Lecture 4
Fundamentals of features

“图像处理的洪荒之力（二）”

七月在线 金老师

2016年9月11日

An edge is not a line...

- How can we detect *lines* ?

Features: Topics

- Advanced Edge Detection (last time)
- Global Image Features (Hough Transform)

- Corner Detector
- SIFT Features
- Learning with Many Simple Features

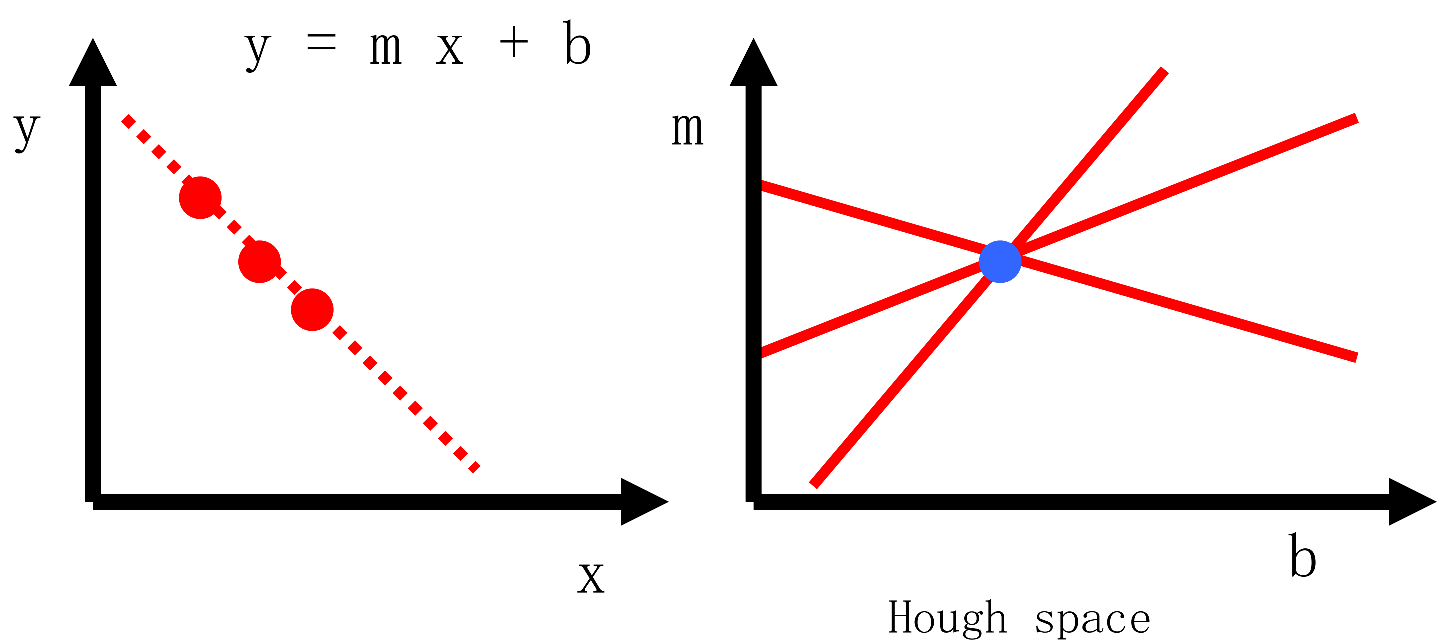
Towards Global Features

Local versus global

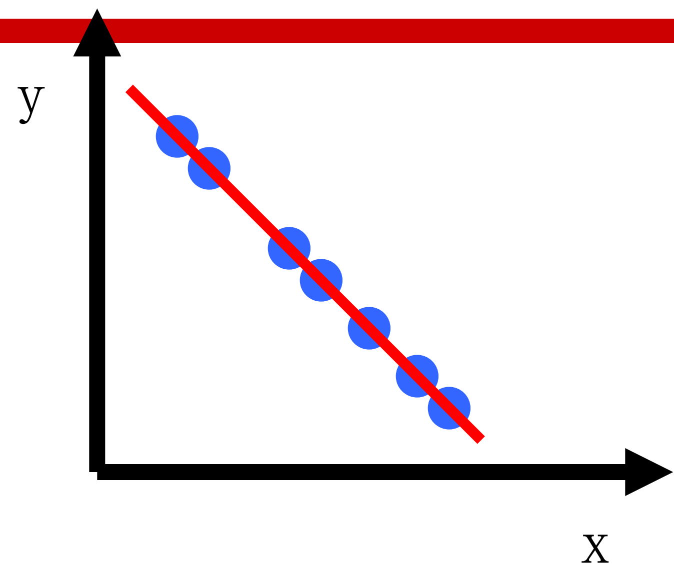
Hough transform

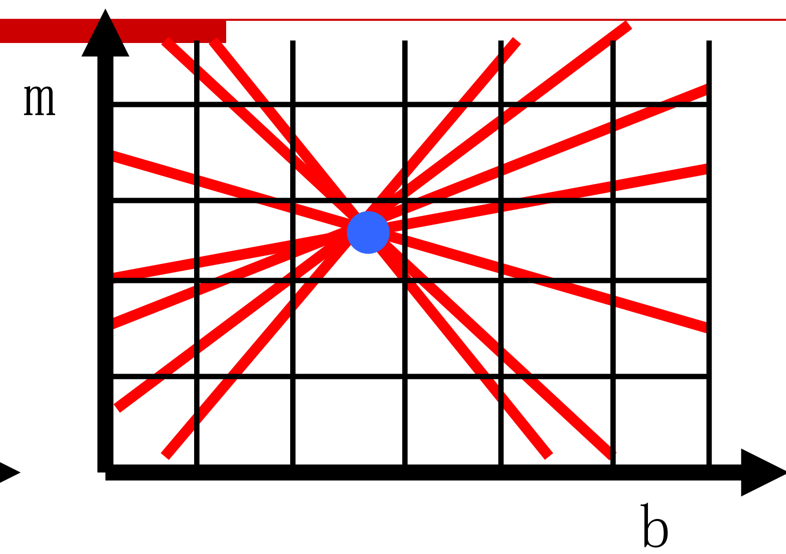
P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

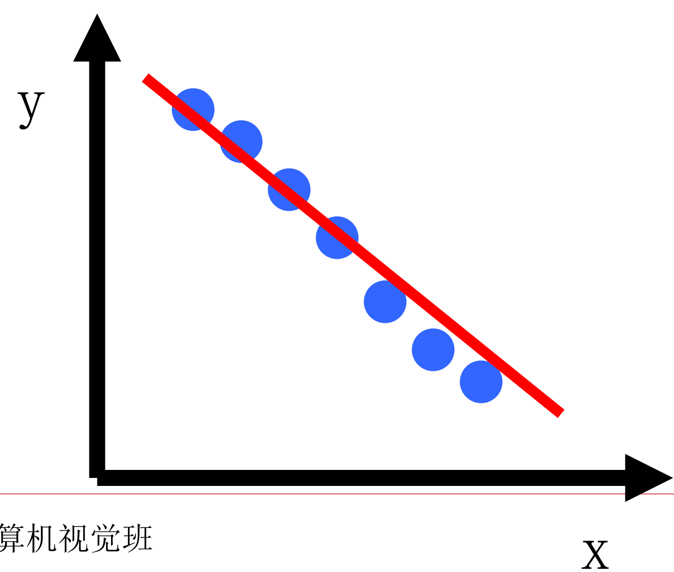
Given a set of points, find the curve or line that explains the data points best

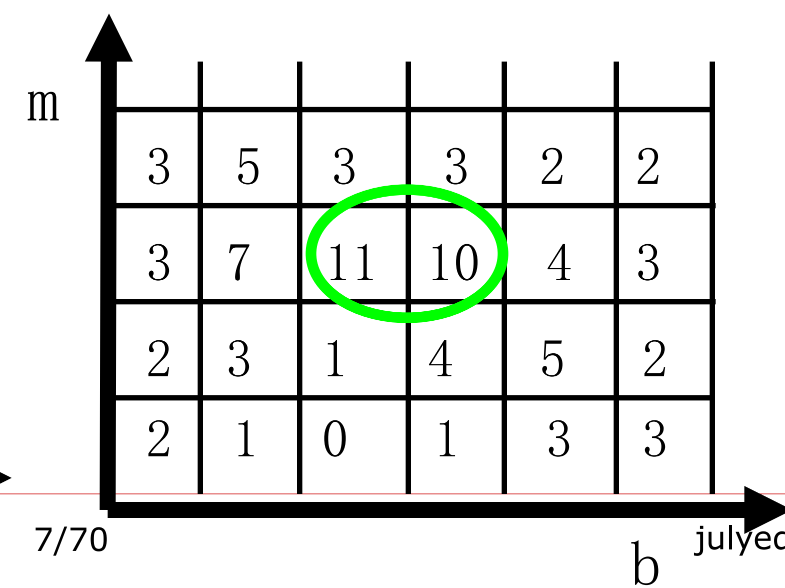


Hough transform



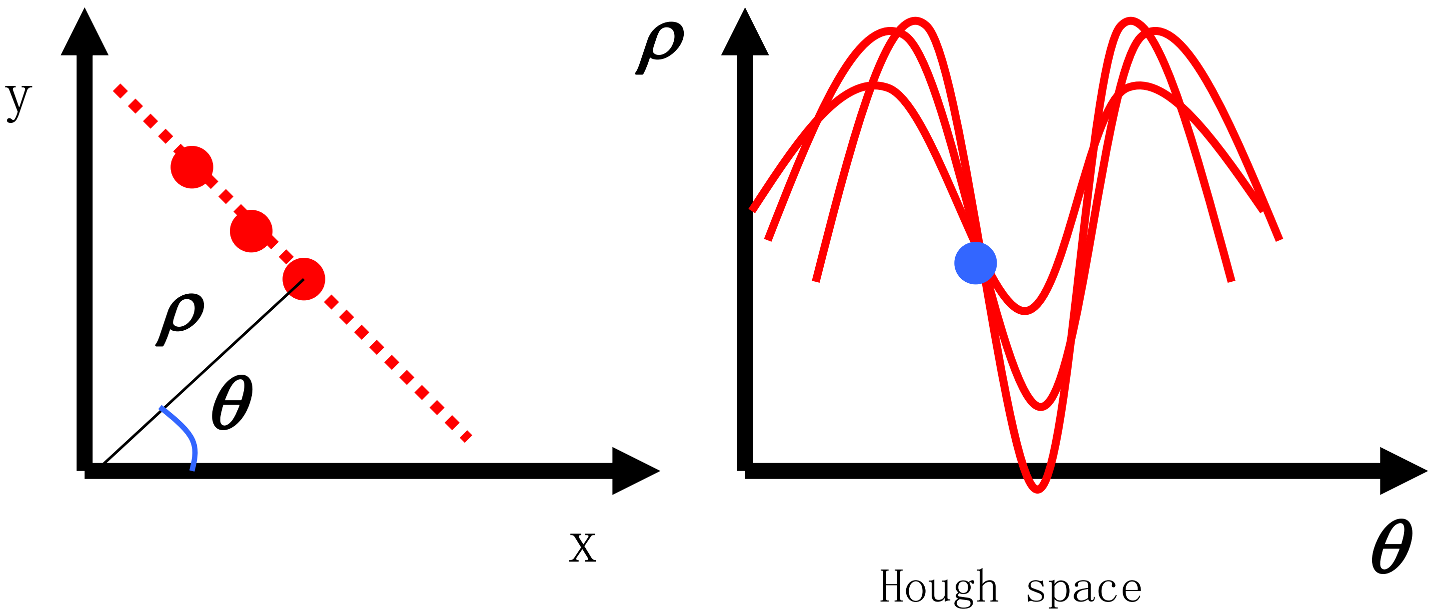






Hough transform

Use a polar representation for the parameter space



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

Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. For each ***edge*** point in $E(x, y)$ in the image

    for θ = 0 to 180 // some quantization; why not 2pi?

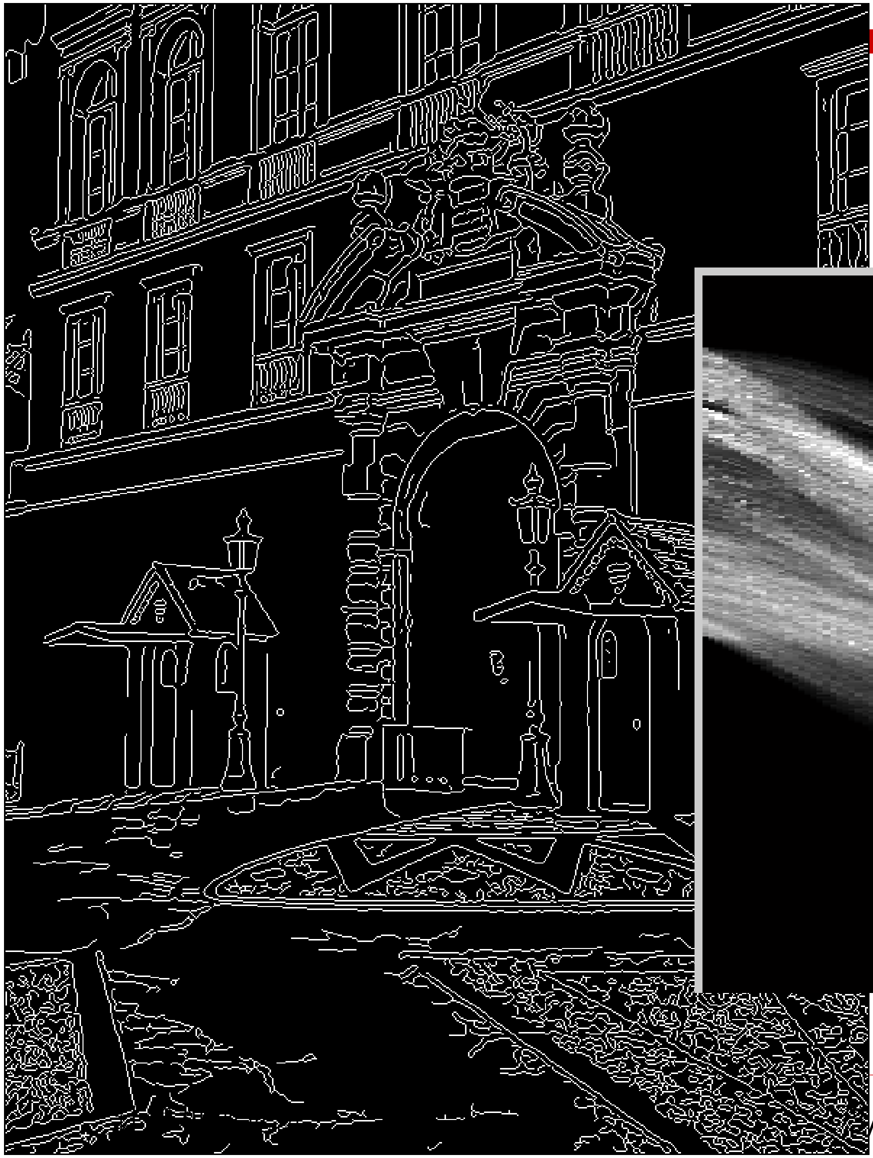
    $d = x\cos\theta - y\sin\theta$ // maybe negative

