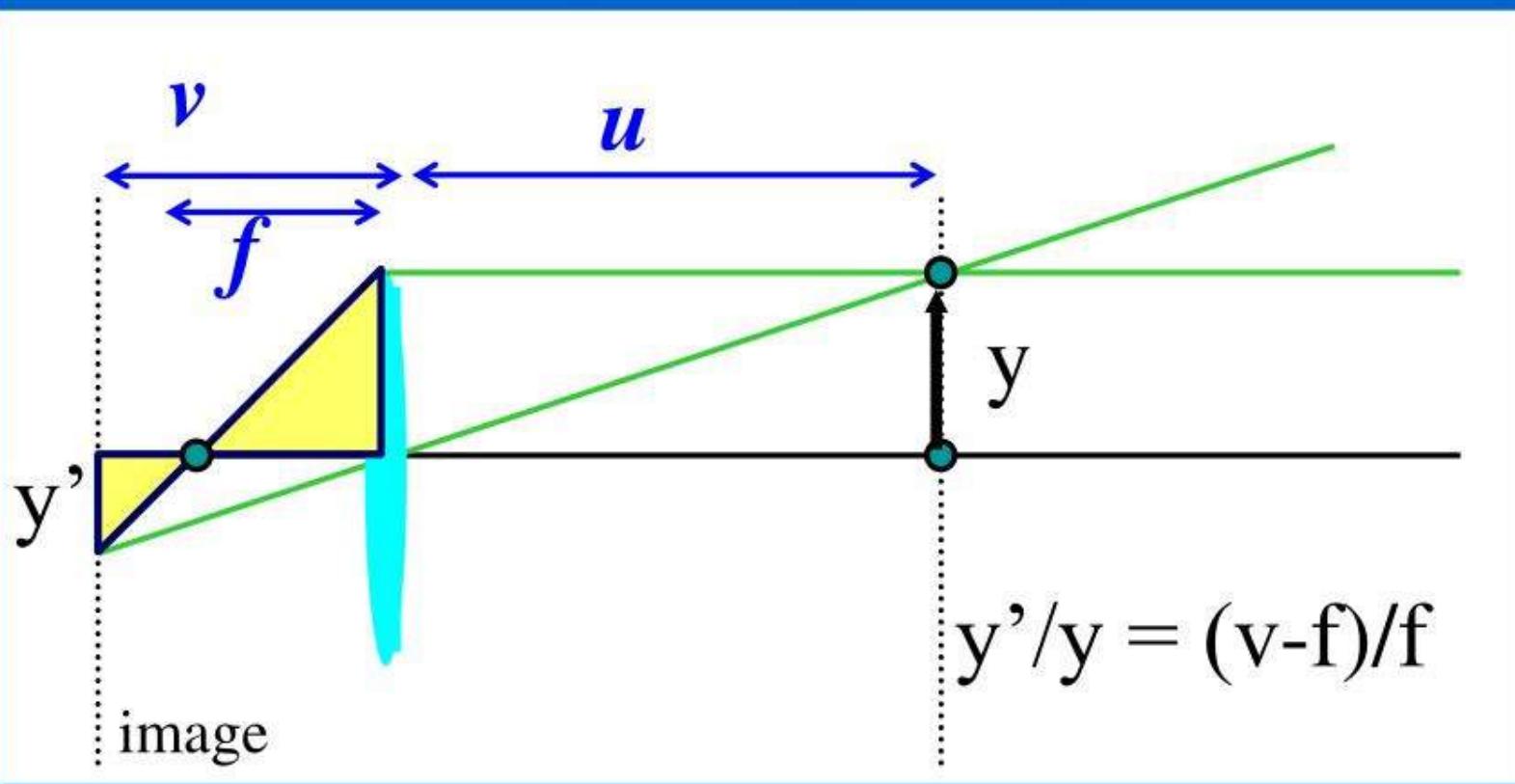
## Thin lens equation (cont'd)

Using similar triangles:



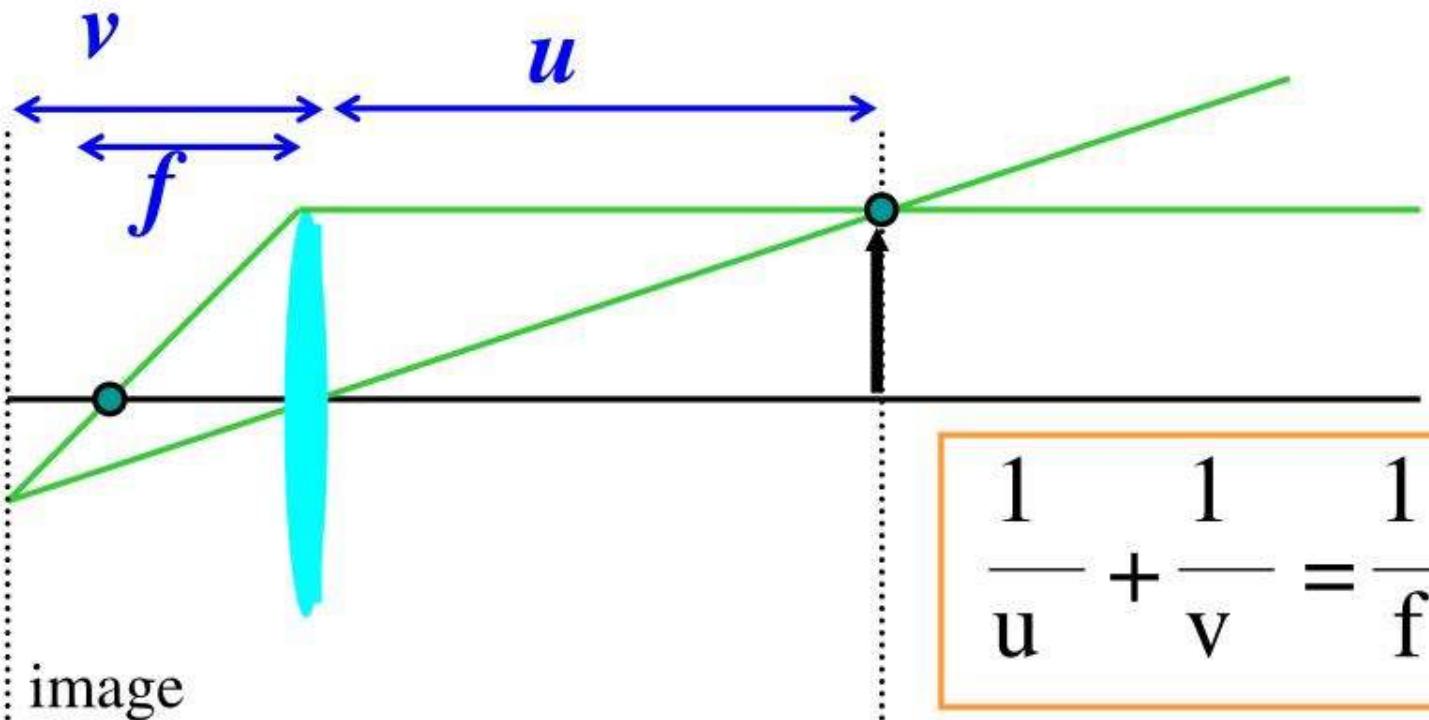
## Thin lens equation (cont'd)

Using similar triangles:

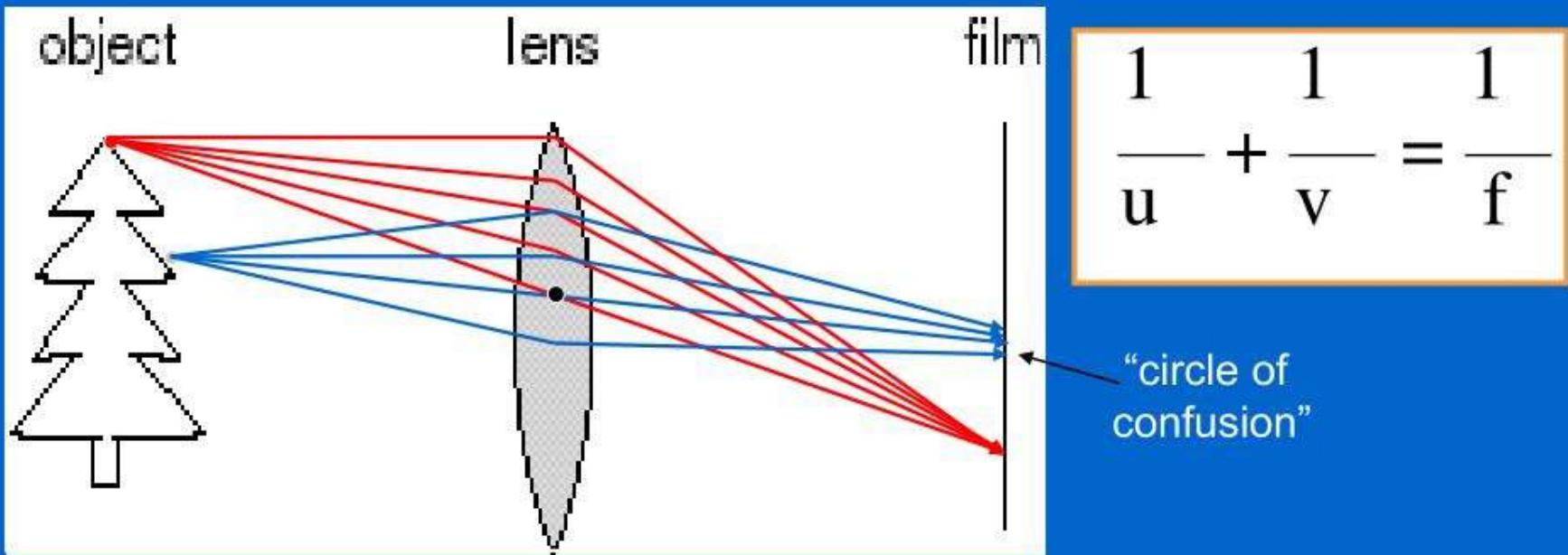


# Thin lens equation (cont'd)

Combining the equations:

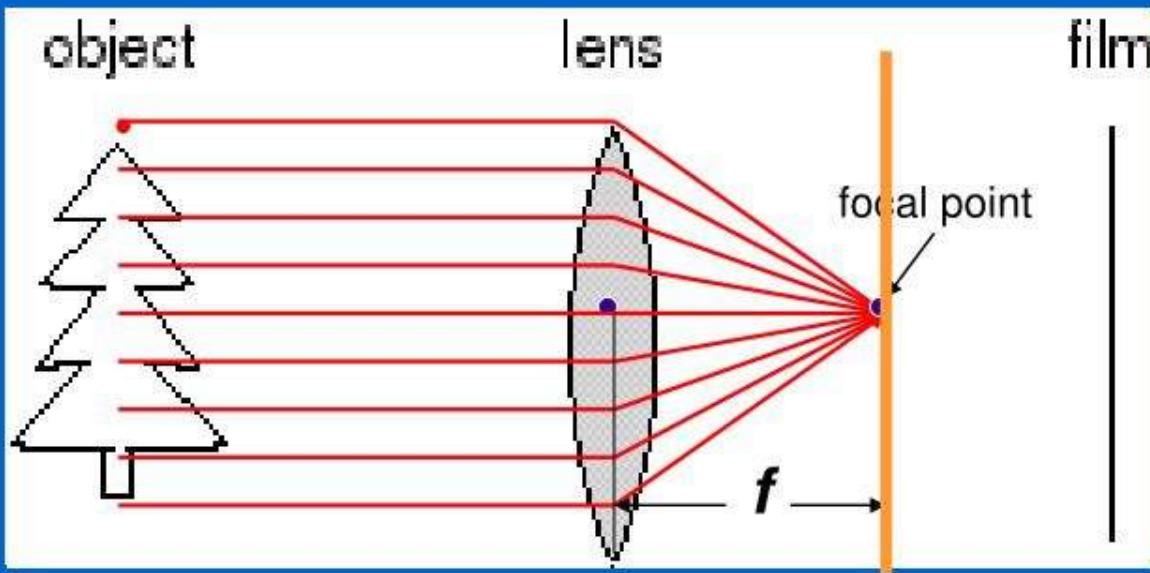


# Thin lens equation (cont'd)



- The thin lens equation implies that only points at distance  $u$  from the lens are “in focus” (i.e., focal point lies on image plane).
- Other points project to a “blur circle” or “circle of confusion” in the image (i.e., blurring occurs).

# Thin lens equation (cont'd)



$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

- When objects move far away from the camera, then the focal plane approaches the image plane.

