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

BÁO CÁO
DIGITAL IMAGE PROCESSION AND COMPUTER VISION
PROJECT 3
GRADIENT DOMAIN EDITING

GVHD: Ths. Võ Thanh Hùng

—o0o—

SVTH: Nguyễn Tiến Phát - 2011797

TP. HỒ CHÍ MINH, 4/2024

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Giới thiệu

1.1 Khái niệm về Gradient domain editing

Trong lĩnh vực chỉnh sửa ảnh, Gradient domain editing được xem như một phương pháp mạnh mẽ để thực hiện các tác vụ liên quan đến chỉnh sửa các chi tiết hình ảnh mượt mà và chính xác. Các thức hoạt động dựa trên sự khác biệt giữa các pixel lân cận, thay vì trên các giá trị pixel với mục tiêu là xây dựng hình ảnh mới bằng cách tích hợp gradient thông qua việc giải phương trình Poisson.

Quá trình thực hiện gồm 2 bước. Đầu tiên, chọn một gradient hình ảnh (trích xuất từ một hoặc nhiều hình ảnh và sau đó được sửa đổi). Bước hai là giải phương trình Poisson để tìm một hình ảnh mới có thể tạo ra gradient từ bước đầu tiên.

Có rất nhiều ứng dụng của phương pháp này có thể kể đến như: Seamless Cloning, Texture Flattening, Local Color Changes, Seamless Tiling,... Tuy nhiên trong bài tập lớn lần này, chúng ta chỉ thực hiện Seamless Cloning thông qua bài toán "chọn ra 2 hình, hình 1 làm nền và hình 2 để lấy object ghép vào hình 1".

Chương 2

Hiện thực

2.1 Cơ sở lý thuyết

Guided interpolation (tạm dịch: nội suy hướng dẫn)

Chúng ta sử dụng một trường vector hướng để hướng dẫn việc nội suy hình ảnh. Trong trường hợp này, trường vector hướng được xác định bằng gradient của hình ảnh nguồn.

Cho S, một tập con đóng của R^2, là miền định nghĩa hình ảnh, và ω là một tập con đóng của S với $\partial\omega$ biên. Cho f^* là một hàm vô hướng đã biết được xác định trên S trừ đi phần trong của ω và cho f là một hàm vô hướng chưa biết được xác định trên phần trong của ω. Cuối cùng, giả sử v là một trường vectơ được định nghĩa theo . Nội suy f đơn giản nhất của f^* trên ω là nội suy màng được định nghĩa là giải pháp của bài toán tối thiểu:

$$\min_f \int\int_\Omega \|\nabla f\|^2 \text{ with } f|_{\delta\Omega} = f^*|_{\delta\Omega} \tag{2.1}$$

Trong trường hợp bài toán ghép vật thể của chúng ta, Mix gradient là lựa chọn hợp lý. Ví dụ để thêm các đối tượng với kết cấu phức tạp hoặc có lỗ hoặc trong suốt một phần lên nền có nhiều chi tiết. Chúng ta sẽ định nghĩa trường hướng dẫn v là sự kết hợp tuyến tính của các trường gradient nguồn và đích. Tại mỗi điểm ω, ta giữ lại biến thể mạnh hơn trong f^* hoặc g, sử dụng trường hướng dẫn sau:

$$\text{for all } x \in \omega, v(x) = \begin{cases} \nabla f^*(x) & \text{if } |\nabla f^*(x)| > |\nabla g(x)|, \\ \nabla g(x) & \text{otherwise.} \end{cases}$$

Từ đây, ta xác định được v theo công thức:

$$v(x) = \begin{cases} \nabla f^*(x) & \text{if } |\nabla f^*(x)| > |\nabla g(x)|, \\ \nabla g(x) & \text{otherwise.} \end{cases}$$