    H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given

    by $d = x\cos\theta - y\sin\theta$

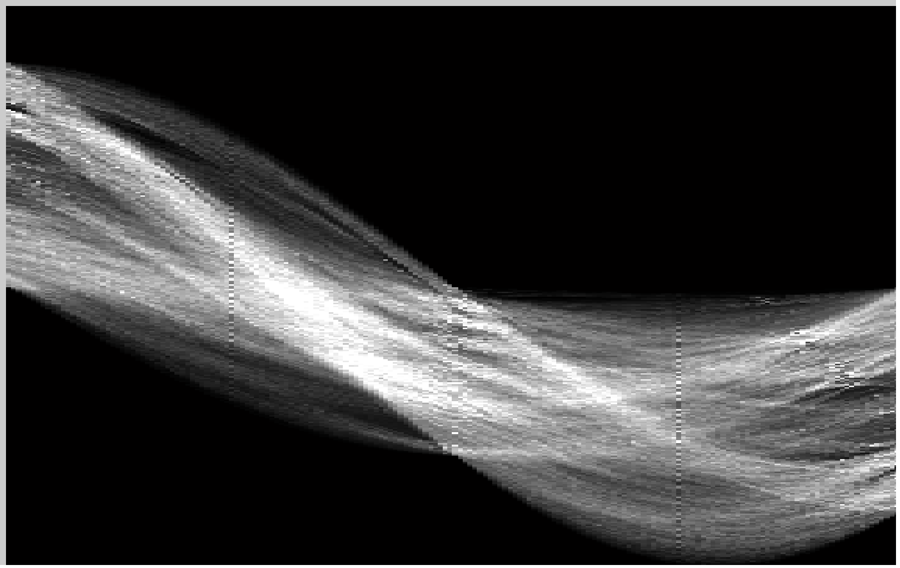
1. Image → Canny

2. Canny → Hough votes



3. Hough votes → Edges

Find peaks and post-process



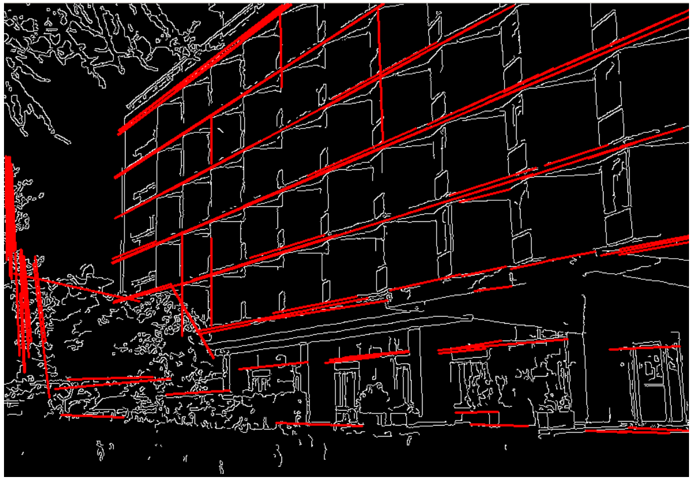
Hough transform in OpenCV

- HOUGH_STANDARD cv::HoughLines

```
C++: void HoughLines(InputArray image, OutputArray lines, double rho, double theta, int threshold, double srn=0, double stn=0 )
```

- HOUGH_PROBABILISTIC

```
C++: void HoughLinesP(InputArray image, OutputArray lines, double rho, double theta, int threshold, double minLineLength=0, double maxLineGap=0 )
```



cvHoughLines2
Opencv 1.X

- *CvSeq* cvHonghLines2(*

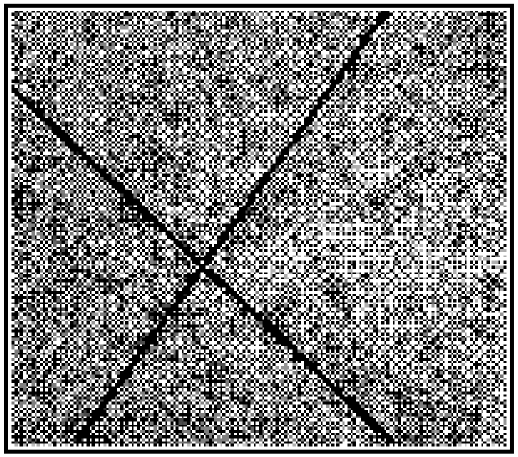
  CvArr image, void* line_storage, int mehtod, double rho, double theta, int threshold, double param1 =0, double param2 =0*

  );

```
void cv::HoughLines( InputArray _image,OutputArray _lines,
                     double rho, double theta,int threshold,
                     double srn, double stn )
{
   Ptr<CvMemStorage> storage = cvCreateMemStorage(STORAGE_SIZE);
   Mat image = _image.getMat();
   CvMat c_image = image;
   CvSeq* seq = cvHoughLines2( &c_image, storage, srn == 0 &&stn == 0 ?
                    CV_HOUGH_STANDARD :CV_HOUGH_MULTI_SCALE,
                    rho, theta, threshold, srn,stn );
   seqToMat(seq, _lines);
}
```

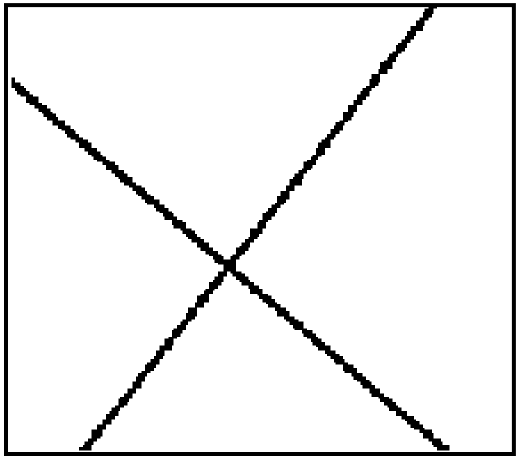
```
void cv::HoughLinesP( InputArray _image,OutputArray _lines,
                      double rho, double theta,int threshold,
                      double minLineLength,double maxGap )
{
   Ptr<CvMemStorage> storage = cvCreateMemStorage(STORAGE_SIZE);
   Mat image = _image.getMat();
   CvMat c_image = image;
    CvSeq*seq = cvHoughLines2( &c_image, storage, CV_HOUGH_PROBABILISTIC,
                 rho, theta, threshold,minLineLength, maxGap );
   seqToMat(seq, _lines);
}
```

Hough Transform: Results



Image

Edge detection



Hough Transform

Hough Transform

- How would we find circles?
  - Of fixed radius
  - Of unknown radius

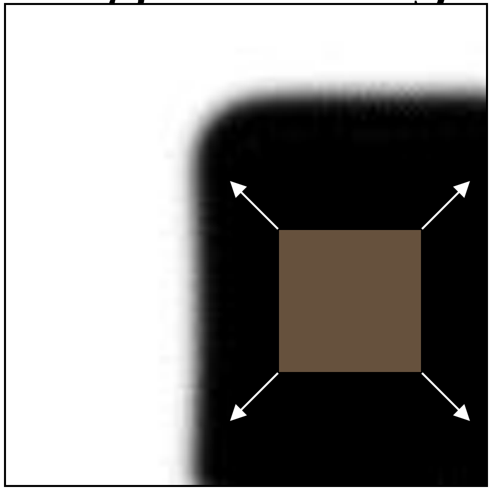
C++: void HoughCircles(InputArray image,OutputArray circles, int method, double dp, double minDist, double param1=100,double param2=100, int minRadius=0, int maxRadius=0)

图像局部特征

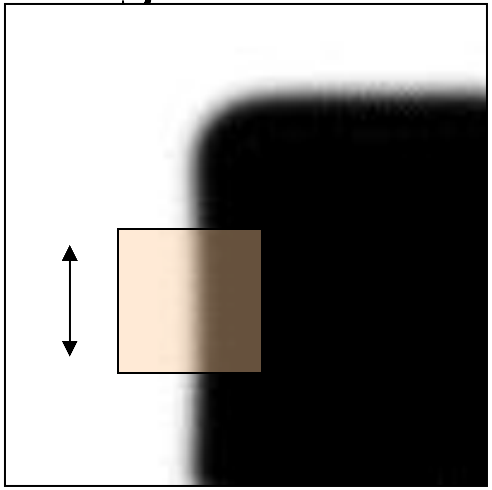
- 全局特征与局部特征
- 局部特征检测之blob and corner
  - Blob detection: 高斯拉普拉斯算子 LOG
    &像素点Hessian矩阵的行列式 DOH
    eg: SIFT SURF…
  - Corner detection
    eg: Harris FAST…
- 特征描述
  - 梯度统计直方图 或 二进制字符串特征描述子

Corner Detection: Basic Idea

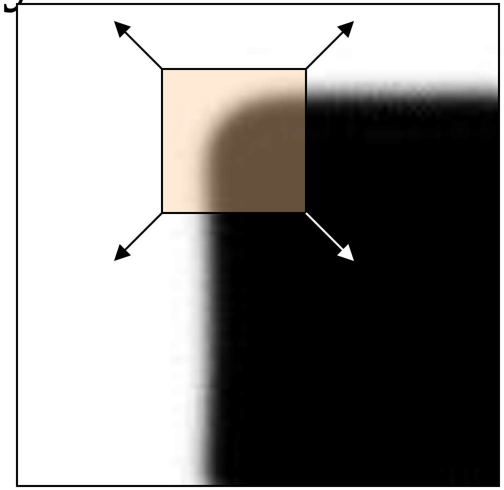
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region: no change in all directions



"edge": no change along the edge direction



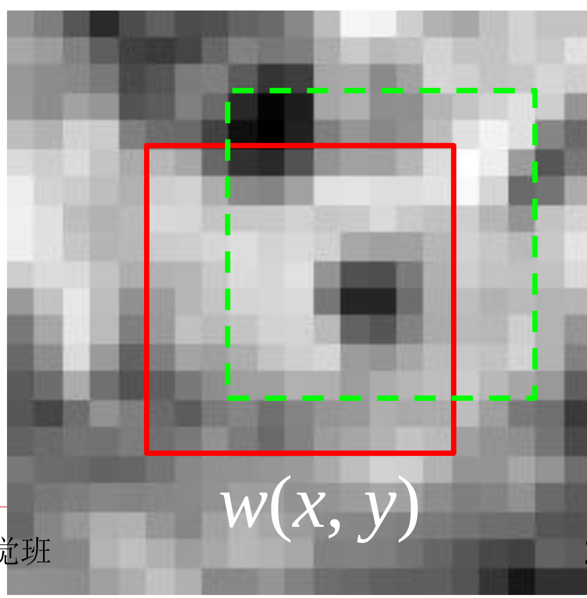
"corner": significant change in all directions

Corner Detection: Mathematics

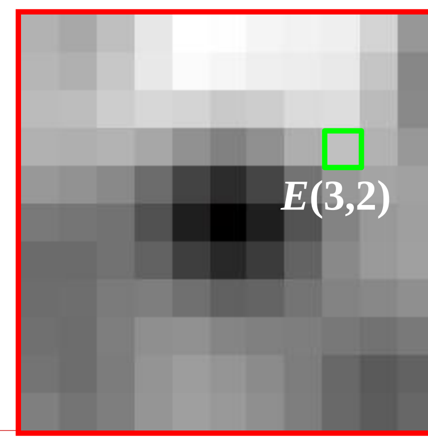
Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

$I(x, y)$



$E(u, v)$



Corner Detection: Mathematics

Change in appearance of window $w(x,y)$
for the shift $[u,v]$:

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

$$E(u,v) \approx E(0,0) + [u \quad v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u \quad v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

Corner Detection: Mathematics

The quadratic approximation simplifies to

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

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

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

Interpreting the second moment matrix

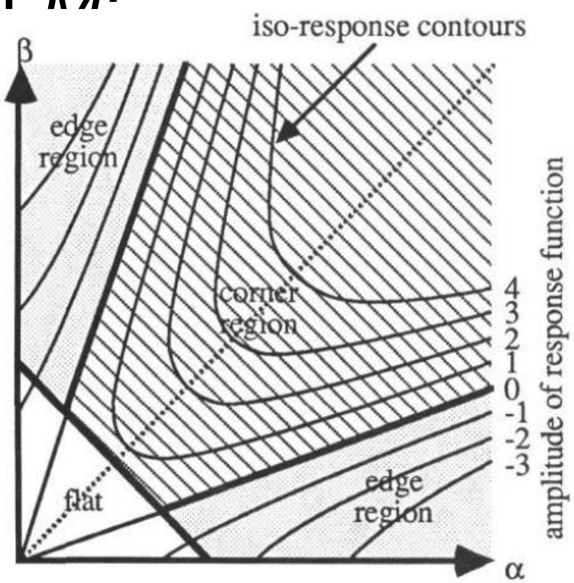
First, consider the axis-aligned case (gradients are either horizontal or vertical)

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

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

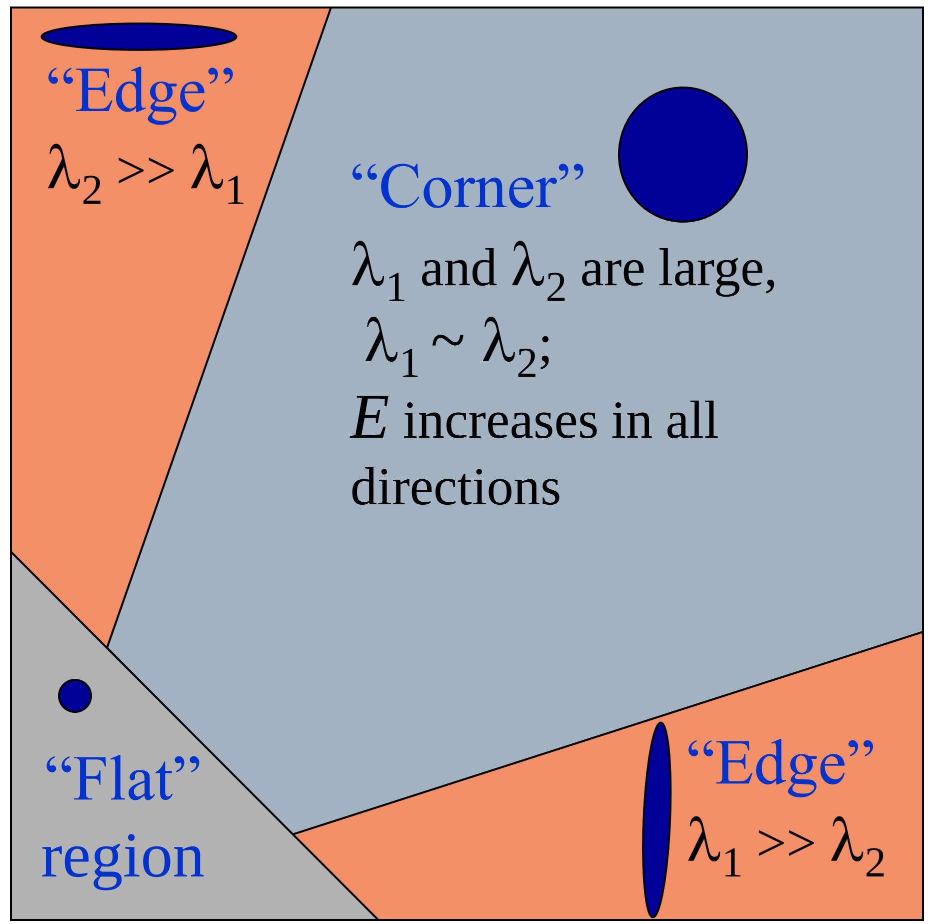
Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:



