Project 4: Image Blending

Computational Science

KTHFAN

December 2021

Outline

Introduction

Introduction

The purpose of this project is to stitch multiple images together to generate a single image. However, the edges of the stitched images may not be continuous, making the picture look unnatural.

In this project, laplacian pyramid blending is utilized to make the edges of the stitched images more continuous and smooth.

Models

Models

First, there are N images to be blended, denoted by $I_1, I_2, ..., I_N$. Each image has a corresponding binary mask, denoted by $M_1, M_2, ..., M_N$, which are mutually disjoint, namely $M_i \wedge M_j = \phi$. Therefore, the blended image will be composed of images $I_1, I_2, ..., I_N$ without overlapping. So, the blended image in this stage will be

$$I_{\text{simple}} = \sum_{i=1}^{N} I_i \circ M_i$$

where $\circ$ is element-wise product.

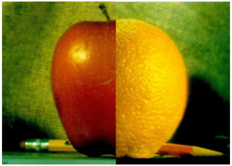


Figure: Example: I_{simple}

Models

Since the result seems terrible, Laplacian Pyramid Blending was introduced in this project to solve this problem.

Knowing that Laplacian Pyramid can reconstruct the entire image without loss of information, we build Laplacian Pyramid for the images $I_1, I_2, ..., I_N$, denoted by $L_i = \{L_i^{(1)}, L_i^{(2)}, ..., L_i^{(k)}, L_i^{(k+1)}\}$. Then we can reconstruct image I_i by

$$I_i = L_i^{(k+1)} + \sum_{j=1}^{k} \text{upsample}(L_i^{(j)})$$

Models

In order to construct a blended image, apply each mask $M_1, M_2, ..., M_N$ to the corresponding Laplacian Pyramid $L_1, L_2, ..., L_N$. By the reconstruct method mentioned in former paragraph, we are able to construct the blended image:

$$I_{\text{Laplacian}} = \sum_{i=1}^{N} (L_i^{(k+1)} \circ M_i) + \sum_{j=1}^{k} \text{upsample}(L_i^{(j)} \circ M_i)$$

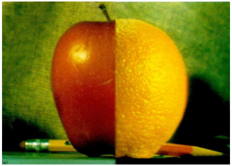


Figure: Example: $I_{\text{Laplacian}}$

Models

After constructing the blended image by Laplacian Pyramid, we found that the edge still not smooth enough. Therefore, the solution in this project is to "blur" the mask, by apply convolution on the mask with a blur filter F_{blur} defined by

$$F_{\text{blur}} = \frac{1}{p^4} \begin{bmatrix} 1 \\ 2 \\ \vdots \\ p-1 \\ p \\ p-1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & \dots & p-1 & p & p-1 & \dots & 1 \end{bmatrix}$$

Models

The filter F_{blur} is a $(2p-1)\times(2p-1)$ matrix where p determines the degree of blur.

Figure: Apply the blur mask

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Figure: F_{blur} with $p = 2$

Models

Therefore, we can use the blurred masks again, and then apply blurred masks on the corresponding Laplacian Pyramids, sum the masked Laplacian Pyramids and get the blended image.

Figure: Example: blended image with $p = 50$

Experiments

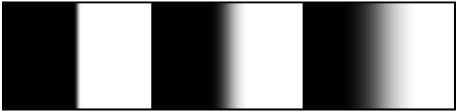
Experiments

With the proper masks, we can do any stitching of images. The following examples are image blending using different masks.

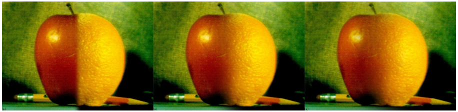
$\Downarrow$

Experiments

As p increase, the blur filter F_{blur} become larger, therefore the edge will be looked more continuous. The following examples are images blending with $p = 20, 100, 200$.



$\Downarrow$



Summary

Summary

- Build Laplacian Pyramids for each images to construct the blended image.
- Apply each mask on the corresponding Laplacian Pyramid to stitch the images.
- Apply blur filter on each mask to make the stitched edges looked continuous.
- The shape of blur filter used in this project is square, but if the stitching edges are irregular, a blur filter with circle shape may achieve better performance.

