Final Project
Panorama Stitching

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

Content

- SURF
    - Xxx
    - Yyy
    - zzz
- SIFT
    - Xxx
    - Yyy
    - zzz
- 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

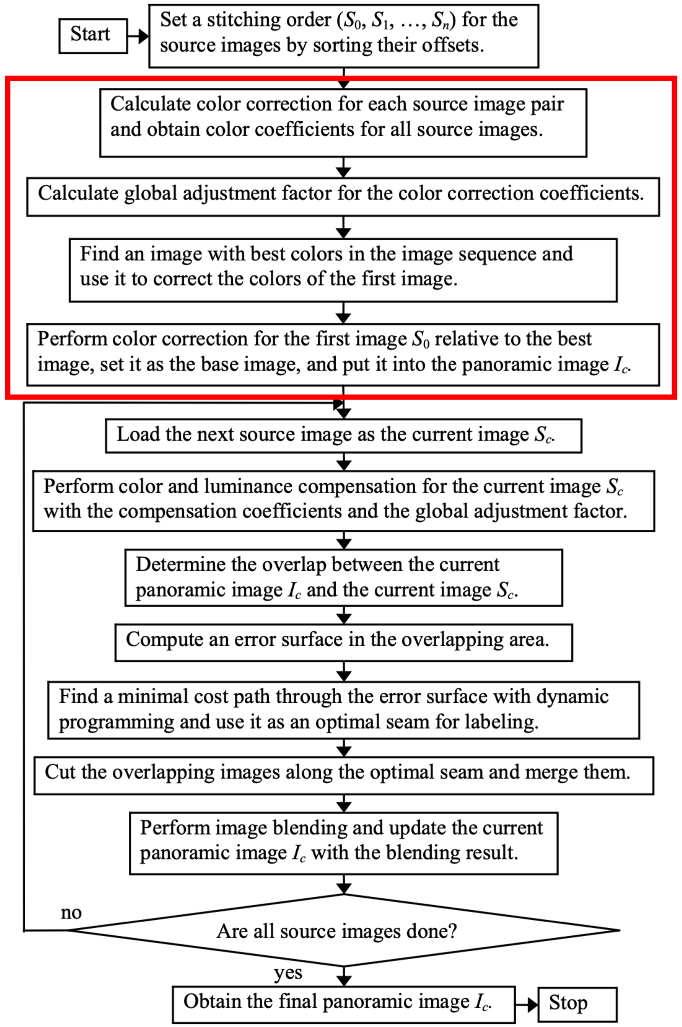
Solution

- ✓ 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

Fast Panorama Stitching for High-Quality Panoramic Images on Mobile Phones

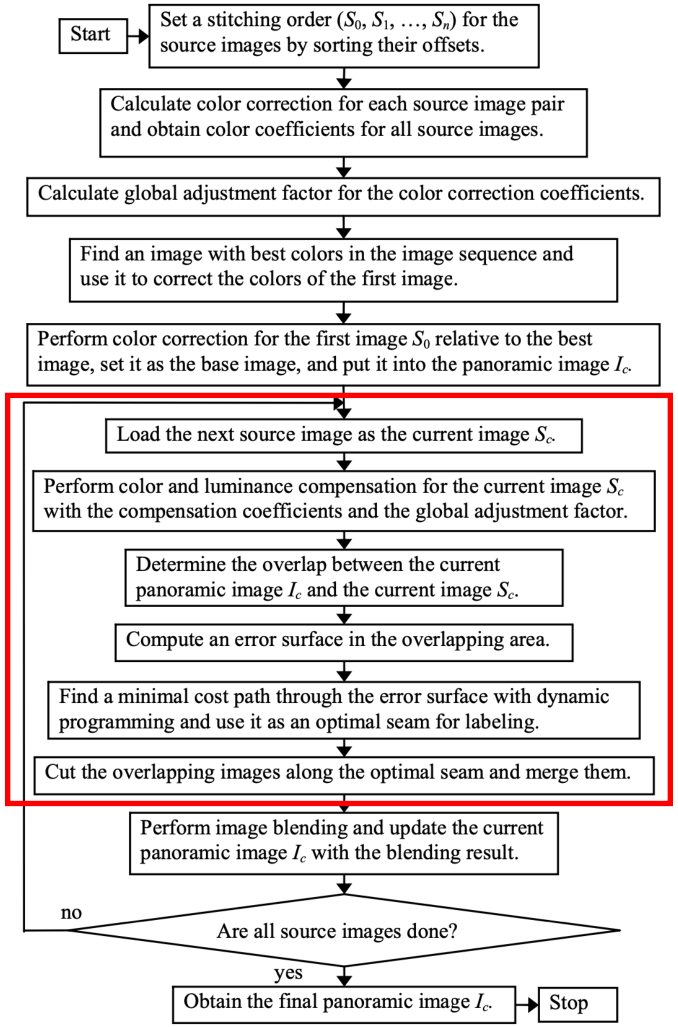
Yingen Xiong and Kari Pulli, Member, IEEE

