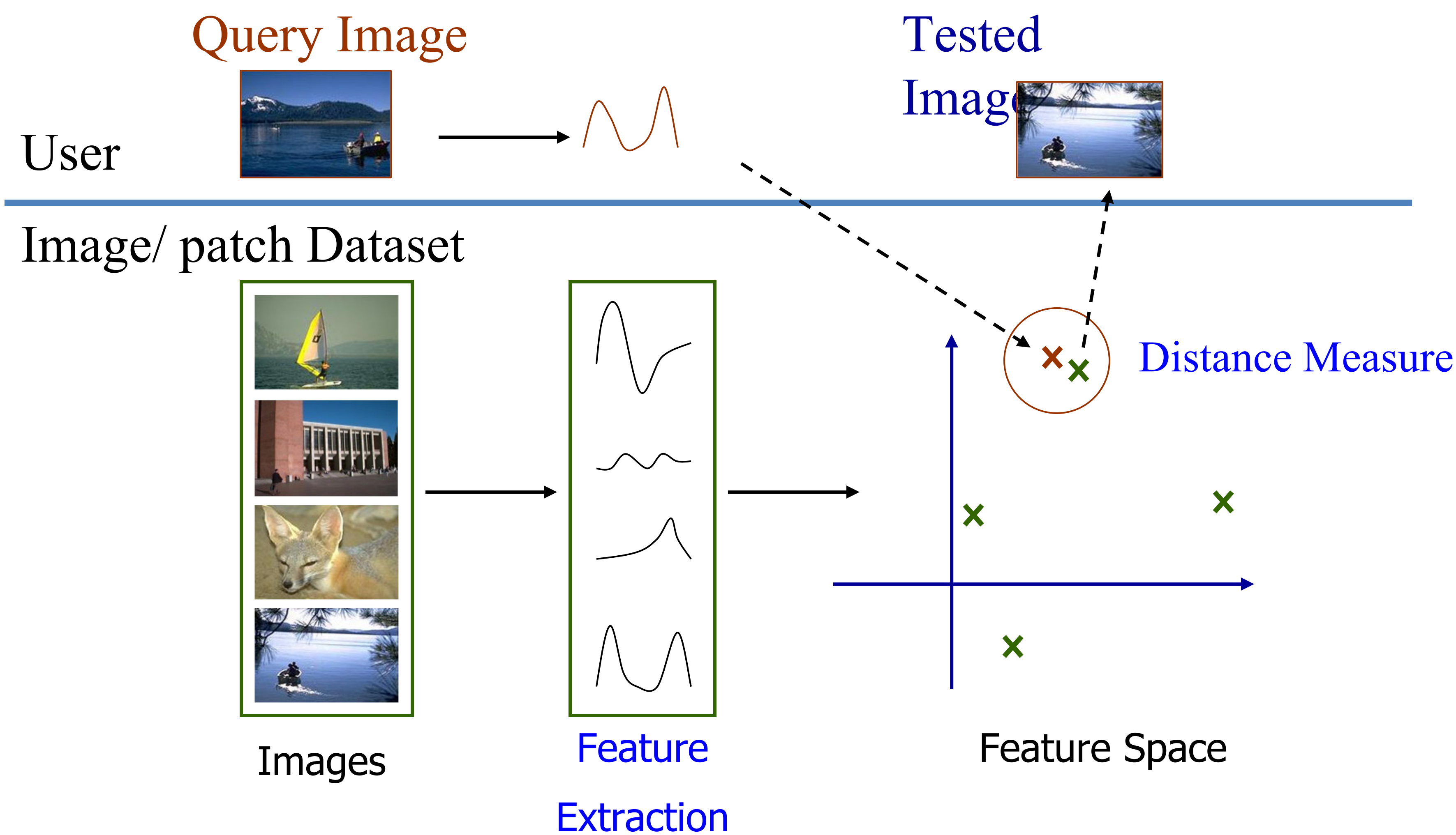
Course Project Tutorial 3

CS4185 Multimedia Technologies and Applications

Image Features / Distance Measures

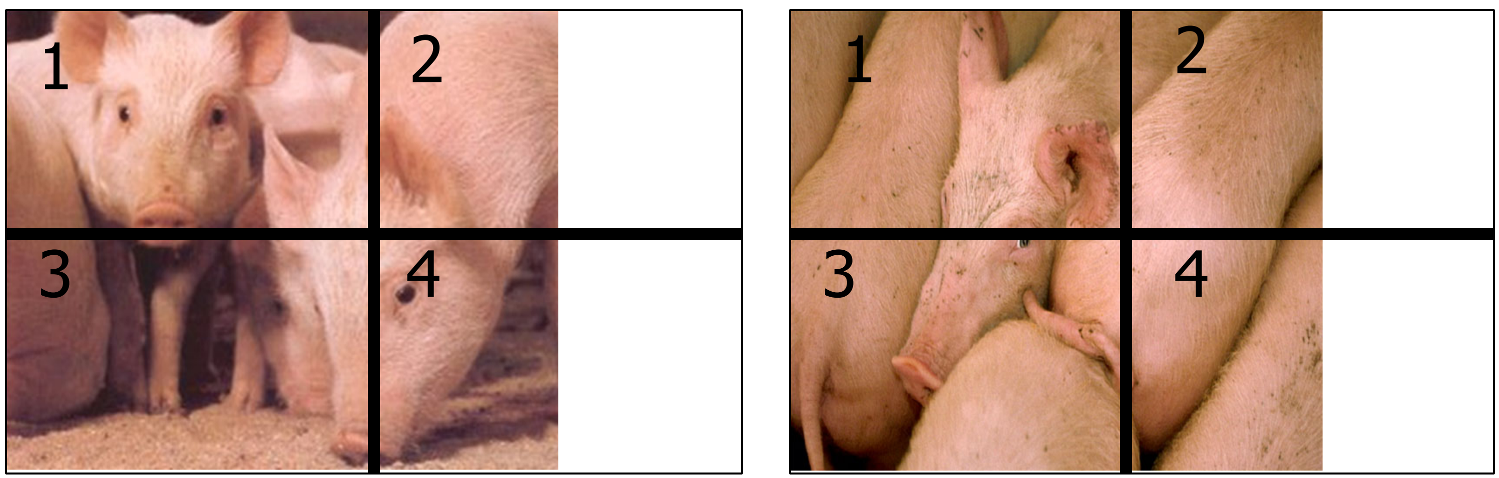


How to improve the performance?

- Possible solutions:
  - Utilize color information.
  - Utilize edge and shape information.
  - Using different layout.
  - Features fusion.

Color Layout

Color Layout (or gridded color) distance is the sum of the color distances in each of the corresponding grid squares.



Color Layout

- Need for Color Layout
  - Global color features give too many false positives.
- How it works:
  - Divide the whole image into sub-blocks.
  - Extract features from each sub-block.
- Can we go one step further?
  - Divide the image into regions based on color feature concentration.
