Final Project
Panorama Stitching

Zhe ZHANG, 1754060
Kaixin CHEN, 1753188
Yunxin SUN, 1551534
School of Software Engineering
Tongji University
Spring 2020

Content

- SIFT
  - Scale Space and Image Pyramids
  - Localizing Extrema
  - Generating Orientations and Descriptors
- SURF
  - Hessian Detector
  - Math Behind Hessian Matrix
  - Construction of feature vector
  - Comparision between SIFT and SURF
  - Results
- Fast Panorama Stitching on Mobile
  - Current Approaches
  - Summary
  - Color and Luminance Compensation
  - Optimal Seam Finding and Image Labeling
  - Image Blending
  - Results

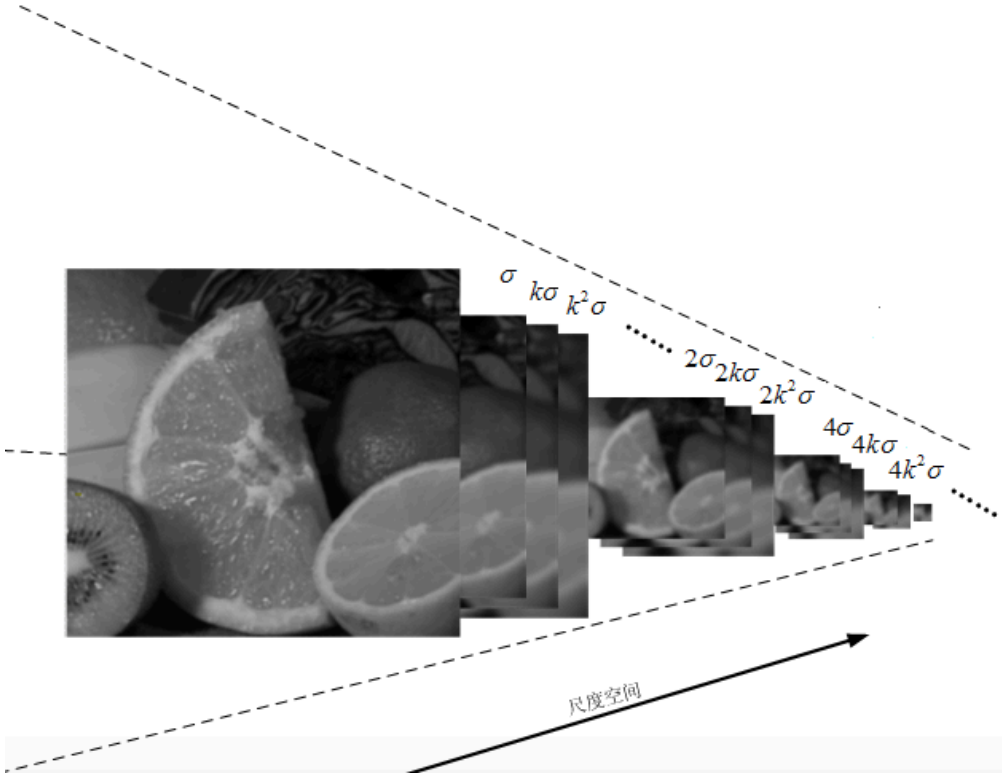
Content

- SIFT
  - Scale Space and Image Pyramids
  - Localizing Extrema
  - Generating Orientations and Descriptors
- SURF
  - Hessian Detector
  - Math Behind Hessian Matrix
  - Construction of feature vector
  - Comparision between SIFT and SURF
  - Results
- Fast Panorama Stitching on Mobile
  - Current Approaches
  - Summary
  - Color and Luminance Compensation
  - Optimal Seam Finding and Image Labeling
  - Image Blending
  - Results

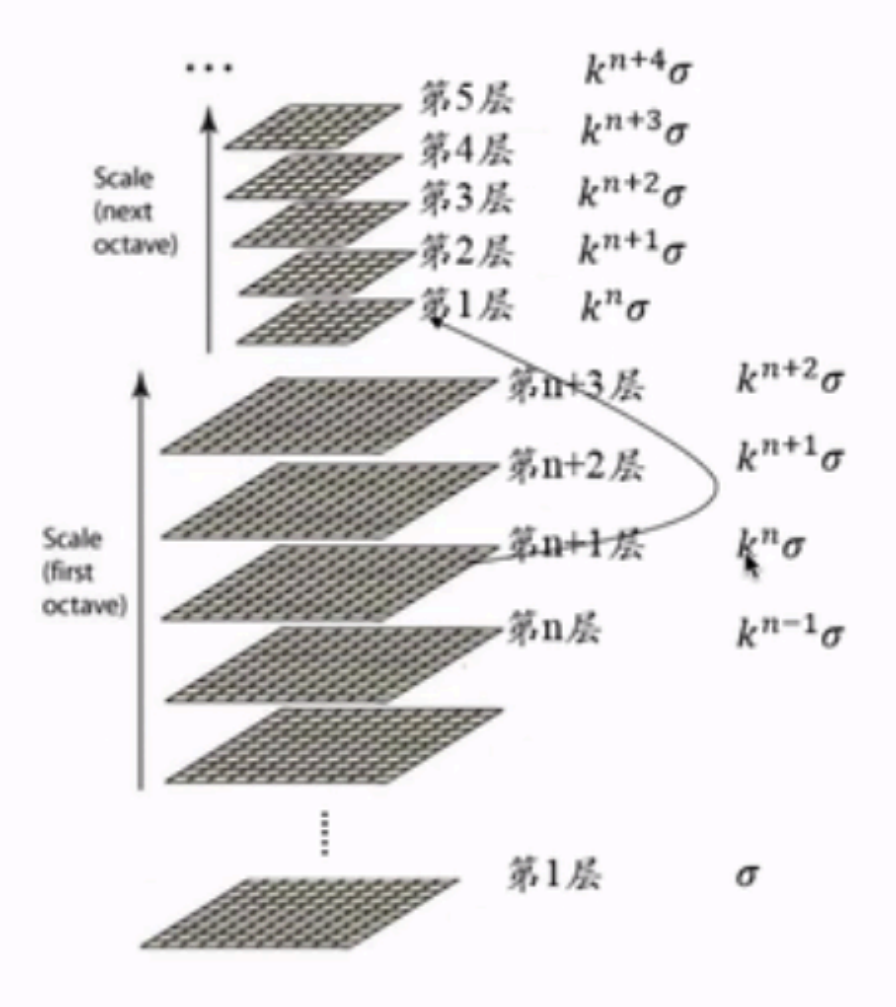
SIFT —— Scale Space & Image Pyramids

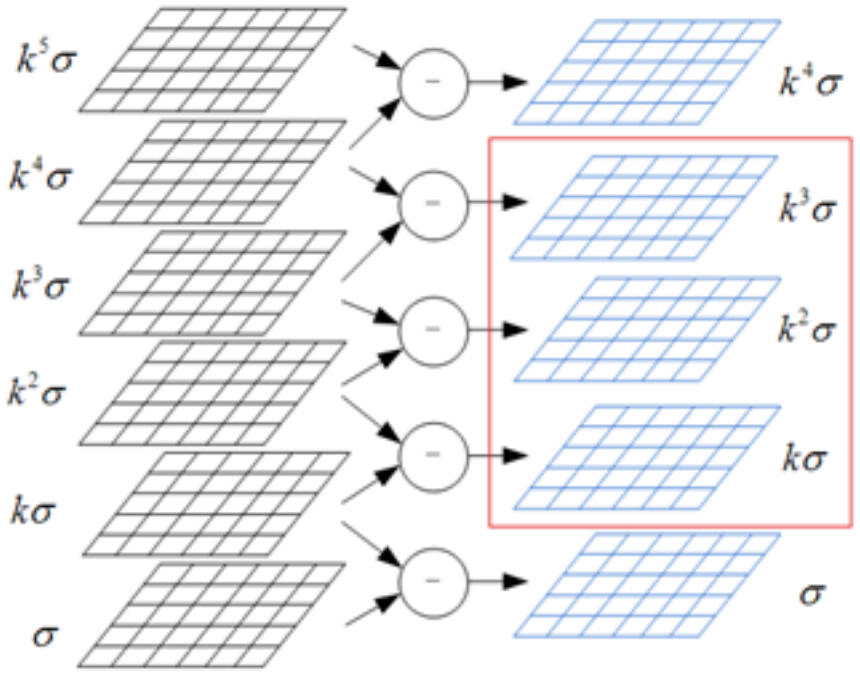
The Gaussian pyramid imitates different scales of the image --"the depth of the image"

The problem solved by the scale space of the image is how to describe the image at all scales.