Summary



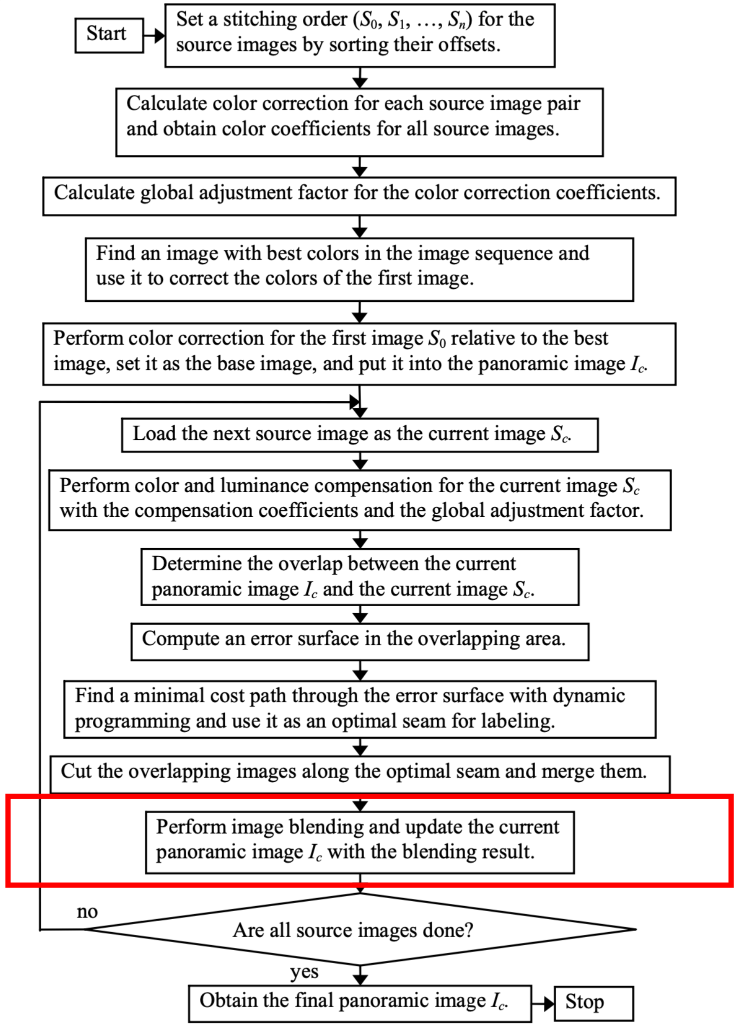
- **Color Correction**
  - color correction for all source images to reduce color differences
  - smoothen remaining color transitions between adjacent images
- **Image Labeling**
  - error surface is constructed with squared differences between overlapping images
  - low-cost path is found through the 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

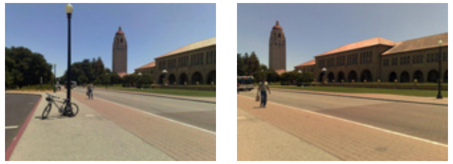
$$\left.\begin{aligned}\alpha_{c,i} &= \frac{\sum_p \left(P_{c,i-1}(p)\right)^{\gamma}}{\sum_p \left(P_{c,i}(p)\right)^{\gamma}} \quad c \in \{R, G, B\}(i = 1,2,3,\cdots, n) \\ g_c &= \frac{\sum_{i=0}^{n} \alpha_{c,i}}{\sum_{i=0}^{n} \alpha_{c,i}^2} c \in \{R, G, B\}(i = 0,1,\ldots, n)\end{aligned}\right\} \quad P_{c,i}(p) \leftarrow \left(g_c \alpha_{c,i}\right)^{1/\gamma} P_{c,i}(p), c \in \{R, G, B\}(i = 0,1,\ldots, n)$$

Color and Luminance Compensation

```
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)
```

```
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

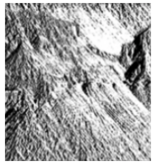


Input images

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

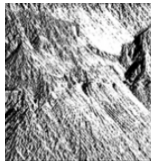
Panorama image

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 wi
        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

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

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

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

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

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

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