Appendix: Matlab Code

```
%% load images
img_0 = imread('https://cs.brown.edu/courses/csci1290/labs/lab_compositing/img/burt_apple.png');
img_1 = imread('https://cs.brown.edu/courses/csci1290/labs/lab_compositing/img/burt_orange.png');
[M, N, ~] = size(img_0);
img_1 = imresize(img_1, [M, N]);
img_0 = double(img_0) / 255;
img_1 = double(img_1) / 255;

%% blend with different masks
mask = fspecial('gaussian', 9);

v_mask = build_mask({@(x, y) x < N/2, @(x, y) x >= N/2}, [M, N], 50);
diag_mask = build_mask({@(x, y) M*x - N*y < 0, @(x, y) M*x - N*y >= 0}, [M, N], 50);
circ_mask = build_mask({@(x, y) (y-M/2).^2 + (x-N/2).^2 > M^2/25, ...
        @(x, y) (y-M/2).^2 + (x-N/2).^2 <= M^2/25}, [M, N], 50);

v_blend = blend_images({img_0, img_1}, v_mask, mask, 10);
diag_blend = blend_images({img_0, img_1}, diag_mask, mask, 10);
circ_blend = blend_images({img_0, img_1}, circ_mask, mask, 10);

figure; imshow([v_blend, diag_blend, circ_blend]);

%% blend mask with N
mask = fspecial('gaussian', 9);

v_mask20 = build_mask({@(x, y) x < N/2, @(x, y) x >= N/2}, [M, N], 20);
v_mask100 = build_mask({@(x, y) x < N/2, @(x, y) x >= N/2}, [M, N], 100);
v_mask200 = build_mask({@(x, y) x < N/2, @(x, y) x >= N/2}, [M, N], 200);

v_blend20 = blend_images({img_0, img_1}, v_mask20, mask, 10);
v_blend100 = blend_images({img_0, img_1}, v_mask100, mask, 10);
v_blend200 = blend_images({img_0, img_1}, v_mask200, mask, 10);

figure; imshow([v_blend20, v_blend100, v_blend200]);
```

```
function I = downsample_image(I, shape)
  [H, W, ~] = size(I);
  h = floor(shape(1)); w = floor(shape(2));
  Ws = 1:W; Hs = 1:H;
  rw = mod(W, w); rh = mod(H, h);
  mw = floor(W / w); mh = floor(H / h);

  if mh >= 2
    math = 1:H-rh;
    math((1:mh:H-rh) + floor(mh/2)) = [];

    Hs(math) = [];
  end
  if mw >= 2
    matw = 1:W-rw;
    matw((1:mw:W-rw) + floor(mw/2)) = [];
    Ws(matw) = [];
  end

  if rh ~= 0
    len_Hs = length(Hs);
    mrh = floor(len_Hs/rh);
    del_rh = downsample(1:mrh*rh, mrh, mrh-1) + mod(len_Hs, rh);
    Hs(del_rh) = [];
  end
  if rw ~= 0
    len_Ws = length(Ws);
    mrw = floor(len_Ws/rw);
    del_rw = downsample(1:mrw*rw, mrw, mrw-1) + mod(len_Ws, rw);
    Ws(del_rw) = [];
  end
  I = I(Hs, Ws, :);
end
```

```matlab
function result = gaussian_pyramid(I, step, shape, mask)
  [h, w, c] = size(I);
  shape_list = zeros(2, step);
  step_h = (h - shape(1))/step;
  step_w = (w - shape(2))/step;
  shape_list(1,:) = flip(shape(1):step_h:h-step_h);
  shape_list(2,:) = flip(shape(2):step_w:w-step_w);
  shape_list = floor(shape_list);

  result = cell(step+1, 1);
  result{1} = I;
  for i=1:step
    I = imfilter(I, mask, 'same');
    I = downsample_image(I, [shape_list(1, i), shape_list(2, i)]);
    result{i+1} = I;
  end
  result = flip(result);
end

function result = laplacian_pyramid(U)
  step = length(U);

  result = cell(step, 1);
  result{1} = U{1};
  for i=2:step
    result{i} = U{i} - imresize(U{i-1}, size(U{i}, 1, 2));
  end
end
```

```
% Image blending for images.
% Args:
%   imgs  {cell<MxNx3 matrix>} Images for blending.
%   masks {cell<matrix>} Masks for each imgs.
%   f     {matrix}         Conv filter.
%   level {number}  Level of the blending.
% Returns:
%   I {matrix} Blended image.
function I = blend_images(imgs, masks, f, level)
  n_images = length(imgs);
  L_list = cell(n_images, 1);
  [M, N, ~] = size(imgs{1});
  I = zeros(M/level, N/level, 3);

  for i=1:n_images
    U = gaussian_pyramid(imgs{i}, level, [M/level, N/level], f);
    L = laplacian_pyramid(U);
    L_list{i} = L;
    I = I + imresize(masks{i}, size(I, 1, 2)) .* L{1};
  end
  for j=2:level+1
    I = imresize(I, size(L_list{1}{j}, 1, 2));
    for i=1:n_images
      I = I + imresize(masks{i}, size(I, 1, 2)) .* L_list{i}{j};
    end
  end
end
```

```
% Returns the mask(weight) for image blending
% Args:
%   mask_fun {cell<fun(x, y)>} Functions return boolean to determine mask.
%   shape    {matrix}
%   blur_size{number} Size for the blur conv filter.
% Returns:
%   masks {cell<matrix>}
function masks = build_mask(mask_fun, shape, blur_size)
  M = shape(1); N = shape(2);
  [Y, X] = ndgrid(1:M, 1:N);

  masks = cell(length(mask_fun), 1);
  blur_mask = blur_filter(blur_size);

  for i=1:length(mask_fun)
    mask = double(mask_fun{i}(X, Y));
    mask = imfilter(mask, blur_mask, 'replicate');
    masks{i} = mask;
  end
end

% Returns the blur filter
function blur_mask = blur_filter(blur_size)
  blur_mask = 1:blur_size;
  blur_mask = [blur_mask, flip(blur_mask(1:blur_size-1))];
  blur_mask = blur_mask' * blur_mask;
  blur_mask = blur_mask / blur_size.^4;
end
```