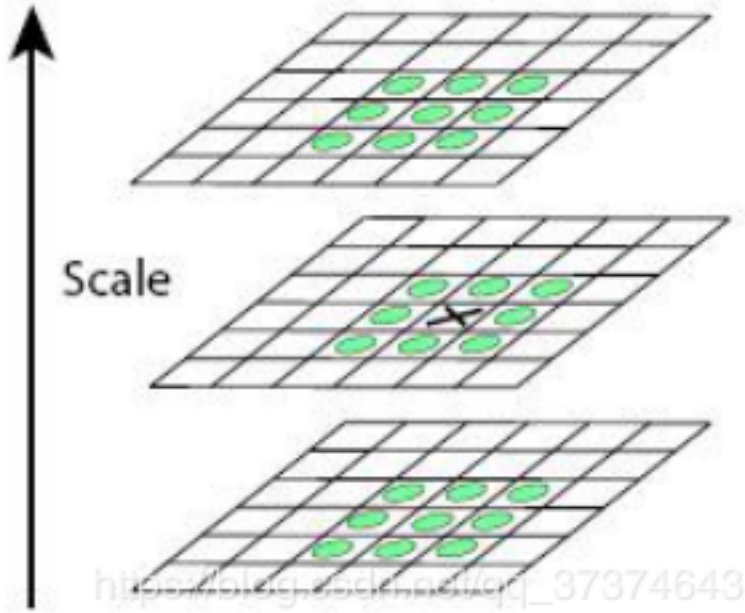


SIFT —— Scale Space & Image Pyramids



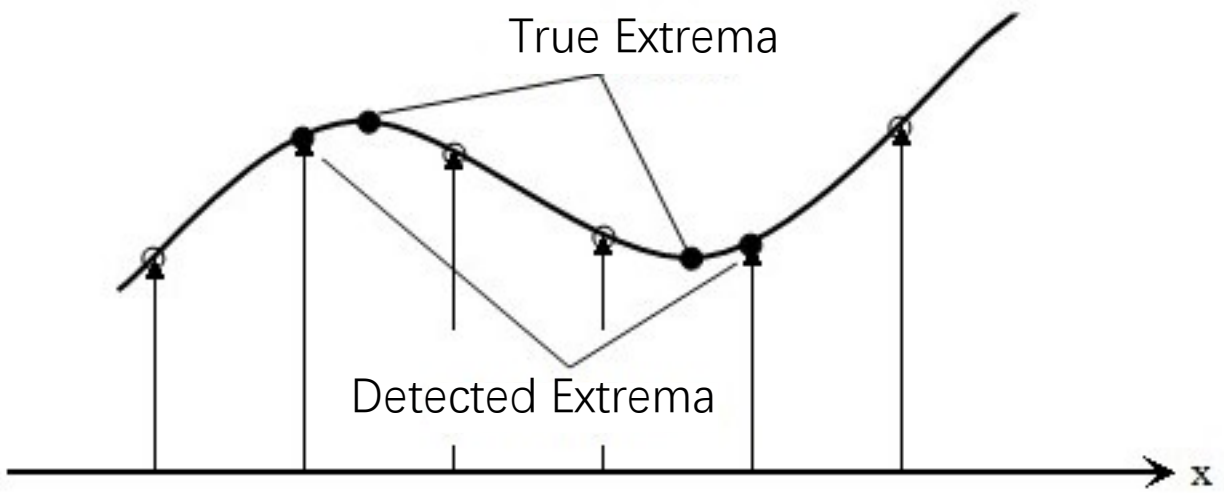


SIFT —— Localizing Extrema



SIFT —— Localizing Extrema

Sub-pixel interpolation

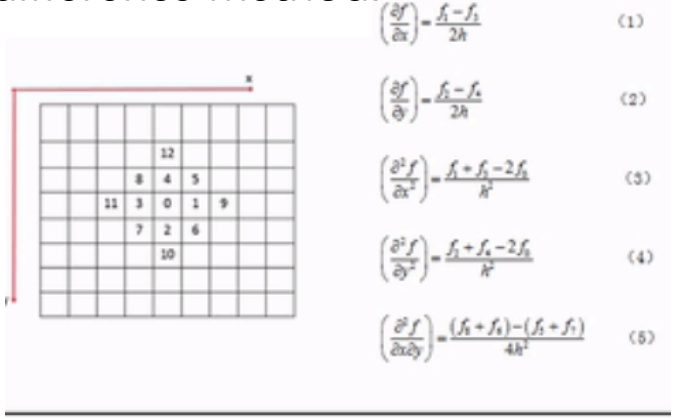


SIFT —— Localizing Extrema

Sub-pixel interpolation

$$D(\Delta x, \Delta y, \Delta\sigma) = D(x, y, \sigma) + \begin{bmatrix} \frac{\partial D}{x} & \frac{\partial D}{y} & \frac{\partial D}{\sigma} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\sigma \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta x & \Delta y & \Delta\sigma \end{bmatrix} \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial \sigma} \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial \sigma} \\ \frac{\partial^2 D}{\partial \sigma \partial x} & \frac{\partial^2 D}{\partial \sigma \partial y} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\sigma \end{bmatrix}$$

Finite difference method:



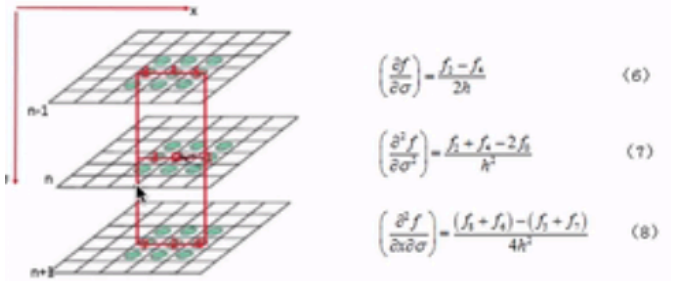
$$\left(\frac{\partial f}{\partial x}\right) = \frac{f_1 - f_3}{2h} \quad (1)$$

$$\left(\frac{\partial f}{\partial y}\right) = \frac{f_2 - f_4}{2h} \quad (2)$$

$$\left(\frac{\partial^2 f}{\partial x^2}\right) = \frac{f_1 + f_3 - 2f_0}{h^2} \quad (3)$$

$$\left(\frac{\partial^2 f}{\partial y^2}\right) = \frac{f_2 + f_4 - 2f_0}{h^2} \quad (4)$$

$$\left(\frac{\partial^2 f}{\partial x \partial y}\right) = \frac{(f_5 + f_8) - (f_6 + f_7)}{4h^2} \quad (5)$$



$$\left(\frac{\partial f}{\partial \sigma}\right) = \frac{f_1 - f_4}{2h} \quad (6)$$

$$\left(\frac{\partial^2 f}{\partial \sigma^2}\right) = \frac{f_1 + f_4 - 2f_0}{h^2} \quad (7)$$

$$\left(\frac{\partial^2 f}{\partial x \partial \sigma}\right) = \frac{(f_1 + f_4) - (f_2 + f_3)}{4h^2} \quad (8)$$

$$D(x) = D + \frac{\partial D^T}{\partial x}\Delta x + \frac{1}{2}\Delta x^T \frac{\partial^2 D^T}{\partial x^2}\Delta x$$

Derivation, substitution, iterative process to get new expression

SIFT —— Localizing Extrema

Edge effect removal

A flat DoG response peak tends to have a larger main curvature across the edge and a smaller main curvature in the direction of the vertical edge.

$$H(x, y) = \begin{bmatrix} D_{xx}(x, y) & D_{xy}(x, y) \\ D_{xy}(x, y) & D_{yy}(x, y) \end{bmatrix}$$