2.2 Hiện thực với python

Toàn bộ source code có thể tìm thấy tại đây: Google colab

2.2.1 Cài đặt một vài hàm phụ trợ tính toán cần thiết

```
from PIL import Image
import numpy as np
from skimage.segmentation import find_boundaries
import scipy.signal
import scipy.linalg
import scipy.sparse

def read_image(filename, gray=False):
  img = Image.open(filename)
  if gray:
      img = img.convert("L")
```

```
  img = np.array(img)
  if len(img.shape) == 3:
      img = img[..., :3]
  return img.astype(np.float64) / 255
def compute_gradient(img, forward=True):
  if forward:
      kx = np.array([
          [0, 0, 0],
          [0, -1, 1],
          [0, 0, 0]
      ])
      ky = np.array([
          [0, 0, 0],
          [0, -1, 0],
          [0, 1, 0]
      ])
  else:
      kx = np.array([
          [0, 0, 0],
          [-1, 1, 0],
          [0, 0, 0]
      ])
      ky = np.array([
          [0, -1, 0],
          [0, 1, 0],
          [0, 0, 0]
      ])
  Gx = scipy.signal.fftconvolve(img, kx, mode="same")
  Gy = scipy.signal.fftconvolve(img, ky, mode="same")
  return Gx, Gy

def get_masked_values(values, mask):
  assert values.shape == mask.shape
  nonzero_idx = np.nonzero(mask) # get mask 1
  return values[nonzero_idx]

def separate_mask(mask):
    boundary = find_boundaries(mask, mode="inner").astype(int)
    inner = mask - boundary
    return inner, boundary
def get_pixel_ids(img):
    pixel_ids = np.arange(img.shape[0] * img.shape[1]).reshape(img.shape[0], img.shape[1])
    return pixel_ids
```

2.2.2 Tạo ma trận tham số

Ma trận này được xây dựng để giải phương trình Poisson. Nó biểu diễn các hệ số của các phương trình tuyến tính.

```
def construct_A_matrix():
    n1_pos = np.searchsorted(mask_ids, inner_ids - 1)
    n2_pos = np.searchsorted(mask_ids, inner_ids + 1)
    n3_pos = np.searchsorted(mask_ids, inner_ids - img_w)
    n4_pos = np.searchsorted(mask_ids, inner_ids + img_w)

    A = scipy.sparse.lil_matrix((len(mask_ids), len(mask_ids)))
    A[inner_pos, n1_pos] = 1
    A[inner_pos, n2_pos] = 1
    A[inner_pos, n3_pos] = 1
    A[inner_pos, n4_pos] = 1
    A[inner_pos, inner_pos] = -4
    A[boundary_pos, boundary_pos] = 1
    A = A.tocsr()
    print("matrix A: ", A)
    return A
```

Vector b biểu diễn phần bên phải của các phương trình tuyến tính trong phương trình Poisson.

```
def construct_b(inner_gradient_values, boundary_pixel_values):
    b = np.zeros(len(mask_ids))
    b[inner_pos] = inner_gradient_values
    b[boundary_pos] = boundary_pixel_values
    return b
```

2.2.3 Tiến hành giải phương trình Poisson với phương pháp Mixing gradient

```
def get_alpha_blended_img(src, target, alpha_mask):
    return src * alpha_mask + target * (1 - alpha_mask)

def compute_mixed_gradients( src, target):
  Ix_src, Iy_src = compute_gradient(src)
  Ix_target, Iy_target = compute_gradient(target)
  I_src_amp = (Ix_src**2 + Iy_src**2)**0.5
  I_target_amp = (Ix_target**2 + Iy_target**2)**0.5
  Ix = np.where(I_src_amp > I_target_amp, Ix_src, Ix_target)
  Iy = np.where(I_src_amp > I_target_amp, Iy_src, Iy_target)
  Ixx, _ = compute_gradient(Ix, forward=False)
  _, Iyy = compute_gradient(Iy, forward=False)
  print("compute mix gradients: Ixx + Iyy=", Ixx+ Iyy)
  return Ixx + Iyy

def poisson_blend_channel(src, target):
    mixed_gradients = compute_mixed_gradients(src, target)

    boundary_pixel_values = get_masked_values(target, boundary_mask).flatten()
    inner_gradient_values = get_masked_values(mixed_gradients, inner_mask).flatten()

    # Construct b
    b = construct_b(inner_gradient_values, boundary_pixel_values)

    # Solve Ax = b
    x = solver_func(A, b)
    if isinstance(x, tuple): # solvers other than spsolve
        x = x[0]
        print("Solve x = ", x)
```

```
    new_src = np.zeros(src.size)
    new_src[mask_pos] = x
    new_src = new_src.reshape(src.shape)
    print("new src: ", new_src)
    poisson_blended_img = get_alpha_blended_img(new_src, target, mask)
    poisson_blended_img = np.clip(poisson_blended_img, 0, 1)

    return poisson_blended_img


def calculate( ):
  result_img = []
  for i in range(src.shape[-1]):
      result_img.append(
          poisson_blend_channel(src[..., i], target[..., i])
      )
  return np.dstack(result_img)
```

2.2.4 Khởi chạy và hiển thị kết quả

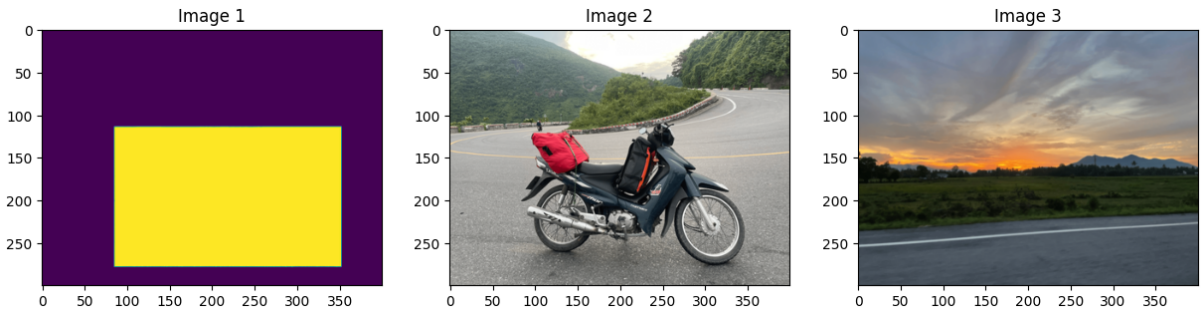
```
path = '/content/drive/MyDrive/ColabNotebooks/ComputerVision/data1/'
mask = read_image(path + '111.png', True)
src = read_image(path + '222.png')
target = read_image(path + '333.png')
A = construct_A_matrix()
final = calculate()
img = (final*255).astype(np.uint8)
plt.imshow(img)
plt.show()
```

Chương 3

Kết quả đạt được và tổng kết

3.1 Kết quả

3.1.1 Bộ ảnh để hiện thực



Hình 3.1: *Origin image*

3.1.2 Ảnh sau khi thêm vật thể

Hình 3.2: *Result*

3.2 Tổng kết

Sau khi hiện thực bài toán ghép vật thể vào một ảnh bằng phương pháp gradient domain editing, ta đã đạt được những kết quả đáng kinh ngạc cùng những bài học hữu ích không chỉ trên phương diện lý thuyết mà cả ở thực hành với python, cụ thể:

1. Phương pháp nội suy hướng dẫn (Guided Interpolation) thông qua việc sử dụng trường vector hướng được xác định bằng gradient của hình ảnh nguồn để hướng dẫn việc nội suy hình ảnh. Quá trình nội suy được thực hiện thông qua giải phương trình tối thiểu, nhằm tìm ra hàm vô hướng f thỏa mãn điều kiện biên nhất định.

2. Trong bài toán ghép vật thể, ta đã lựa chọn phương pháp Mix Gradient là lựa chọn hợp lý. Điều này phản ánh trong việc xác định trường hướng dẫn v, là sự kết hợp tuyến tính của các trường gradient từ hình ảnh nguồn và đích. Điều này cho phép giữ lại biến thể mạnh hơn giữa hai hình ảnh để tạo ra kết quả hợp lý và tự nhiên.

3. Phần hiện thực với Python đã được thực hiện thông qua việc sử dụng các thư viện xử lý ảnh như OpenCV và NumPy, cùng với các phương pháp và công cụ hỗ trợ như giải phương trình Poisson và xây dựng ma trận. Điều này đã tạo ra một cách tiếp cận linh hoạt và hiệu quả trong quá trình hiện thực.

4. Ảnh sau khi xử lý ghép vật thể xong tuy vẫn còn nhiều "sạn" - nguyên nhân bởi độ phức tạp của cả ảnh nguồn lẫn ảnh đích. Tuy nhiên, nó đã làm khá tốt việc gắn vật thể vào ảnh nền mới.

Cuối cùng, việc áp dụng gradient domain editing đã mang lại những kết quả ấn tượng trong việc ghép vật thể vào một ảnh, đồng thời cung cấp một cái nhìn rõ ràng về cách thức thực hiện và ứng dụng của phương pháp này trong lĩnh vực xử lý ảnh.

3.3 Tài liệu tham khảo

1. Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing* (4th Edition). Pearson, 2022.

2. Sonka, Milan, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision* (4th Edition). Cengage Learning, 2019.

3. Fattal, Raanan; Lischinski, Dani; Werman, Michael (2002). "Gradient domain high dynamic range compression" (PDF). Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02. p. 249. doi:10.1145/566570.566573. ISBN 1581135211. S2CID 1650337.

4. McCann, James; Pollard, Nancy S. (2008). "Real-time gradient-domain painting" (PDF). ACM Transactions on Graphics. 27 (3): 1–7. doi:10.1145/1360612.1360692.

5. Pérez, Patrick; Gangnet, Michel; Blake, Andrew (2003). "Poisson image editing" (PDF). ACM SIGGRAPH 2003 Papers on - SIGGRAPH '03. p. 313. doi:10.1145/1201775.882269. ISBN 1581137095. S2CID 6541990.

3.4 Phụ lục