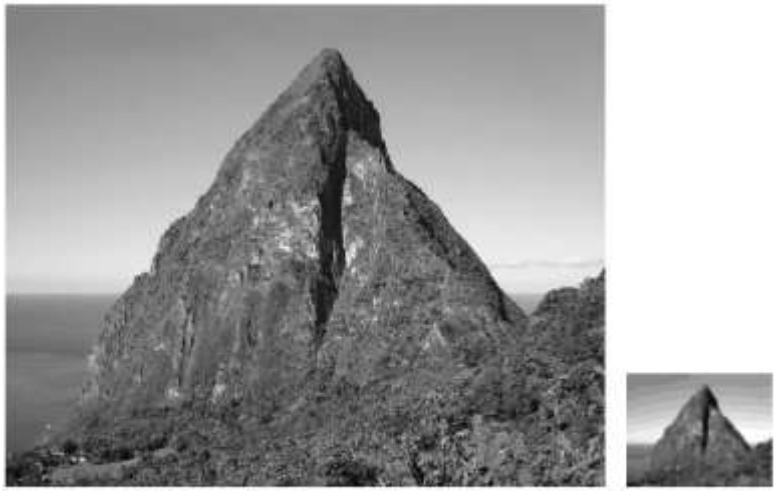
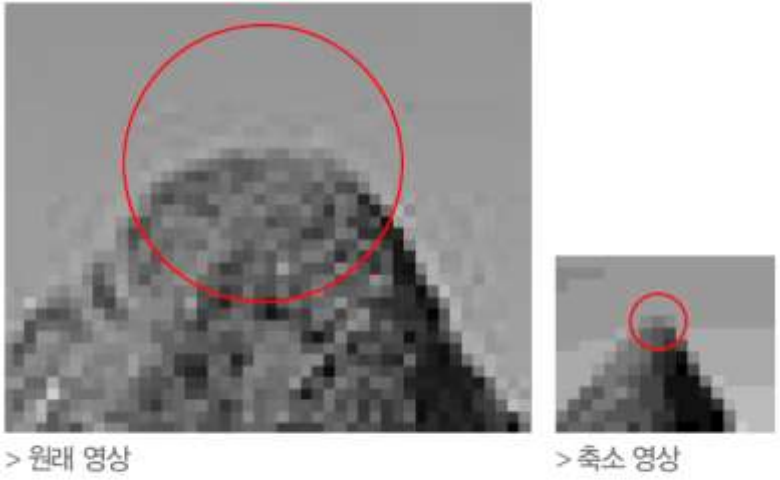
5.3 스케일 불변한 지역 특징

- 거리에 따른 스케일 변화
  - 예) 멀면 작고 윤곽만 어렴풋이 보이다가, 가까워지면 커지면서 세세한 부분 보임
  - 사람은 강인하게 대처하는데, 컴퓨터 비전도 대처 가능한가?



(a) 원래 영상과 1/10로 축소한 영상



(b) 산꼭대기를 확대한 부분 영상

그림 4-12 카메라와 물체 간의 거리에 따른 스케일 변화

연산자의 크기가 달라야 하지 않을까?

5.3 스케일 불변한 지역 특징

- **사람은 스케일 불변인 특징 사용**
  - 거리에 상관없이 같은 물체를 같다고 인식
  - 거리에 따라 세세한 내용에만 차이가 있음
- **스케일 공간$_{\text{scale space}}$ 이론은 컴퓨터 비전에 스케일 불변 가능성 열어줌**

[알고리즘 5-1] 스케일 공간에서 특징점 검출

입력: 명암 영상 f

출력: 스케일에 불변한 특징점 집합

1. 입력 영상 f로부터 다중 스케일 영상 $\tilde{f}$를 구성한다.
2. $\tilde{f}$에 적절한 미분 연산을 적용하여 다중 스케일 미분 영상 $\tilde{f}'$를 구한다.
3. $\tilde{f}'$에서 극점을 찾아 특징점으로 취한다.

스케일 공간

- 다중 스케일 접근 방법(스케일 불변 특징을 찾기 위해)
  - 스케일 축이 추가된 3차원 공간에서 극점(지역 최대 또는 최소점) 검출
  - 극점을 찾기 위해서는 적절한 미분 연산을 적용해서다중 스케일 미분 영상을 구해야 한다.

알고리즘 4-2 다중 스케일 접근 방법

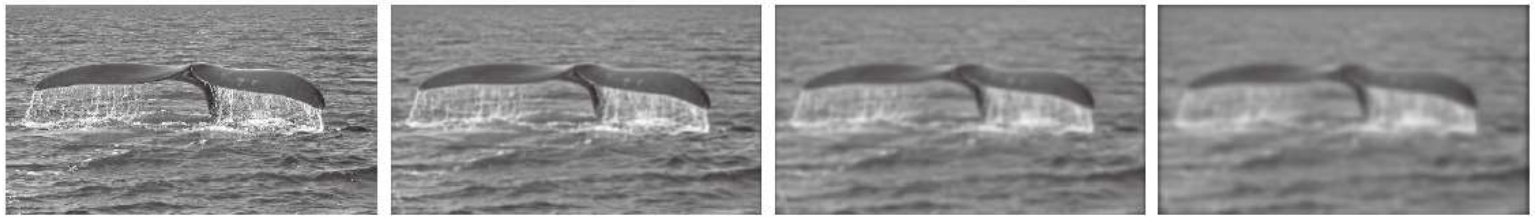
입력 : 영상 $f(j,i), 0 \le j \le M-1, 0 \le i \le N-1$, 임계값 T

출력 : 특징점 리스트 F

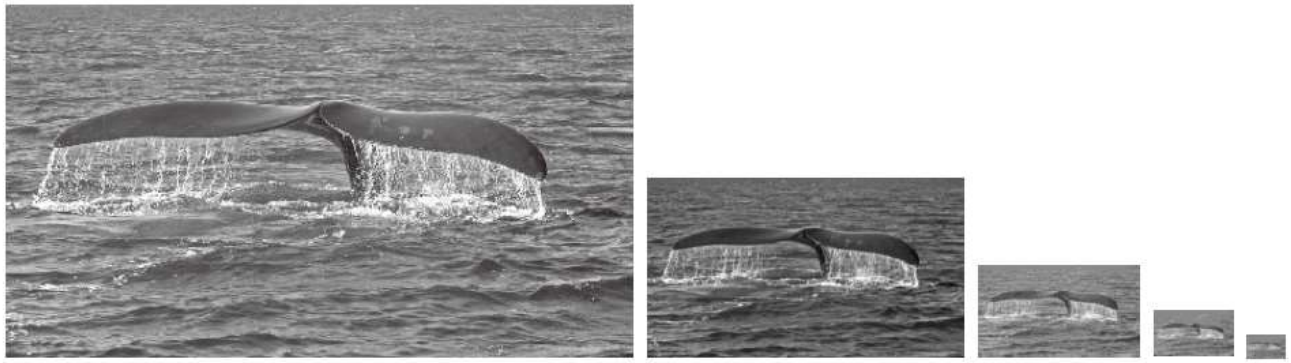
1	f에서 다중 스케일 영상, $M=\{f^{s0}, f^{s1}, f^{s2}, \cdots\}$를 구성한다. // f^{si}는 스케일이 s_i인 영상
2	M에서 3차원 극점을 찾아 특징점 집합 F로 취한다. // 극점 (y,x,s)는 스케일 불변이어야 함.

5.3 스케일 불변한 지역 특징

- 다중 스케일 영상을 구현하는 두 가지 방식
  - 가우시안 스무딩: 스케일에 해당하는 σ가 연속 공간에 정의
    - 거리가 멀어지면 물체의 세밀한 부분은 사라지고 윤곽은 점점 흐릿해 짐
  - 피라미드: ½씩 줄어들므로 이산적인 단점
    - 물체가 멀어지면 물체의 크기가 작아지는 효과 모방



(a) 가우시안 스무딩 방법



(b) 피라미드 방법

그림 5-7 다중 스케일 영상을 구성하는 두 가지 방법[오일석2014]

5.3 스케일 불변한 지역 특징

- 가우시안 스무딩에 의한 스케일 공간: 다중 스케일 영상
  - 스케일 축을 추가한 3차원 공간: (y,x,t)공간, $t = \sigma^2$: 스케일 매개변수

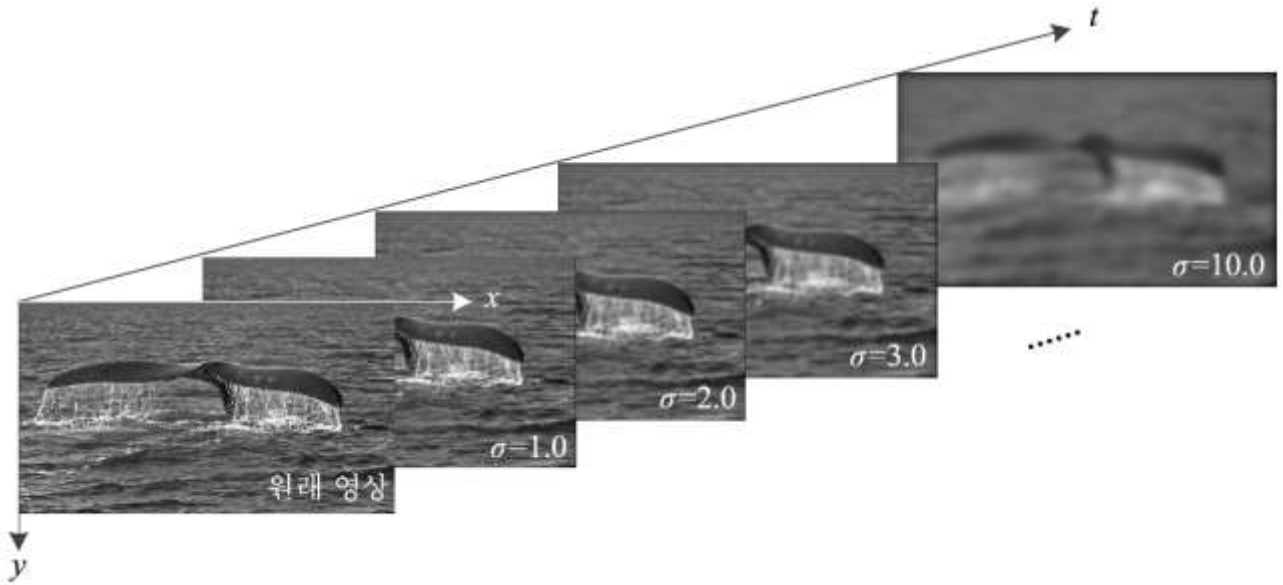


그림 5-8 스케일 공간 (y, x, t)에 정의된 다중 스케일 영상[오일석2014]