  - This process is called segmentation.

More Info: http://en.wikipedia.org/wiki/Color_layout_descriptor

Edge and shape:

- An edge is where change occurs. So most edge operators are based on gradient.
- Sobel:

$$s_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad s_y = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Canny (size = 5)

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

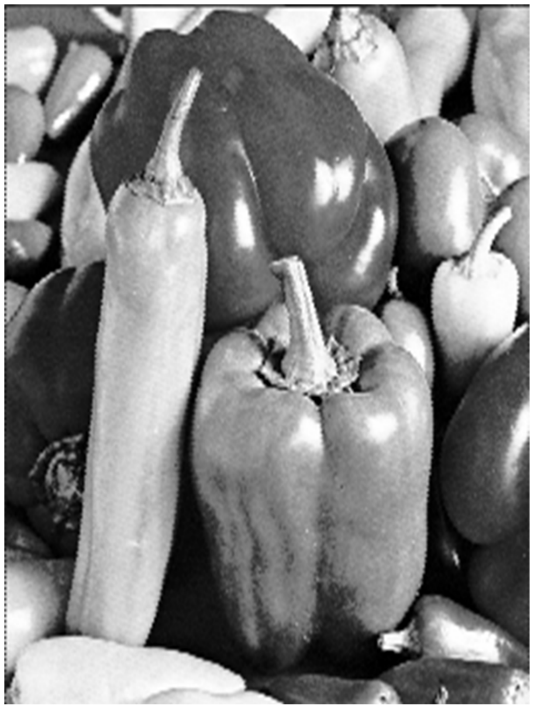
Edge and shape:

- A soccer is always a circle in image space. So we can claim that a circular object is more likely to be a soccer than, say a rectangular object.
- Circle Hough transform:
  - https://en.wikipedia.org/wiki/Circle_Hough_Transform
  - http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