# Depth of Field

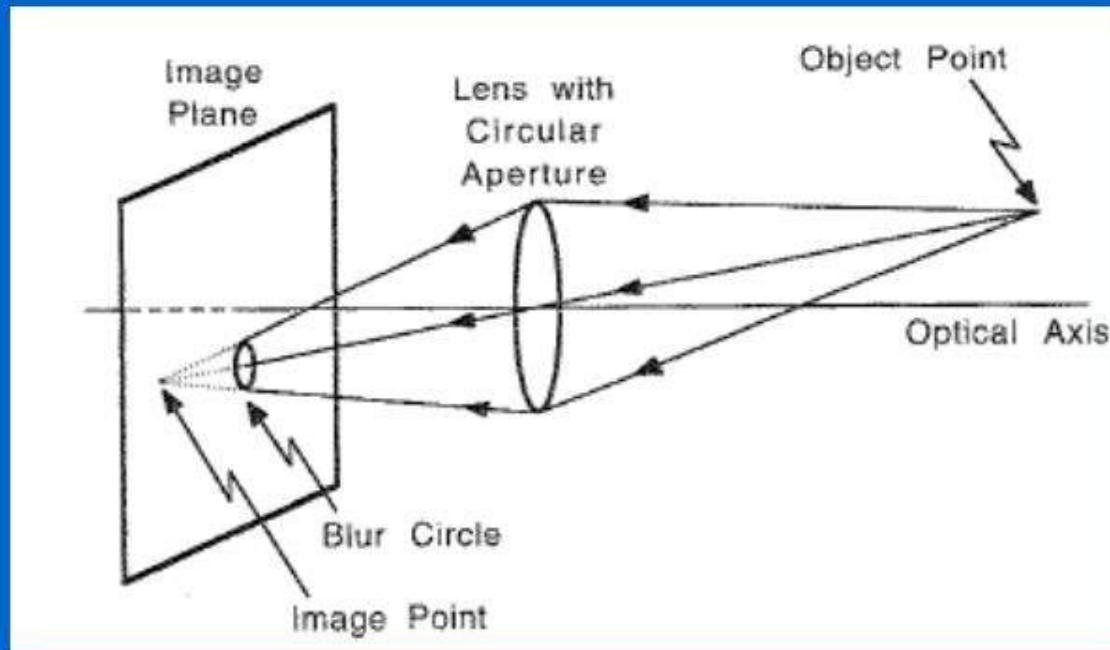
The range of depths over which the world is approximately sharp (i.e., in focus).



<http://www.cambridgeincolour.com/tutorials/depth-of-field.htm>

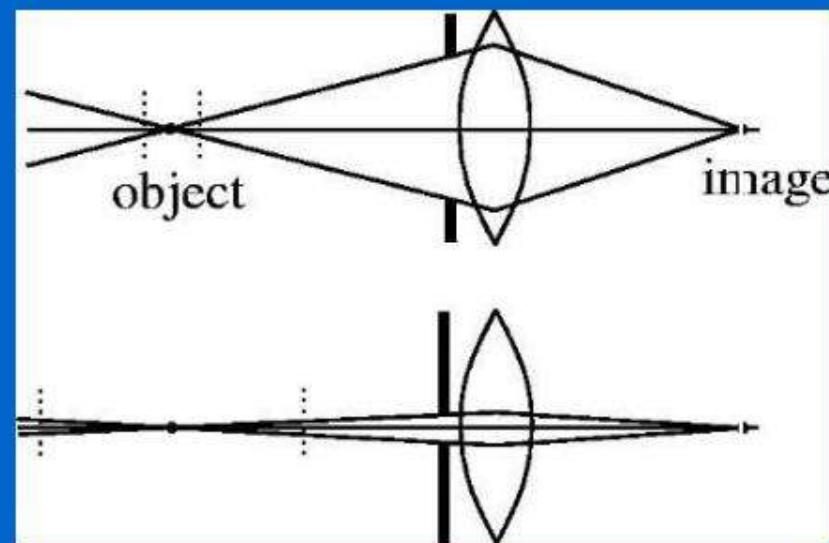
# How can we control depth of field?

- The size of blur circle is proportional to aperture size.



# How can we control depth of field? (cont'd)

- Changing aperture size (controlled by diaphragm) affects depth of field.
  - A smaller aperture increases the range in which an object is approximately in focus (but need to increase **exposure time**).
  - A larger aperture decreases the depth of field (but need to decrease **exposure time**).



# Varying aperture size

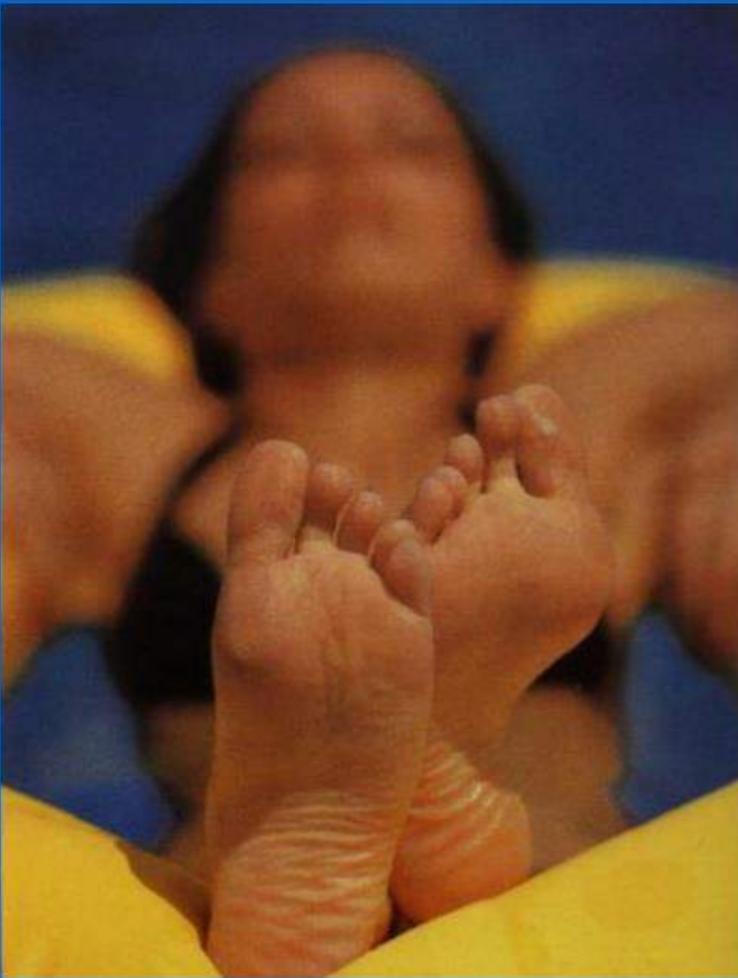


Large aperture = small DOF



Small aperture = large DOF

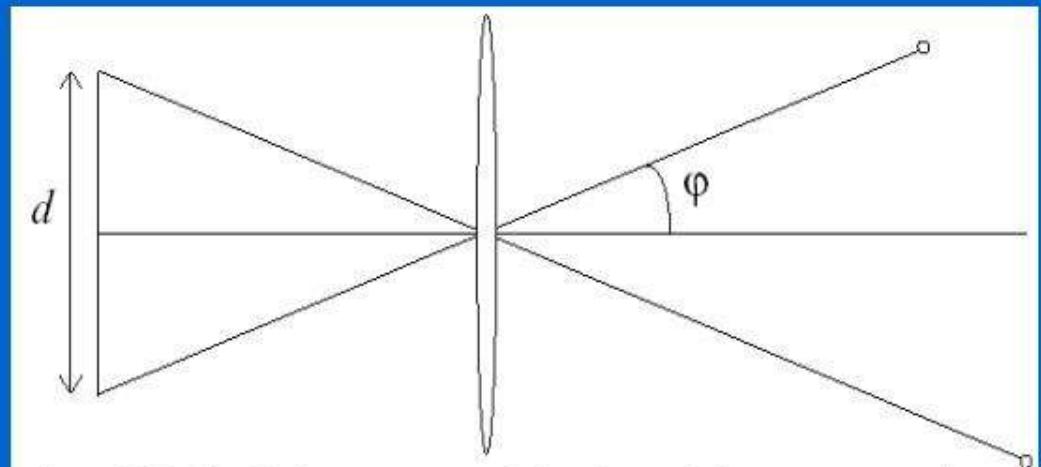
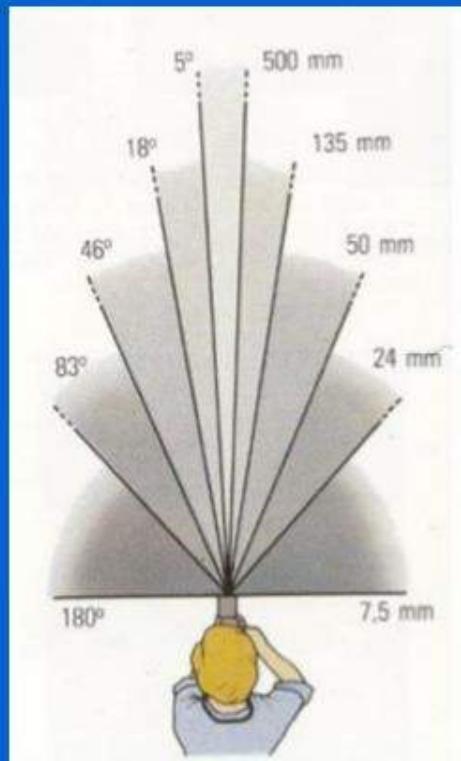
## Another Example



Large aperture = small DOF

# Field of View (Zoom)

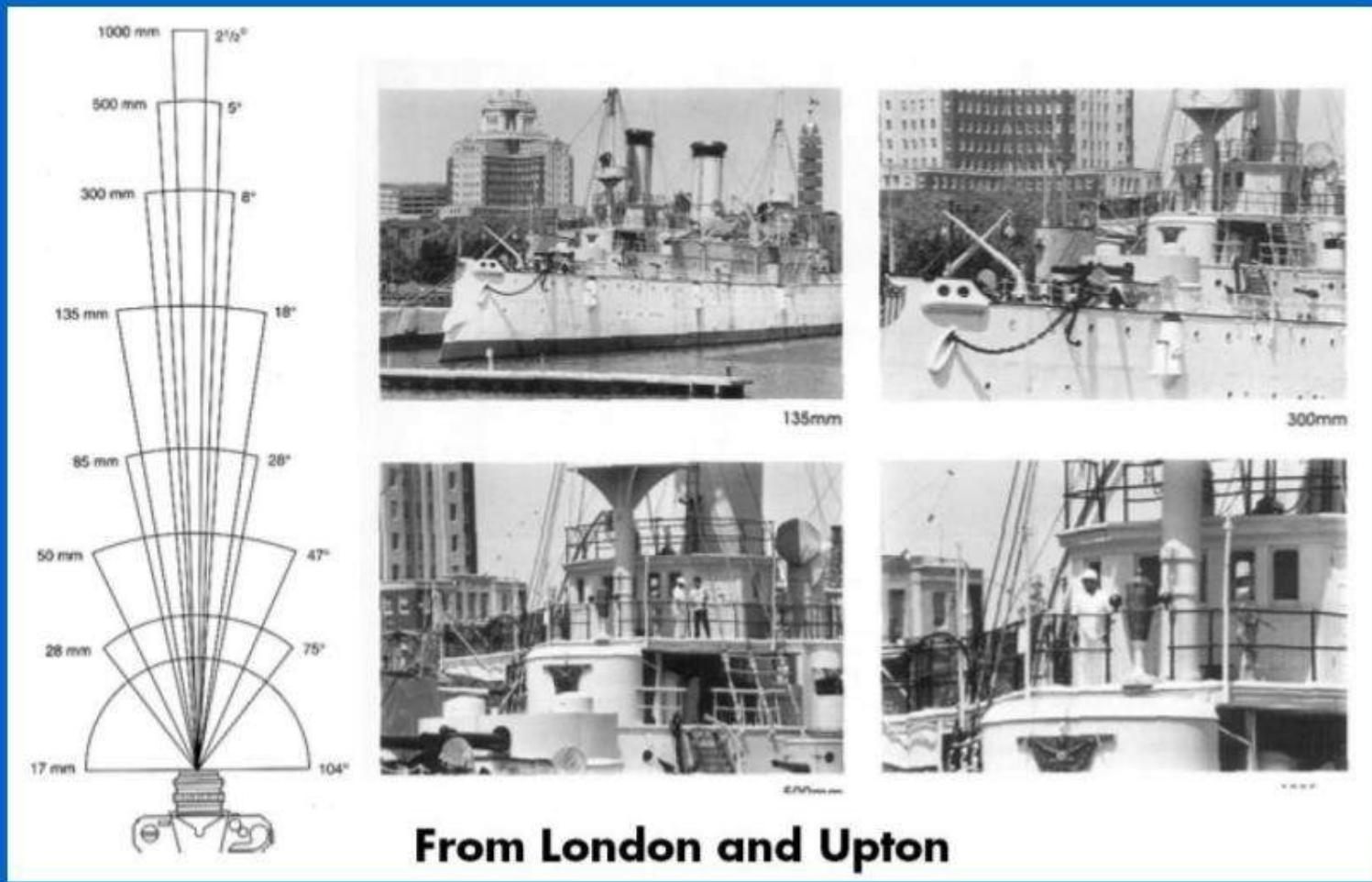
- The cone of viewing directions of the camera.
- Inversely proportional to focal length.