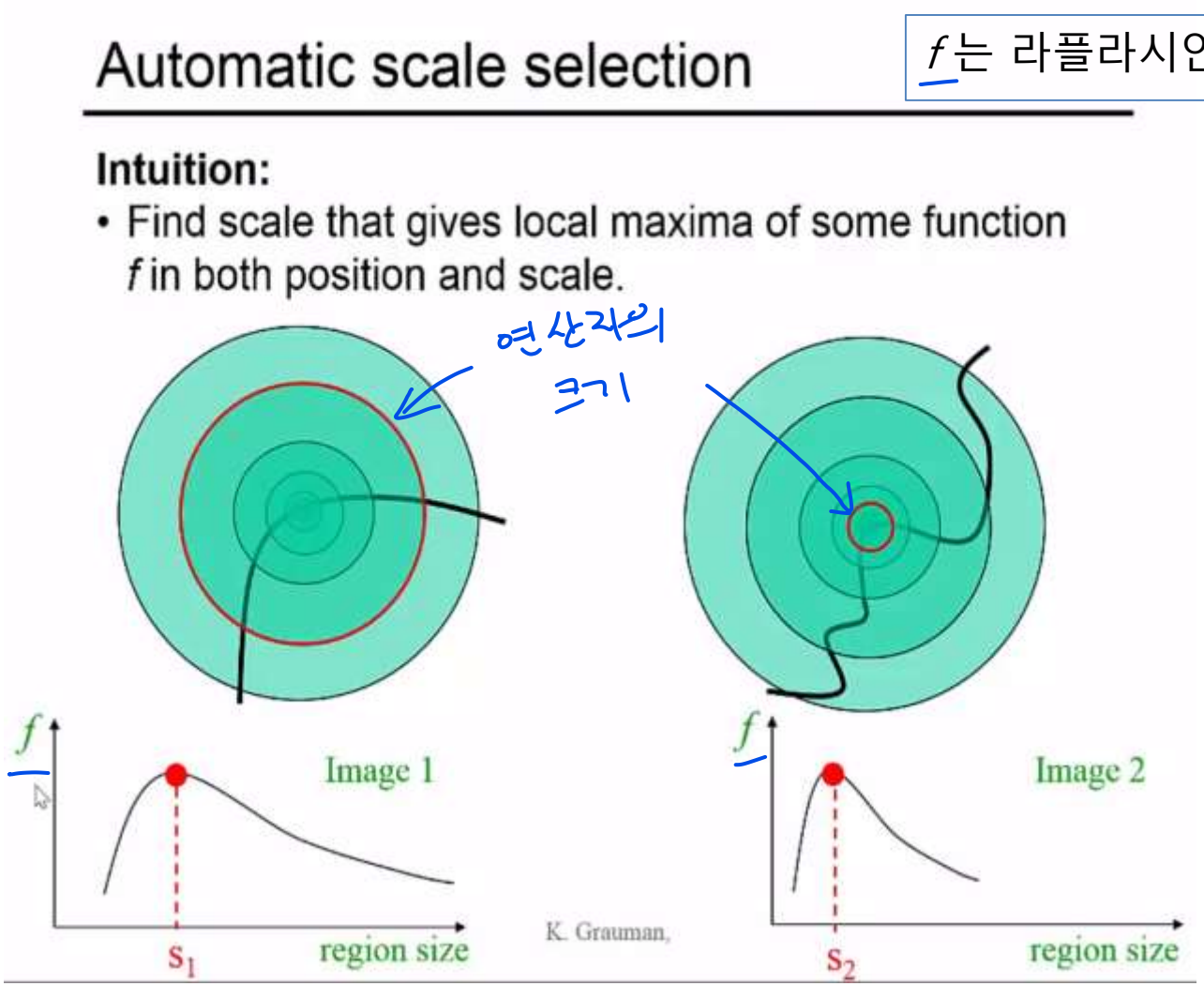
- 스케일 공간의 미분 ([알고리즘 5-1]의 2행)은 정규 라플라시안 사용

$$\text{라플라시안: } \nabla^2 f = d_{yy} + d_{xx} \qquad (5.10)$$

$$\text{정규 라플라시안: } \nabla^2_{normal} f = \sigma^2 \mid d_{yy} + d_{xx} \mid \qquad (5.11)$$

스케일 공간

- t 축에서 지역 극점 탐색: t 가 커질 수록 디테일은 사라지고 큰 에지만 남는다

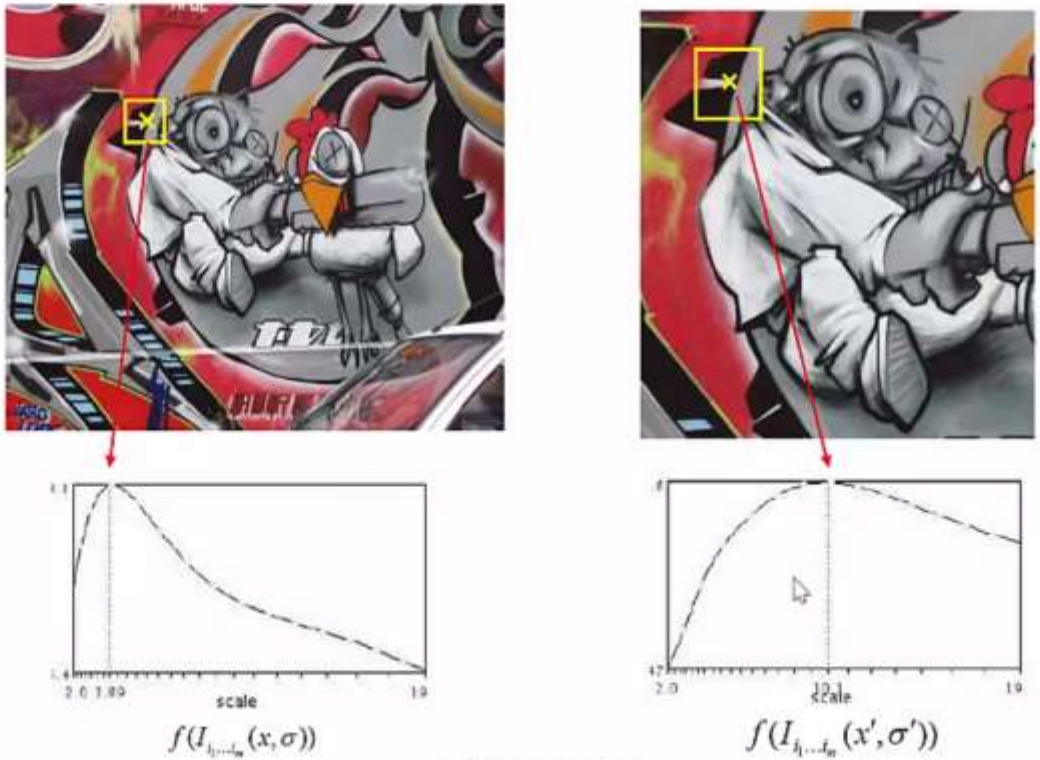


스케일 공간

- t 축에서 지역 극점 탐색: σ 를 다르게 함으로 연산자 크기 조정 가능

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

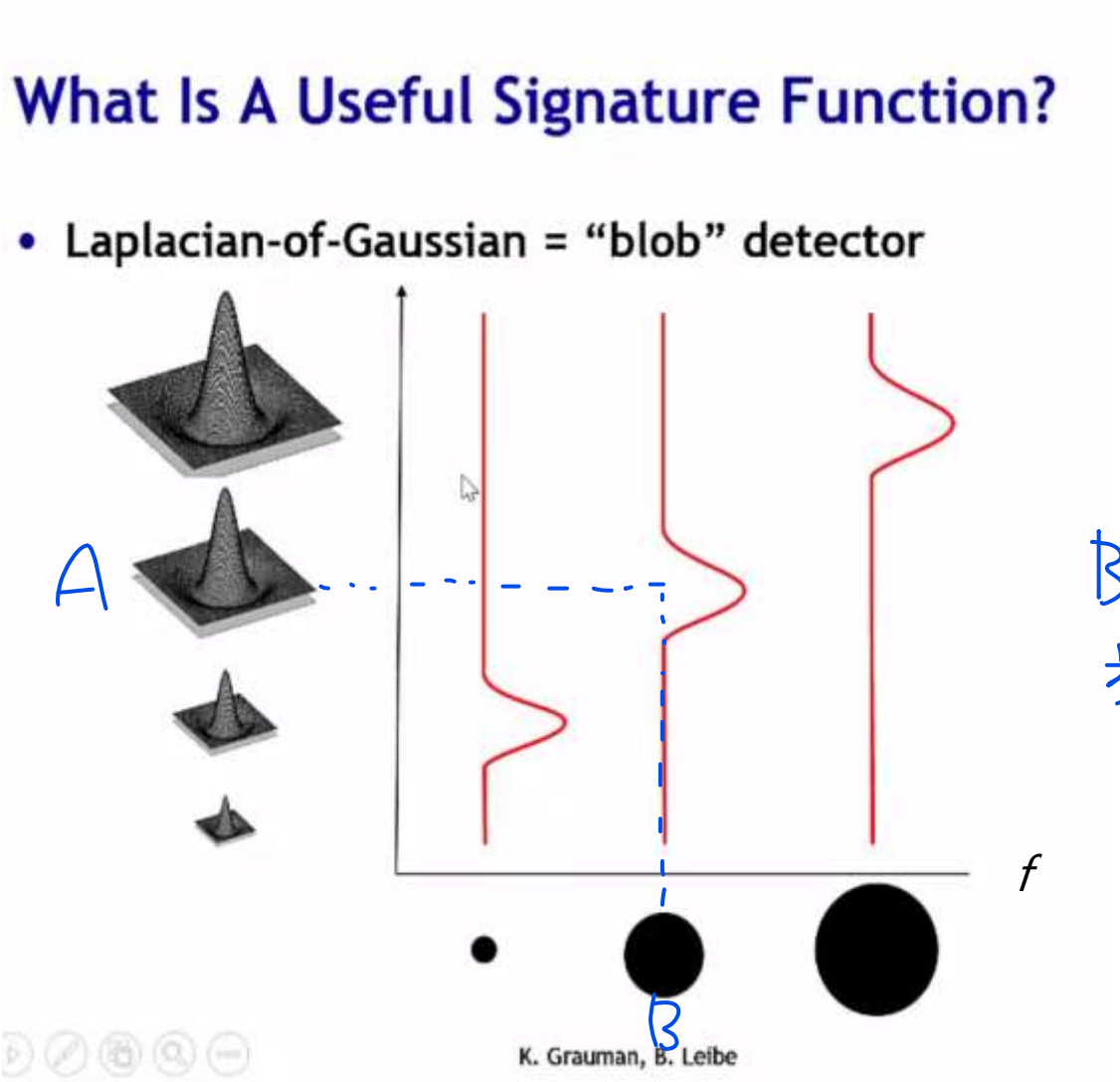


K. Grauman, B. Leibe

f 는 라플라시안 LoG

스케일 공간

- t 축에서 지역 극점 탐색: σ 를 다르게 함으로 연산자 크기 조정 가능



f 는 LOG를 blob의 센터에 적용한 결과 값

blob이란 이미지에서 extrema같은 부분들이다. 이 부분들은 순간적으로 밝아지거나 어두워지는 부분을 말한다.

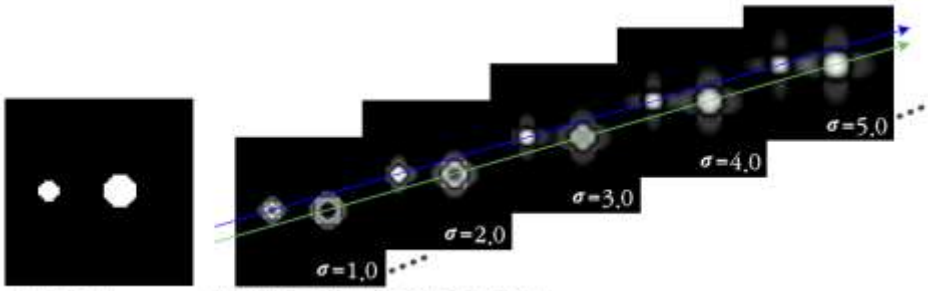
B 같은 특징점을 찾으려면 A 와 같은 LOG 필터 필요

스케일 공간

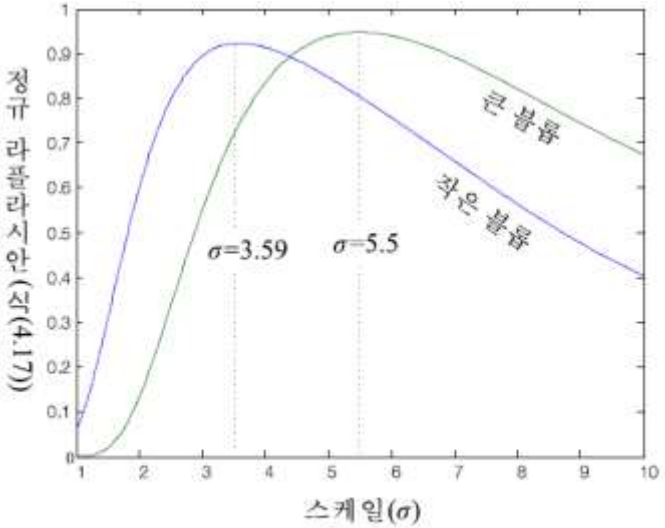
- t 축에서 지역 극점 탐색: 관심점
  - t 축을 따라 정규 라플라시안 측정해 보면, 극점 발생함

$$\nabla^2_{norm} f = \sigma^2 \left| d_{yy}(\sigma) + d_{xx}(\sigma) \right| \qquad (4.17)$$

  - 실험에 따르면 t 축에서 정규 라플라시안이 가장 안정적으로 극점 생성
  - 극점의 σ 값은 물체의 스케일에 해당
  - 동일한 객체의 두 다른 크기의 영상에서 얻은 극점의 σ 비율은 영상 크기의 비율에 대응
  - 직경 7, 11인 원형 blob의 중점((y,x) 공간에서 극점)
  - t축에서의 극점 발생: σ= 3.59, 5.6
    - 11/7 = 1.57 ≈ 5.6/3.59 = 1.53



(a) 원래 영상 (b) 스케일 공간에서 정규 라플라시안 영상

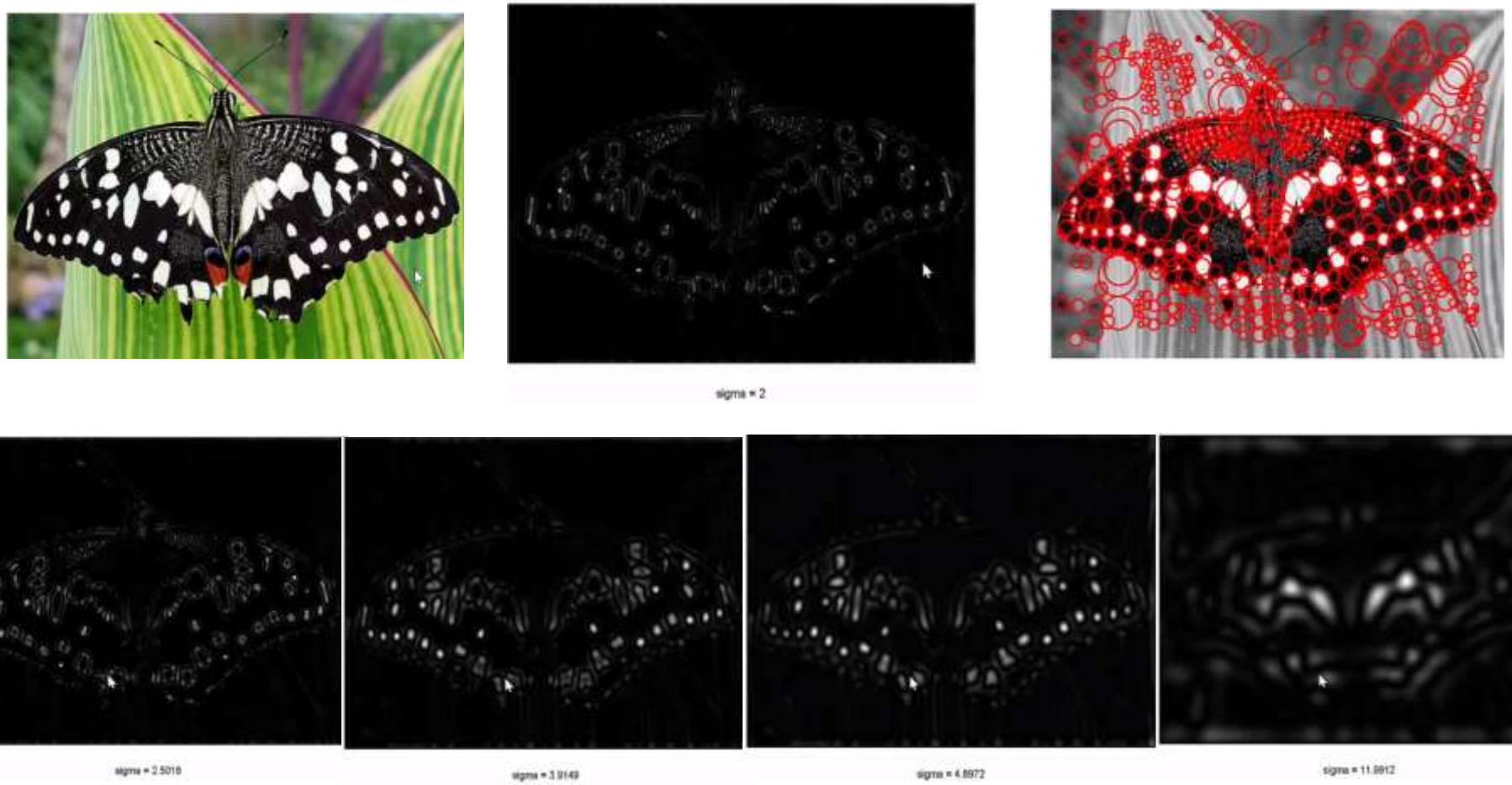


(c) 스케일 축에서 극점

그림 4-15 스케일 공간 (y, x, t)에서 극점 검출

스케일 공간

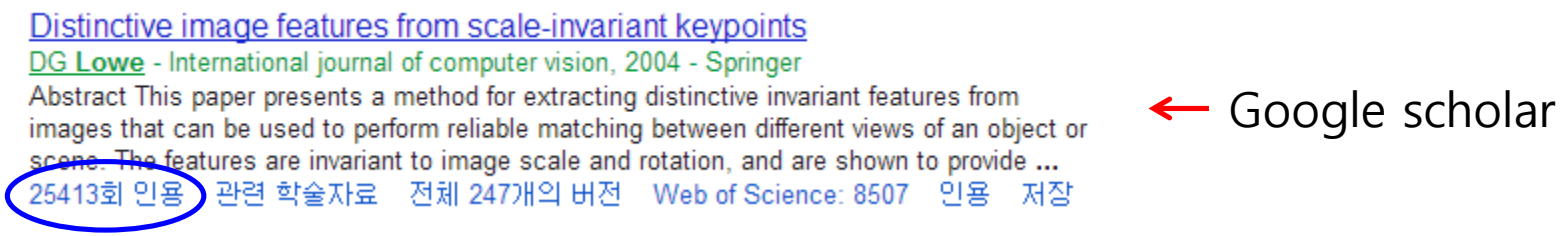
- t 축에서 지역 극점 탐색: 모든 픽셀에 정규 라플라시안 적용



원의 크기는 극점(관심점, interesting poinyt, key point)을 생성하는 스케일(σ) 크기

5.4 SIFT

- SIFT의 등장(scale invariant feature transform)
  - 1999년 David Lowe 교수의 논문 [Lowe99]
  - 2004년 IJCV에 확장된 논문 발표 [Lowe2004]
  - 성능이 뛰어나 현재 가장 널리 사용되며, 다양한 변형이 개발되어 있음
    - 반복성이 뛰어나다
    - 계산 시간이 빨라서 실시간 처리 가능



5.4 SIFT

■ SIFT는 [알고리즘 5-1]을 구현하는 가장 성공적인 방법

- 5.4.1절은 [알고리즘 5-1]로 SIFT 검출하는 과정
- 5.4.2절은 특징점에서 기술자를 추출하는 과정
- 5.5절은 특징점을 빠르게 매칭하는 방법

[알고리즘 5-1] 스케일 공간에서 특징점 검출

입력: 명암 영상 f

출력: 스케일에 불변한 특징점 집합

1. 입력 영상 f로부터 다중 스케일 영상 $\tilde{f}$를 구성한다.
2. $\tilde{f}$에 적절한 미분 연산을 적용하여 다중 스케일 미분 영상 $\tilde{f}'$를 구한다.
3. $\tilde{f}'$에서 극점을 찾아 특징점으로 취한다.

5.4.1 SIFT 검출

- 1단계: SIFT의 스케일 공간
  - 피라미드+가우시안 구조
    - 피라미드는 4X4가 될 때 까지 축소
  - 각 층은 여섯 영상의 묶음(옥타브)으로 구성
  - 옥타브의 영상은 σ_i로 스무딩
    - $\sigma_{i+1} = k\sigma_i$ (σ_0=1.6, k=$2^{1/3}$)
  - 옥타브 내에서 위로 올라갈 수록 마스크의 크기(6σ)가 커져서 계산 시간 증가
  - 반복성이 매우 좋음