Introduction to segmentation

- The main purpose is to find meaningful regions with respect to a particular application.
  - To detect homogeneous regions
  - To detect edges (boundaries, contours)
- Segmentation of non-trivial images is one of the difficult tasks in image processing. Still under research.
- Applications of image segmentation include:
  - Objects in a scene (for object-based retrieval)
  - Objects in a moving scene *(MPEG4)*
  - Spatial layout of objects (Path planning for a mobile robots)

Introduction to segmentation

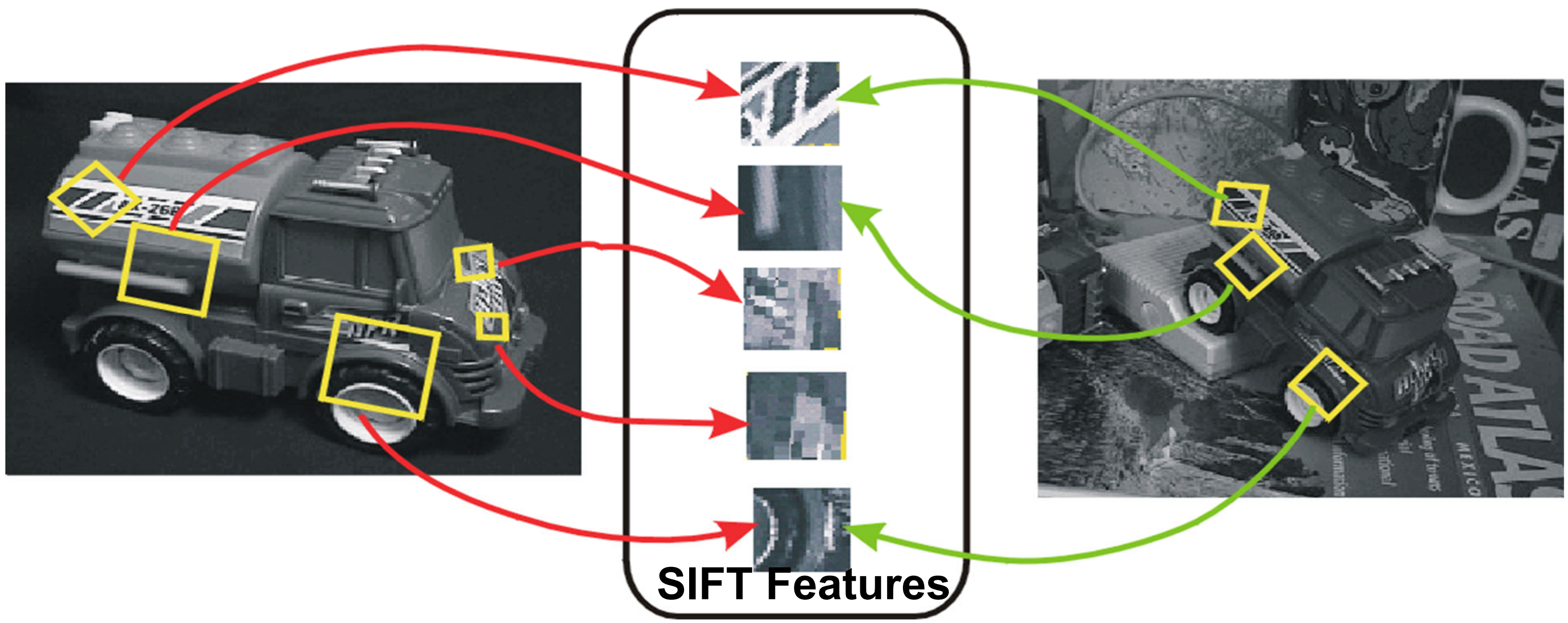


Local descriptors

- Features for local regions in the image
  - Regions obtained by segmentation
  - Regions of interest (ROI) – around interest points (keypoints)
- Interest points: corners, edges and others
- Keypoints: points in images, which are invariant to image translation, scale and rotation, and are minimally affected by noise and small distortions
- Scale-invariant feature transform (SIFT) by David Lowe

Idea of SIFT

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



Claimed Advantages of SIFT

- **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

SIFT Program

- Detect keypoints using the SIFT detector

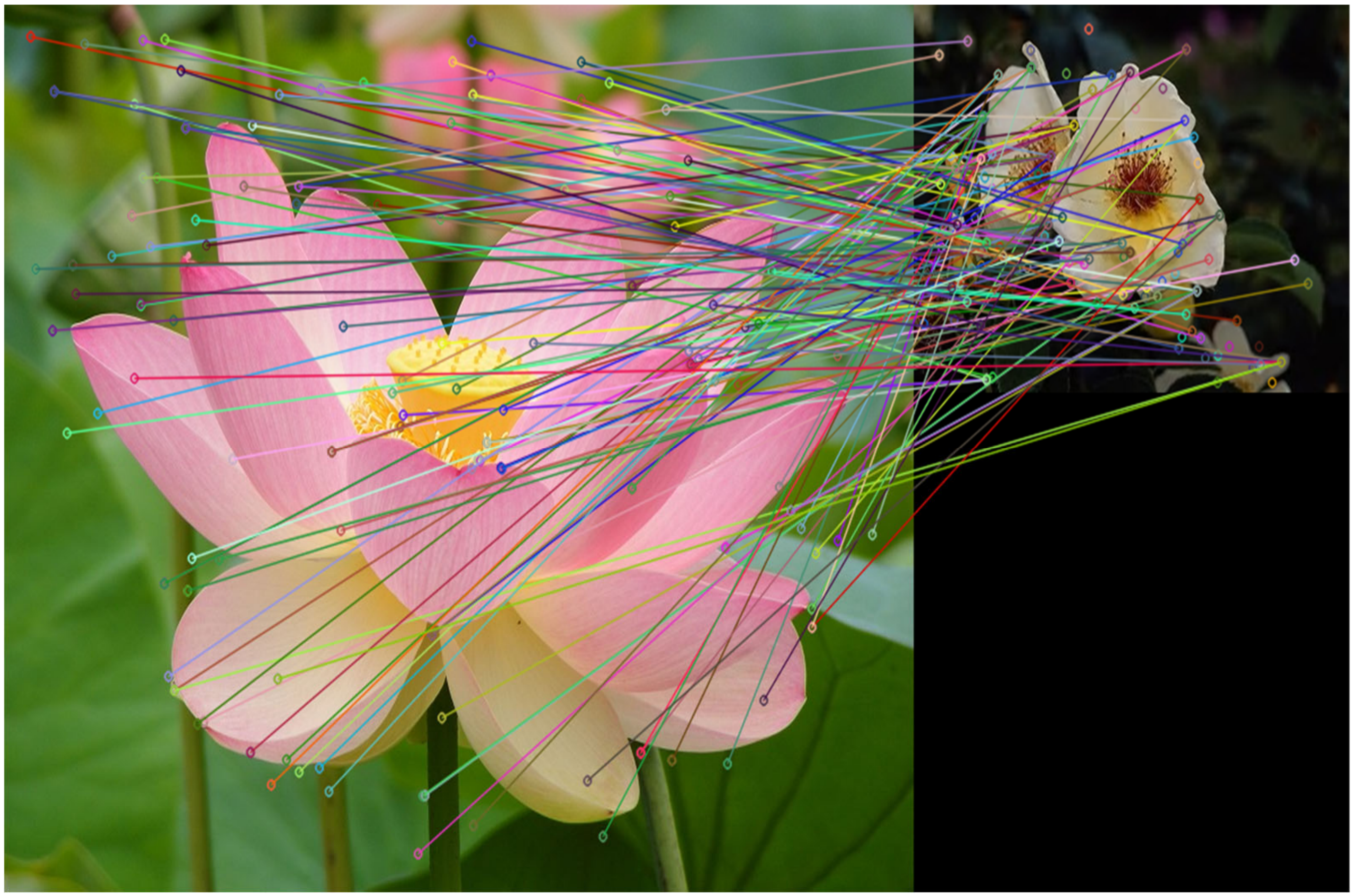
```
def SIFT():
    img1 = cv.imread("flower.jpg")
    img2 = cv.imread("image.orig/685.jpg")
    if img1 is None or img2 is None:
        print('Error loading images!')
        exit(0)
    #-- Step 1: Detect the keypoints using SIFT Detector, compute the descriptors
    minHessian = 400
    detector = cv.SIFT_create()
    keypoints1, descriptors1 = detector.detectAndCompute(img1, None)
    keypoints2, descriptors2 = detector.detectAndCompute(img2, None)
```