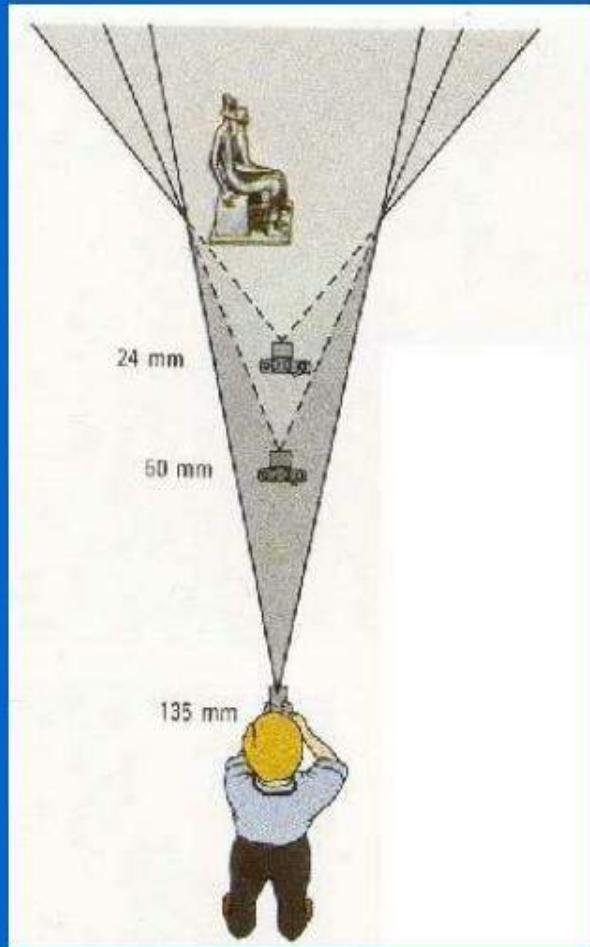
Size of field of view governed by size of the camera retina:

$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

# Field of View (Zoom)



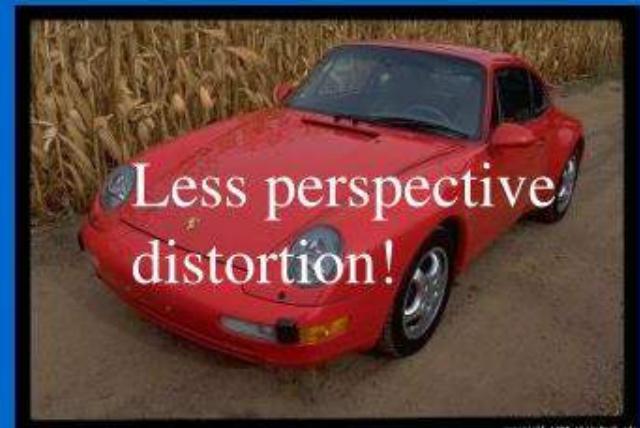
# Reduce Perspective Distortions by varying Distance / Focal Length



Small f (i.e., large FOV),  
camera close to car



Large f (i.e., small FOV),  
camera far from car



Less perspective  
distortion!

- 
- 
- 

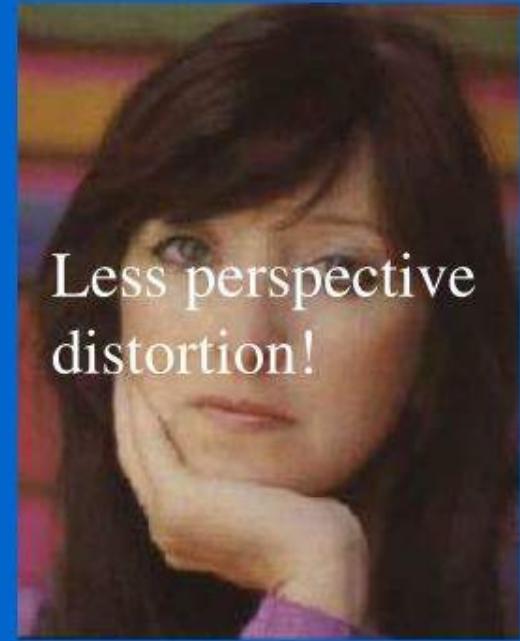
## Same effect for faces



wide-angle



standard



telephoto

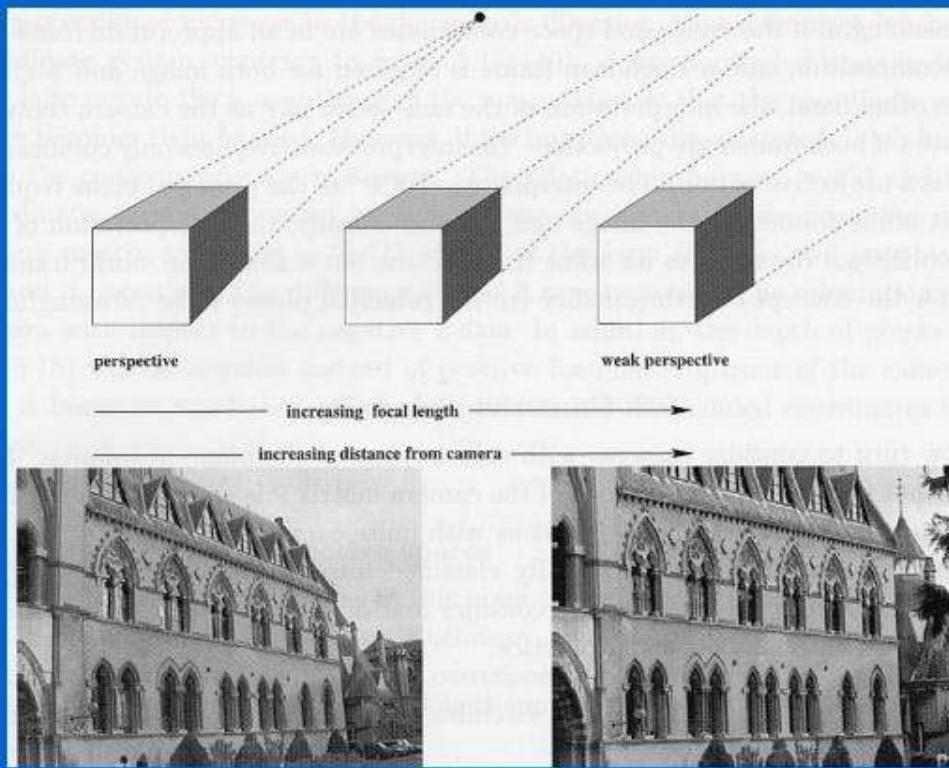
Less perspective distortion!

Practical significance: we can approximate perspective projection using a simpler model when using telephoto lens to view a distant object that has a relatively small range of depth.

- 
- 
- 
- 
- 
-

# Approximating an “affine” camera

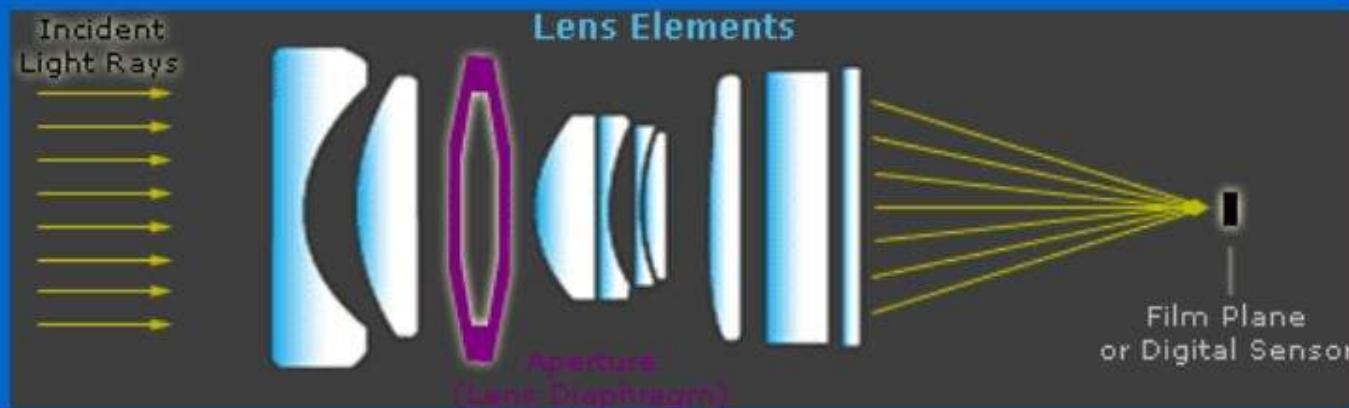
Center of projection is at infinity!



- 
- 
- 

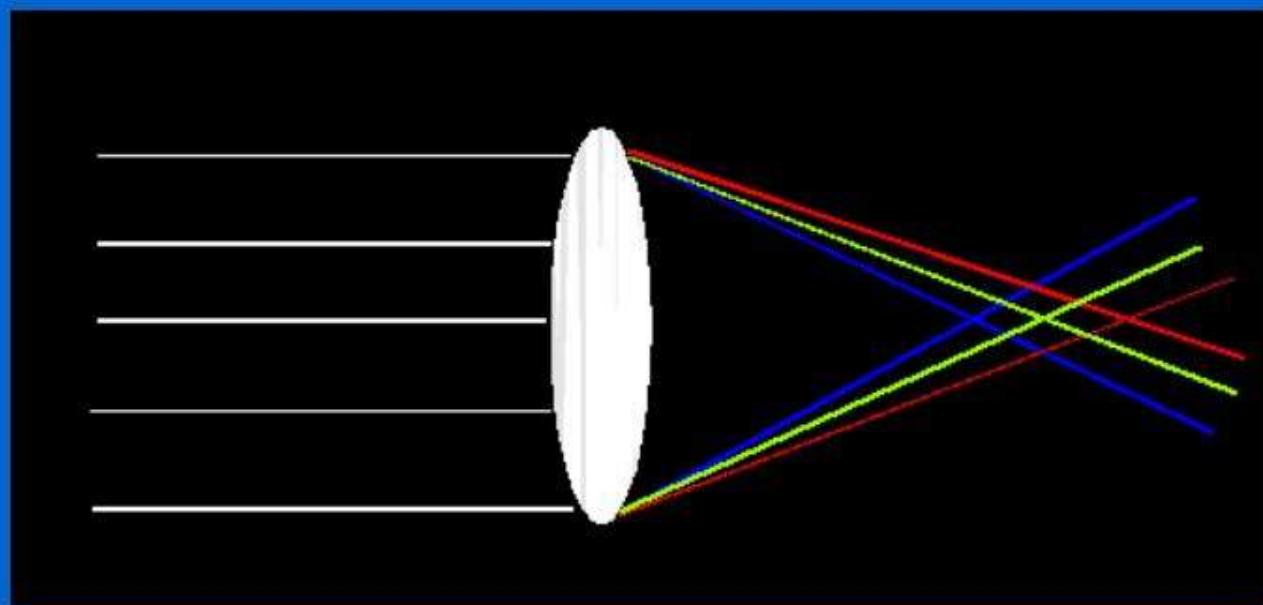
# Real lenses

- All but the simplest cameras contain lenses which are actually comprised of several "lens elements."
- Each element aims to direct the path of light rays such that they recreate the image as accurately as possible on the digital sensor.



# Lens Flaws: Chromatic Aberration

- Lens has different refractive indices for different wavelengths.
- Could cause color **fringing**:
  - i.e., lens cannot focus all the colors at the same point.

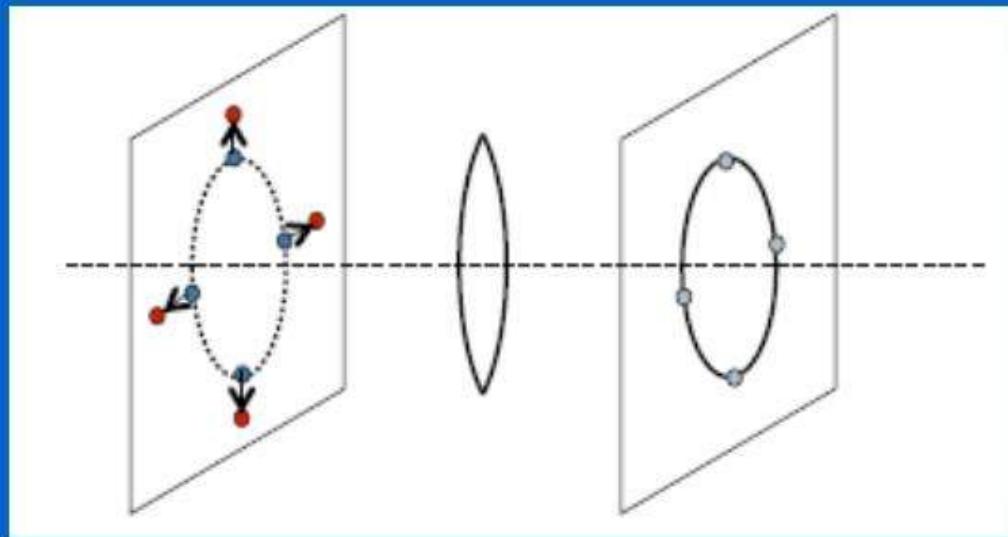


- 
- 
- 

## Chromatic Aberration - Example



# Lens Flaws: Radial Distortion

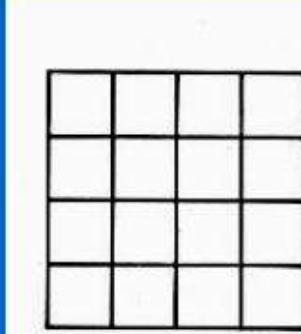


- Straight lines become distorted as we move further away from the center of the image.
- Deviations are most noticeable for rays that pass through the edge of the lens.

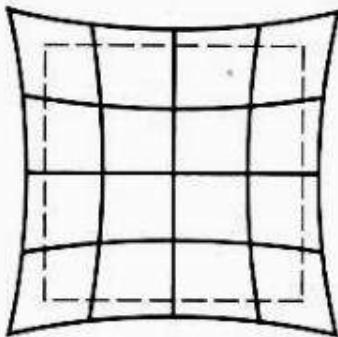
- 
- 
- 

## Lens Flaws: Radial Distortion (cont'd)

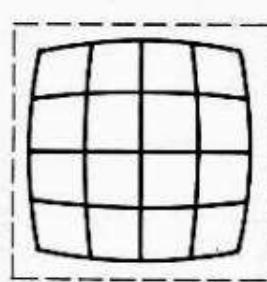
No distortion



Pin cushion

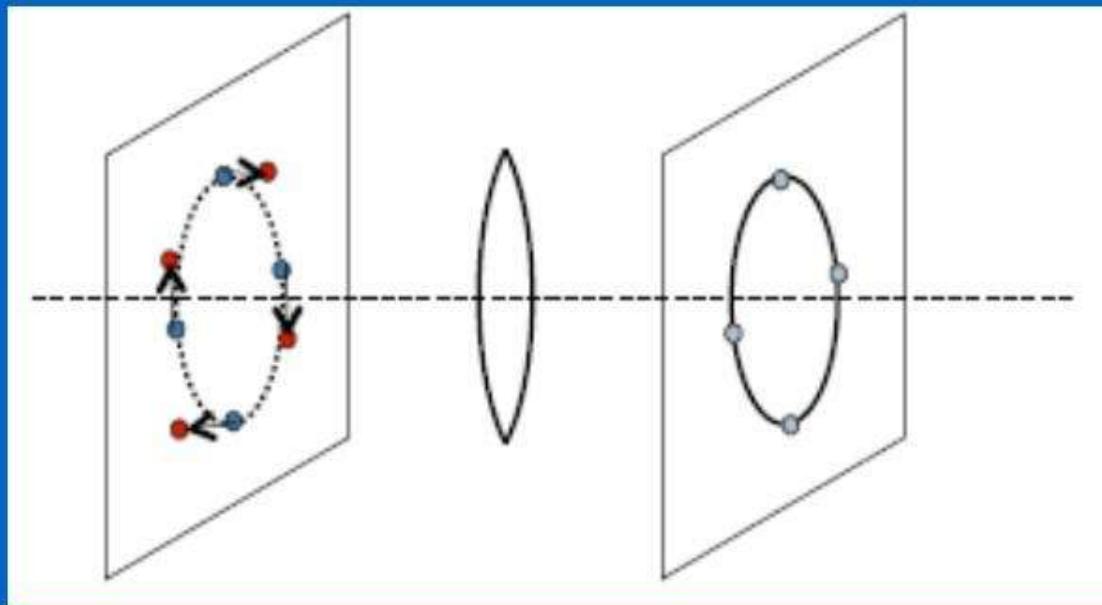


Barrel



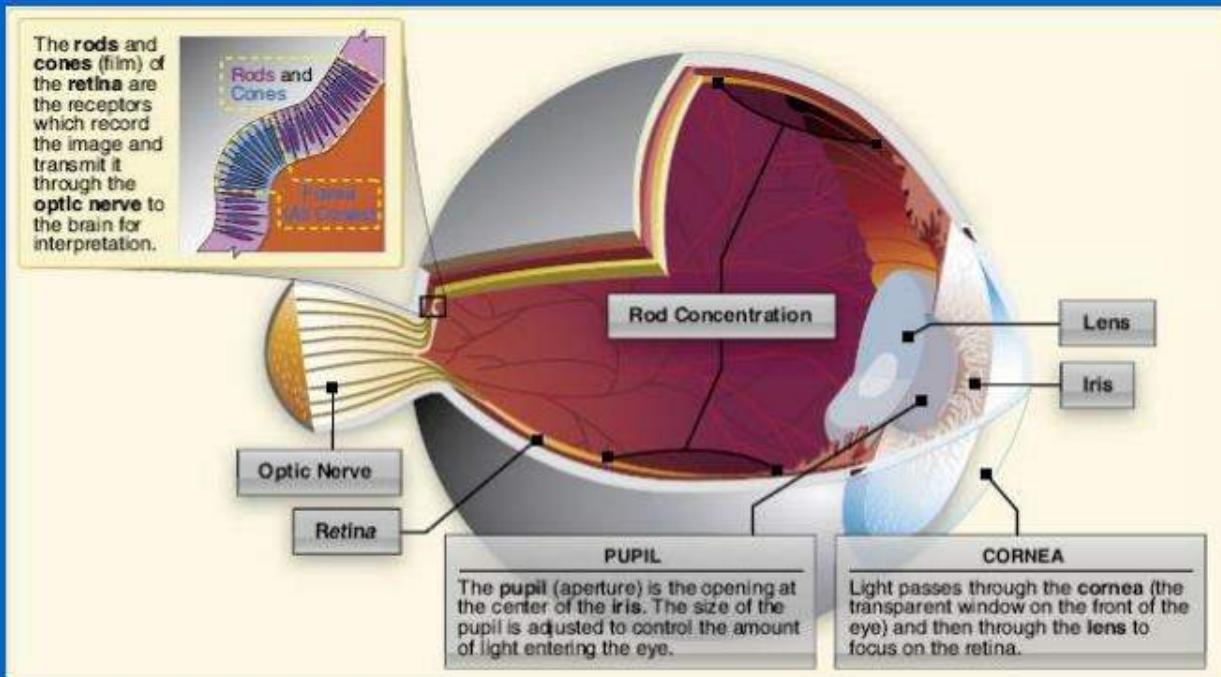
# Lens Flaws: Tangential Distortion

- Lens is not exactly parallel to the imaging plane!



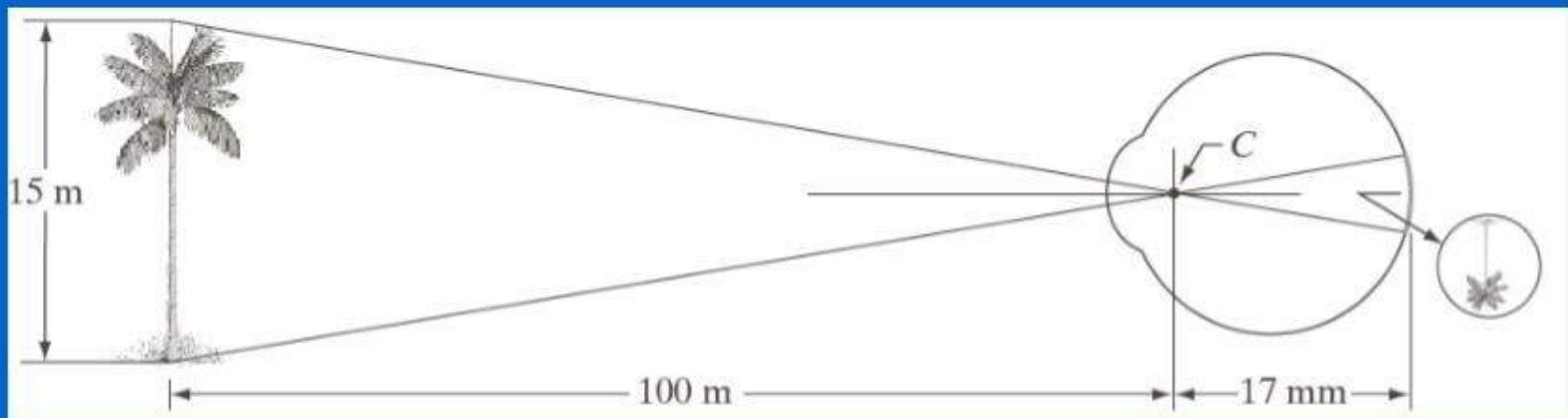
# Human Eye

- Functions much like a camera:
  - aperture (i.e., pupil), lens, mechanism for focusing (zoom in/out) and surface for registering images (i.e., retina)



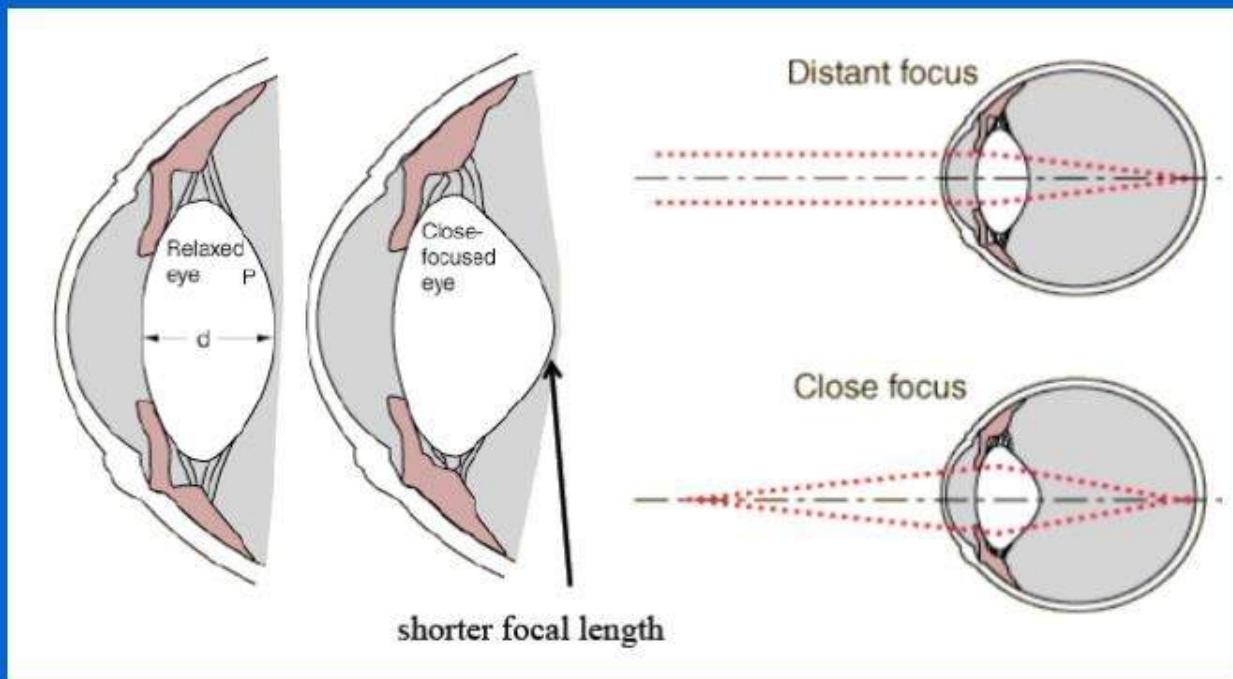
## Human Eye (cont'd)

- In a camera, focusing at various distances is achieved by varying the distance between the lens and the imaging plane.
- In the human eye, the distance between the lens and the retina is fixed (i.e., 14mm to 17mm).



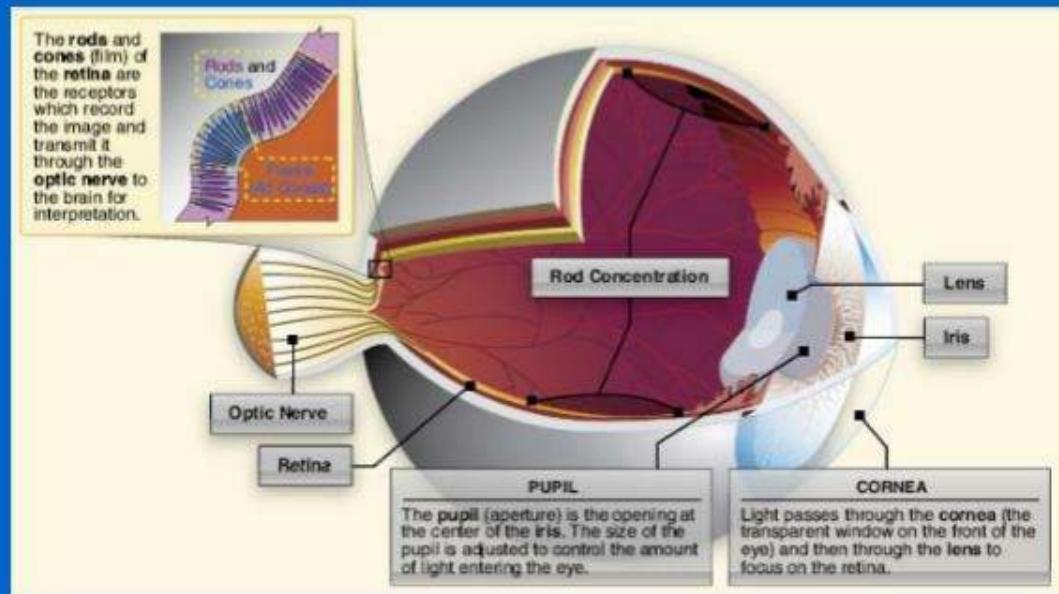
# Human Eye (cont'd)

- Focusing is achieved by varying the shape of the lens (i.e., flattening or thickening).



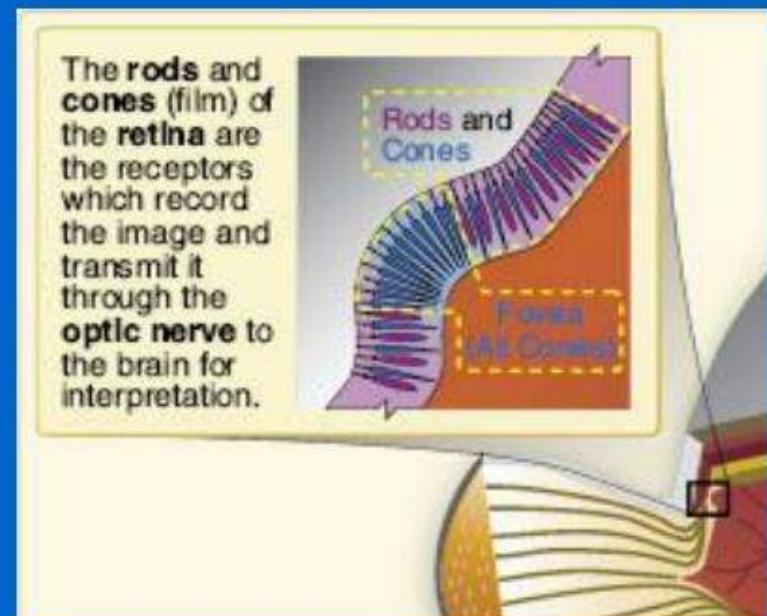
# Human Eye (cont'd)

- Retina contains light sensitive cells that convert light energy into electrical impulses that travel through nerves to the brain.
- Brain interprets the electrical signals to form images.



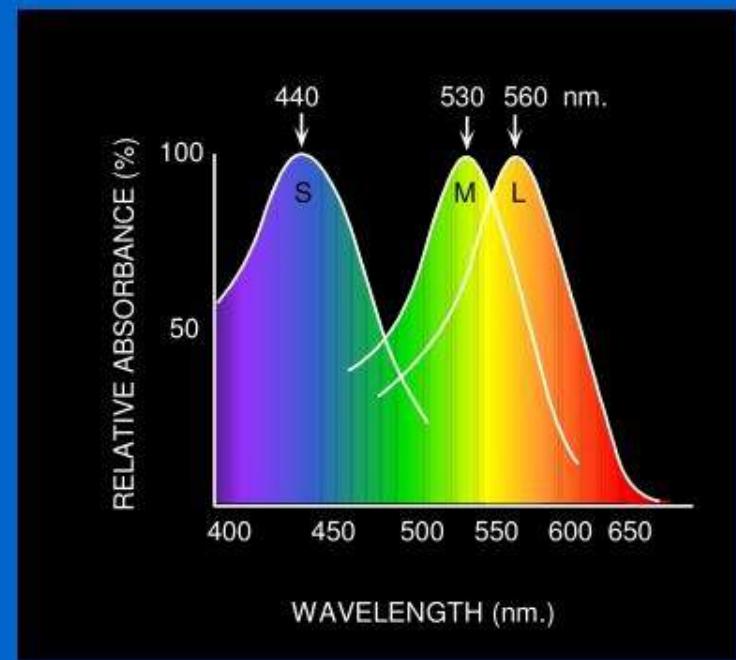
## Human Eye (cont'd)

- Two kinds of light-sensitive cells: rods and cone (unevenly distributed).
- Cones (6 – 7 million) are responsible for all color vision and are present throughout the retina, but are concentrated toward the center of the field of vision at the back of the retina.
- Fovea – special area
  - Mostly cones.
  - Detail, color sensitivity, and resolution are highest.



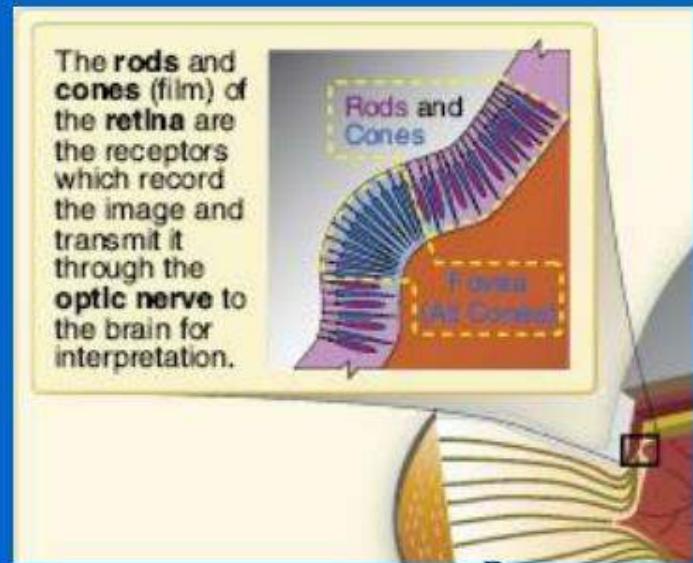
# Human Eye (cont'd)

- Three different types of cones; each type has a special pigment that is sensitive to wavelengths of light in a certain range:
  - Short (S) corresponds to blue
  - Medium (M) corresponds to green
  - Long (L) corresponds to red
- Ratio of L to M to S cones:
  - approx. 10:5:1
- Almost no S cones in the center of the fovea



## Human Eye (cont'd)

- Rods (120 million) more sensitive to light than cones but cannot discern color.
  - Primary receptors for night vision and detecting motion.
  - Large amount of light overwhelms them, and they take a long time to “reset” and adapt to the dark again.
  - Once fully adapted to darkness, the rods are 10,000 times more sensitive to light than the cones



- 
- 
- 

# Digital cameras

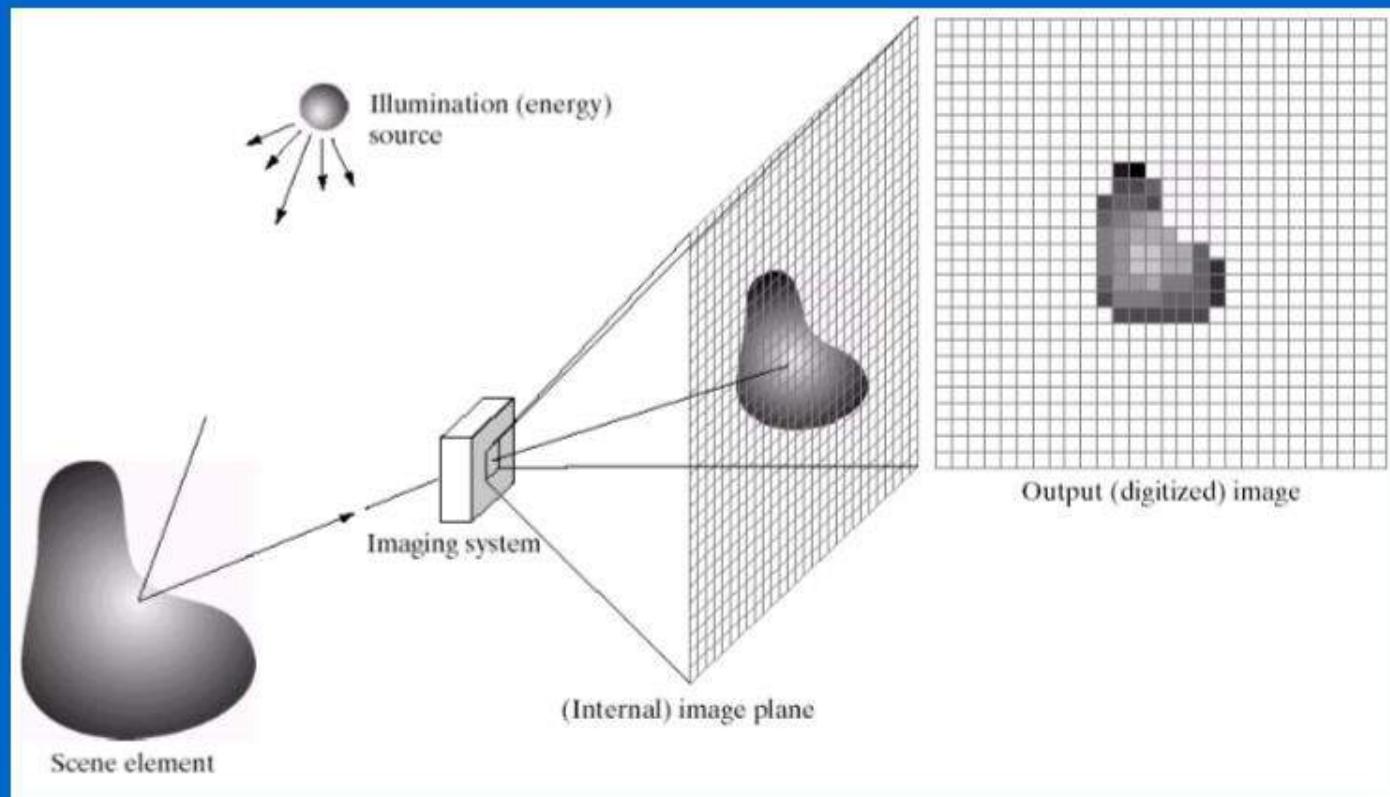
- A digital camera replaces film with a sensor array.
  - Each cell in the array is light-sensitive diode that converts photons to electrons
  - Two common types
    - Charge Coupled Device (CCD)
    - Complementary metal oxide semiconductor (CMOS)



<http://electronics.howstuffworks.com/digital-camera.htm>

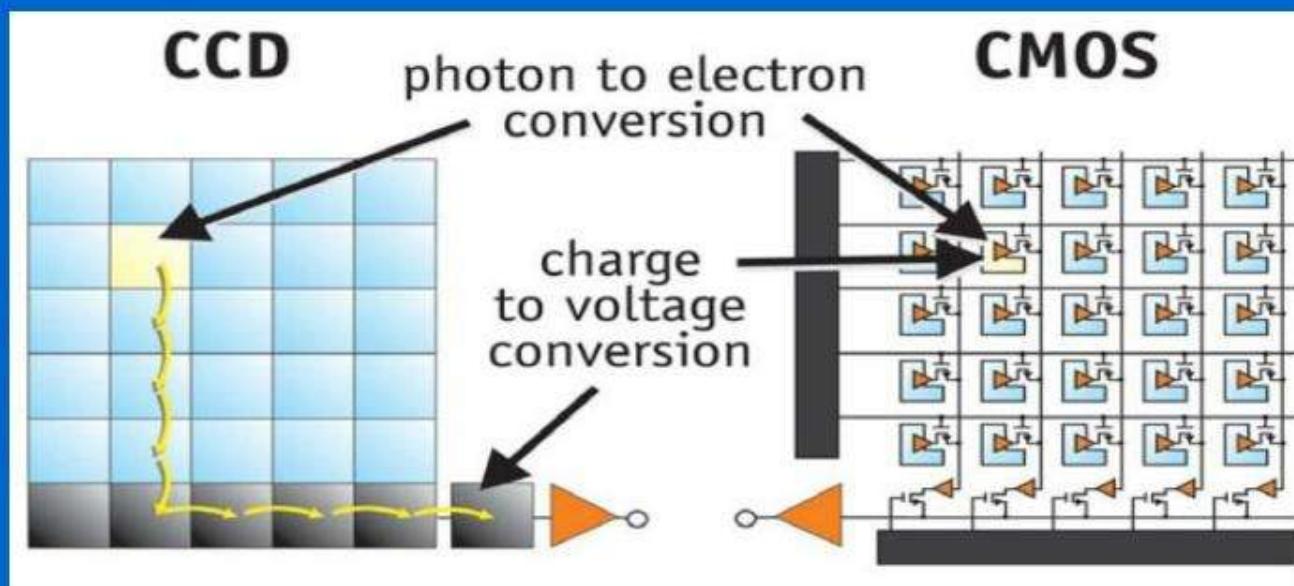
- 
- 
- 
- 
-

# Digital cameras (cont'd)



# CCD Cameras

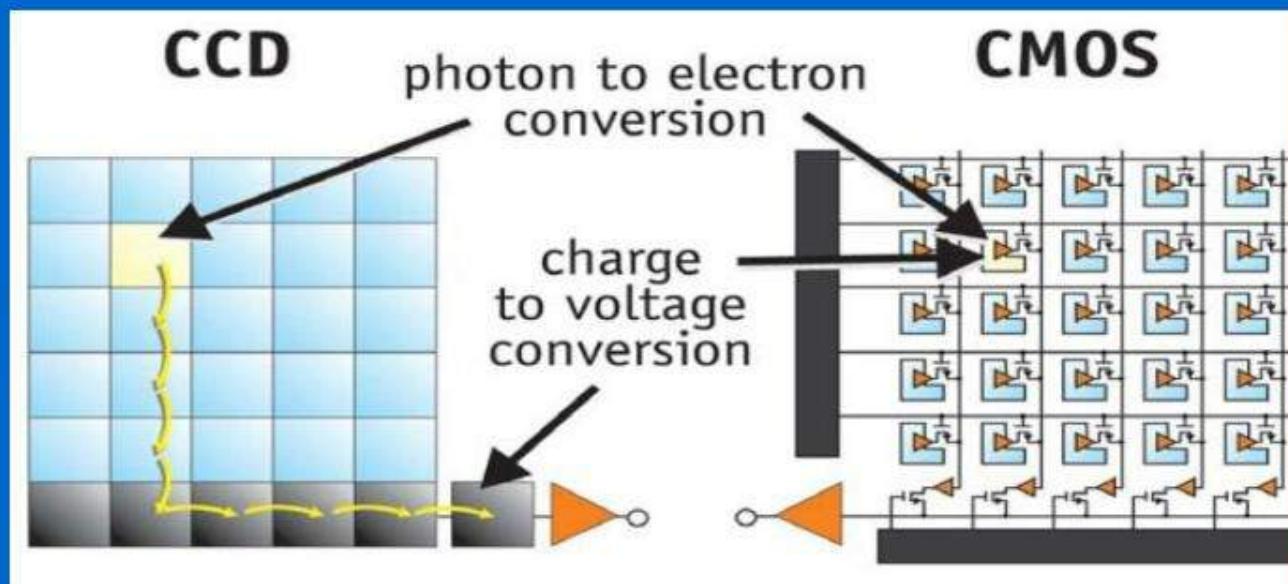
- CCDs move photogenerated charge from pixel to pixel and convert it to voltage at an output node.
- An **analog-to-digital converter (ADC)** then turns each pixel's value into a digital value.



