Seam Carving Algorithm

Bài viết gốc: Seam Carving Algorithm - K. Lykov Blog

Giới thiệu

Seam carving là một thuật toán dùng để thay đổi kích thước hình ảnh, nó được giới thiệu trong bài báo cáo khoa học của S. Avidan & A. Shamir. Trong bài báo, việc thay đổi kích thước ảnh được thực hiện bằng cách loại bỏ đi các điểm ảnh ít quan trọng và giữ lại các điểm ảnh quan trọng. