ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Computer Vision

Lớp L01

Assignment 3: Gradient Domain Editing

Giảng viên hướng dẫn: Võ Thanh Hùng
Sinh viên: Đinh Vũ Hà - 2113269

Thành phố Hồ Chí Minh, Tháng 03/2024

Mục lục

1 Yêu cầu

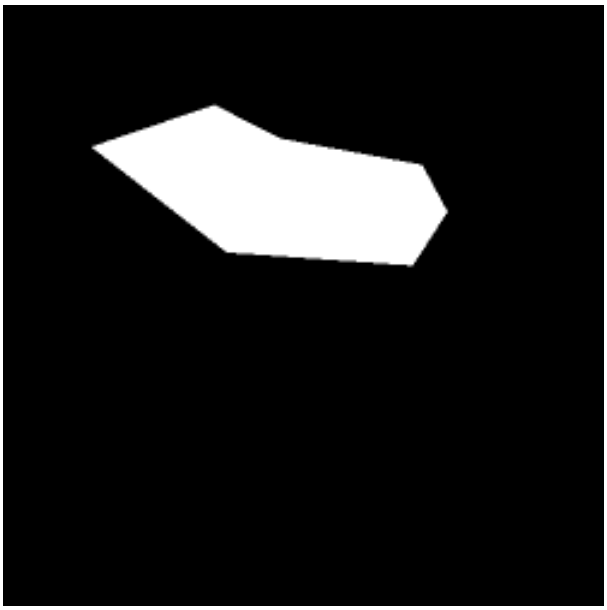
Ta cần chọn ra 2 hình, hình 1 làm nền và hình 2 để lấy object ghép vào hình 1.

2 Lý thuyết

Trong bài tập lớn này, chúng ta sử dụng hai ảnh: một ảnh nguồn (foreground.jpg) và một ảnh nền (background.jpg), chúng ta cắt phần của ảnh nguồn mà chúng ta muốn ghép vào ảnh nền, chuẩn bị một mask có hình dạng giống với phần cắt của ảnh nguồn (mask.png)

Ảnh nguồn ta muốn cắt

Ảnh nền



mask

Để làm được bài tập lớn này, 3 ảnh trên phải có cùng kích cỡ, và ở đây đã được chuyển về 250x250 px. Đối với mỗi pixel trong ảnh mục tiêu (target image) mà nằm ngoài mask phần nguồn, chúng ta sao

chép giá trị pixel từ ảnh nền. Đối với mỗi pixel trong mask phần nguồn, chúng ta giải quyết bài toán lease squares (bình phương tối thiểu):

$$\mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - (s_i - s_j))^2 + \sum_{i \in S, j \in N_i \cap \neg S} ((v_i - t_j) - (s_i - s_j))^2$$

Trong phương trình trên, biến v đại diện cho giá trị pixel từ ảnh mục tiêu, s đại diện cho giá trị pixel từ vùng nguồn dưới mask và t đại diện cho giá trị của các pixel lân cận của pixel nằm tại biên của mask, giá trị của chúng được lấy từ ảnh mục tiêu. Kết quả hệ phương trình sau đó được giải quyết như một bài toán bình phương tối thiểu, từ đó cho chúng ta các giá trị của vùng mask trong ảnh mục tiêu, sau đó được thế vào để có được ảnh đầu ra.

3 Code

Link tới thư mục drive của Assignment: https://drive.google.com/drive/folders/1Lzm__YaDx1wtT9JX7aZl2QIAK0THps9O?usp=drive_link

4 Thực hiện

Trước tiên, ta cần import các thư viện cần thiết. Đặt tên file gồm ảnh nguồn (foreground.jpg) là F, ảnh nền (background.jpg) là B và mask (mask.png) là M

```
1   import matplotlib.pylab as plt
2   from PIL import Image
3   import numpy as np
4   from scipy import sparse
5   import scipy.sparse.linalg as splinalg
6
7
8   F = 'foreground.jpg'
9   B = 'background.jpg'
10  M = 'mask.png'
11
```

Import các hàm, thư viện và cho ra ảnh

Chúng ta có class GradientDomainCloning. Class này có các hàm như sau:

- Hàm __init __(self, F, B, M): Hàm này sẽ nhận vào 3 file bao gồm ảnh nguồn là F, ảnh nền là B và mask là M. Nó chuẩn bị dữ liệu cần thiết để thực hiện quá trình Gradient Domain Cloning sau này.

```
14      def __init__(self, F, B, M):
15          # foreground
16          self.f = np.asarray(Image.open(F),dtype=int)
17
18          # background
19          self.b = np.asarray(Image.open(B),dtype=int)
20          # mask
21          self.m = np.asarray(Image.open(M),dtype=int)
22
23          # width and height
24          self.h = self.b.shape[0]
25          self.w = self.b.shape[1]
26          # new image after gradient domain cloning
27          self.new = Image.new('RGB',(self.h,self.w))
28          # map coordinate of pixels to be calculated to index_map according to mask
29          self.idx_map = []
30
31          # map coordinates of neigbourhoods to mask indices
32          ngb_map = []
33
34          # map coordinates to mask indices
35          self.pMap = [[-1 for i in range (self.w)] for j in range(self.h)]
36          counter = 0;
37
38          for i in range(self.h):
39              for j in range(self.w):
40                  if self.m[:,:,0][i][j]==255:
41                      self.idx_map.append([i,j])
42                      ngb_map.append([self.m[:,:,0][i-1][j]==255,
43                                      self.m[:,:,0][i+1][j]==255,
```

```
                                self.m[:,:,0][i+1][j]==255,
                                self.m[:,:,0][i][j-1]==255,
                                self.m[:,:,0][i][j+1]==255])
                    self.pMap[i][j] = counter
                    counter = counter+1

        # nxn matrix A, nx1 vector b are used to solve poisson equation Au=b
        # for nx1 unknown pixel color vector u
        # r, g, b, 3 channels are calculated seperately
        n = len(self.idx_map)
        self.b_r = np.zeros(n)
        self.b_g = np.zeros(n)
        self.b_b = np.zeros(n)
        self.A = [[0 for i in range (n)] for j in range(n)]


        # set up sparse matrix A, 4's on main diagnal, -1's and 0's off main diagnal
        for i in range(n):
            self.A[i][i] = 4;
            xx = self.idx_map[i][0]
            yy = self.idx_map[i][1]
            if (ngb_map[i][0] == True):
                self.A[i][self.pMap[xx-1][yy]] = -1
            if (ngb_map[i][1] == True):
                self.A[i][self.pMap[xx+1][yy]] = -1
            if (ngb_map[i][2] == True):
                self.A[i][self.pMap[xx][yy-1]] = -1
            if (ngb_map[i][3] == True):
                self.A[i][self.pMap[xx][yy+1]] = -1
        self.A = sparse.lil_matrix(self.A, dtype=int)
```

hàm init

- Hàm count_neighbor(self, pix _idx): Được sử dụng để đếm số lượng hàng xóm trong vùng clone của một pixel và xác định xem pixel đó có nằm ở biên của vùng clone không.

```python
def count_neighbor(self, pix_idx):
    count = 0
    boundary_flag = [0,0,0,0]
    y = pix_idx[0]
    x = pix_idx[1]
    # has left neighbor or not
    if (y >= 0 and y < self.h):
        if (y==0 or self.pMap[y-1][x] == -1):
            boundary_flag[0] = 1
        else:
            count +=1
        if (y == self.h-1 or self.pMap[y+1][x] == -1):
            boundary_flag[1] = 1
        else:
            count +=1
    if ( x >= 0 and x < self.w):
        if (x==0 or self.pMap[y][x-1] == -1):
            boundary_flag[2] = 1
        else:
            count +=1
        if (x == self.w-1 or self.pMap[y][x+1] == -1):
            boundary_flag[3] = 1
        else:
            count +=1
    return count,boundary_flag
```

Hàm count_neighbor

- Hàm poisson_solver(self): Được sử dụng để giải phương trình Poisson trong quá trình Gradient Domain Cloning.

```python
def poisson_solver(self):
    # split into r, g, b 3 channels and
    # iterate through all pixels in the cloning region indexed in idx_map
    for i in range(len(self.idx_map)):
        neighbors, flag = self.count_neighbor(self.idx_map[i])
        x, y = self.idx_map[i]
        if neighbors == 4:
            # degraded form if neighbors are all within clone region
            self.b_r[i] = 4*self.f[x,y,0] - (self.f[x-1,y,0] +self.f[x+1,y,0] + self.f[x,y-1,0] + self.f[x,y+1,0])
            self.b_g[i] = 4*self.f[x,y,1] - (self.f[x-1,y,1] +self.f[x+1,y,1] + self.f[x,y-1,1] + self.f[x,y+1,1])
            self.b_b[i] = 4*self.f[x,y,2] - (self.f[x-1,y,2] +self.f[x+1,y,2] + self.f[x,y-1,2] + self.f[x,y+1,2])
        # have neighbor(s) on the clone region boundary, include background terms
        else:
            self.b_r[i] = 4*self.f[x,                    y,0] +self.f[x+1,y,0] + self.f[x,y-1,0] + self.f[x,y+1,0])
            self.b_g[i] = 4*self.f[x,y,1] - (self.f[x-1,y,1] +self.f[x+1,y,1] + self.f[x,y-1,1] + self.f[x,y+1,1])
            self.b_b[i] = 4*self.f[x,y,2] - (self.f[x-1,y,2] +self.f[x+1,y,2] + self.f[x,y-1,2] + self.f[x,y+1,2])
            self.b_r[i] += flag[0] * self.b[x-1,y,0] + flag[1] * self.b[x+1,y,0] + flag[2] * self.b[x,y-1,0] + flag[3] * self.b[x,y+1
            self.b_g[i] += flag[0] * self.b[x-1,y,1] + flag[1] * self.b[x+1,y,1] + flag[2] * self.b[x,y-1,1] + flag[3] * self.b[x,y+1
            self.b_b[i] += flag[0] * self.b[x-1,y,2] + flag[1] * self.b[x+1,y,2] + flag[2] * self.b[x,y-1,2] + flag[3] * self.b[x,y+1


    # use conjugate gradient to solve for u
    u_r = splinalg.cg(self.A, self.b_r)[0]
    u_g = splinalg.cg(self.A, self.b_g)[0]
    u_b = splinalg.cg(self.A, self.b_b)[0]

    return u_r, u_g, u_b
```

(variable) y: Any

poisson_solver

- Hàm combine(self): Được sử dụng để kết hợp các kết quả sau khi giải phương trình Poisson và thực hiện Gradient Domain Cloning cho vùng clone trong hình ảnh

```python
def combine(self):
    self.new = np.array(self.new,dtype=int)
    u_r,u_g,u_b = self.poisson_solver()


    # naive copy
    for i in range(3):
        self.new[:,:,i] = self.b[:,:,i];

    # fix cloning region
    for i in range(len(self.idx_map)):
        x, y = self.idx_map[i]
        self.new[x,y,0] = min(255,u_r[i])
        self.new[x,y,1] = min(255,u_g[i])
        self.new[x,y,2] = min(255,u_b[i])
    self.new = np.asarray(self.new, dtype='uint8')
```

Hàm combine

Kết quả của việc ghép ảnh sẽ được lưu trong file output.png. Và đây là kết quả:

Kết quả