SIFT Program

- Match descriptor vectors with a brute force matcher

```
#-- Step 2: Matching descriptor vectors with a brute force matcher
matcher = cv.DescriptorMatcher_create(cv.DescriptorMatcher_BRUTEFORCE)
matches = matcher.match(descriptors1, descriptors2)
#-- Draw matches
img_matches = np.empty((max(img1.shape[0], img2.shape[0]), img1.shape[1]+img2.shape[1], 3), dtype=np.uint8)
cv.drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches)
#-- Show detected matches
cv.imshow('Matches: SIFT (Python)', img_matches)
cv.waitKey()
```

SIFT Program



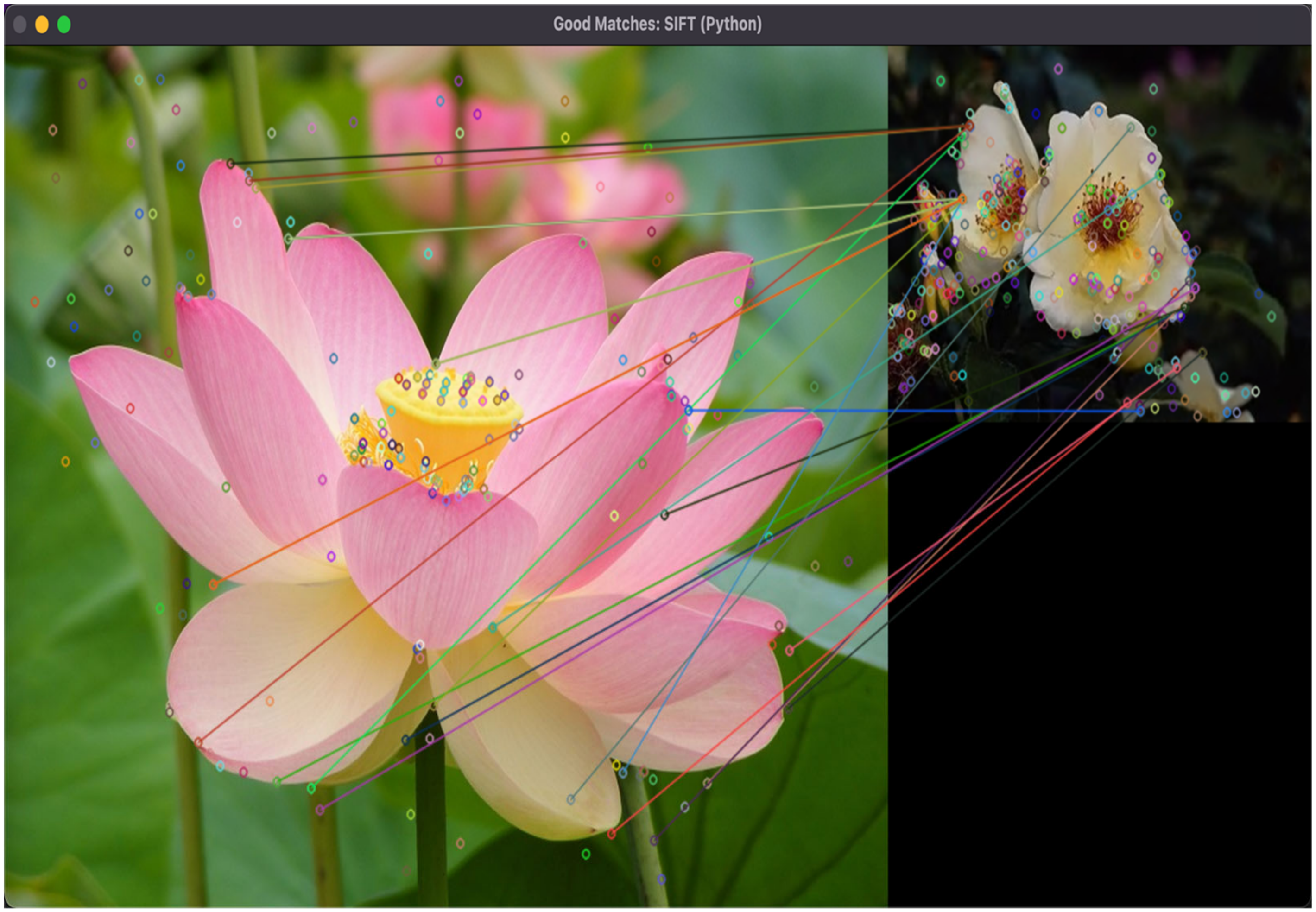
SIFT Program

- Only show “good” matches

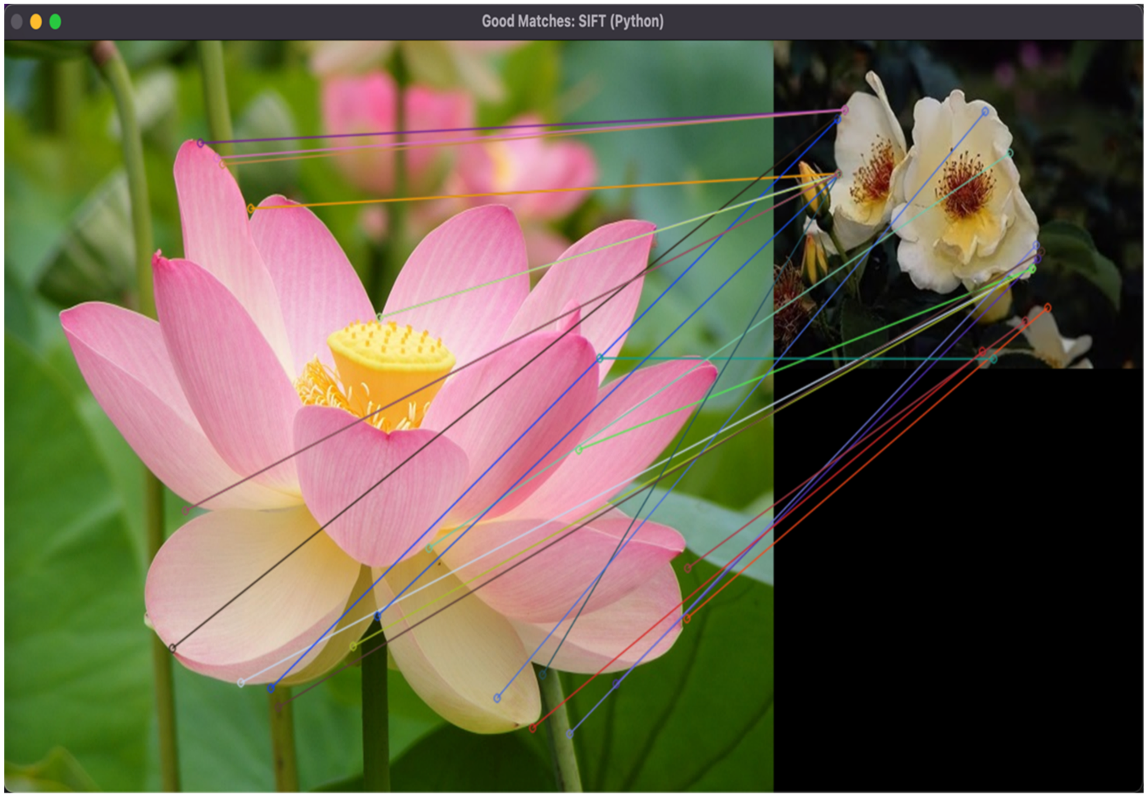
```
# draw good matches
matches = sorted(matches, key = lambda x:x.distance)
min_dist = matches[0].distance
good_matches = tuple(filter(lambda x:x.distance <= 2 * min_dist, matches))

img_matches = np.empty((max(img1.shape[0], img2.shape[0]), img1.shape[1]+img2.shape[1], 3), dtype=np.uint8)
cv.drawMatches(img1, keypoints1, img2, keypoints2, good_matches, img_matches)
#-- Show detected matches
cv.imshow('Good Matches: SIFT (Python)', img_matches)
cv.waitKey()
```