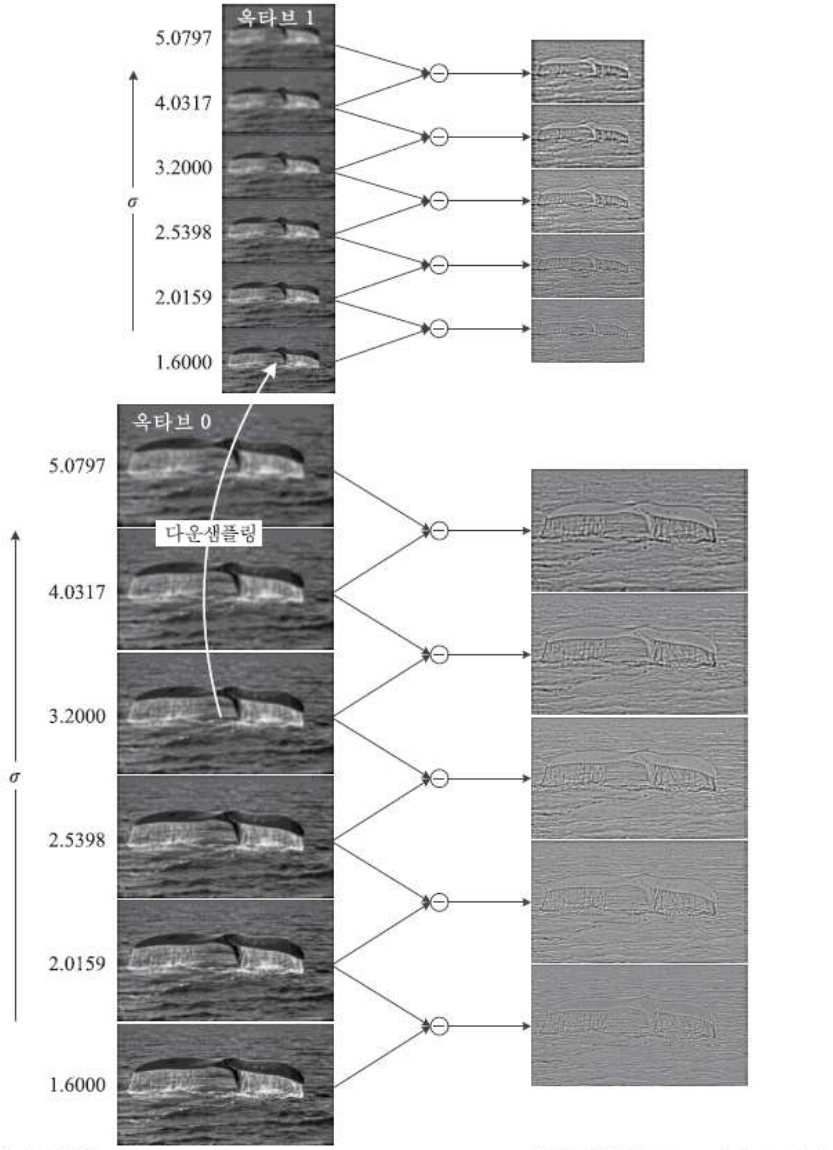


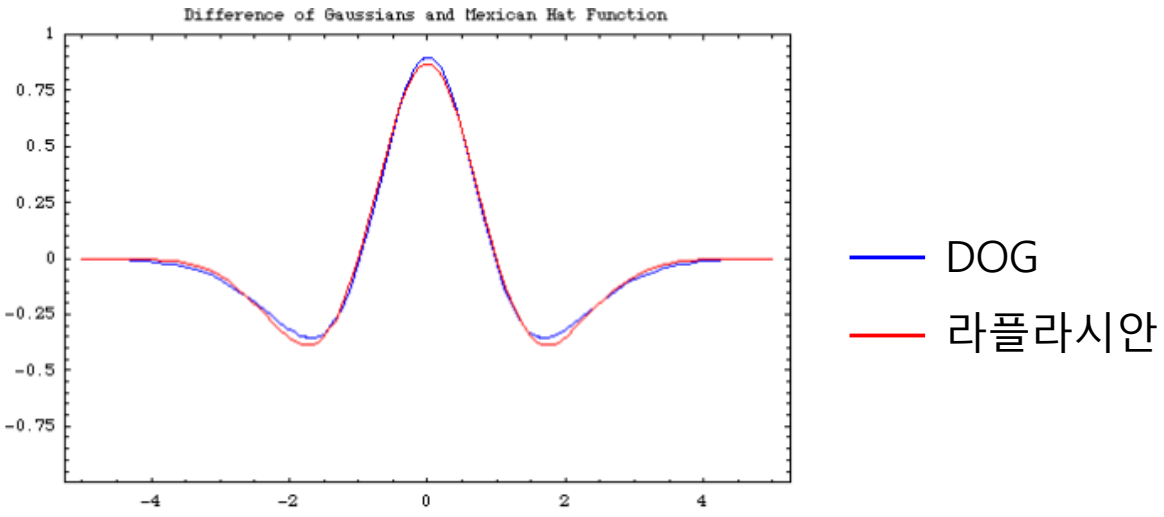
그림 4-16 SIFT가 사용하는 스케일 공간(DOG 피라미드 구조)

5.4.1 SIFT 검출

- 2단계: 정규 라플라시안 맵 구축 정규 라플라시안 ↑, 특징 가능성↑
  - [Mikolajczik2002a]의 실험 결과에 따르면, 정규 라플라시안이 가장 안정적으로 극점 형성
  - 정규 라플라시안과 유사한 DOG 계산으로 대치
    - DOG는 단지 차영상을 계산하므로 매우 빠름

둘은 매우 비슷하다

$$DOG(\sigma_i) = G(\sigma_{i+1}) \circledast f - G(\sigma_i) \circledast f$$
$$= G(k\sigma_i) \circledast f - G(\sigma_i) \circledast f = (G(k\sigma_i) - G(\sigma_i)) \circledast f \quad (4.20)$$



5.4.1 SIFT 검출

- 정규 라플라시안 맵 구축

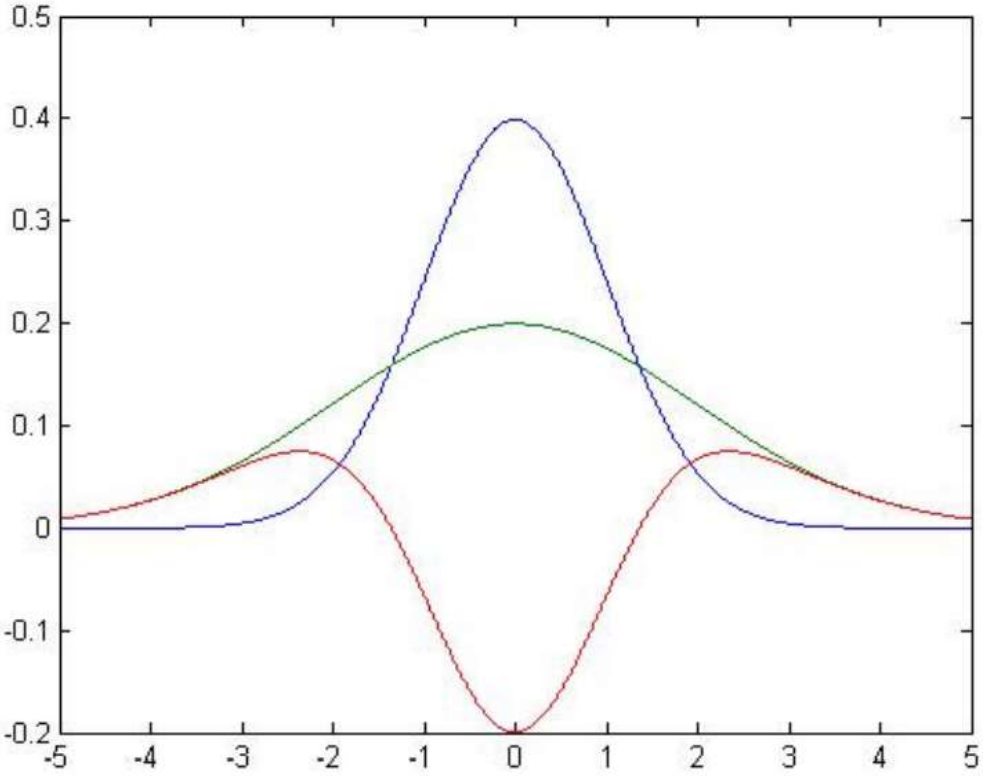
$$\frac{\partial G}{\partial \sigma} = \sigma \Delta^2 G \qquad \text{Heat Equation}$$

$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \Delta^2 G$$

$$\text{Typical values}: \sigma = 1.6;\ k = \sqrt{2}$$

DOG 필터



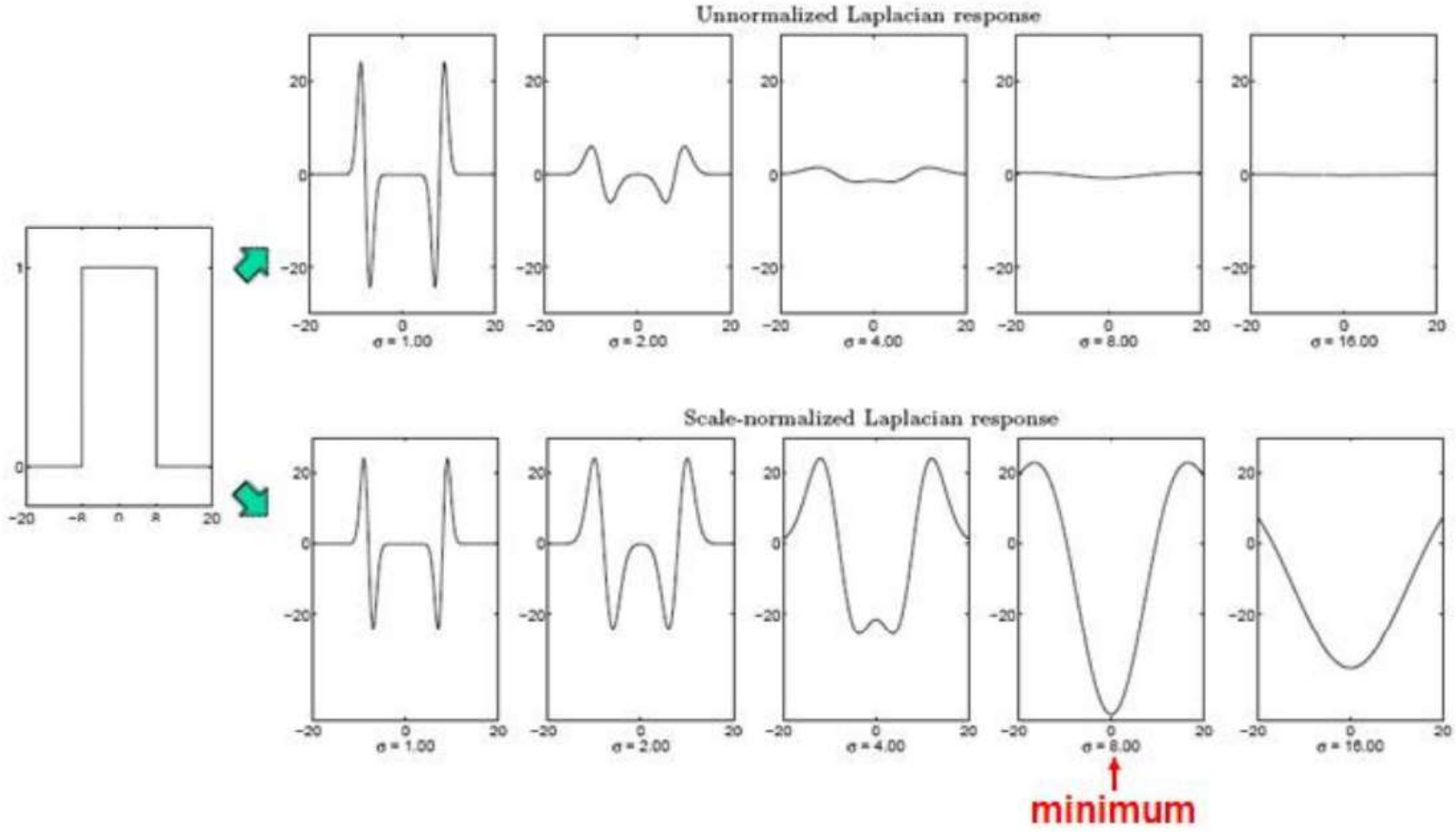
[그림 8]

[그림 8]은 각각 파랑-σ=1 초록-σ=2 빨강-DOG을 나타내는데 보면 DOG는 LOG와 유사한 형태를 보여주고 있다. 이 DOG는 [수식 8]의 과정을 통해서 LOG에 근사되게 된다.