[http://www.dalsa.com/shared/content/pdfs/CCD\\_vs\\_CMOS\\_Litwiller\\_2005.pdf](http://www.dalsa.com/shared/content/pdfs/CCD_vs_CMOS_Litwiller_2005.pdf)

# CMOS Cameras

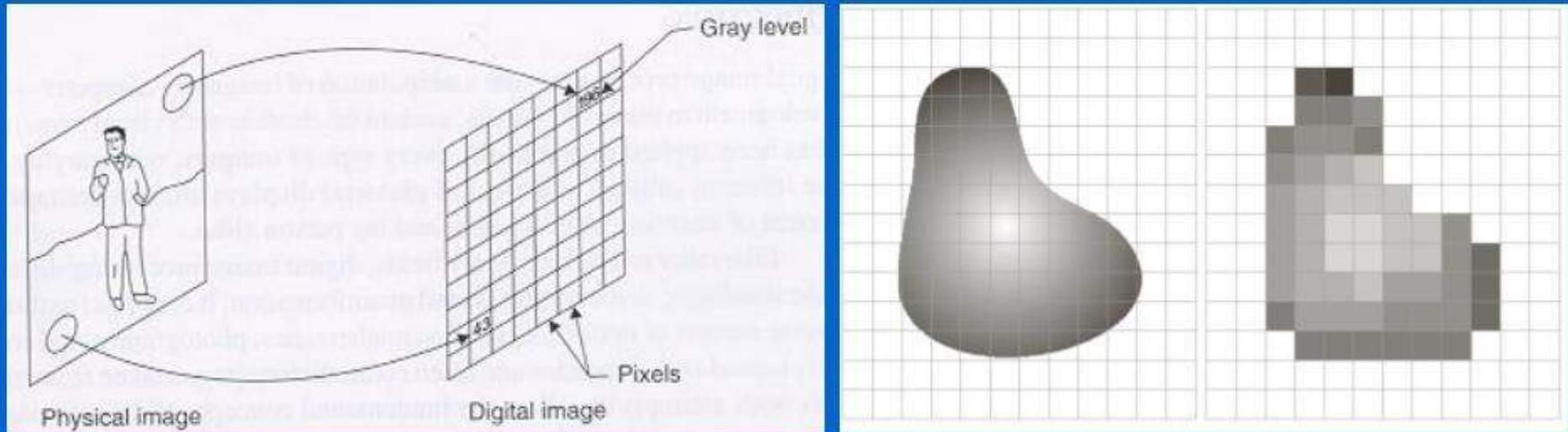
- CMOS convert charge to voltage inside each element.
- Uses several transistors at each pixel to amplify and move the charge using more traditional wires.
- The CMOS signal is digital, so it needs no ADC.



[http://www.dalsa.com/shared/content/pdfs/CCD\\_vs\\_CMOS\\_Litwiller\\_2005.pdf](http://www.dalsa.com/shared/content/pdfs/CCD_vs_CMOS_Litwiller_2005.pdf)

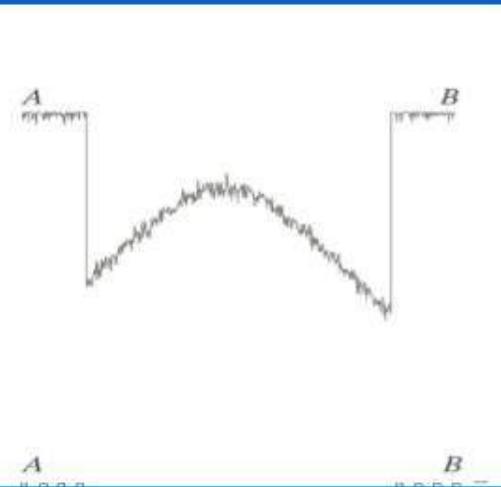
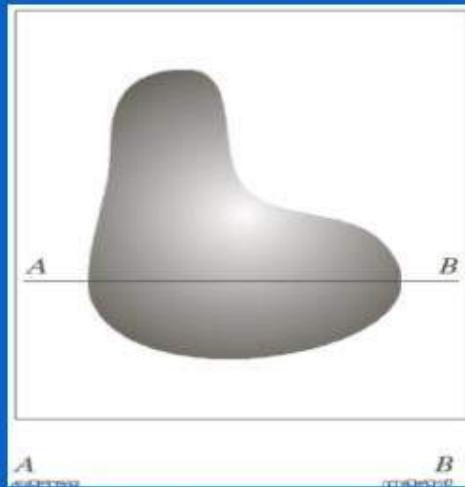
- 
- 
- 

# Image digitization

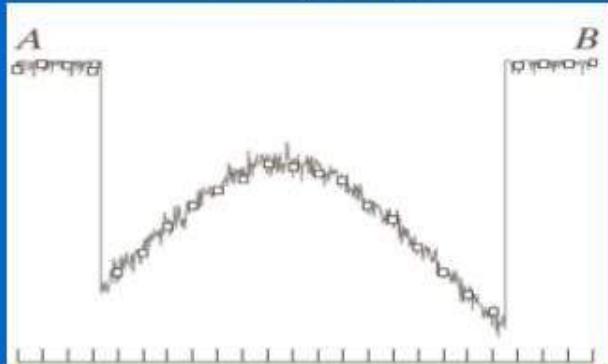


- **Sampling:** measure the value of an image at a finite number of points.
- **Quantization:** represent measured value (i.e., voltage) at the sampled point by an integer.

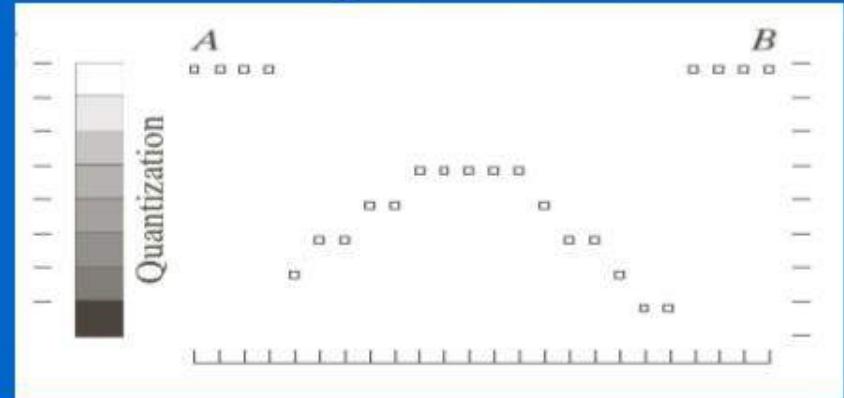
# Image digitization (cont'd)



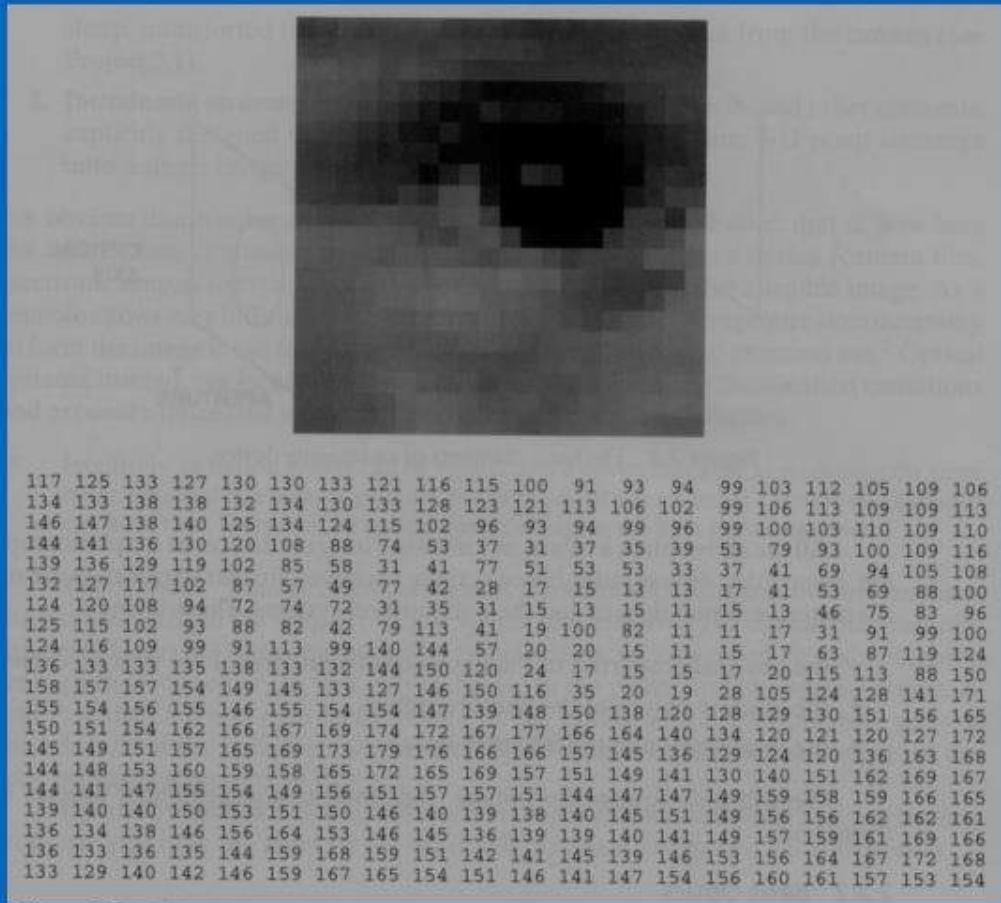
Sampling



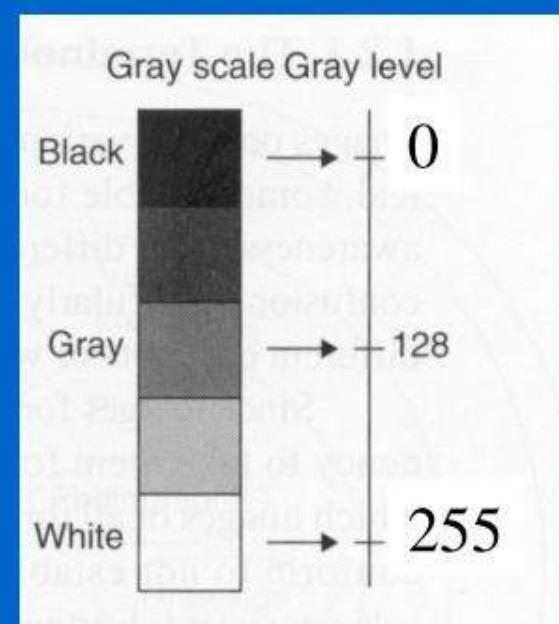
Quantization



# What is an image?

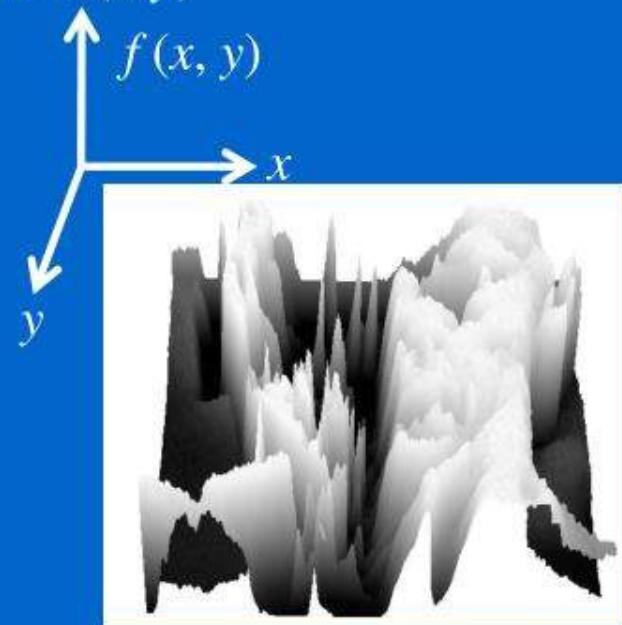


8 bits/pixel



## What is an image? (cont'd)

- We can think of a (grayscale) image as a **function**,  $f$ , from  $\mathbf{R}^2$  to  $\mathbf{R}$  (or a 2D *signal*):
  - $f(x,y)$  gives the **intensity** at position  $(x,y)$