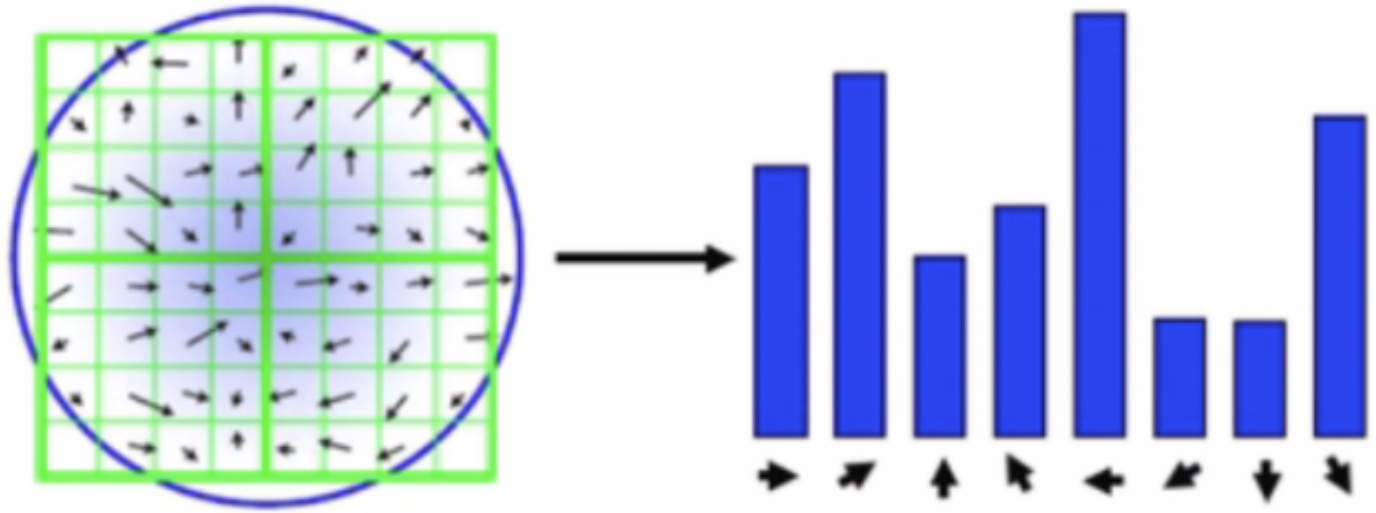
$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(\gamma + 1)^2}{\gamma}$$

$$\frac{Tr(H)^2}{Det(H)} < \frac{(\gamma + 1)^2}{\gamma}$$

SIFT —— Generating Orientations & Descriptors



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

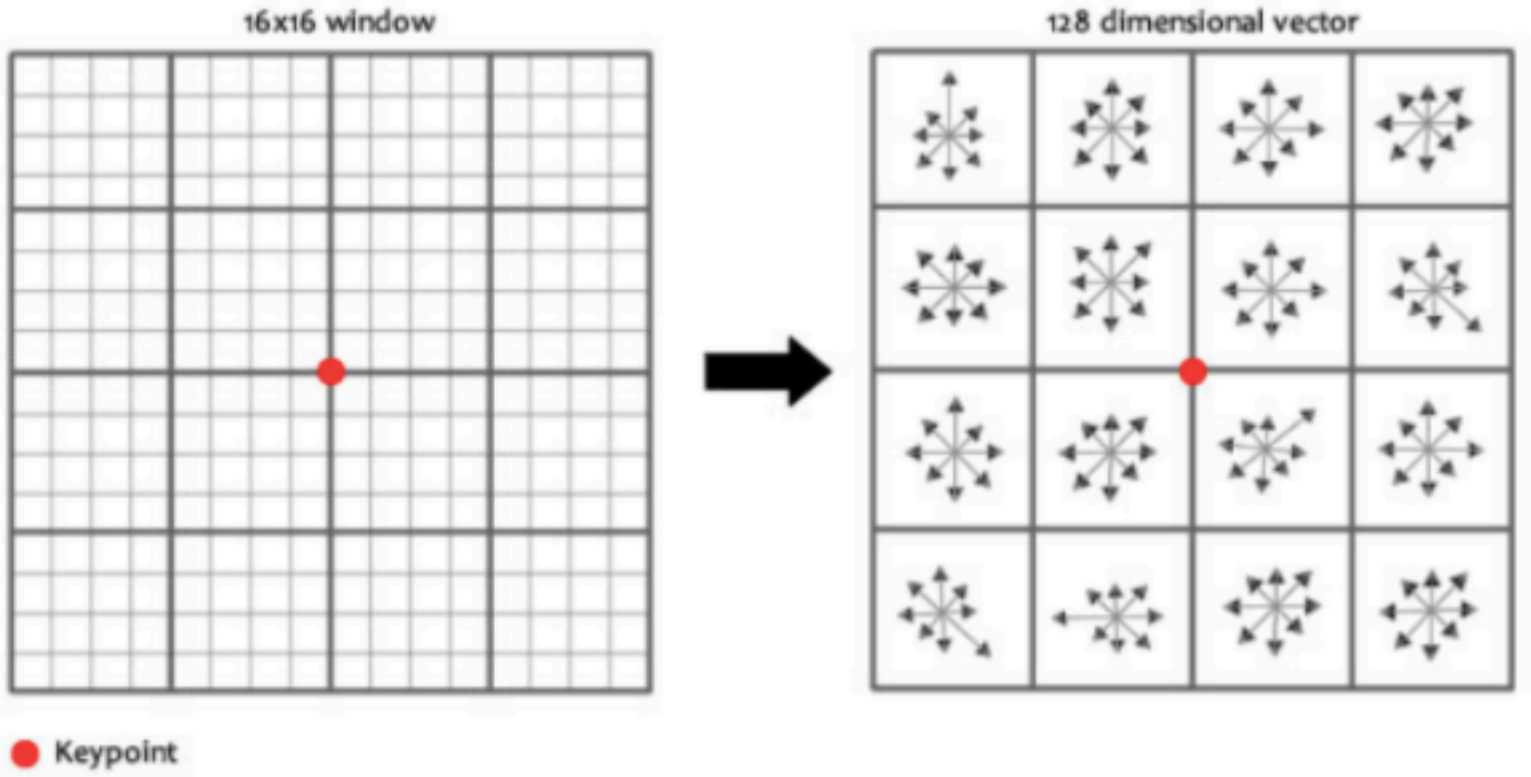
$$\theta(x,y)=\tan^{-1}\left((L(x,y+1)-L(x,y-1))/(L(x+1,y)-L(x-1,y))\right)$$

Robustness requirements:

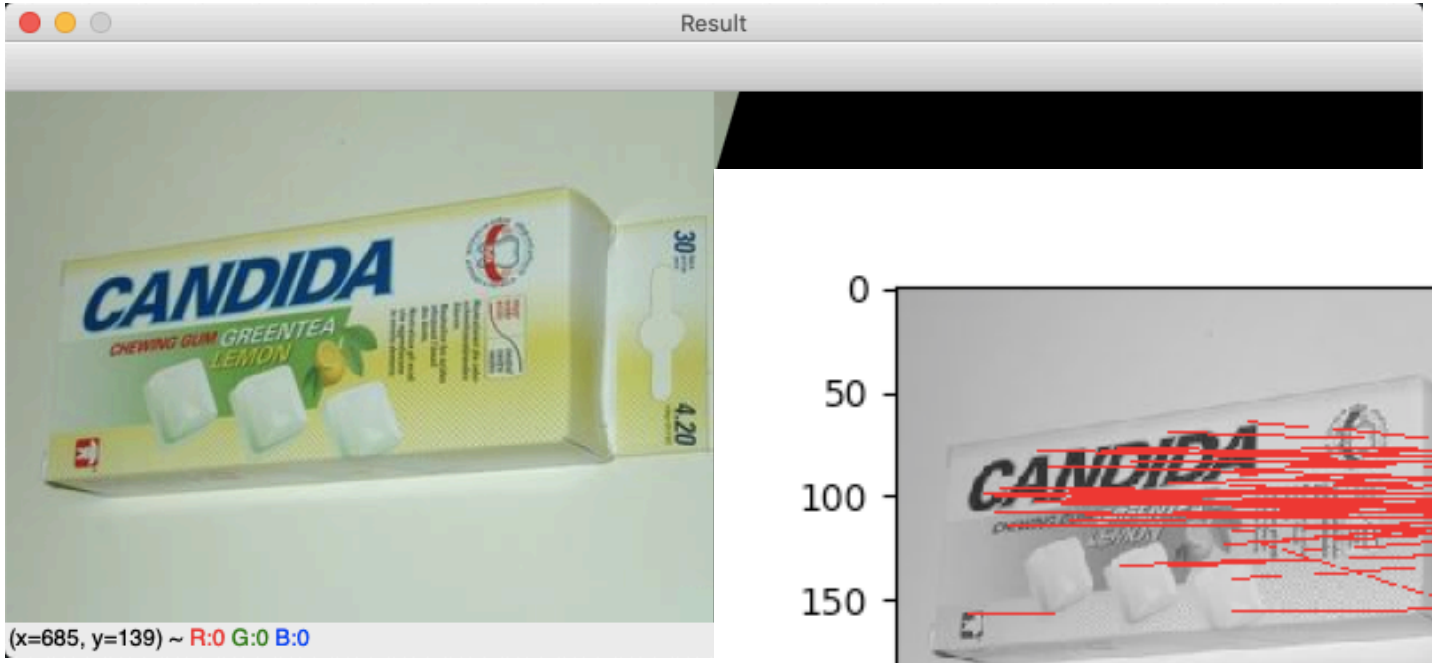
- Secondary direction

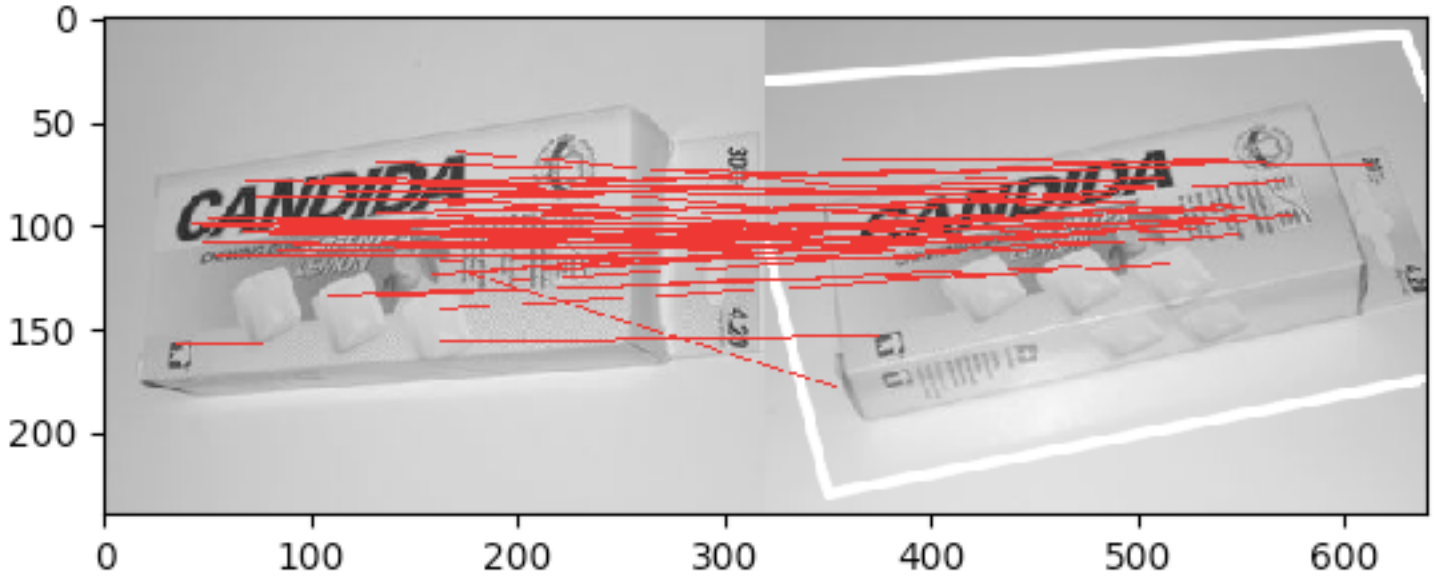
- Smoothing

SIFT —— Generating Orientations & Descriptors



SIFT —— Result


Result
CANDIDA
CHEWING GUM GREENTEA LEMON
(x=685, y=139) ~ R:0 G:0 B:0


0
50
100
150
200
0
100
200
300
400
500
600

Content

- SIFT
  - Scale Space and Image Pyramids
  - Localizing Extrema
  - Generating Orientations and Descriptors
- SURF
  - Hessian Detector
  - Math Behind Hessian Matrix
  - Construction of feature vector
  - Comparision between SIFT and SURF
  - Results
- Fast Panorama Stitching on Mobile
  - Current Approaches
  - Summary
  - Color and Luminance Compensation
  - Optimal Seam Finding and Image Labeling
  - Image Blending
  - Results

Overview

The task of finding correspondences can be divided into three stages:

1. Detection: Find “Interesting points” in two images. => repeatability
2. Feature Descriptor: =>
    1. Distinctive, robust, simple, rotation and scale invariant
    2. Only scale invariant version: USURF
3. Match => Not so interesting in SURF

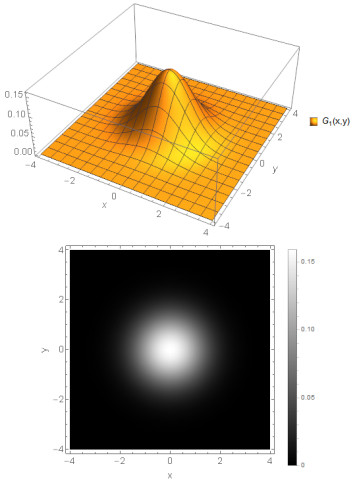
Definition 1: Hessian Matrix

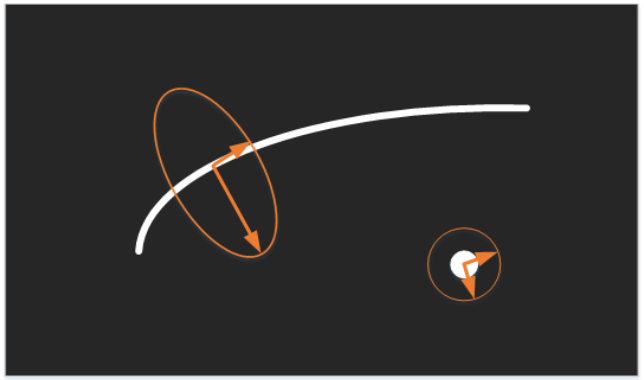
$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix}$$

$$\det \mathcal{H} = L_{xx}L_{yy} - L_{xy}^2$$
$$\operatorname{tr} \mathcal{H} = L_{xx} + L_{yy}$$

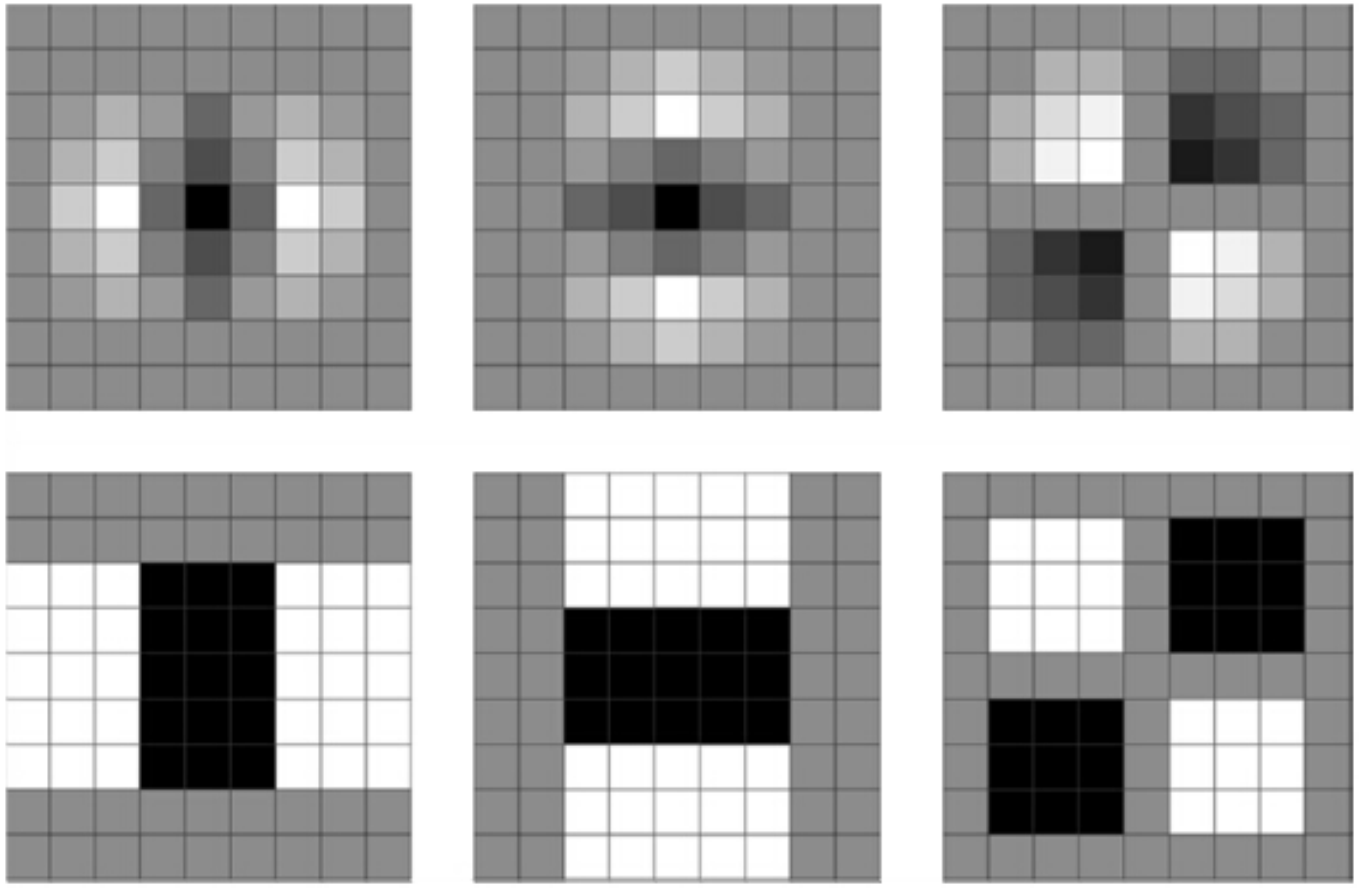
Remarks

1. Hessian Matrix is used to detect blob like structure.
2. At a certain point in an image, the two most "interesting" directions are H' s two eigenvectors
3. The mul of two eigenvalues happen to be the det of H.
4. We can use the det of H as a indicator to find interesting points.
5. Only when det of H is greater than zero.
6. We need to compute $L_{xx}L_{yy} - L_{xy}^2$