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{(k-1)\sigma}$$

$$DOG \approx (k-1)\sigma^2\nabla^2 G = (k-1)\sigma \times NLOG(Nomalized\,Laplacian\,of\,Gaussian)$$

정규화 라플라시안 필터



[그림 7]

여기서 정규화(Nomalization)을 하기 위해서는 σ를 LOG에 곱해주면 된다.

5.4.1 SIFT 검출

- 3단계: 특징점 (키포인트) 검출

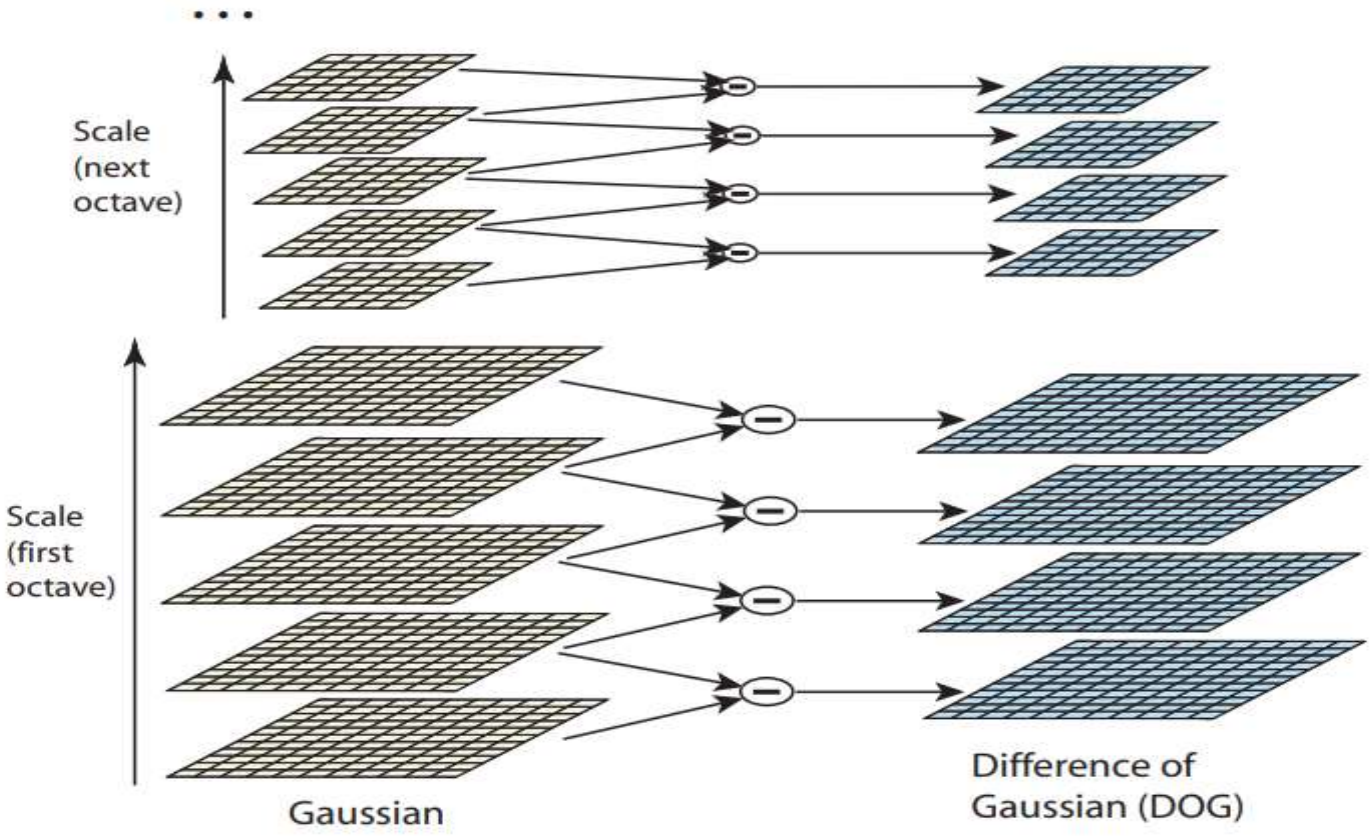


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

5.4.1 SIFT 검출

- 3단계: 특징점 (키포인트) 검출
  - 한 옥타브에는 다섯 장의 DOG 영상
  - 중간에 끼인 세 장의 DOG 맵에서 극점 검출: 실험 결과 반복성이 높음
  - 주위 26개 이웃에 대해 최저 또는 최대인 점
    - 임계값보다 작은 점은 잡음 처리
  - 검출된 극점을 키포인트라 부름

LoG filter extrema locates "blobs"
- maxima = dark blobs on light background
- minima = light blobs on dark background

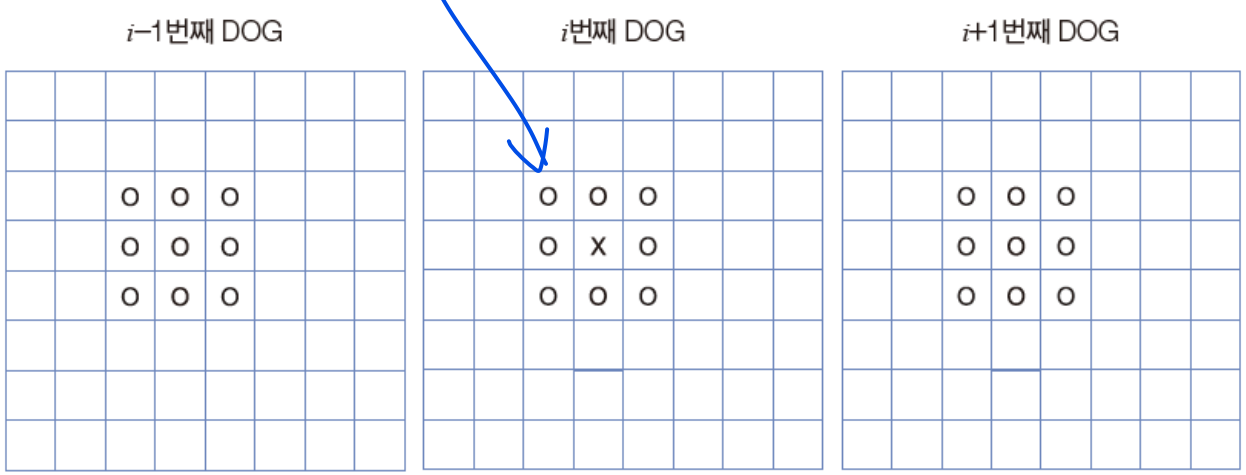


그림 5-10 3차원 구조의 DOG 영상에서 특징점(키포인트) 검출

5.4.1 SIFT 검출

위치와 스케일 계산

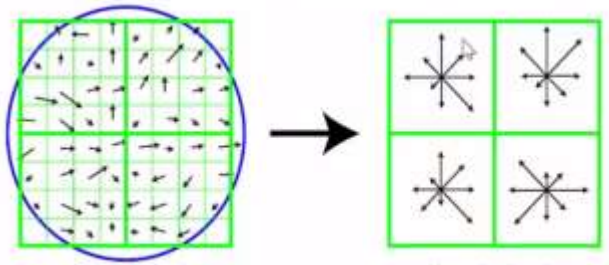
- 키포인트는 <*y*,*x*,*o*,*i*> 정보를 가짐 (옥타브 *o*의 *i* 번째 DOG 영상의 (*y*,*x*)에서 검출)
- 미세 조정 (부분 화소 정밀도)을 거쳐 <*y*′,*x*′,*o*,*i*′>로 변환됨
- 위치와 스케일 계산 식 적용하여 옥타브 0(시작)에 있는 영상의 위치 (y, x)로 변경

$$(y,x) = (y' \times 2^o, x' \times 2^o)$$
$$s = 1.6 \times 2^{\frac{o+i'}{3}} \qquad (4.21)$$

공개 소프트웨어

- David Lowe
- Rob Hess
- Andrea Vedaldi
- OpenCV

- Compute relative orientation and magnitude in a 16x16 neighborhood at key point
- Form weighted histogram (8 bin) for 4x4 regions
  - Weight by magnitude and spatial Gaussian
  - Concatenate 16 histograms in one long vector of 128 dimensions
- Example for 8x8 to 2x2 descriptors



5.4.1 SIFT 검출

original image

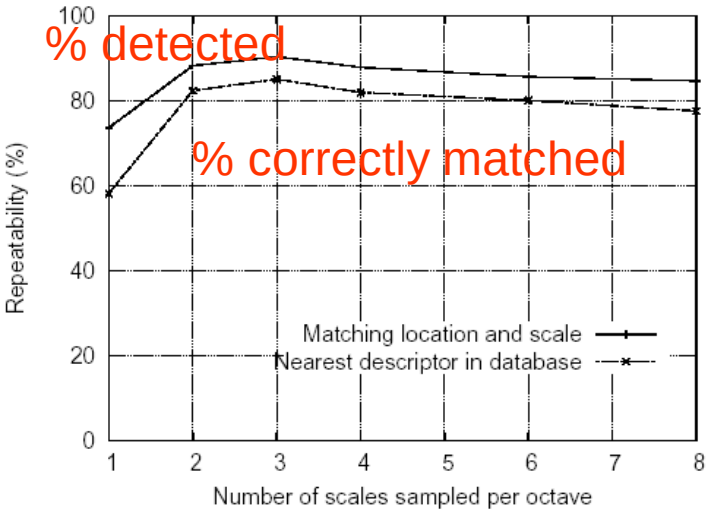
extrema locations

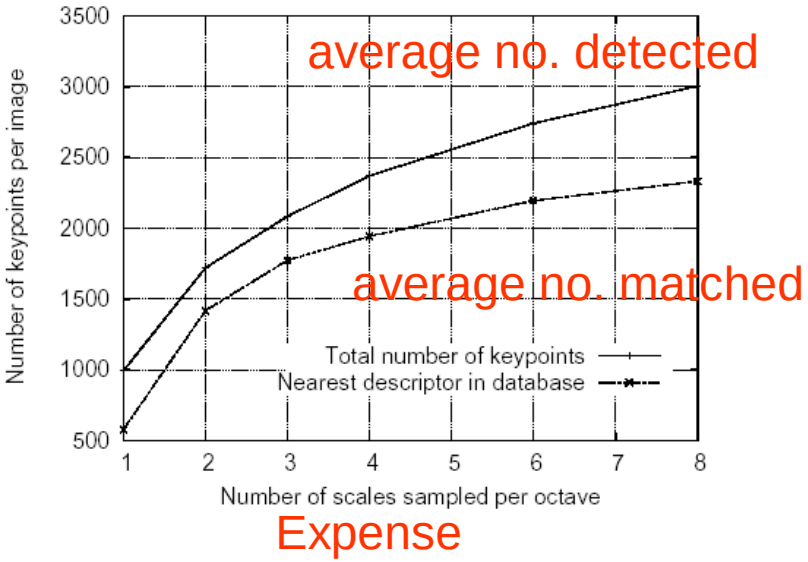
Initial weak-feature Outlier rejection
length is the scale of the position

Further edge-like Outlier rejection

from 729 key points to 536 key points.

Scale-space extrema detection: experimental results over 32 images that were synthetically transformed and noise





- Sampling in scale for efficiency
  - How many scales should be used per octave? S=?
    - More scales evaluated, more keypoints found
    - S < 3, stable keypoints increased too
    - S > 3, stable keypoints decreased
    - S = 3, maximum stable keypoints found

SIFT example

Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.