- A **digital** image is a discrete (**sampled, quantized**) version of this function

# Image Sampling - Example

original image



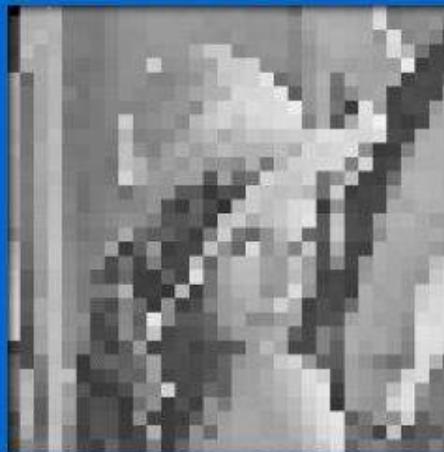
sampled by a factor of 2



sampled by a factor of



sampled by a factor of 8



Images have  
been resized  
for easier  
comparison

# Image Quantization - Example

- 256 gray levels (8bits/pixel)    32 gray levels (5 bits/pixel)    16 gray levels (4 bits/pixel)



- 8 gray levels (3 bits/pixel)    4 gray levels (2 bits/pixel)    2 gray levels (1 bit/pixel)



# Color Images

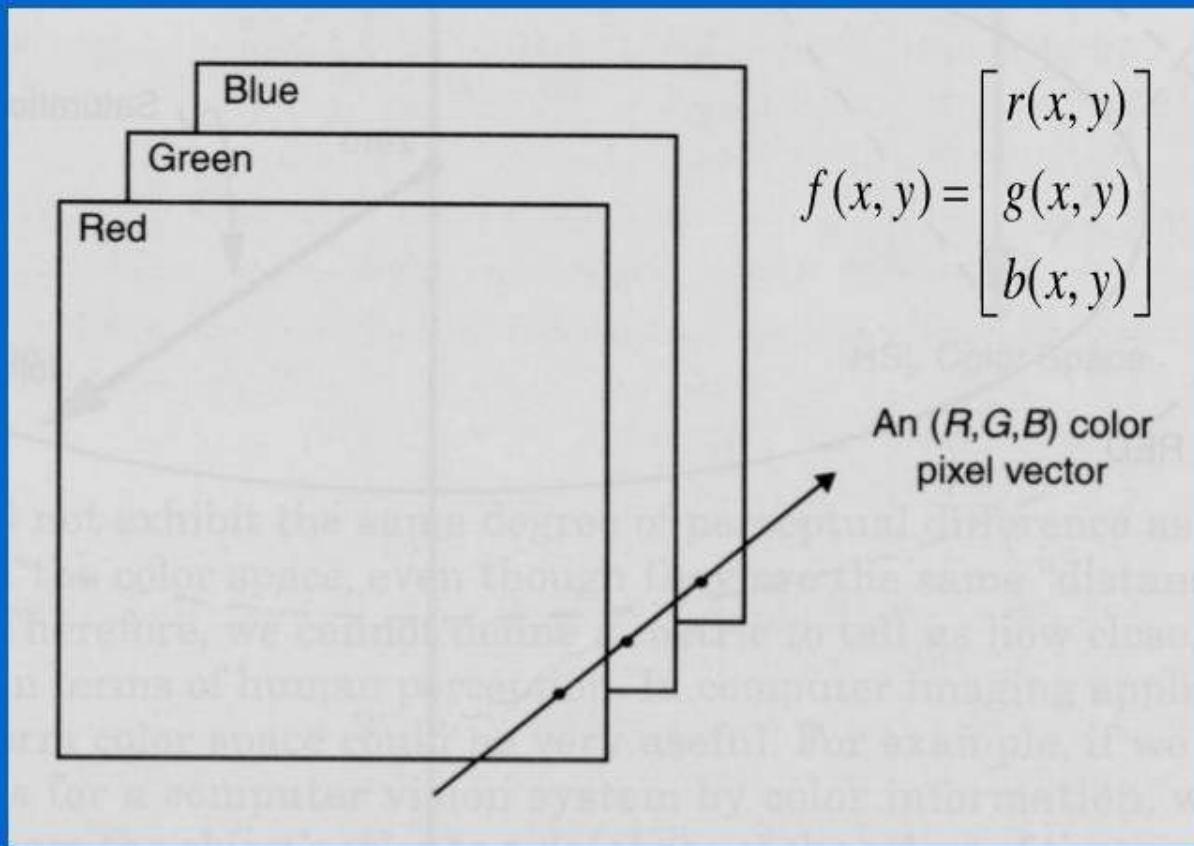
- Color images are comprised of three color channels – **red**, **green**, and, **blue** – which combine to create most of the colors we can see.



=

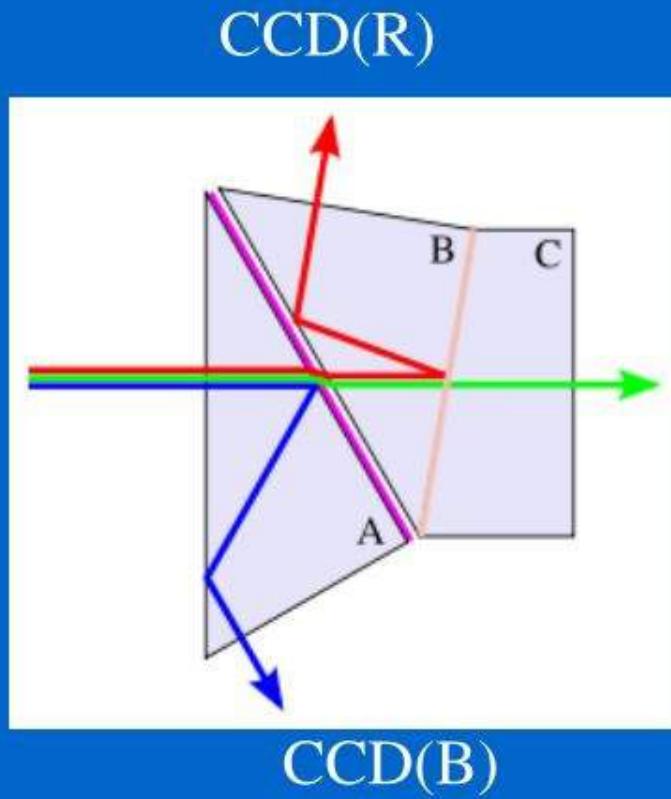


# Color images



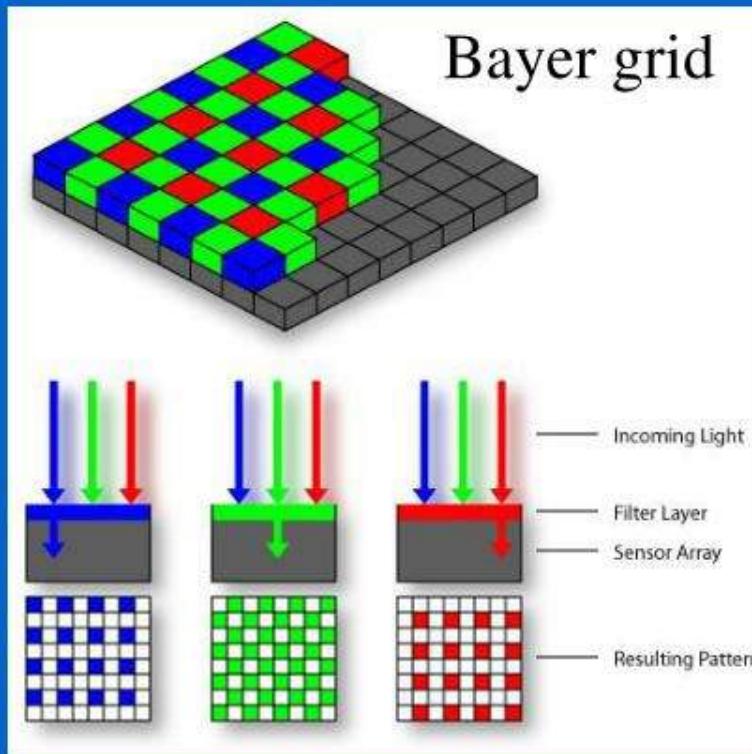
# Color sensing in camera: Prism

- Requires three chips and precise alignment.

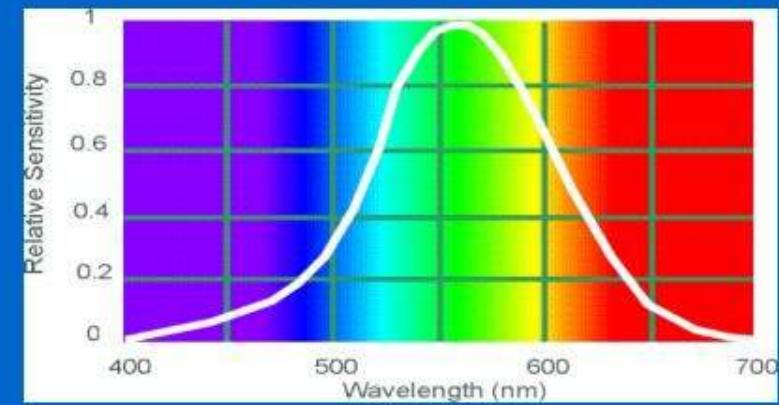


# Color sensing in camera: Color filter array

- In traditional systems, color filters are applied to a single layer of photodetectors in a tiled mosaic pattern.

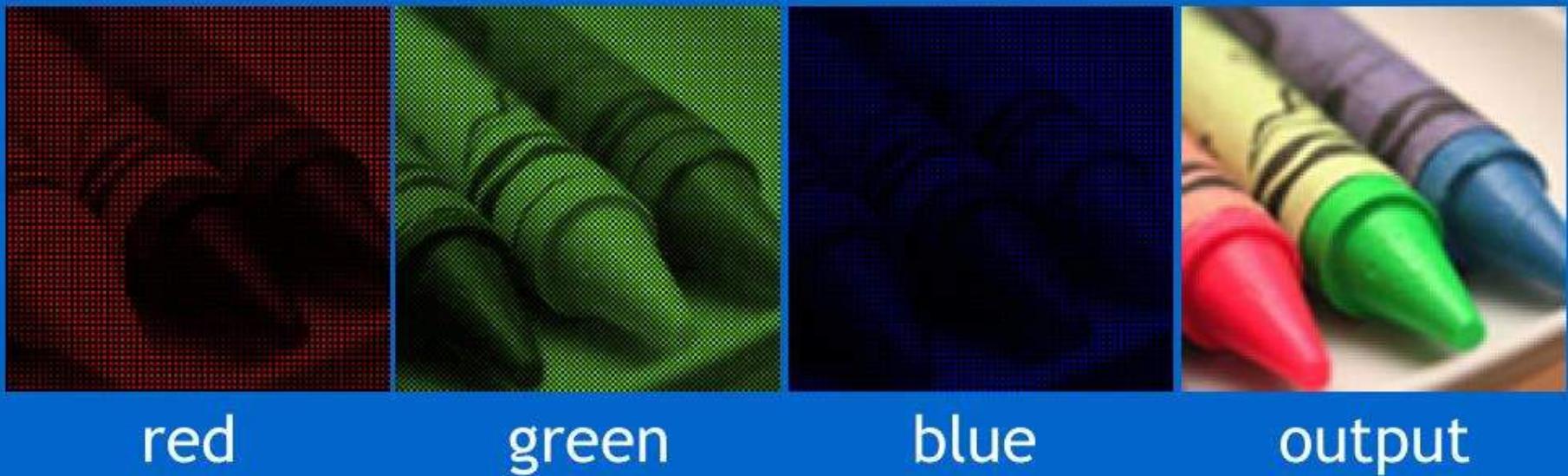


Why more green?