λ_2

λ_1 and λ_2 are small;
E is almost constant in all directions

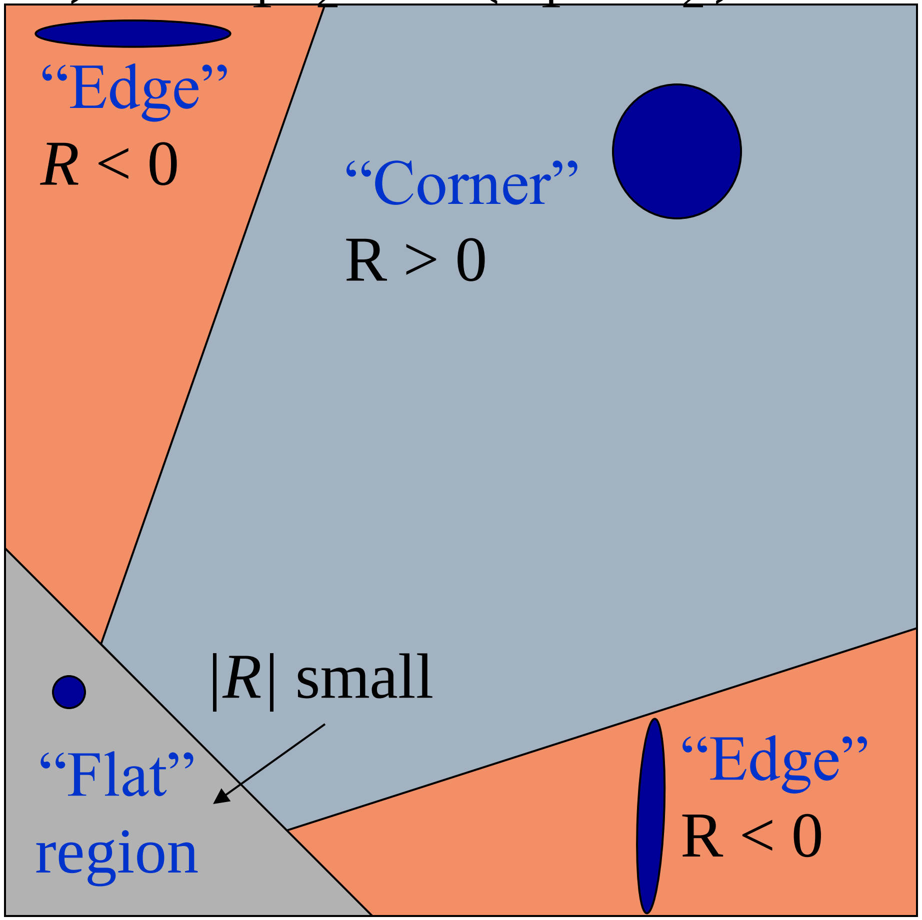


λ_1

Corner response function

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

α: constant (0.04 to 0.06)



Harris corner detector

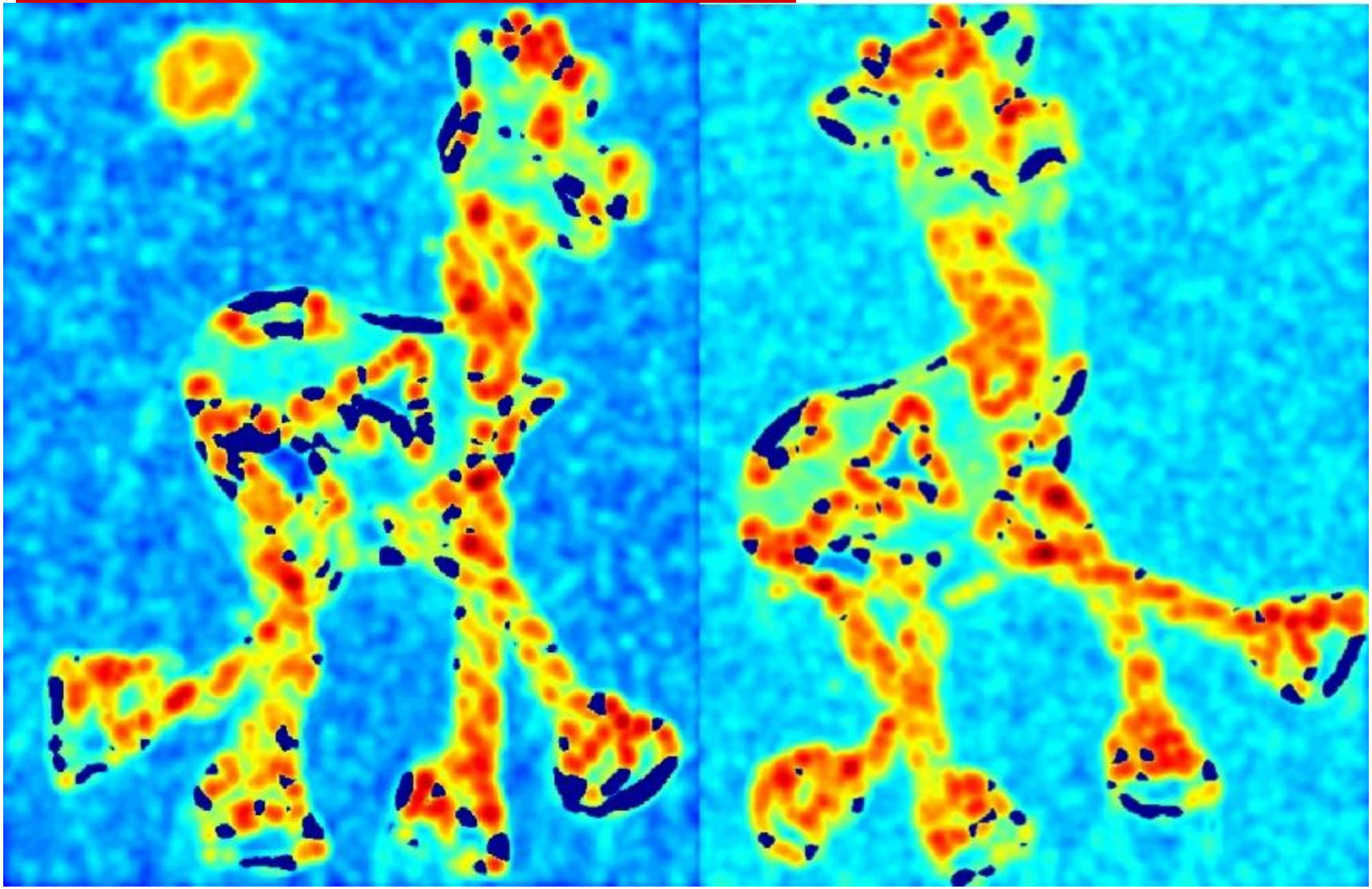
1) Compute M matrix for each image window to get their *cornerness* scores.
2) Find points whose surrounding window gave large corner response (f > threshold)
3) Take the points of local maxima, i.e., perform non-maximum suppression