5.4.2 SIFT 기술자

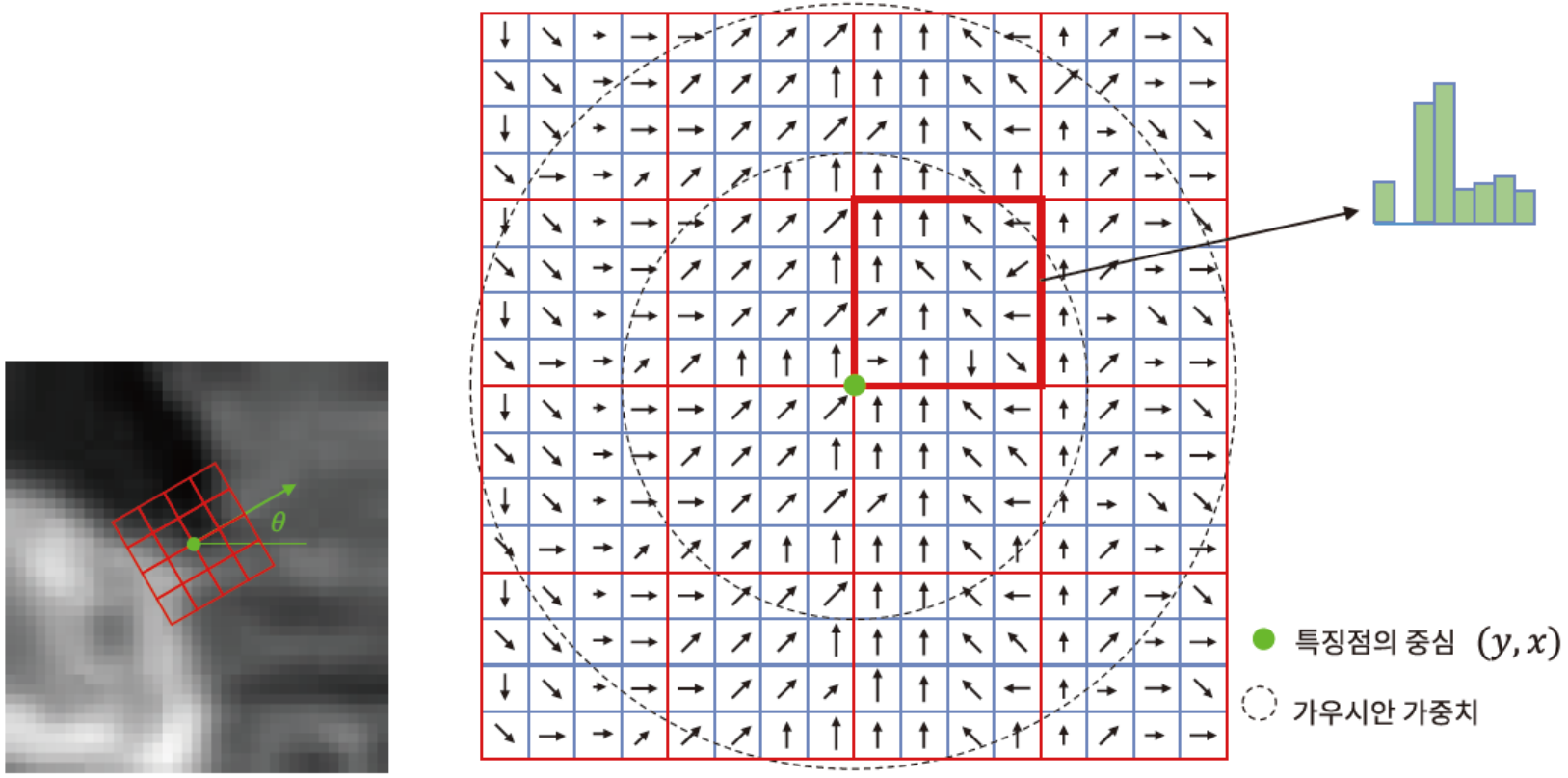
■ **특징점 주위를 살펴 풍부한 정보를 가진 기술자 추출**

- 불변성 달성이 매우 중요

■ **기술자 추출 알고리즘(그림은 다음 슬라이드)**

- o와 i로 가장 가까운 가우시안을 결정하고 거기서 기술자 추출하여 스케일 불변성 달성
- 기준 방향을 정하고 기준 방향을 중심으로 특징을 추출하여 회전 불변성 달성
- 기술자 $\mathbf{x}$를 단위 벡터로 바꿈으로써 조명 불변성 달성

5.4.2 SIFT 기술자

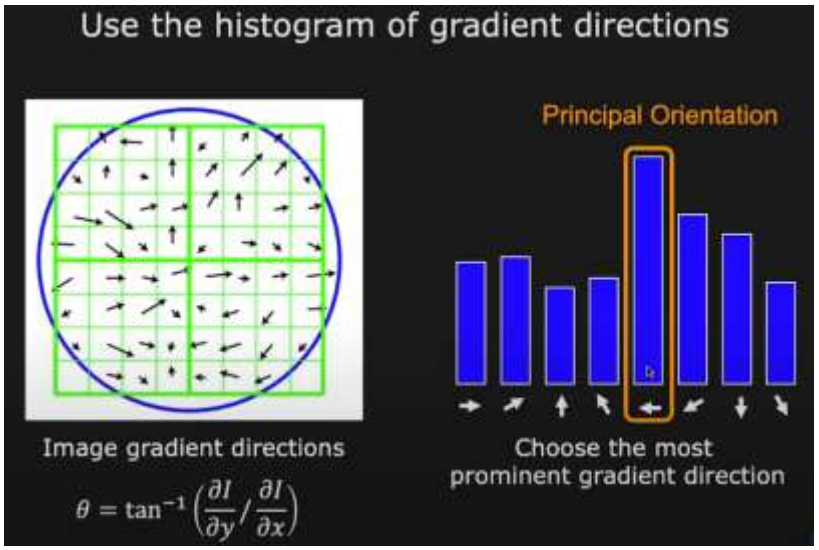


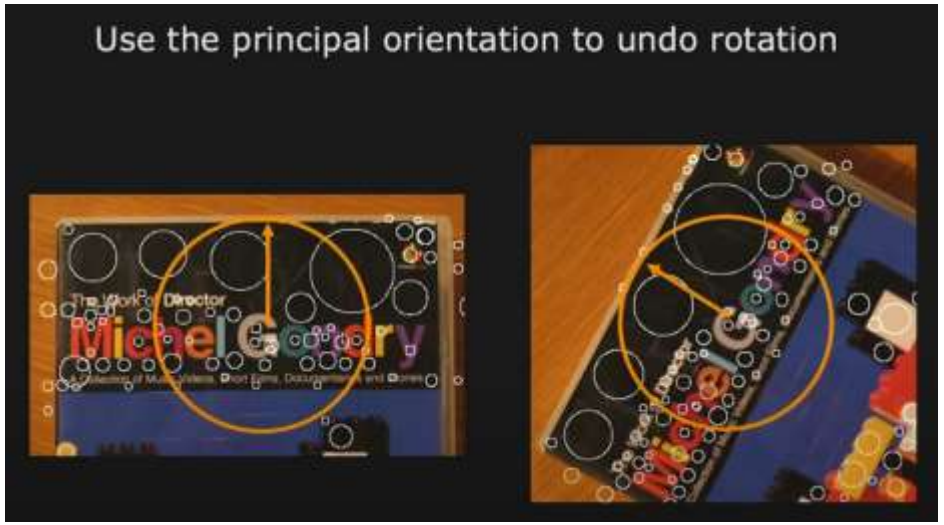
(a) 지배적인 방향의 윈도우 (b) 16×16 부분 영역 샘플링과 기술자 추출

그림 5-11 SIFT 특징점에서 기술자 추출

5.4.2 SIFT 기술자

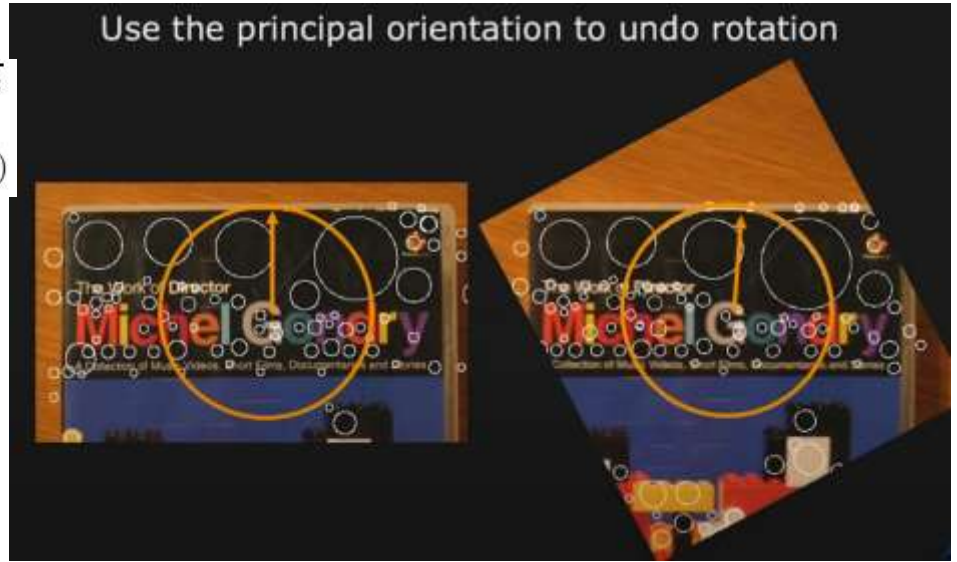
- 회전 불변(rotation invariance)
  - 지배적인 방향(dominant orientation): 키포인트의 주요 방향





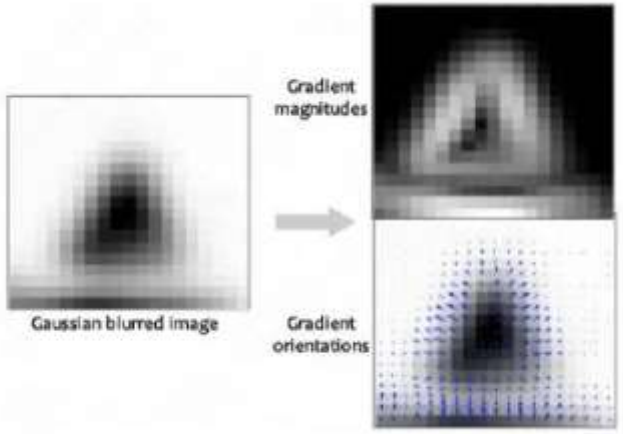
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

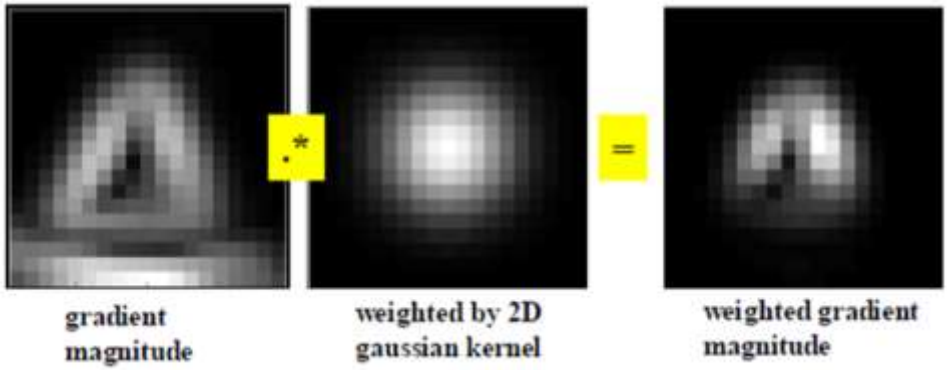


5.4.2 SIFT 기술자

- 회전 불변(rotation invariance)
  - 지배적인 방향(dominant orientation): 키포인트의 주요 방향
    - keypoint 주변으로 16×16 크기의 윈도우를 잡은 뒤 그 안의 이미지(픽셀들)를 Gaussian blurring한 다음 각 점에 대 해서 gradient의 방향과 크기를 구한다.