Approx Trick: Box Filter

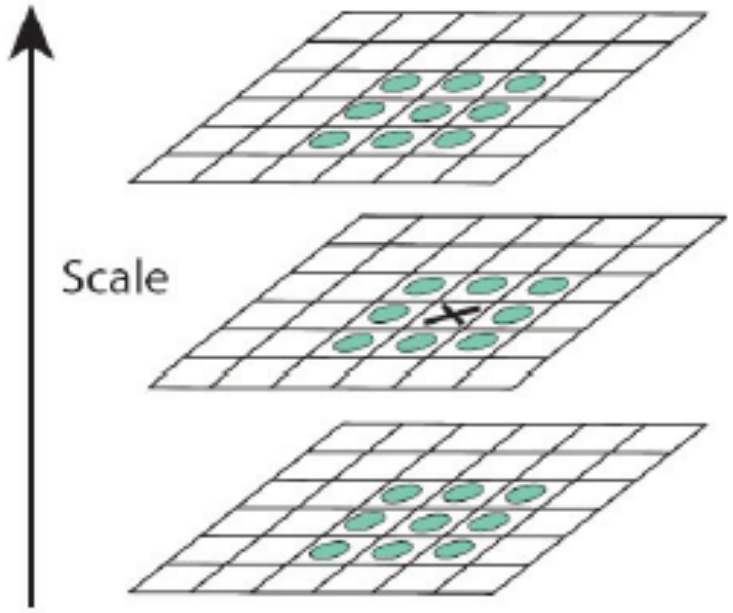


$$\det \mathcal{H} \approx D_{xx}D_{yy} - 0.9D_{xy}^2$$

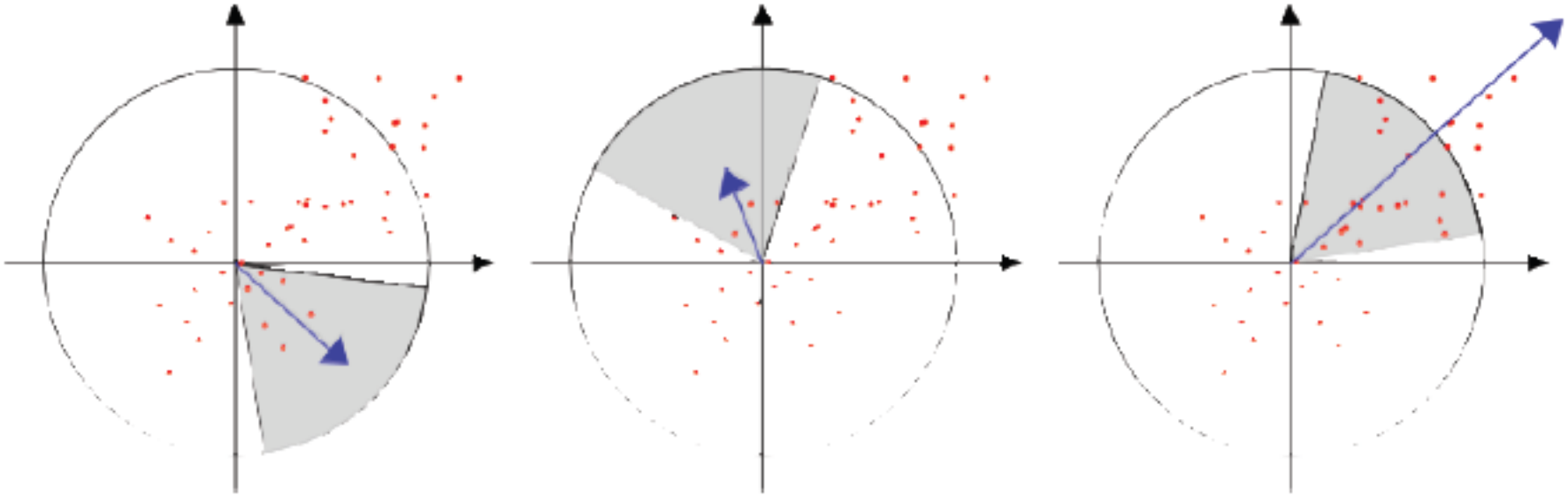
Definition 2: Integral Image

$$I_{\Sigma}(x, y) = \Sigma_{i=0}^{x} \Sigma_{j=0}^{y} I(i, j)$$

Scale Space



Orientation of each Interesting Point

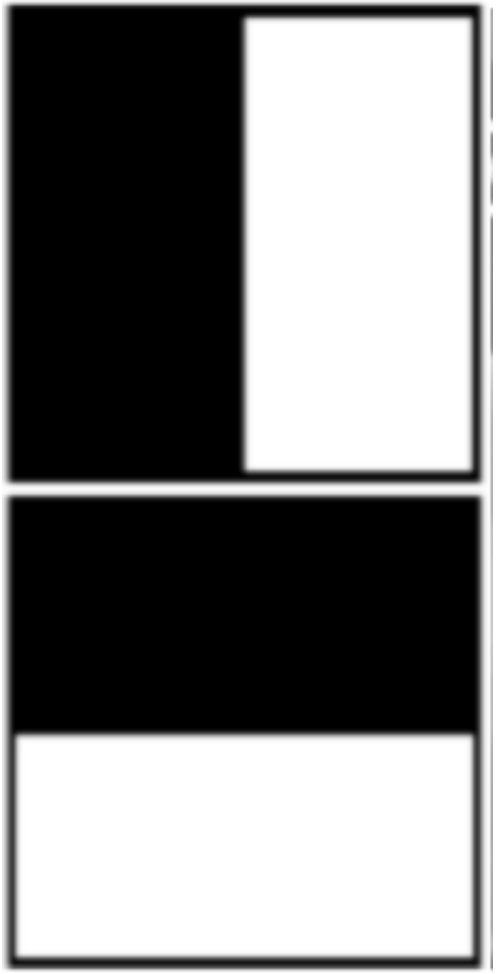


Haar Wavelet Response

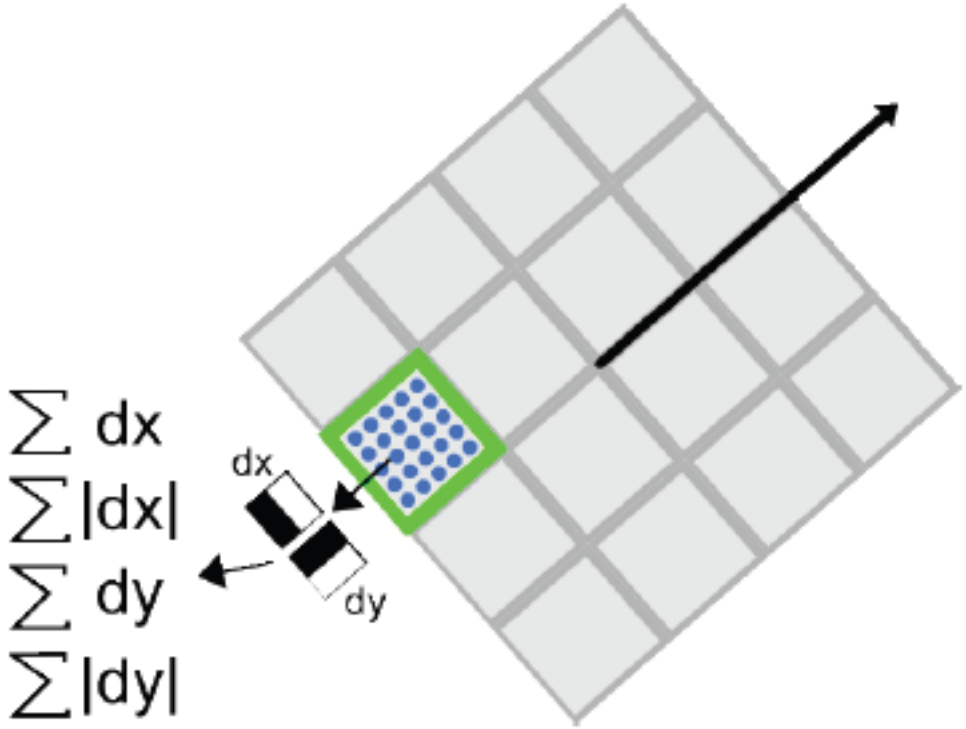
White stands for average of pixels,

Black stands for difference of pixels.

$$R(1,1)_x = \frac{I(1,1)-I(1,2)}{2}$$
$$R(1,5)_x = \frac{I(1,1)+I(1,2)}{2}$$
$$R(1,1)_y = \frac{I(1,1)-I(2,1)}{2}$$
$$R(5,1)_y = \frac{I(1,1)+I(2,1)}{2}$$



Descriptor Component



Final Output

1. N interesting points' positions(n * 2)
2. N interesting points' feature vectors(n * 64)
3. N interesting points' characteristic scale(n * 1)
4. N interesting points' orientation(n * 1)

SIFT versus SURF

	SIFT	**SURF**
Speed	Slow	Fast
Feature Vector Dimension	128	64
Detector	$\operatorname{tr}\mathcal{H}(\mathbf{x},\sigma)$	$\det\mathcal{H}(\mathbf{x},\sigma)$
Approx Tricks	DoG LoG	Box Filter
Construction of Feature Vector	gradient	Haar Wavelet Response

Result

Result

Result


http://blog.csdn.net/dcrmg


http://blog.csdn.net/dcrmg


http://blog.csdn.net/dcrmg

SIFT vs SURF

	SIFT	SURF
Case 1	1.78s	1.57s
Case 2	0.64s	0.32s
Case 3	0.21s	0.14s

Content

- SIFT
  - Scale Space and Image Pyramids
  - Localizing Extrema
  - Generating Orientations and Descriptors
- SURF
  - Hessian Detector
  - Math Behind Hessian Matrix
  - Construction of feature vector
  - Comparision between SIFT and SURF
  - Results
- Fast Panorama Stitching on Mobile
  - Current Approaches
  - Summary
  - **Color and Luminance Compensation**
  - **Optimal Seam Finding and Image Labeling**
  - **Image Blending**
  - Results

Current Approaches

- **transition smoothing**: reduce color differences between source images to make seams invisible and remove stitching artifacts
  - **Alpha blending**: fast transition smoothing approach, but it cannot avoid ghosting problems caused by object motion and small spatial alignment errors
  - **Gradient Domain Image Blending approaches**: can reduce color differences and smooth color transitions using gradient domain operations, producing high-quality composite images
- **optimal seam finding**: search for seams in overlapping areas along paths where differences between source images are minimal
- **combination**: optimal seams first, if seams and stitching artifacts are visible, transition smoothing to reduce color differences to hide the artifacts then
  - **graph cut** -> find optimal seams
  - **poisson blending** -> smoothing color transitions

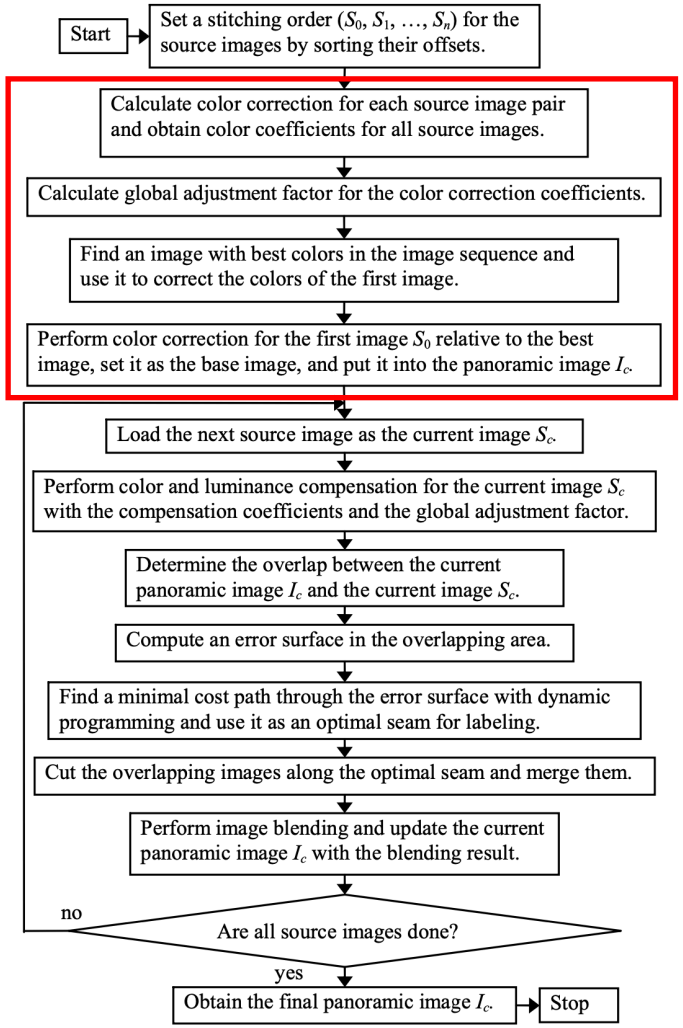
Current Approaches Problem

- computational and memory costs are high
- pixels are easy saturated in color correction
- don't work well for source images in very different colors and luminance
- linear blending, moving objects on the overlapping areas will cause ghosting artifacts