SIFT Program



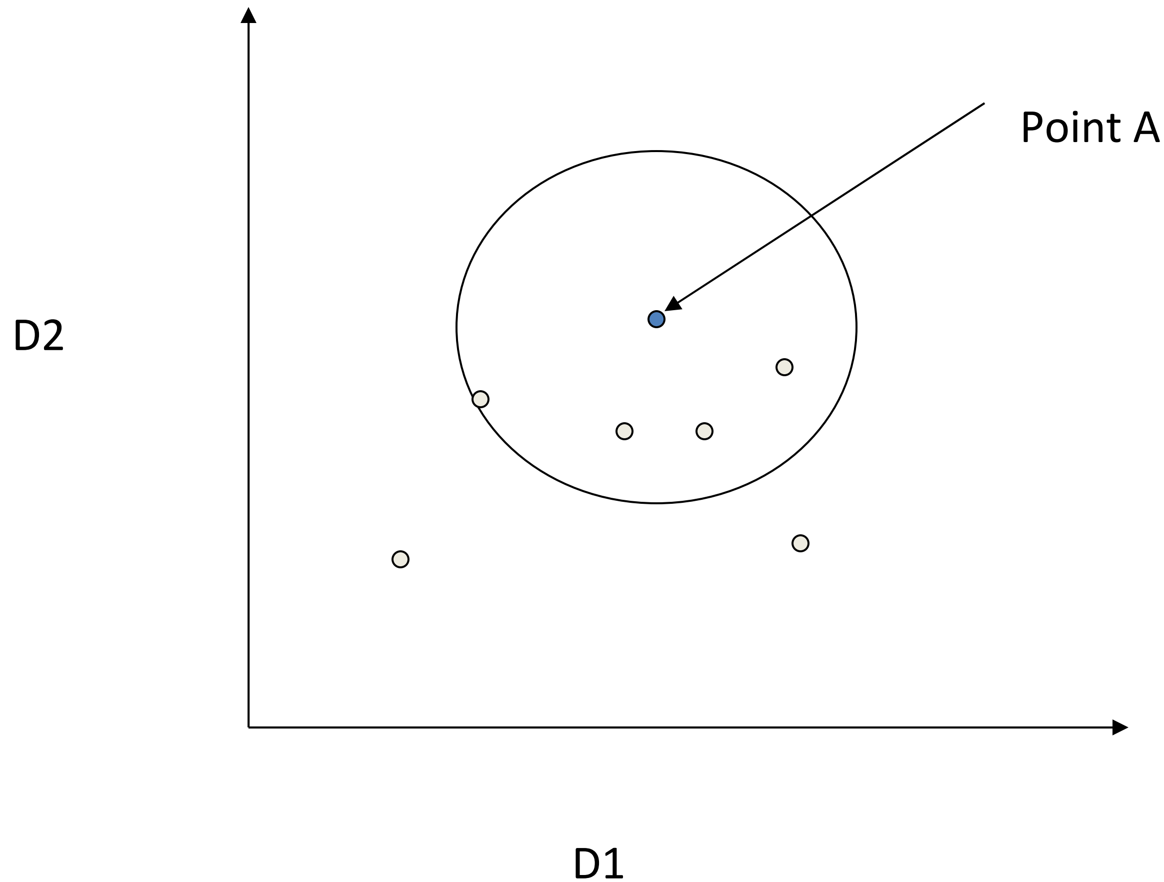
- Not showing single keypoints
  - cv.drawMatches(img1, keypoints1, img2, keypoints2, good_matches, img_matches, flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)