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

Harris Detector: Steps

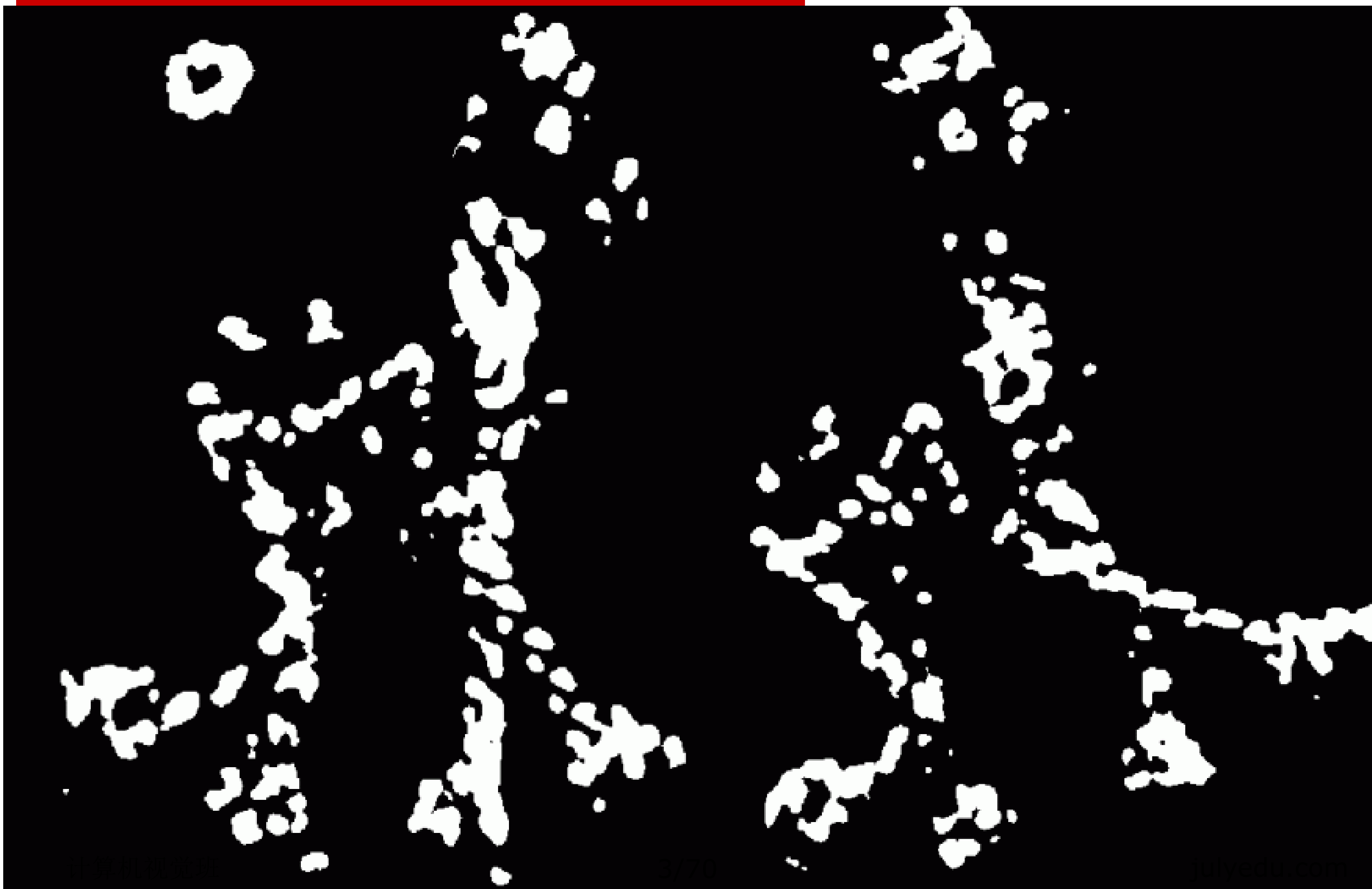
Harris Detector: Steps

Compute corner response R



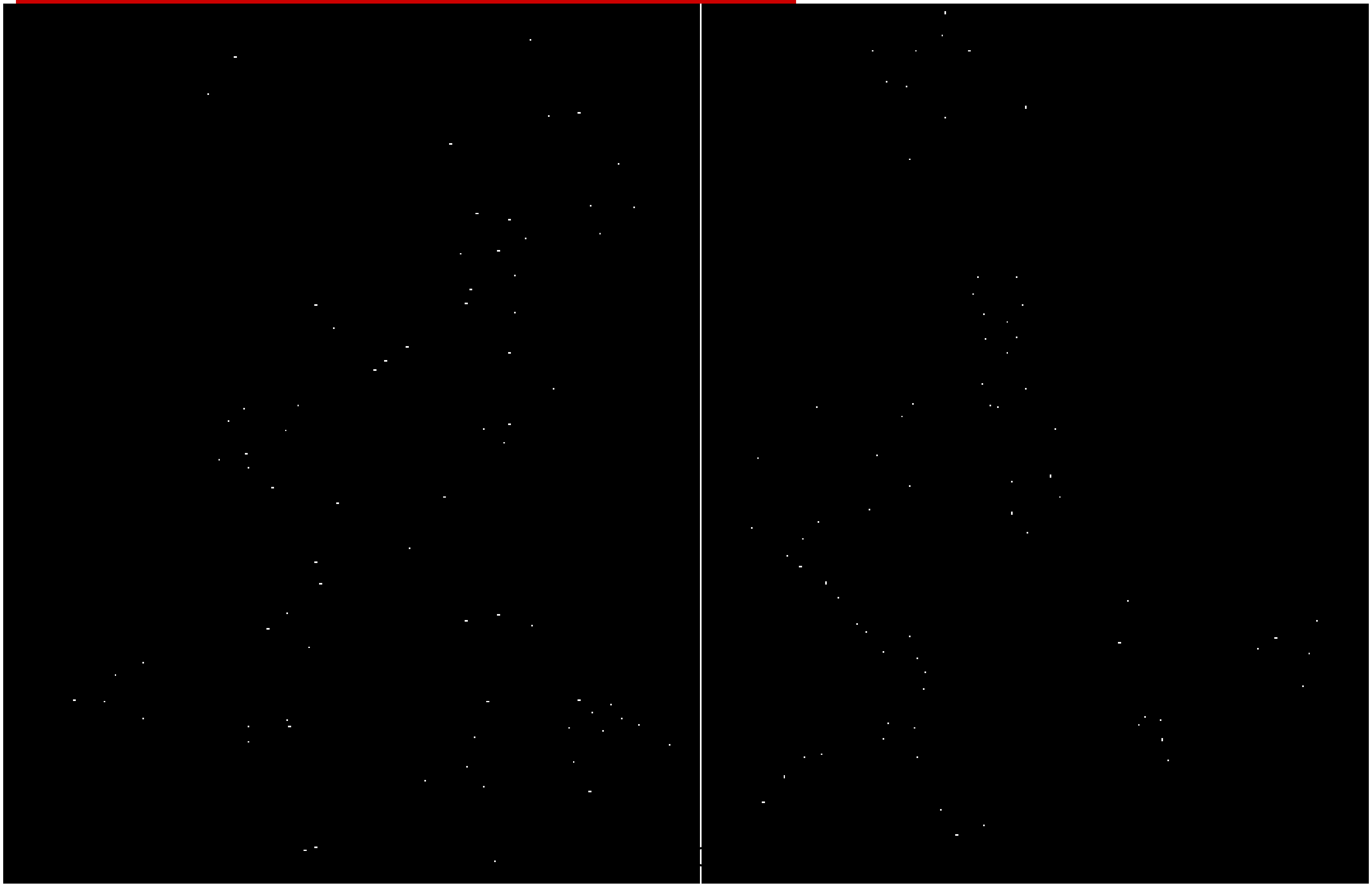
Harris Detector: Steps

Find points with large corner response: R>threshold



Harris Detector: Steps

Take only the points of local maxima of R



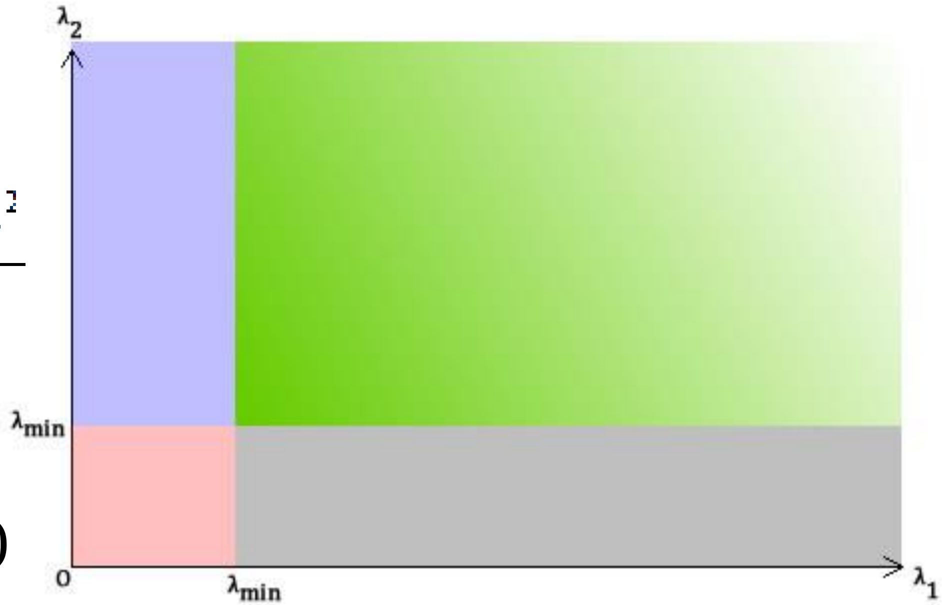
Harris Detector: Steps

Other methods

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

Nobel, 1988

$$cim = \frac{I_x^2 I_y^2 - (I_x I_y)^2}{I_x^2 + I_y^2}$$



Shi-Tomasi, 2000

$$R = \min(\lambda_1\lambda_2)$$

```
cv::goodFeaturesToTrack(image,corners,
    500,    // maximum number of corners to be returned
    0.01,    // quality level
    10);    // minimum allowed distance between points
```

Advanced Features: Topics

- Advanced Edge Detection (last time)
- Global Image Features (Hough Transform)

- Corner Detector
- SIFT Features
- Learning with Many Simple Features

So far: can localize in x-y, but not scale

Automatic Scale Selection

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

How to find corresponding patch sizes?

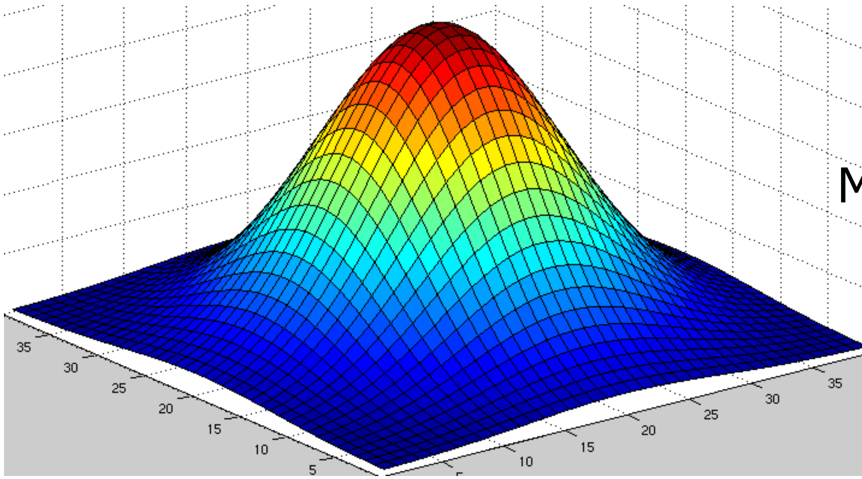
Scale Space

- Gaussian Blur

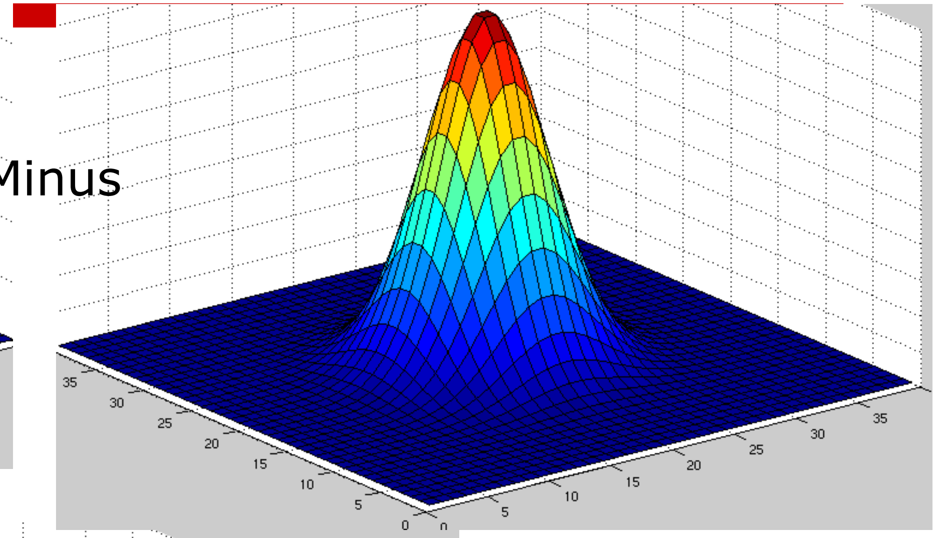
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

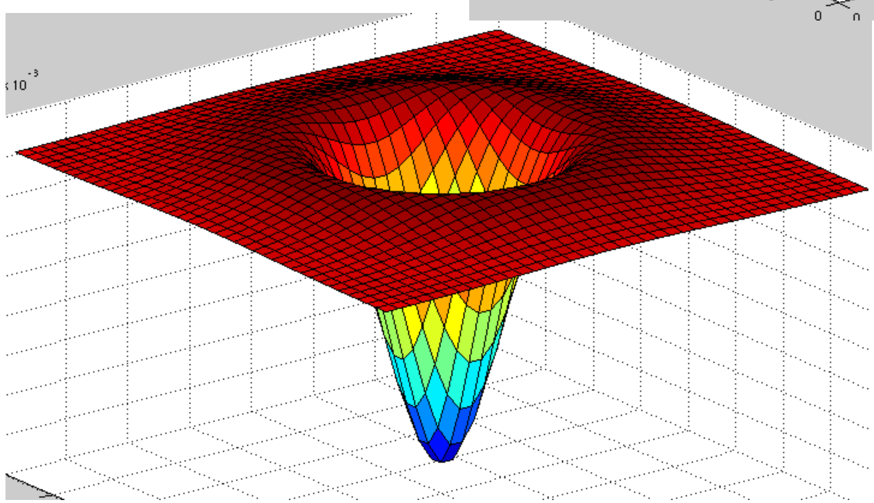
Difference of Gaussians



Minus

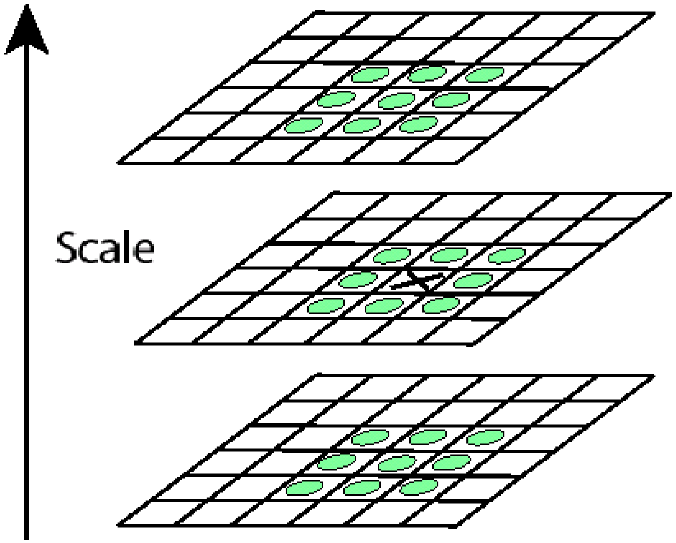


Equals

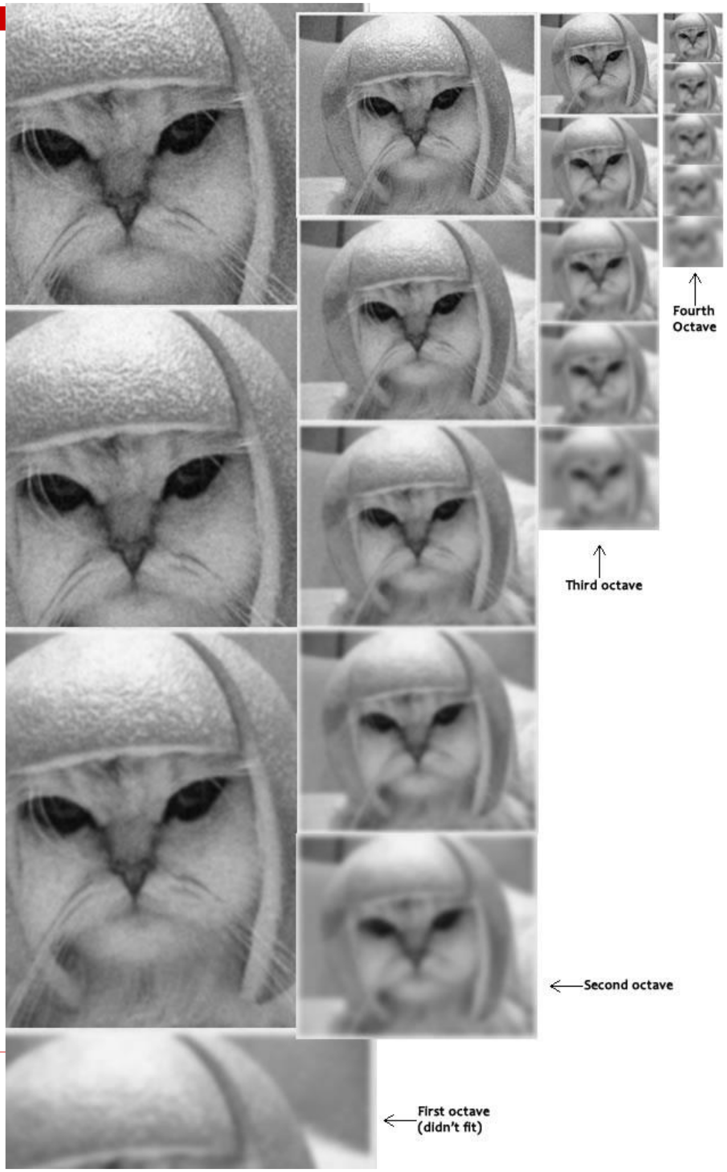


Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space

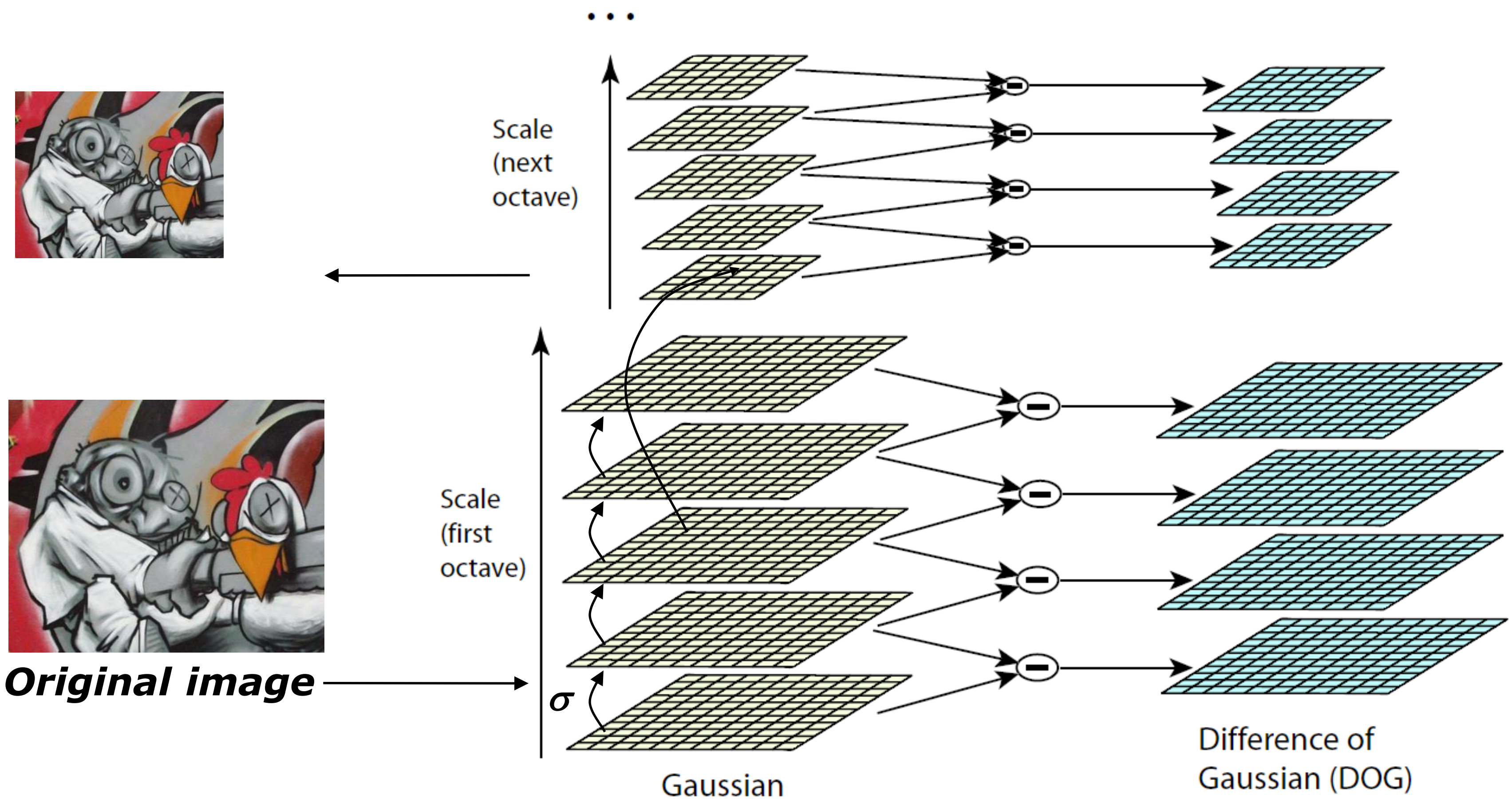


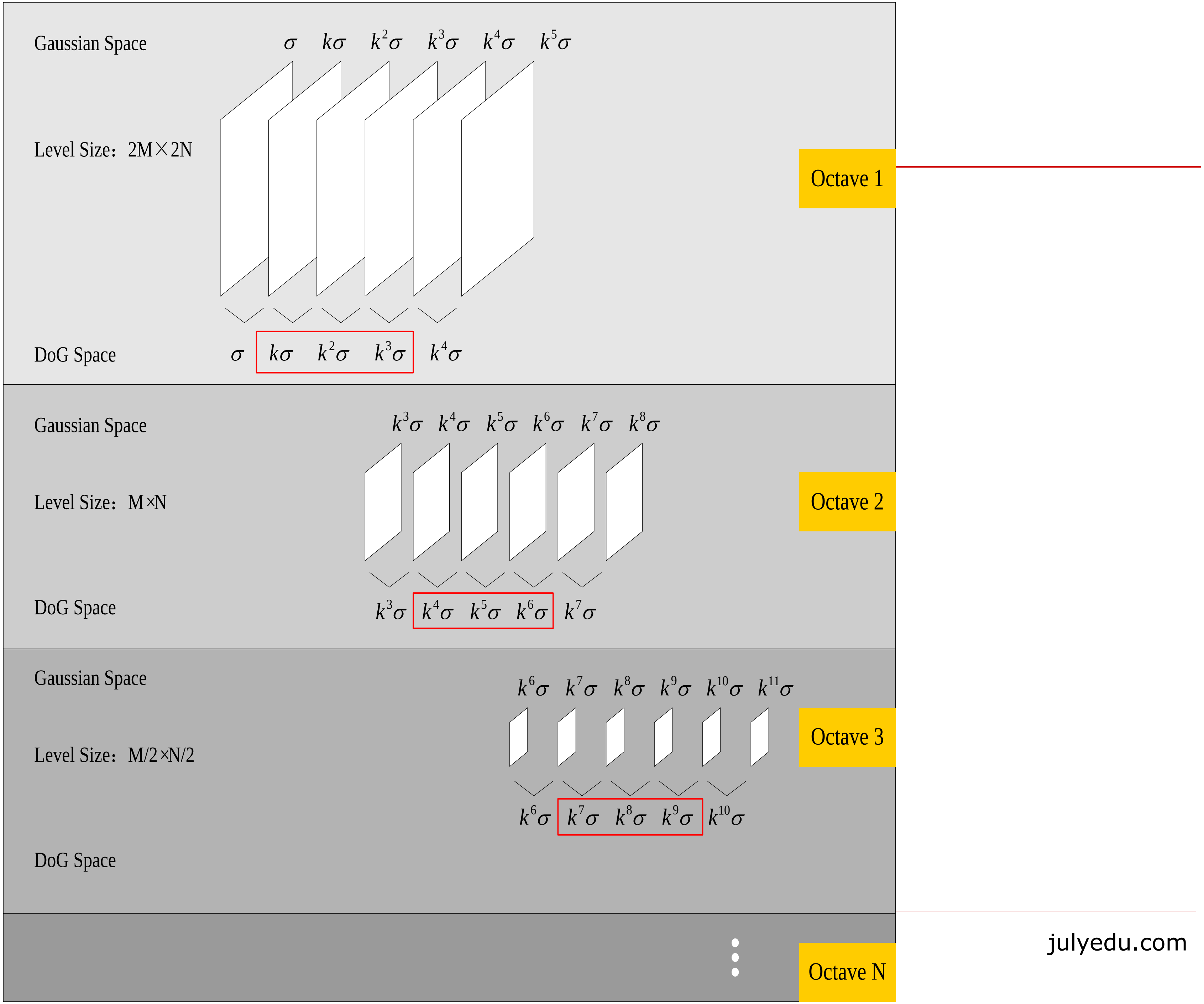
Scale Spaces in SIFT



DoG – Efficient Computation

- Computation in Gaussian scale pyramid

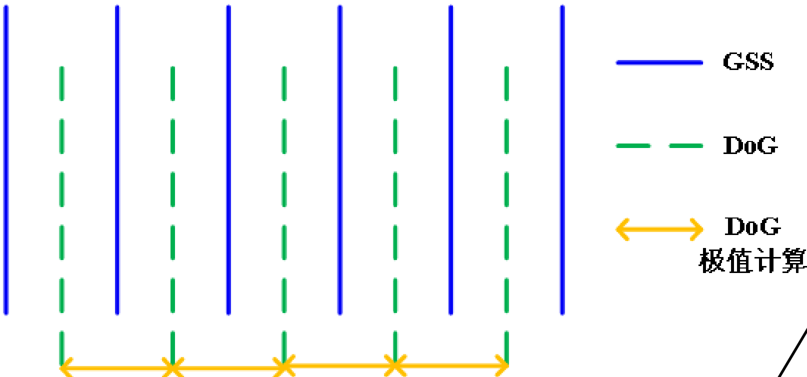


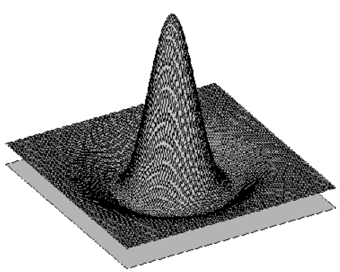


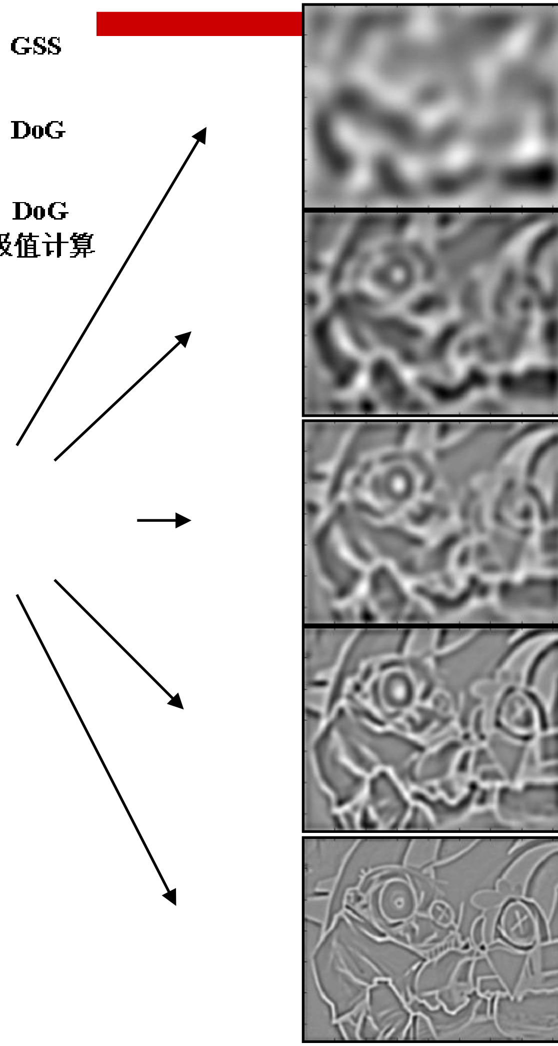
Gaussian Space
σ $k\sigma$ $k^2\sigma$ $k^3\sigma$ $k^4\sigma$ $k^5\sigma$
Level Size：2M×2N
Octave 1
DoG Space
σ $k\sigma$ $k^2\sigma$ $k^3\sigma$ $k^4\sigma$
Gaussian Space
$k^3\sigma$ $k^4\sigma$ $k^5\sigma$ $k^6\sigma$ $k^7\sigma$ $k^8\sigma$
Level Size：M×N
Octave 2
DoG Space
$k^3\sigma$ $k^4\sigma$ $k^5\sigma$ $k^6\sigma$ $k^7\sigma$
Gaussian Space
$k^6\sigma$ $k^7\sigma$ $k^8\sigma$ $k^9\sigma$ $k^{10}\sigma$ $k^{11}\sigma$
Level Size：M/2×N/2
Octave 3
$k^6\sigma$ $k^7\sigma$ $k^8\sigma$ $k^9\sigma$ $k^{10}\sigma$
DoG Space
Octave N

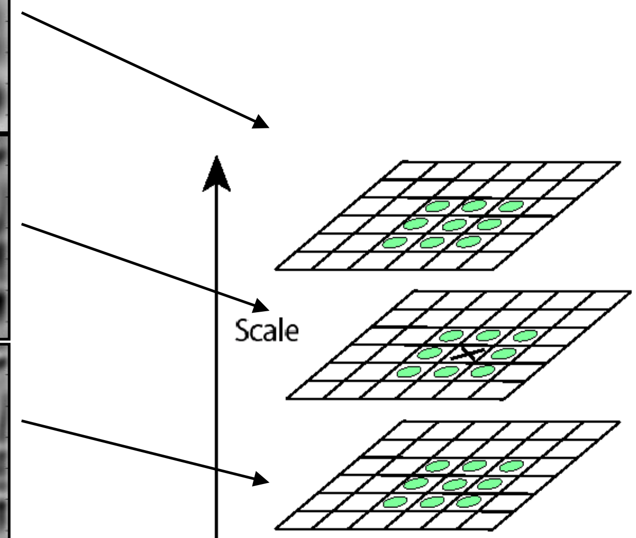
julyedu.com

Find local maxima in position-scale space of **Difference-of-Gaussian**

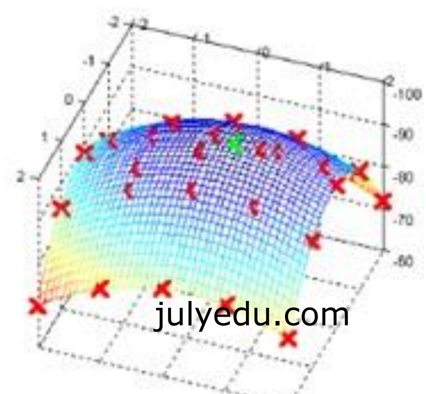




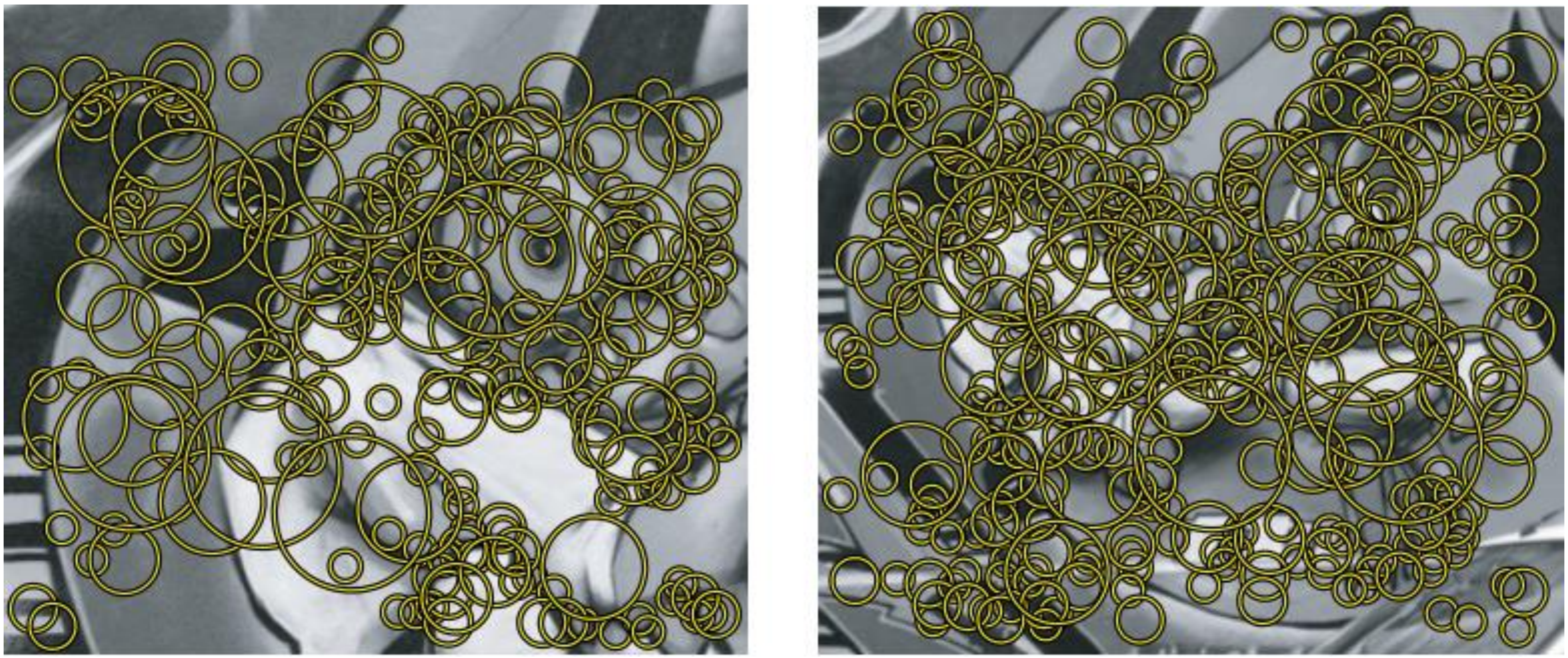




⇒ List of
(x, y, s)



Results: Difference-of-Gaussian

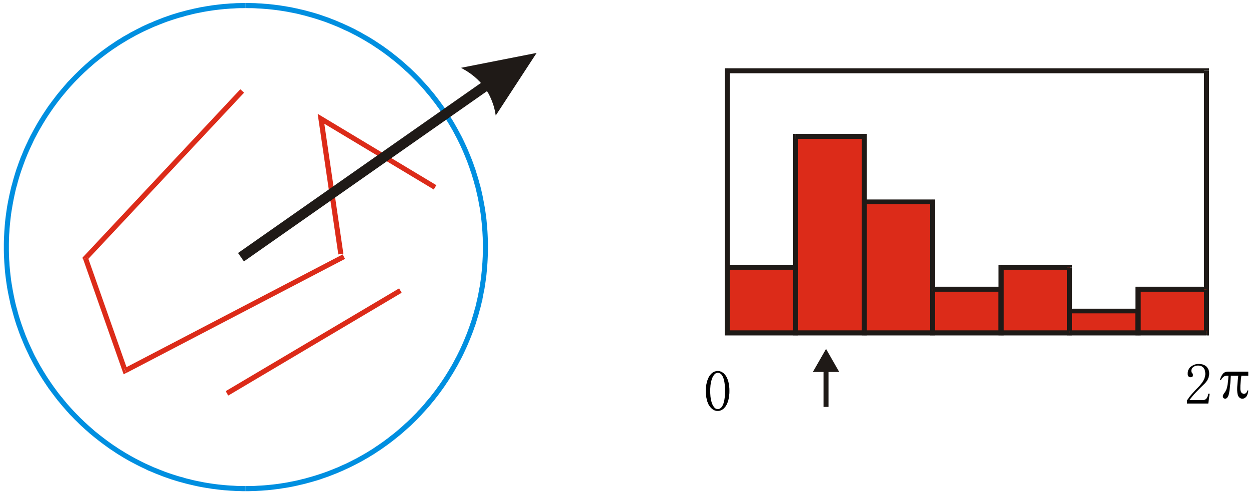


kpt.size = sigma*powf(2.f, (layer + xi) / nOctaveLayers)*(1 << octv)*2;

- How do we represent the patches around the interest points?
- How do we make sure that representation is invariant?

Keypoint orientations

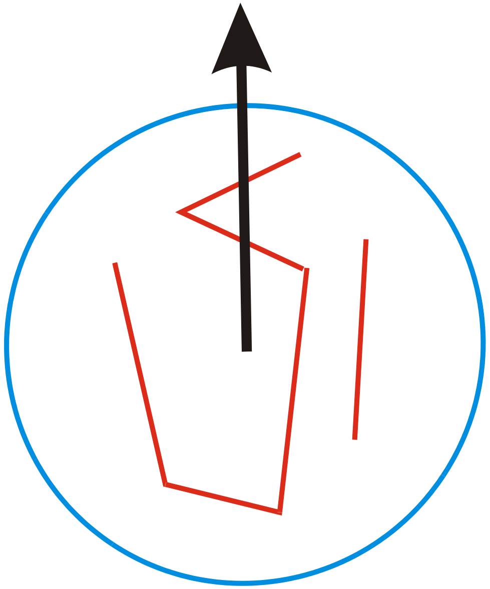
- Compute orientation histogram
- Select dominant orientation

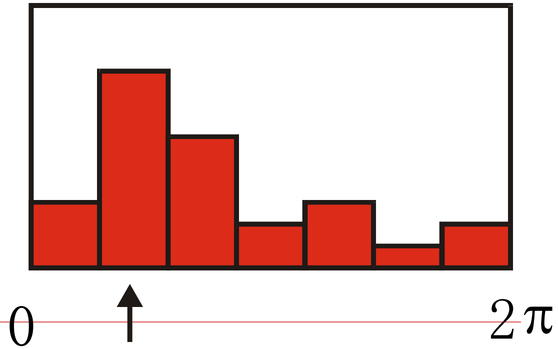


Orientation Normalization

[Lowe, SIFT, 1999]

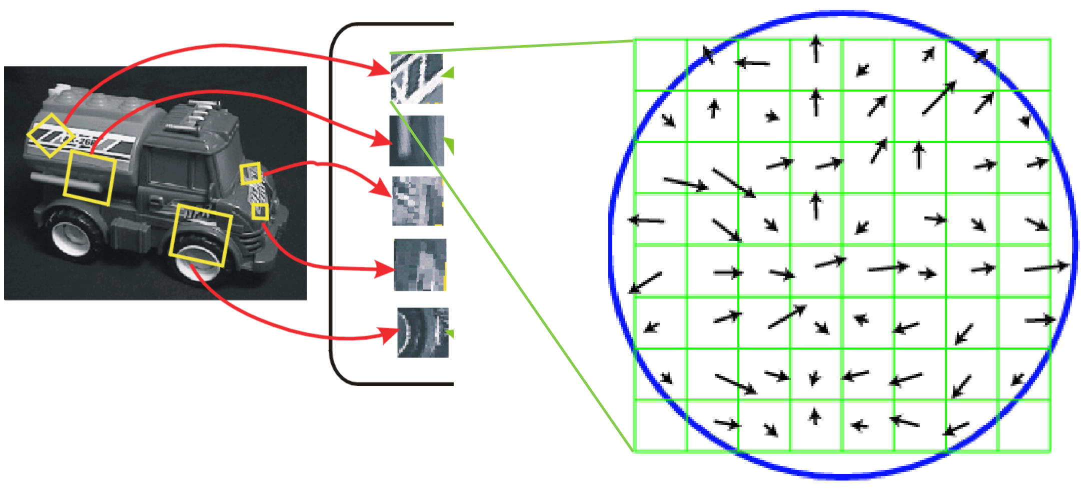
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation





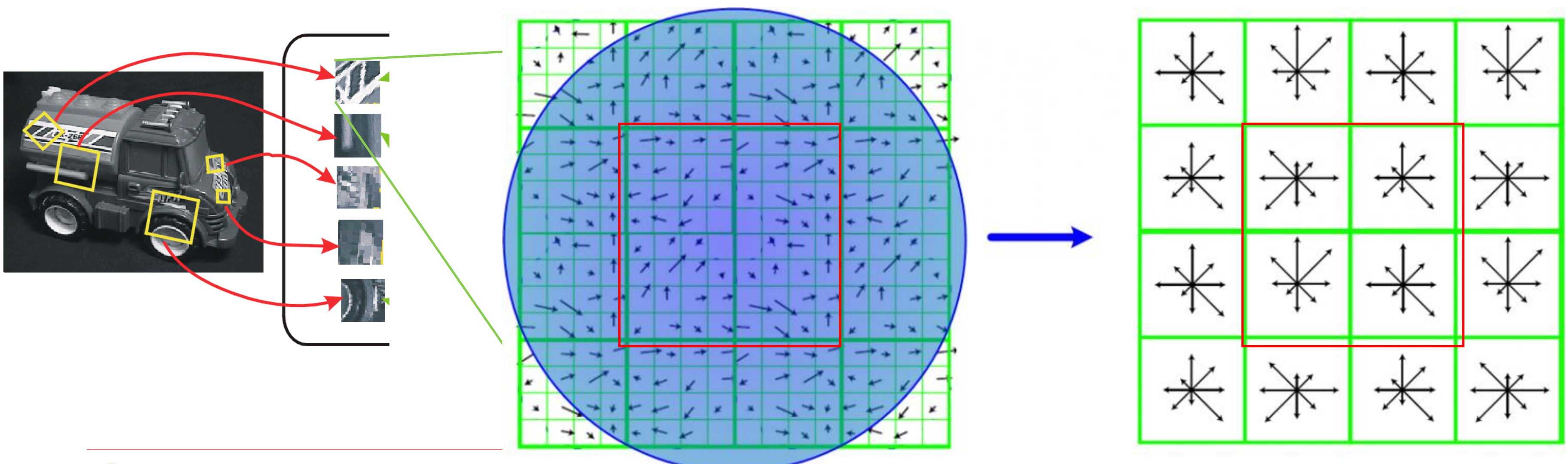
SIFT vector formation

- Computed on rotated and scaled version of window according to computed orientation & scale
  - resample the window
- Based on gradients weighted by a Gaussian of variance half the window (for smooth falloff)



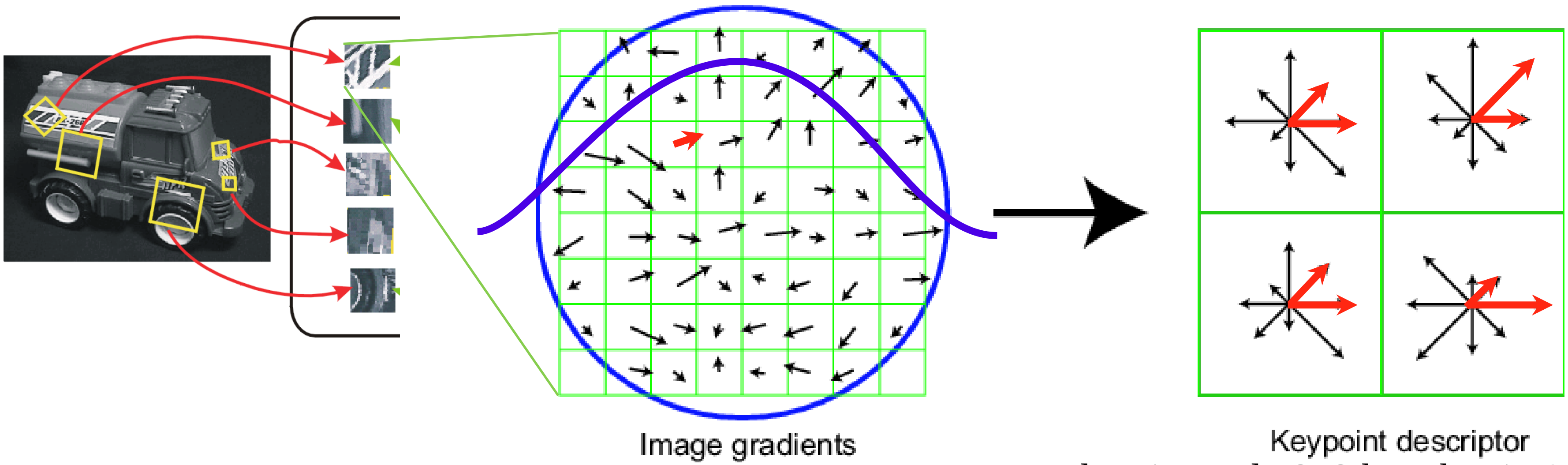
SIFT vector formation

- 4x4 array of gradient orientation histogram weighted by magnitude
- 8 orientations x 4x4 array = 128 dimensions
- Motivation: some sensitivity to spatial layout, but not too much.



Ensure smoothness

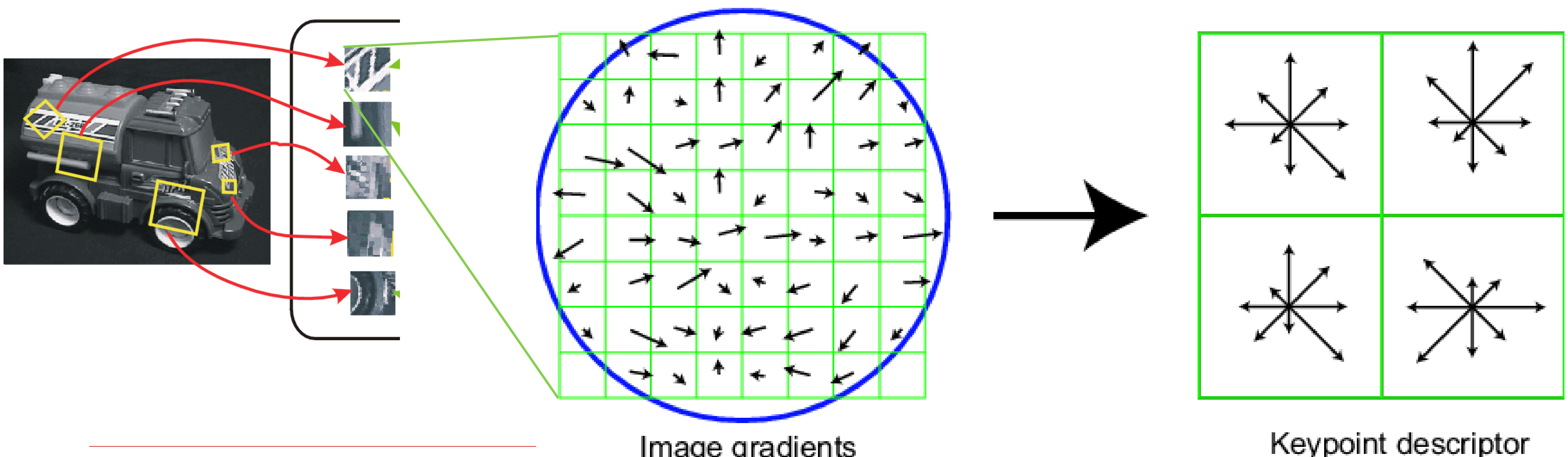
- Gaussian weight
- Trilinear interpolation
  - a given gradient contributes to 8 bins: 4 in space times 2 in orientation



showing only 2x2 here but is 4x4

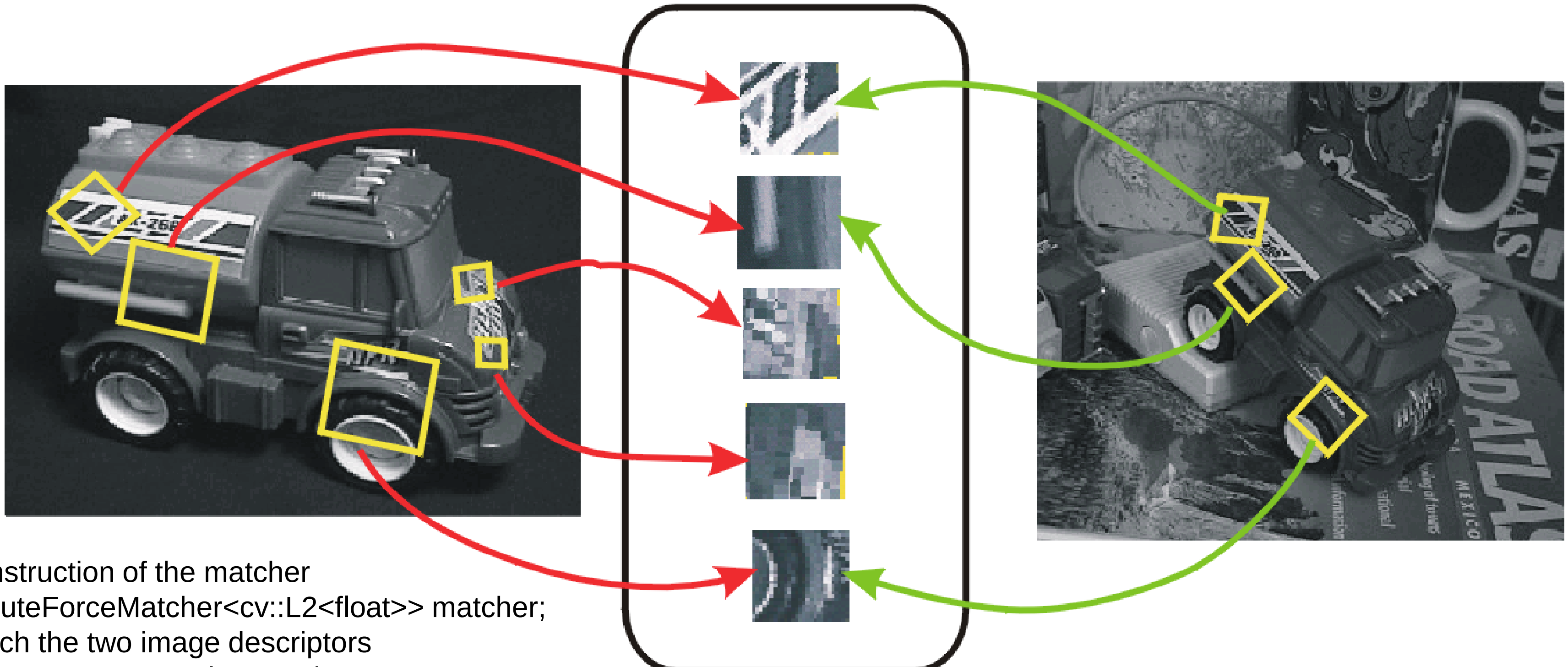
Reduce effect of illumination

- 128-dim vector normalized to 1
- Threshold gradient magnitudes to avoid excessive influence of high gradients
  - after normalization, clamp gradients >0.2
  - renormalize