Human Luminance Sensitivity Function

# Color sensing in camera: Color filter array



red

green

blue

output

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

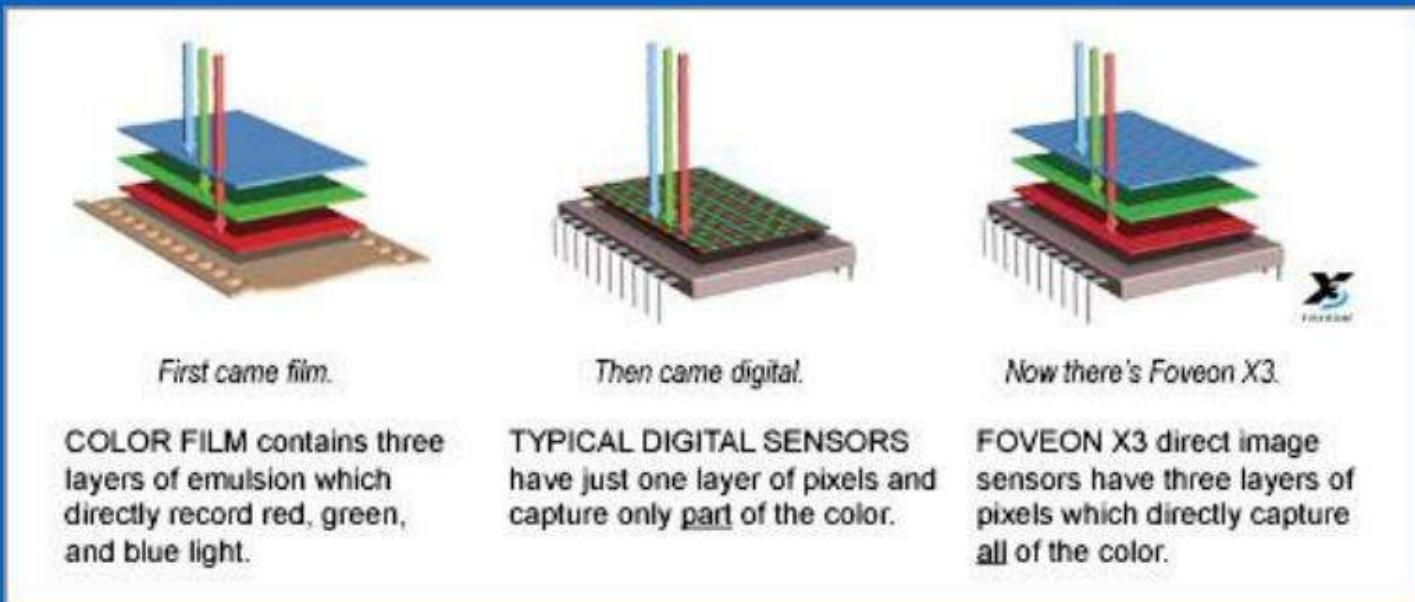
demosaicing  
(interpolation)



rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

# Color sensing in camera: Foveon X3

- CMOS sensor; takes advantage of the fact that red, blue and green light penetrate silicon to different depths.



<http://www.foveon.com/article.php?a=67>

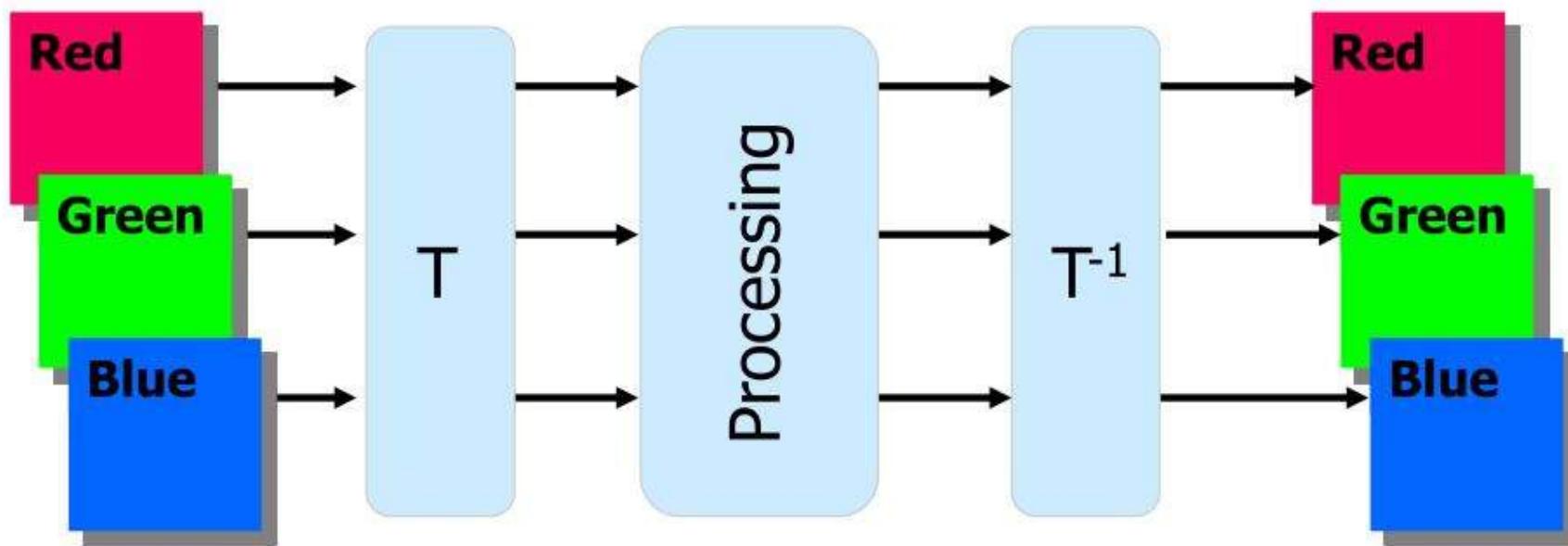
# Alternative Color Spaces

- Various other color representations can be computed from **RGB**.
- This can be done for:
  - Decorrelating the color channels:
    - principal components.
  - Bringing color information to the fore:
    - Hue, saturation and brightness.
  - Perceptual uniformity:
    - CIELuv, CIELab, ...

# Alterative Color paces

- RGB (CIE), RnGnBn (TV - National Television Standard Committee)
- XYZ (CIE)
- UVW (UCS de la CIE), U\*V\*W\* (UCS modified by the CIE)
- YUV, YIQ, YCbCr
- YDbDr
- DSH, HSV, HLS, IHS
- Munsel color space (cylindrical representation)
- CIELuv
- CIELab
- SMPTE-C RGB
- YES (Xerox)
- Kodak Photo CD, YCC, YPbPr, ...

# Processing Strategy



## Color Transformation - Examples

$$H = \arccos \frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{((R-G)^2 + (R-B)(G-B))}}$$

$$S = 1 - 3 \frac{\min(R, G, B)}{R+G+B}$$

$$V = \frac{1}{3}(R+G+B)$$

$$Y = 0.299R + 0.587G + 0.114B$$

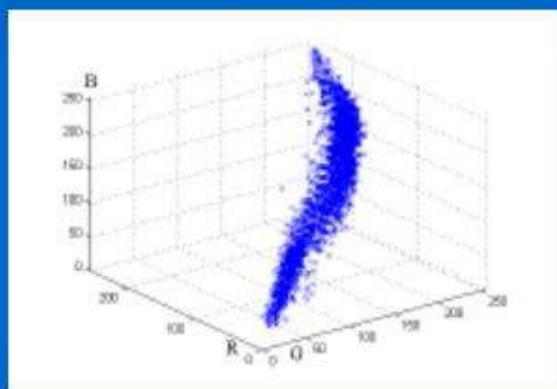
$$C_r = R - Y$$

$$C_b = B - Y$$

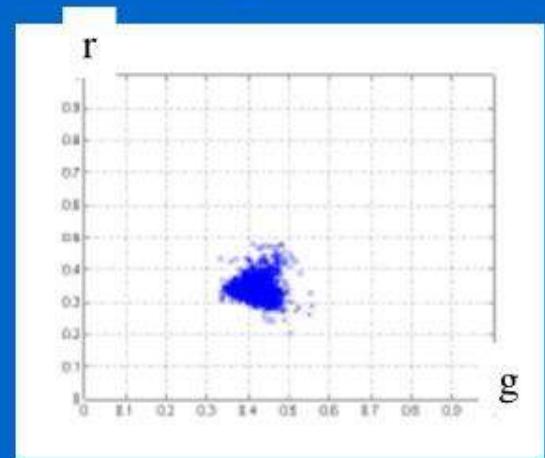
# Skin color



RGB

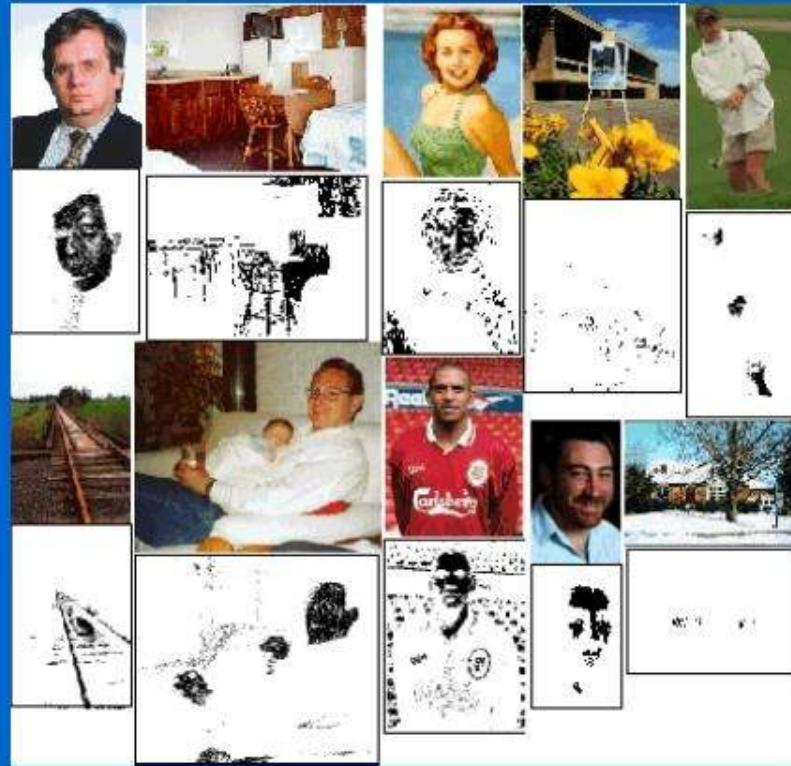


rg



$$r = \frac{R}{R+G+B} \quad g = \frac{G}{R+G+B} \quad b = \frac{B}{R+G+B}$$

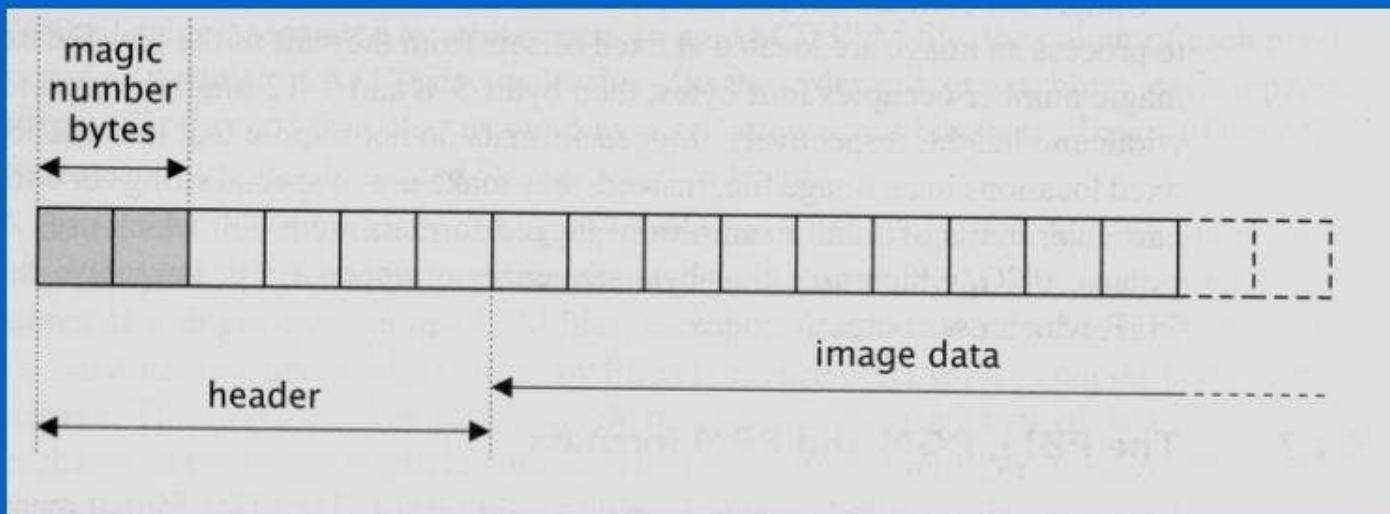
# Skin detection



M. Jones and J. Rehg, Statistical Color Models with Application to Skin Detection, International Journal of Computer Vision, 2002.

# Image file formats

- Many image formats adhere to the simple model shown below (line by line, no breaks between lines).
- The header contains at least the width and height of the image.
- Most headers begin with a signature or “magic number” (i.e., a short sequence of bytes for identifying the file format)



# Common image file formats

- GIF (Graphic Interchange Format) -
- PNG (Portable Network Graphics)
- JPEG (Joint Photographic Experts Group)
- TIFF (Tagged Image File Format)
- PGM (Portable Gray Map)
- FITS (Flexible Image Transport System)

# PBM/PGM/PPM format

- A popular format for grayscale images (8 bits/pixel)
- Closely-related formats are:
  - PBM (Portable Bitmap), for binary images (1 bit/pixel)
  - PPM (Portable Pixelmap), for color images (24 bits/pixel)

ASCII

```
P2
# a simple PGM image
7 7 255
120 120 120 120 120 120 120
120 120 120 33 120 120 120
120 120 120 33 120 120 120
120 33 33 33 33 33 120
120 120 120 33 120 120 120
120 120 120 33 120 120 120
120 120 120 120 120 120 120
```

```
P5
# a simple PGM image
7 7 255
xxxxxxxx!xxxxxx!xxxx!!!xxxx!xxxxxx!xxxxxxxxx
Binary
```

ASCII or binary (raw) storage

Signatures of the various PBM, PGM and PPM image formats.

Signature	Image type	Storage type
P1	binary	ASCII
P2	greyscale	ASCII
P3	RGB	ASCII
P4	binary	raw bytes
P5	greyscale	raw bytes
P6	RGB	raw bytes