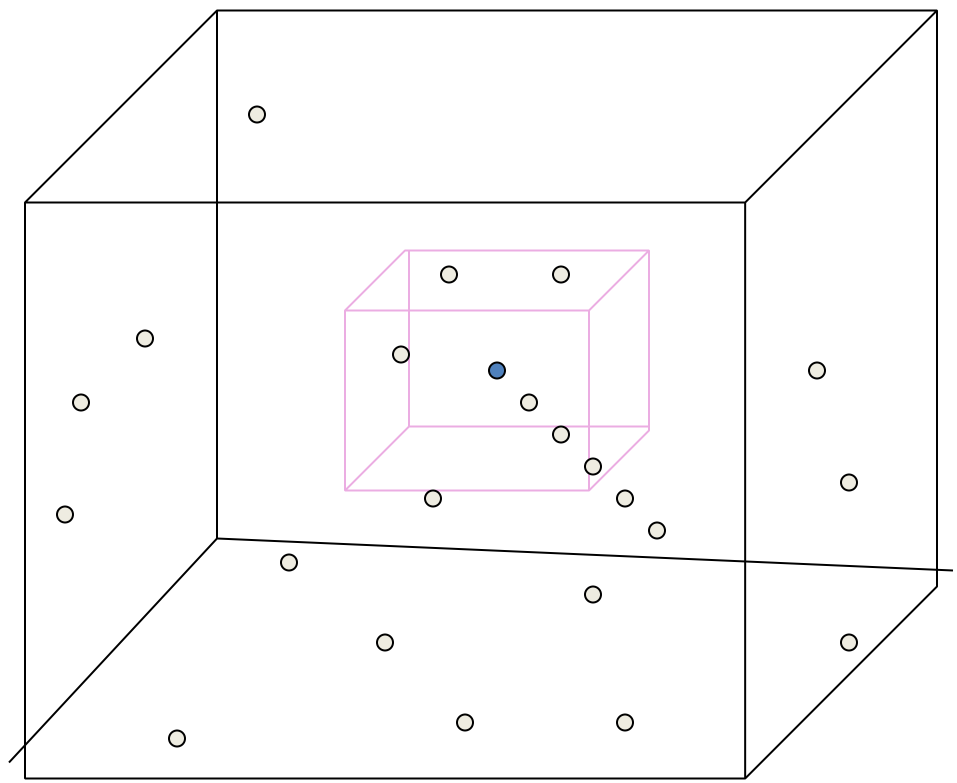
Problem of high dimensions

- Mean Color = RGB = 3 dimensional vector
- Color Histogram = 256 dimensions
- Effective storage and speedy retrieval needed
- Traditional data-structures not sufficient
- R-trees, SR-Trees etc…

2-dimensional space

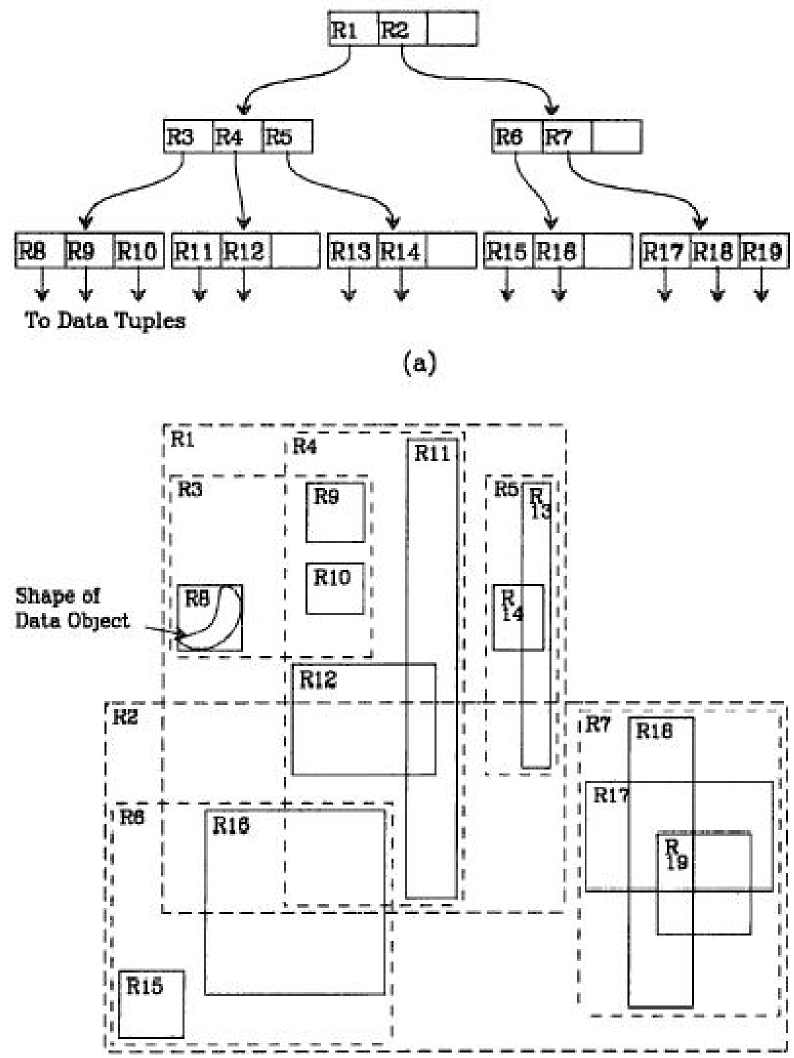


3-dimensional space



Now, imagine…

- An N-dimensional box!!
- We want to conduct a nearest neighbor query.
- R-trees are designed for speedy retrieval of results for such purposes.
- Designed by Guttmann in 1984.



Feature fusion:

- We need multiple characters to identify an object. For example, first it is a ball, but not head, not basketball. So only using shape or edge feature is not enough. We need additional information, such as color, texture to further check the object.
- Feature fusion is significant to performance!

Last but not least…

Feature fusion is significant to performance!