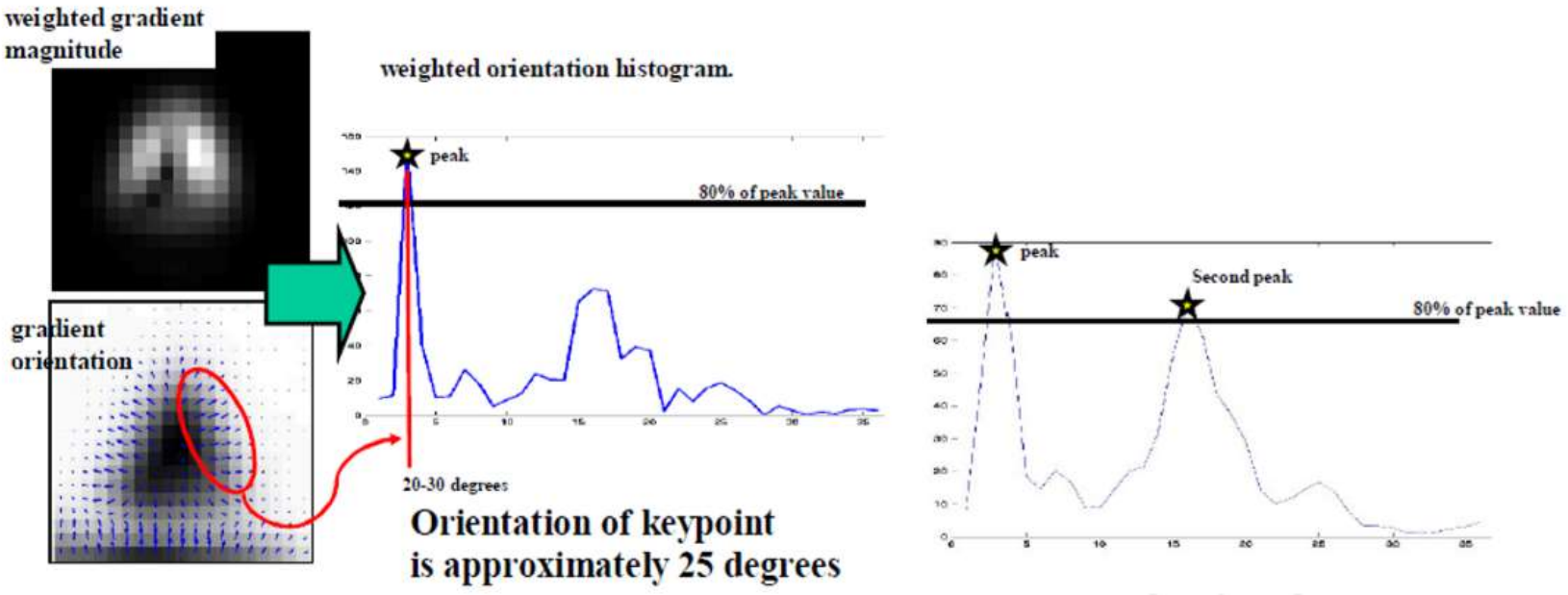


    - gradient magnitude에 16×16크기의 Gaussian weight를 준다.(키포인트와 멀어질 수록 가중값이 작아짐)



5.4.2 SIFT 기술자

- 지배적인 방향 결정: 16x16 윈도우 내에 있는 모든 픽셀에 대하여:
  - 0~360°의 값들을 10도 단위로 36개의 구간으로 분리한 후 해당 픽셀의 방향각이 포함되는 구간에 gradient magnitude를 더한다(누적 에지 강도).



  - 최고점의 80%인 peak가 또 나오게 된다면 이 key point는 2개의 orientation을 가지게 되고 implementing시 복수의 key point로 인식한다

5.4.2 SIFT 기술자

- SIFT 기술자 추출 알고리즘 [Lowe2004]
  - 적절한 크기의 윈도우를 지배적 방향(θ)으로 씌운다
  - 윈도우를 4*4의 16개의 블록으로 분할
    - 각 블록은 그레이디언트 방향 히스토그램 구함
    - 그레이디언트 방향은 8개로 양자화
  - 4*4*8=128차원 특징 벡터 **x**
  - 회전된 화소 값은 보간으로 구함

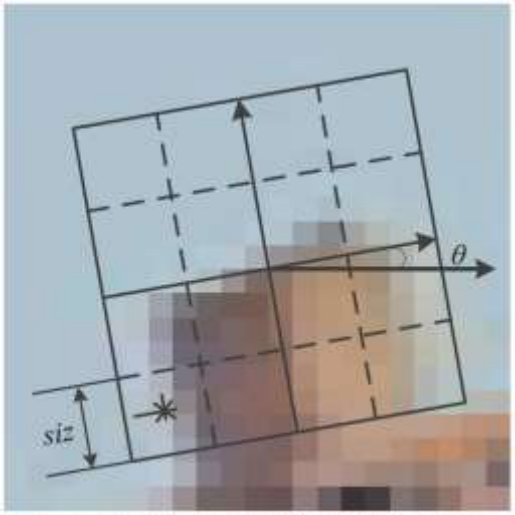


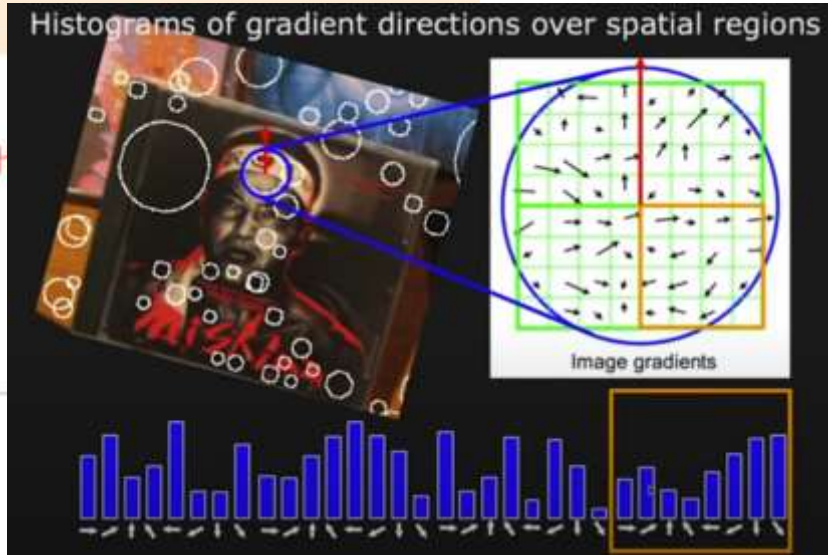
그림 6-4 SIFT 기술자 추출을 위한 좌표계와 4×4 블록

알고리즘 6-1 SIFT 기술자 추출

입력 : 입력 영상 f에서 검출된 키포인트 집합 $p_i=(y_i, x_i, \sigma_i)$, $1 \le i \le n$ // 4.4.3절의 알고리즘으로 추출

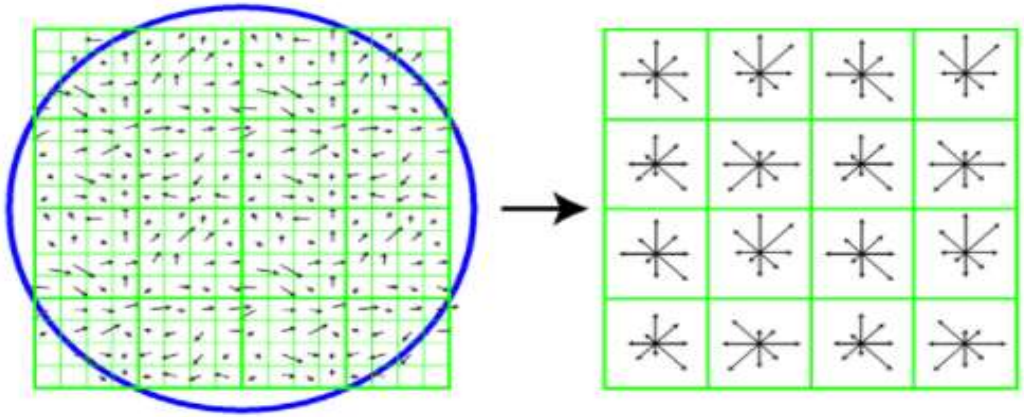
출력 : 기술자가 추가된 키포인트 집합 $p_i=(y_i, x_i, \sigma_i, \theta_i, \mathbf{x}_i)$, $1 \le i \le m$

```
1  for(i=1 to n) {
2    p_i의 지배적인 방향 θ_i를 계산한다.  // 이때 하나의 키포인트가 여러 개로 나
3    p_i의 특징 벡터 x_i를 계산한다.
4    x_i를 정규화한다.
5  }
```



5.4.2 SIFT 기술자

- Descriptor window: 16x16영상에서 4x4 블록으로 구분



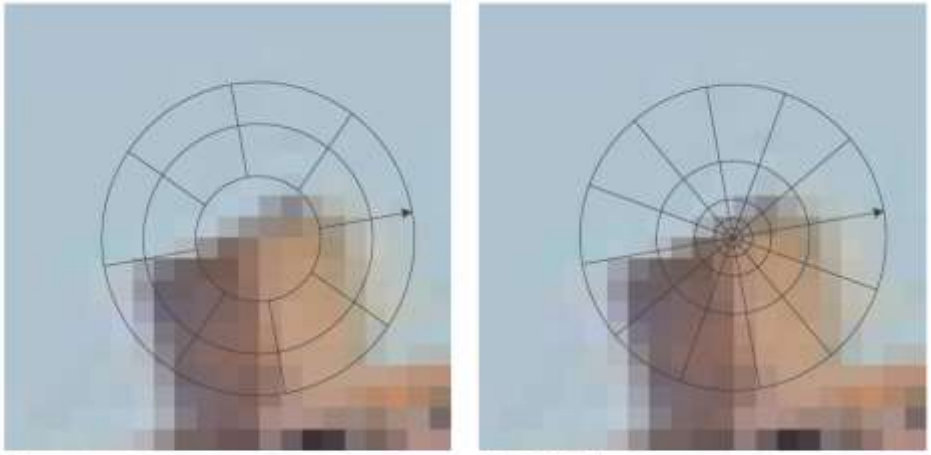
8방향

Fig. 2. A SIFT descriptor computed from the region around the interpret point (the circle): gradient of the image (left), descriptor of the interest point (right)

- 히스토그램을 그릴 때 각 dominant orientation 방향으로 윈도우를 씌움으로 descriptor가 회전 불변이 되게 만듬
- 가우시안 가중치 함수를 적용하여 keypoint로부터 너무 멀리 떨어진 값들에 대해 가중치를 적용
- 히스토그램을 그린 후 밝기에 불변한 값을 가지기 위해 정규화(Normalization)을 해주게 된다. 또한 비선형적(Non-linear)인 밝기 변화 에 대해서는 0.2이상으로 나오면 제대로 검출이 안 되기 때문에 다시 한 번 정규화한다.