Solution

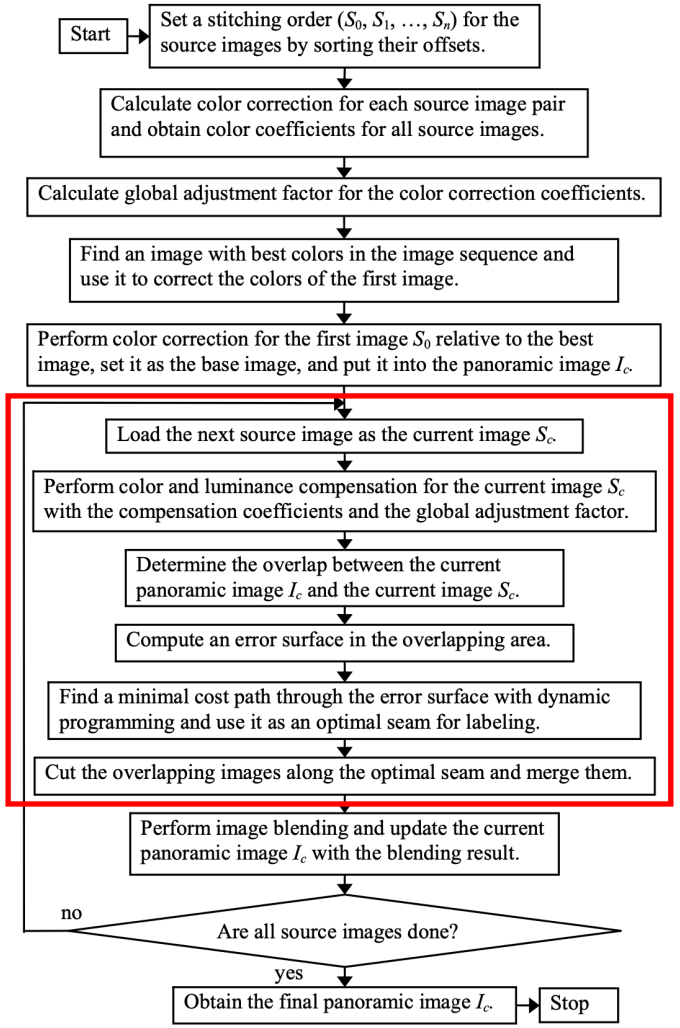
- ✓ don't need to keep all source images in memory due to the sequential stitching
- ✓ dynamic programming for optimal seam finding allowing image labeling much faster than using graph cut
- ✓ combination of color correction and image blending
- ✓ high-quality panoramic images from long image sequences with very different colors and luminance
- ✓ work well on both indoor and outdoor scenes

Summary



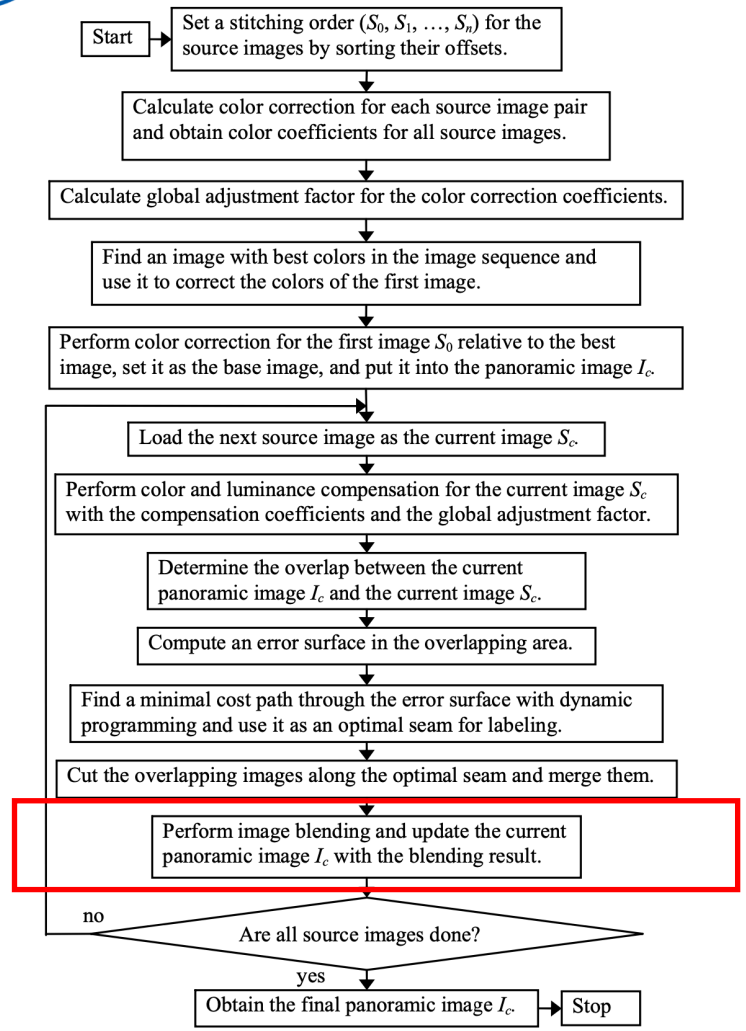
- **Color Correction**
  - color correction for all source images to reduce color differences
  - smoothen remaining color transitions between adjacent images
- **Image Labeling**
  - error surface is constructed with squared differences between overlapping images
  - low-cost path is found through the error surface by dynamic programming and used as an optimal seam to create labeling
- **Image Blending Operations**
  - linear blending -> source images are similar in color and luminance
  - poisson blending -> colors remain too different

Summary



- **Color Correction**
  - color correction for all source images to reduce color differences
  - smoothen remaining color transitions between adjacent images
- **Image Labeling**
  - error surface is constructed with squared differences between overlapping images
  - low-cost path is found through the error surface by dynamic programming and used as an optimal seam to create labeling
- **Image Blending Operations**
  - linear blending -> source images are similar in color and luminance
  - poisson blending -> colors remain too different

Summary



- **Color Correction**
  - color correction for all source images to reduce color differences
  - smoothen remaining color transitions between adjacent images
- **Image Labeling**
  - error surface is constructed with squared differences between overlapping images
  - low-cost path is found through the error surface by dynamic programming and used as an optimal seam to create labeling
- **Image Blending Operations**
  - linear blending -> source images are similar in color and luminance
  - poisson blending -> colors remain too different

Color and Luminance Compensation

Basic idea: compute light averages in the overlap area by linearizing the gamma-corrected RGB

Suppose S_{i-1} and S_i are adjacent images, S_{i-1}^o and S_i^o are where image overlap

$$\alpha_{c,i} = \frac{\sum_p \left(P_{c,i-1}(p)\right)^\gamma}{\sum_p \left(P_{c,i}(p)\right)^\gamma} \quad c \in \{R, G, B\}(i = 1,2,3,\cdots,n)$$

$P_{c,i-1}(p)$: the color value of pixel p in image S_{i-1}^o
$P_{c,i}(p)$: the color value of pixel p in image S_i^o
γ: gamma coefficient(set to 2.2 in paper)

Color and Luminance Compensation

Basic idea: compute light averages in the overlap area by linearizing the gamma-corrected RGB

g_c: global adjustment in the whole image sequence for each color channel c

Solve the least-squares equation $\min_{g_c} \sum_{i=0}^{n} \left(g_c \alpha_{c,i} - 1\right)^2 \quad c \in \{R, G, B\}$

$$g_c = \frac{\sum_{i=0}^{n} \alpha_{c,i}}{\sum_{i=0}^{n} \alpha_{c,i}^2} c \in \{R, G, B\} (i = 0,1,\ldots, n)$$

Color and Luminance Compensation

Basic idea: compute light averages in the overlap area by linearizing the gamma-corrected RGB

$$\alpha_{c,i} = \frac{\sum_p \left(P_{c,i-1}(p)\right)^\gamma}{\sum_p \left(P_{c,i}(p)\right)^\gamma} \quad c \in \{R,G,B\}(i = 1,2,3,\cdots,n)$$

$$g_c = \frac{\sum_{i=0}^{n} \alpha_{c,i}}{\sum_{i=0}^{n} \alpha_{c,i}^2} c \in \{R,G,B\}(i = 0,1,\ldots,n)$$

$$P_{c,i}(p) \leftarrow \left(g_c \alpha_{c,i}\right)^{1/\gamma} P_{c,i}(p), c \in \{R,G,B\}(i = 0,1,\ldots,n)$$

Color and Luminance Compensation

```python
def getBestImgIndex(imgs):
    best_index = 0
    best_value = 255
    for index, img in enumerate(imgs):
        current_mean = np.array([np.mean(img[:,:,0
for three channels
        diff = np.max(current_mean) - np.min(curre
        if diff < best_value:       # choose the b
            best_index = index
            best_value = diff
    return best_index
best_img_index = getBestImgIndex(imgs)
```

```python
def colorCorrection(images_temp, shift, bestIndex, gamma=2.2):      # set gamma to 2.2 by paper
    alpha = np.ones((3, len(images_temp)))

    # compute light averages in the overlap area by linearizing the gamma-corrected RGB values
    for rightBorder in range(bestIndex+1, len(images_temp)):
        for i in range(bestIndex+1, rightBorder+1):
            I = images_temp[i]
            J = images_temp[i-1]
            overlap = I.shape[1] - shift[i-1]
            for channel in range(3):
                alpha[channel, i] = np.sum(np.power(J[:,-overlap-1:,channel], gamma))/np.sum(np.power(I[:,
0:overlap+1,channel],gamma))  # derivative

        G = np.sum(alpha, 1)/np.sum(np.square(alpha), 1)

        for i in range(bestIndex+1, rightBorder+1):
            for channel in range(3):
                images_temp[i][:,:,channel] = np.power(G[channel] * alpha[channel, i], 1.0/gamma) * images_temp
[i][:,:,channel]     # perform using correction coefficients and the global adjustment

    for leftBorder in range(bestIndex-1, -1, -1):
        for i in range(bestIndex-1, leftBorder-1, -1):
            I = images_temp[i]
            J = images_temp[i+1]
            overlap = I.shape[1] - shift[i-1]
            for channel in range(3):
                alpha[channel, i] = np.sum(np.power(J[:,0:overlap+1,channel], gamma))/np.sum(np.power(I[:,
-overlap-1:,channel],gamma))

        G = np.sum(alpha, 1)/np.sum(np.square(alpha), 1)

        for i in range(bestIndex-1, leftBorder-1, -1):
            for channel in range(3):
                images_temp[i][:,:,channel] = np.power(G[channel] * alpha[channel, i], 1.0/gamma) * images_temp
[i][:,:,channel]
    return images_temp
```

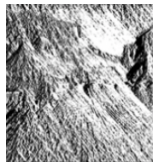
Optimal Seam Finding and Image Labeling


Input images

e: error surface $e = (I_c^o - S_c^o)^2$

E: cumulative minimum squared difference

$$E(h,w) = e(h,w) + min(E(h-1,w-1), E(h-1,w), E(h-1,w+1))$$



All possible paths

Optimal path m_c: tracking back the paths with a minimal cost from bottom to top

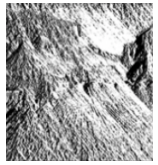
Panorama image

Optimal Seam Finding and Image Labeling


Input images

```python
def calcErrorSurface(panorama, curr_img, overlap, channel):
    left = panorama[:, -overlap-1:, channel]
    right = curr_img[:, 0:overlap+1, channel]
    return np.square(left - right)
```

```python
def calcSeam(e):
    E = np.zeros(e.shape)   # cumulative minimum squared difference
    E[0,:] = e[0,:]
    # dynamic programming
    for h in range(1, e.shape[0]):
        for w in range(0, e.shape[1]):
            if w == 0:
                cost = min(E[h-1, w], E[h-1, w+1])
            elif w == e.shape[1]-1:
                cost = min(E[h-1, w-1], E[h-1, w])
            else:
                cost = min(E[h-1, w-1], E[h-1, w], E[h-1, w+1])
            E[h,w] = e[h,w] + cost
    return E
```



```python
def calcSeamPath(E, e):
    h = e.shape[0]
    path = np.zeros((h, 1))
    idx = np.argmin(E[h-1, :])
    path[h-1] = idx
    for h in range(e.shape[0]-2,-1,-1):     # tracking back the paths w
        w = int(path[h+1][0])
        if w > 0 and E[h, w-1] == E[h+1, w]-e[h+1, w]:
            path[h] = w-1
        elif w < e.shape[1] - 1 and E[h, w+1] == E[h+1, w]-e[h+1, w]:
            path[h] = w+1
        else:
            path[h] = w

    path[path==0] = 1
    return path
```

Panorama image

Image Blending

Simple linear blending: images are similar in color and luminance after color correction

δ pixels width on both sides of the seam

$$P_{I_c,\text{ rew}}(p) = \frac{d_1^n P_{I_c}(p) + d_2^n P_{S_c}(p)}{d_1^n + d_2^n}$$

d_1, d_2: distances from pixel p to boundaries
$P_{I_{c,new}}(p)$: new color of pixel p
n: order

Image Blending

Poisson blending: perform image blending in the gradient domain

(G_x, G_y): gradients of source images using the labeling obtained using optimal seams

$$\mathrm{div}(G) = \frac{\partial G_x}{\partial x} + \frac{\partial G_y}{\partial y}$$

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

$$I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4f(x,y) = G_x(x,y) - G_x(x-1,y) + G_y(x,y) - G_y(x,y-1)$$

solve linear practical differential equation by fixing the colors at the seam and solving new colors $I(x,y)$ over the gradient field

Results

Results

Report

Background Introduction

Panorama Stitching has been a long standing problem in computer vision. The classical steps of find corresponds generally involve detection, finding descriptors invariant to scale and rotation, and finally, matching the interesting points with each other. In this project, we investigated and implemented the well-known algorithms for panorama stitching, namely, SIFT and SURF. We also offered a comparison between these two algorithms. Lastly, we also introduced an algorithm tailored for the mobile phone, a ubiquitous computing platform in the modern society.

SIFT

The SIFT algorithm uses LoG as the detector. After that, it needs to construct a scale space to find the characteristic scale for each interesting point. In this project, we zoomed into this procedure. It constructs a gaussian pyramid to represent different scales of the images. The process of building a pyramid includes using different sigmas to perform Gaussian convolution; continuously down-sampling the original image to obtain a series of images of different sizes; and making a difference between two adjacent images of the same octave to obtain an interpolated image. After that, the location of the extreme point is composed of the local extreme points in the DOG space, and the extreme points of 3X3X3 adjacent points are compared with the center point. Finally, return to the original Gaussian pyramid diagram, look for a Gaussian image close to $\sigma(scale)$, sample in the neighborhood window centered on the key point, and use the histogram to count the gradient direction of the neighborhood pixels.

SURF

The SURF algorithm is considered to be faster and more robust compared with SIFT. The SURF uses the Hessian matrix as the feature detector. The Hessian matrix is known for its ability to detect blob-like structure. In order to become a candidate for an interesting point, the product of the two eigenvalues of the Hessian matrix must be big and greater than zero. We can then use the determinant of the Hessian matrix(which equals the product of two eigenvalues) to detect the interesting points. SURF uses a box filter to approximate the Laplacian filter. It also uses integral image to further speed up the computation. The final descriptor of one SURF interesting point consists of 16 sub-region's haar wavelet response, each with 4 components.

The difference between SIFT and SURF is summarized as follows:

Table 1: Difference between SURF and SIFT

	SIFT	**SURF**
Speed	Slow	Fast
Feature Vector Dimension	128	64
Detector		
Approx Tricks	DoG LoG	Box Filter
Construction of Feature Vector	gradient	Haar Wavelet Response

Mobile Algorithm for Panorama Stitch

This algorithm addresses the problem of creating high-resolution and high-quality panoramic images from long image sequences with very different colors and luminance in source images. A fast stitching approach is proposed for combining a set of source images into a panoramic image using little memory, and implemented on mobile phones. In this approach, color correction reduces color differences of source images and balances colors and luminance in the whole image sequence, dynamic programming finds optimal seams in overlapping areas between adjacent images and merges them together, and image blending further smoothens color transitions and hides visible seams and stitching artifacts. A sequential panorama stitching procedure constructs panoramic images. The advantages include fast processing speed using dynamic programming for optimal seam finding, reducing memory needs by using the sequential panorama stitching.

Figure 1: Image Stitch using SIFT

Figure 2: Image Stitch with SURF

Figure 3: Image Stitch with Mobile Algorithm

Reference

[1] Lowe, David G. "Distinctive image features from scale-invariant keypoints." International journal of computer vision 60.2 (2004): 91-110.

[2] Bay, Herbert, et al. "Speeded-up robust features (SURF)." Computer vision and image understanding 110.3 (2008): 346-359.

[3] Xiong, Yingen, and Kari Pulli. "Fast panorama stitching for high-quality panoramic images on mobile phones." IEEE Transactions on Consumer Electronics 56.2 (2010): 298-306.

Thanks For Listening

Zhe ZHANG, 1754060
Kaixin CHEN, 1753188
Yunxin SUN, 1551534
School of Software Engineering
Tongji University
Spring 2020