Invariant Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



```
// Construction of the matcher
cv::BruteForceMatcher<cv::L2<float>> matcher;
// Match the two image descriptors
std::vector<cv::DMatch> matches;
matcher.match(descriptors1,descriptors2, matches);
```

SIFT Features

Sony Aibo (Evolution Robotics)

SIFT usage:

- Recognize charging station
- Communicate with visual cards

Examples

Advanced Features: Topics

- Advanced Edge Detection (last time)
- Global Image Features (Hough Transform)

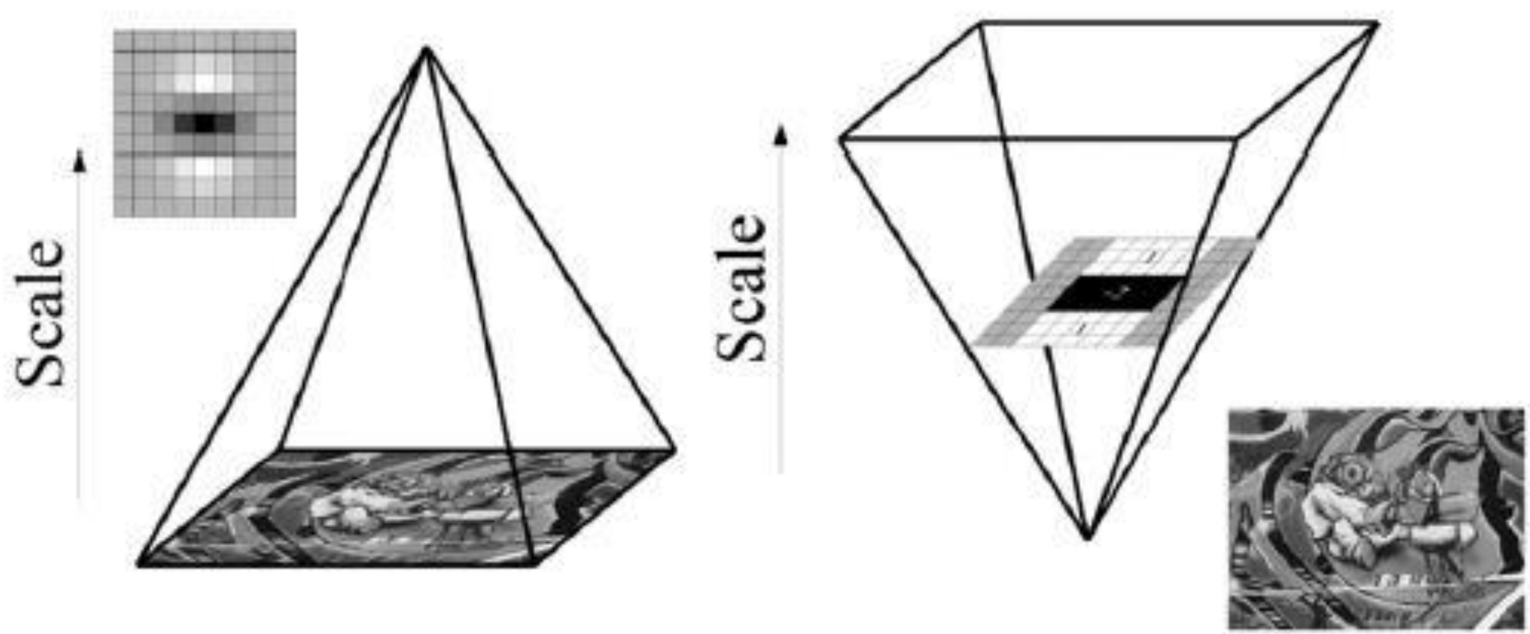
- Corner Detector
- SIFT Features
- Learning with Many Simple Features

SURF

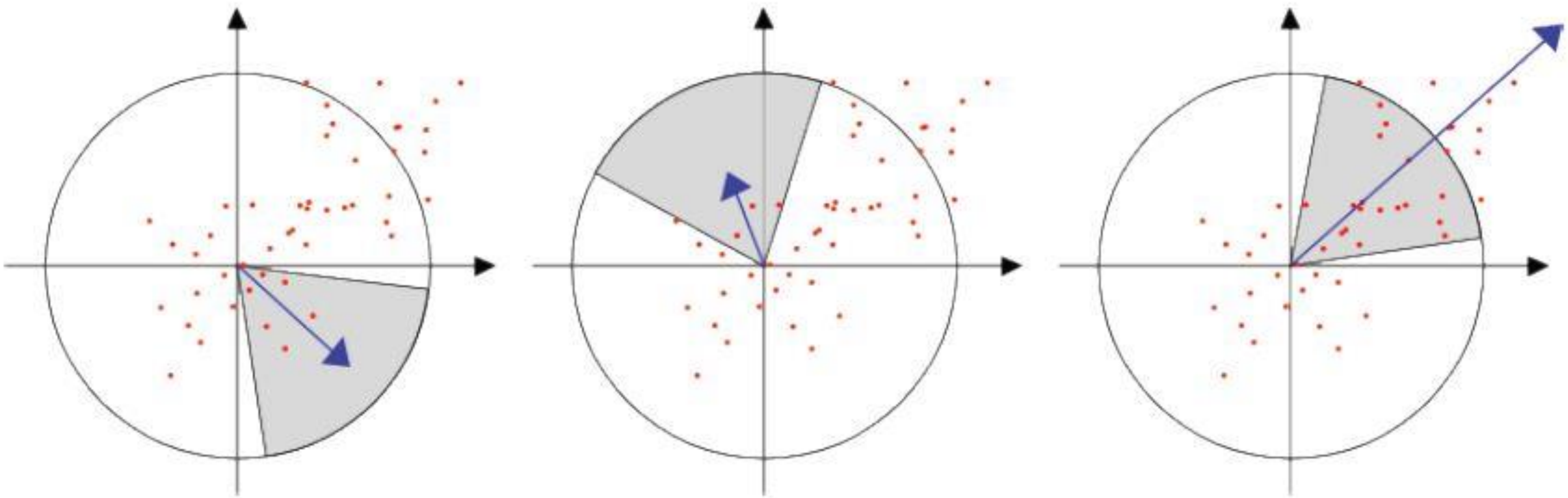
SURF Speeded Up Robust Features，号称是SIFT算法的增强版，SURF算法的计算量小，运算速度快，提取的特征点几乎与SIFT相同，由Bay 2006年提出。

	SIFT	SURF
特征点检测	用不同尺度的图片与高斯函数做卷积	用不同大小的box filter与原始图像(integral image)做卷积， 易于并行
方向	特征点邻接矩形区域内， 利用梯度直方图计算	特征点邻接圆域内， 计算x、y方向上的Haar小波响应
描述符生成	20*20(单位为pixel)区域划分为4*4(或2*2)的子区域， 每个子域计算8bin直方图	20*20(单位为sigma)区域划分为4*4子域， 每个子域计算5*5个采样点的Haar小波响 应， 记录∑dx，∑dy，∑|dx|，∑|dy|。

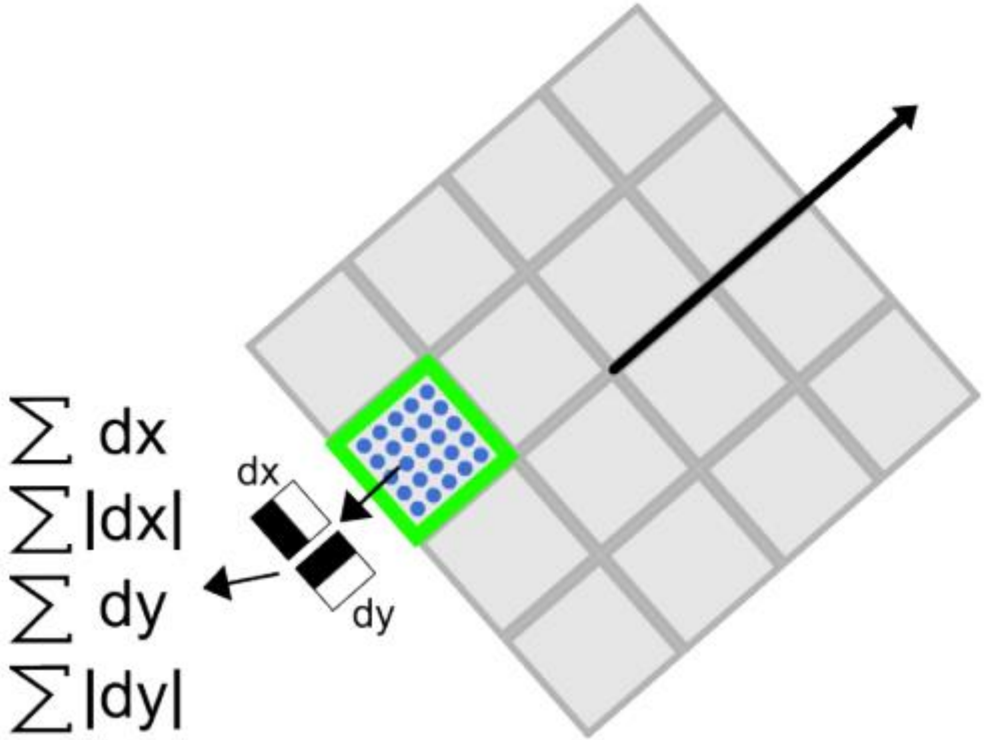
SURF-Scales



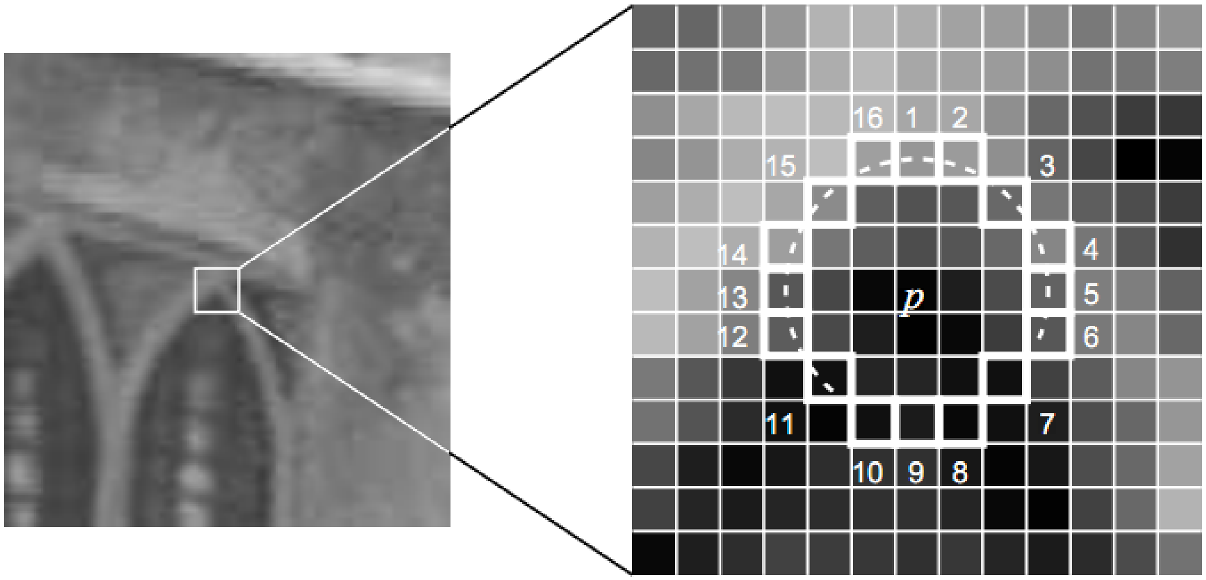
SURF Direction of Keypoints



SURF Descriptor



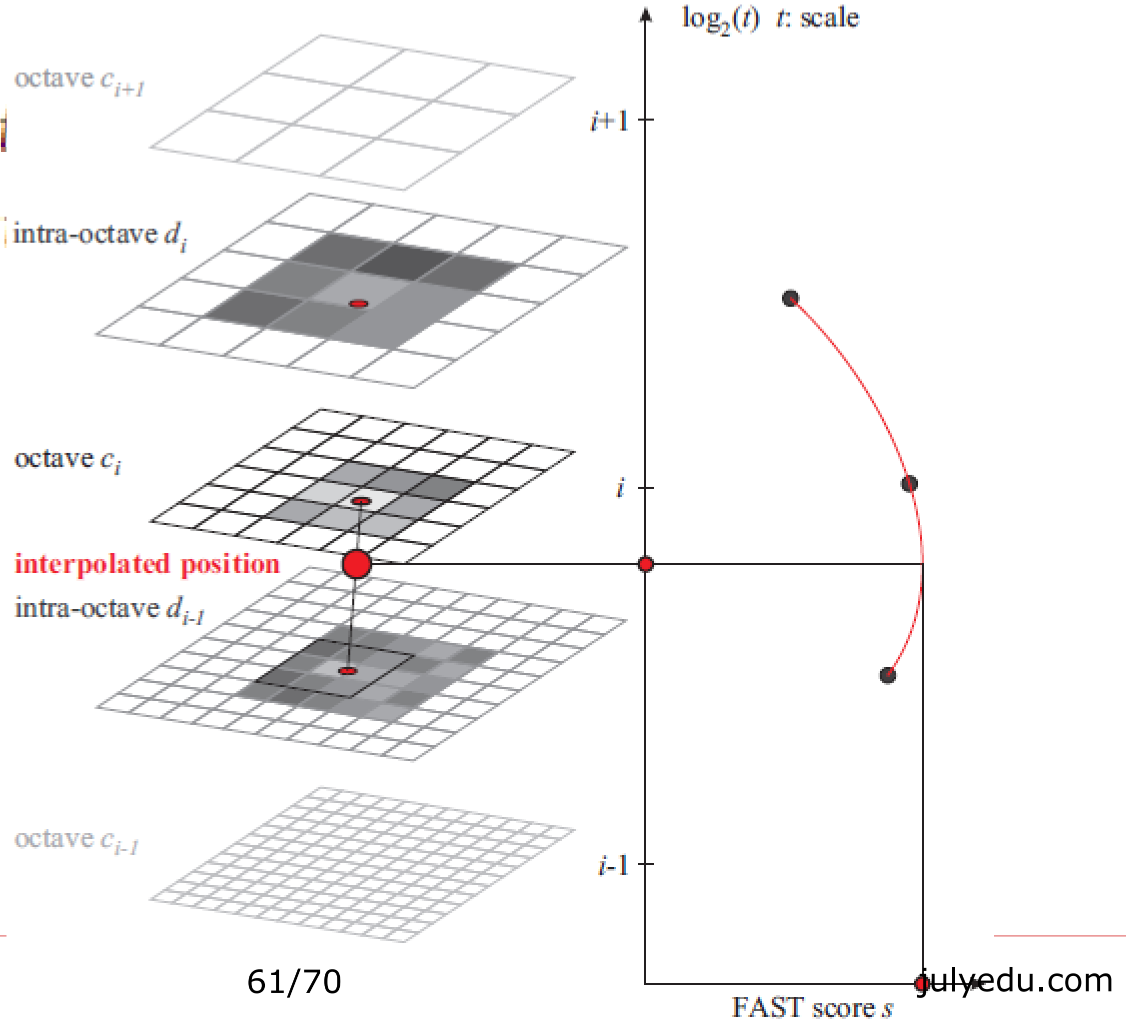
FAST Features from accelerated segment test



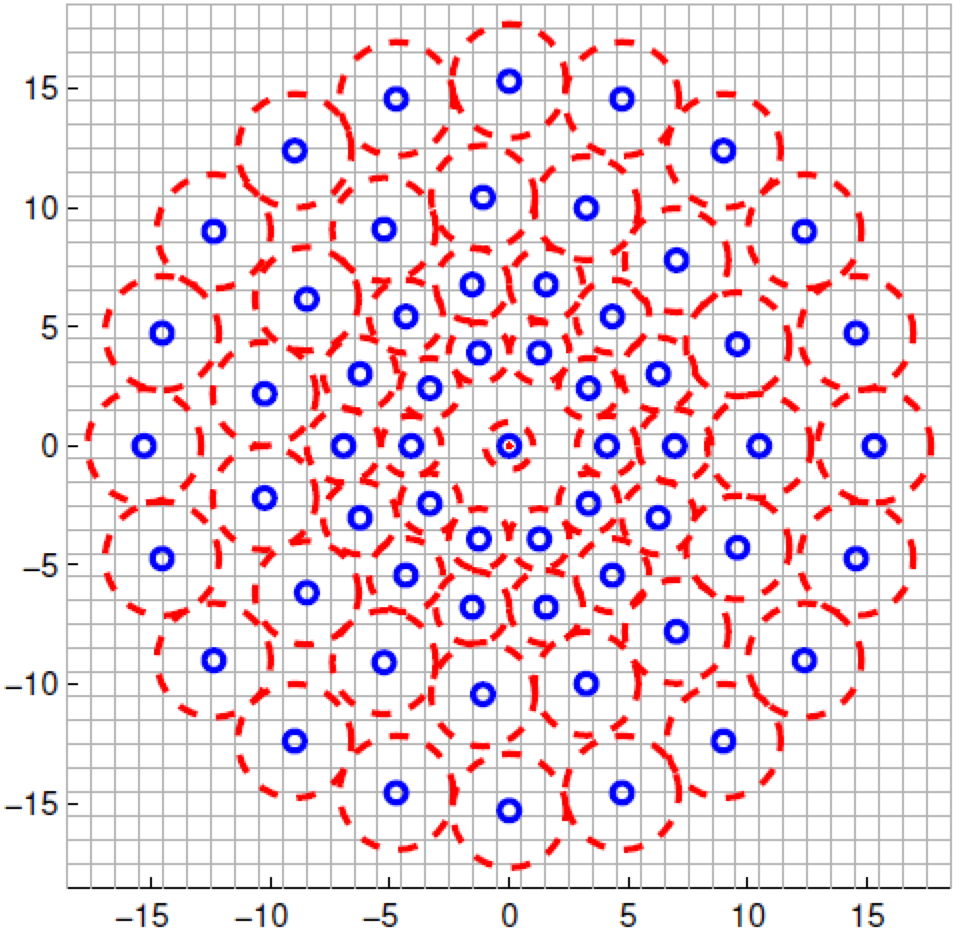

```
// vector of keypoints
std::vector<cv::KeyPoint> keypoints;
// Construction of the Fast feature detector object
cv::FastFeatureDetector fast(
        40); // threshold for detection
// feature point detection
fast.detect(image,keypoints);
```

BRISK:Binary Robust Invariant Scalable Keypoints

img	h(high)	w(width)
c0	h	w
d0	$\frac{2}{3}$h	$\frac{2}{3}$h
c1	$\frac{1}{2}$h	$\frac{1}{2}$h
d1	$\frac{1}{3}$h	$\frac{1}{3}$h
c2	$\frac{1}{4}$h	$\frac{1}{4}$h
d2	$\frac{1}{6}$h	$\frac{1}{6}$h
c3	$\frac{1}{8}$h	$\frac{1}{8}$h
d3	$\frac{1}{12}$h	$\frac{1}{12}$h



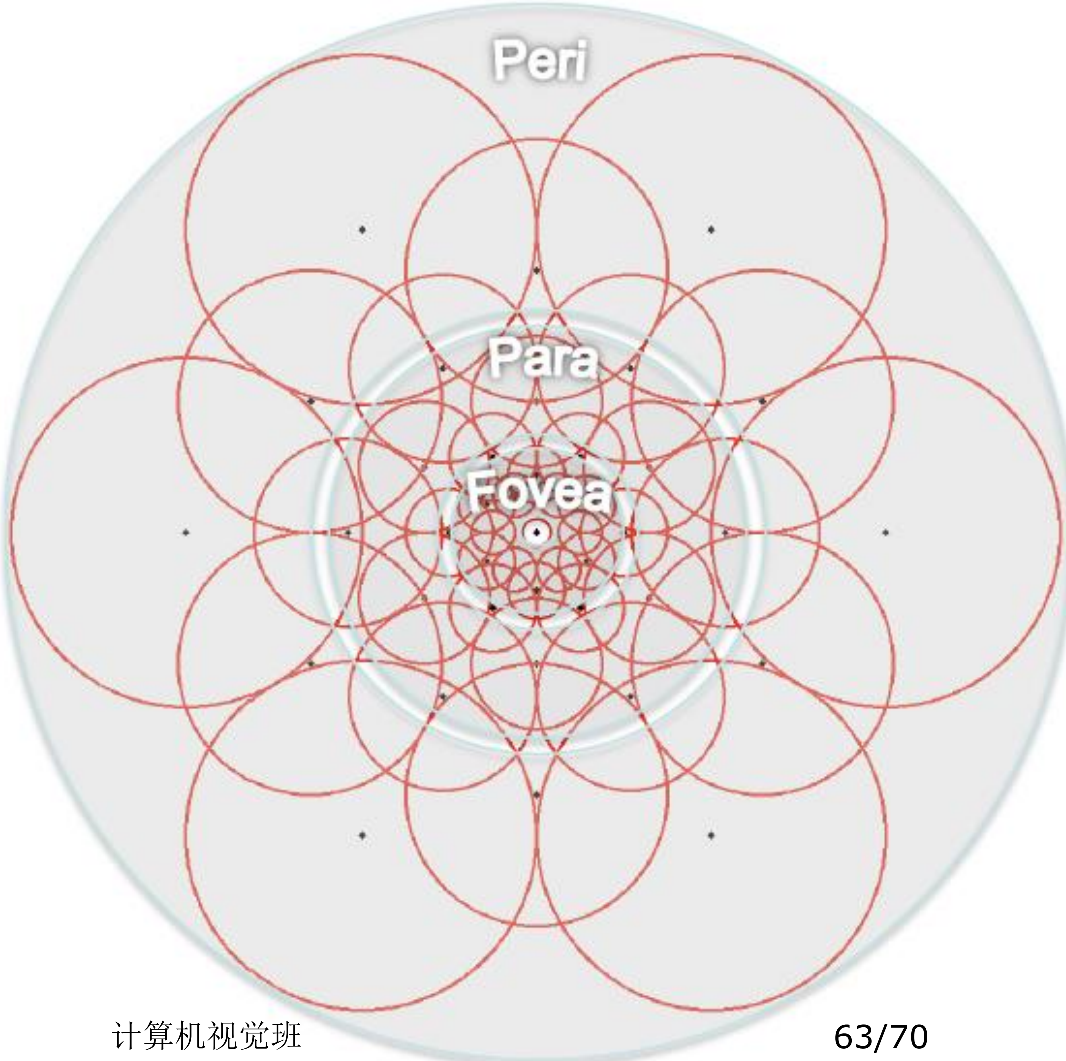
BRISK Descriptor



$$b = \begin{cases} 1, & I(\mathrm{p}_j^\alpha, \sigma_j) > I(\mathrm{p}_i^\alpha, \sigma_i) \\ 0, & \text{otherwise} \end{cases}$$

$$\forall(\mathrm{p}_i^\alpha, \mathrm{p}_j^\alpha) \in \mathcal{S}$$

FREAK: Fast Retina Keypoint



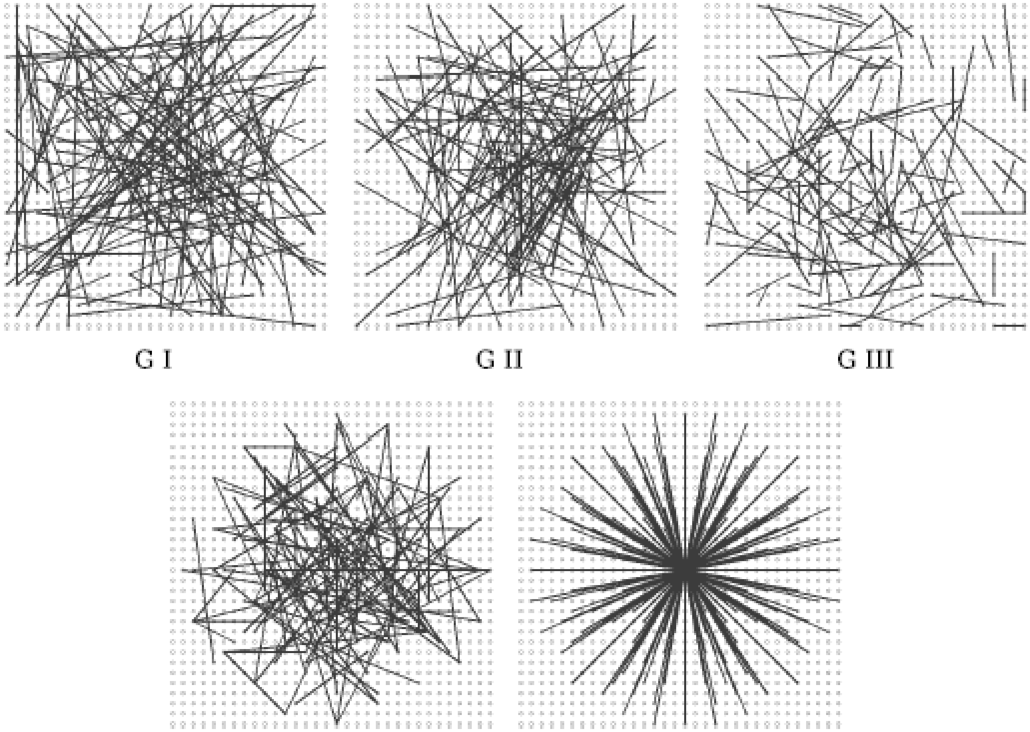
$$T(P_a) = \begin{cases} 1 & \text{if } (I(P_a^{r_1}) - I(P_a^{r_2}) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

$$F = \sum_{0 \le a < N} 2^a T(P_a)$$

BRIEF Binary Robust Independent Elementary Features

- BRIEF is just a descriptor

$$\tau(p;x,y) := \begin{cases} 1 & if p(x) < p(y) \\ 0 & otherwise \end{cases}$$



G I G II G III

G IV G V

ORB An efficient alternative to SIFT or SURF

- Detector

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

$$\theta = \operatorname{atan2}(m_{01}, m_{10})$$

- Descriptor

$$\mathbf{S} = \begin{pmatrix} \mathbf{x}_1, \ldots, \mathbf{x}_n \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \end{pmatrix}$$

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

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S}$$

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases}$$

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

```
ORB::ORB(int nfeatures=500, float scaleFactor=1.2f, int nlevels=8, int dgeThreshold=31,
int firstLevel=0, int WTA_K=2, int scoreType=ORB::HARRIS_SCORE, int patchSize=31)
```

Featuredetector

- Ptr<FeatureDetector> FeatureDetector::create(const string& detectorType)
- Ptr<FeatureDetector> FeatureDetector::create(const string& detectorType)
- // "FAST" – FastFeatureDetector
- // "STAR" – StarFeatureDetector
- // "SIFT" – SIFT (nonfree module)
- // "SURF" – SURF (nonfree module)
- // "ORB" – ORB
- // "MSER" – MSER
- // "GFTT" – GoodFeaturesToTrackDetector
- // "HARRIS" – GoodFeaturesToTrackDetector with Harris detector enabled
- // "Dense" – DenseFeatureDetector
- // "SimpleBlob" – SimpleBlobDetector

DescriptorExtractor

- FREAK
- OpponentColorDescriptorExtractor
- BriefDescriptorExtractor

FeatureDetector&DescriptorExtractor

- BRISK
- **ORB**
- SIFT
- SURF

Advantages of invariant local features

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects
- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to wide range of differing feature types, with each adding robustness

感谢大家！

恳请大家批评指正！