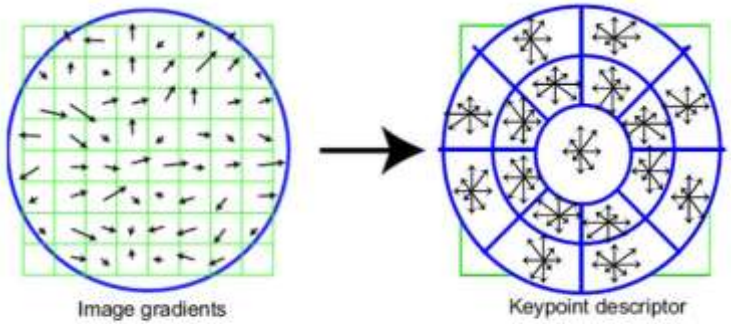
5.4.2 SIFT의 변형

- PCA-SIFT [Ke2004]
  - 키포인트에 39*39 윈도우 씌우고 도함수 d_y와 d_x 계산
  - 39*39*2차원의 벡터를 PCA를 이용하여 20차원으로 축소
- GLOH [Mikolajczyk2005a]: 16단계 방향 히스토그램 사용
  - 원형 윈도우로 17(영역)*16차원의 벡터를 추출하고, PCA를 이용하여 128차원으로 축소
- 모양 콘텍스트 [Mikolajczyk2005a]
  - 원형 윈도우로 60개 영역 안에 에지(캐니 에지) 개수를 사용한 60차원 특징 벡터 추출



(a) GLOH (b) 모양 콘텍스트

그림 6-5 GLOH와 모양 콘텍스트가 사용하는 원형 윈도우



17 location bins, 16 gradient bins per location bin

272 elements -> down to 128 with PCA

5.4.2 이진 기술자

- 빠른 매칭을 위해 특징 벡터를 이진열로 표현
  - 비교 쌍의 대소 관계에 따라 0 또는 1
  - 비교 쌍을 구성하는 방식에 따라 여러 변형

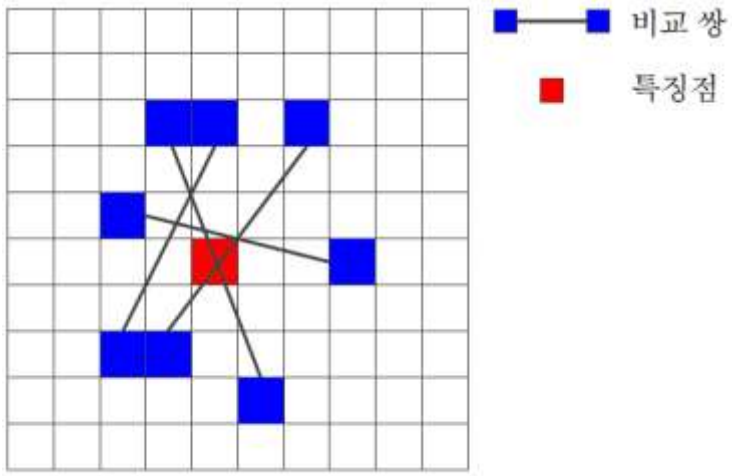


그림 6-6 이진 기술자의 조사 패턴

  - 매칭은 해밍 거리를 이용하여 빠르게 수행
  - $d_h(s,t) = |\{i \in \{0,1,2,\cdots n-1\}: s_i \neq t_i\}| \in \{1,2,\cdots,n\}$
  - '1011101' 과 '1001001' 사이의 해밍 거리는 2이다. (1001001)

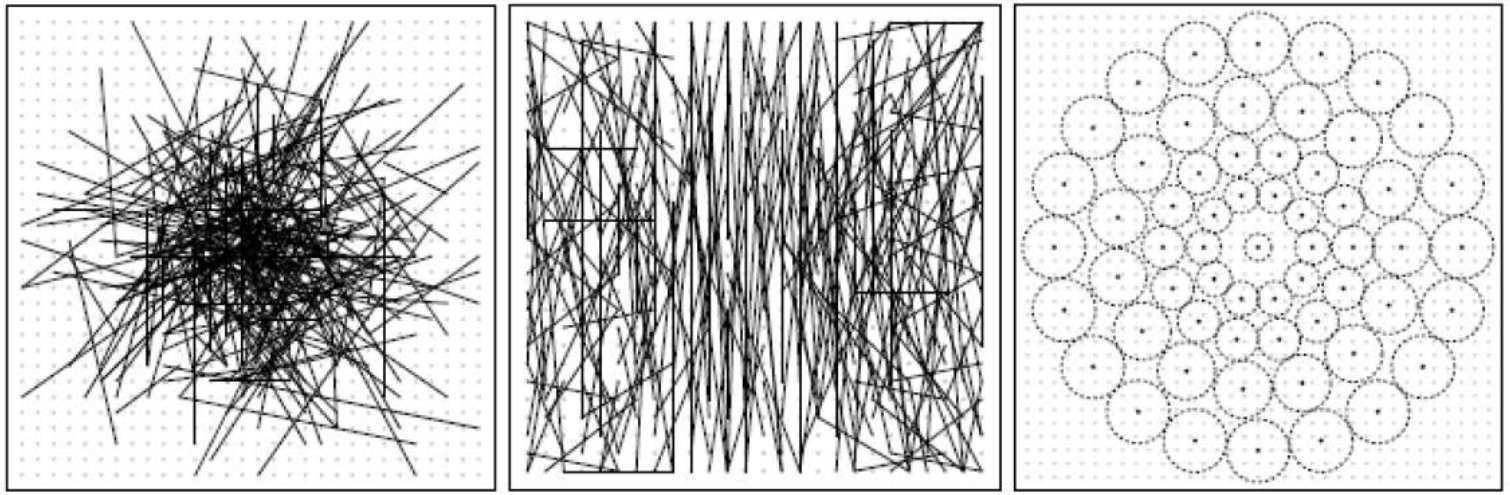
5.4.2 이진 기술자

```
// ORB 특징 검출기 객체 생성
cv::Ptr<cv::ORB> ptrORB =
  cv::ORB::create(75,   // 특징점의 총 개수
                  1.2,  // 계층 간의 크기 조정 인자
                  8);   // 피라미드의 계층 개수
```

- BRIEF, ORB, BRISK

표 6-1 이진 기술자의 특성 비교

	스케일 불변	회전 불변	특징 벡터 비트 수
BRIEF	X	X	256비트
ORB	X	O	512비트
BRISK	O	O	512비트



(a) BRIEF (b) ORB (c) BRISK

그림 6-7 세 가지 이진 기술자가 사용하는 조사 패턴

5.4.2 SIFT 기술자

프로그램 5-2 SIFT 검출

```
01  import cv2 as cv
02
03  img=cv.imread('mot_color70.jpg')              # 영상 읽기
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  sift=cv.SIFT_create()
07  kp,des=sift.detectAndCompute(gray,None)
08
09  gray=cv.drawKeypoints(gray,kp,None,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_
    KEYPOINTS)
10  cv.imshow('sift', gray)
11
12  k=cv.waitKey()
13  cv.destroyAllWindows()
```

5.4.2 SIFT 기술자

5.4.2 SIFT 검출

- static Ptr<SIFT> SFIT::create(int **nfeatures**=0, int **nOctaveLayers**=3, double **contrastThreshold**=0.04, double **edgeThreshold**=10, double **sigma**=1.6)¶
  - **nfeatures** – The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast). 기본값으로 0인 경우 찾을 수 있는 모든 feature 반환
  - **nOctaveLayers** – The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image resolution.(5개 DOG layer=> 3개의 키포인트를 계산할 수 있는 layer 수): 실제 필요한 옥타브의 레이어 수는 이 값에 3을 더한 값이다. 6 blurred layers->5 DOG->3 layers
  - **contrastThreshold** – The contrast threshold used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector.
  - **edgeThreshold** – The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e. the larger the edgeThreshold, the less features are filtered out (more features are retained).
  - **sigma** – The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want to reduce the number.

Python:

```
cv.SIFT.create(  [, nfeatures[, nOctaveLayers[, contrastThreshold[, edgeThreshold[, sigma]]]]]          ) -> retval
cv.SIFT.create(  nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma, descriptorType ) -> retval
```

◆ detectAndCompute()

```
virtual void cv::Feature2D::detectAndCompute ( InputArray                image,
                                               InputArray                mask,
                                               std::vector< KeyPoint > & keypoints,
                                               OutputArray               descriptors,
                                               bool                      useProvidedKeypoints = false
                                             )
```

Python:

```
cv.Feature2D.detectAndCompute( image, mask[, descriptors[, useProvidedKeypoints]] ) -> keypoints, descriptors
```

- 특징 검출기(feature detector)에 의해 검출된 특징점은 KeyPoint 클래스 객체의 벡터로 반환 (features2d.hpp)
  - KeyPoint 클래스 멤버변수
    - pt : keypoint의 (x,y) 좌표, 키포인트 위치를 알려 줌
    - size : keypoint 크기의 반지름
    - angle : keypoint 방향각도[0, 360], -1이면 의미 없음
    - response : keypoint 반응세기
    - octave : keypoint 가 검출된 피라미드 옥타브(layer)
    - class_id : keypoint를 객체별로 클러스터링하는 데 사용된 객체 ID

- void drawKeypoints(const Mat& **image**, const vector<KeyPoint>& **keypoints**, Mat& **outImage**, const Scalar& **color**=Scalar::all(-1), int **flags**=DrawMatchesFlags::DEFAULT)
  - 특징점을 그리는 제네릭 함수
    - **image** – Source image.
    - **keypoints** – Keypoints from the source image.
    - **outImage** – Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below.
    - **color** – Color of keypoints. Scalar::all(-1)이면 각 특징점을 임의의 색상으로 그림
    - **flags** – Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags. See details above in drawMatches
      - struct DrawMatchesFlags { enum { DEFAULT = 0, Output image matrix will be created (Mat::create), i.e. existing memory of output image may be reused. // Two source images, matches, and single keypoints will be drawn. For each keypoint, only the center point will be drawn (without a circle around the keypoint with the keypoint size and orientation). DRAW_OVER_OUTIMG = 1, Output image matrix will not be created (using Mat::create). Matches will be drawn on existing content of output image. NOT_DRAW_SINGLE_POINTS = 2, Single keypoints will not be drawn. DRAW_RICH_KEYPOINTS = 4 For each keypoint, the circle around keypoint with keypoint size and orientation will be drawn. }; };

Python:

cv.drawKeypoints(image, keypoints, outImage[, color[, flags]]) -> outImage

Note

For Python API, flags are modified as cv.DRAW_MATCHES_FLAGS_DEFAULT, cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, cv.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG, cv.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS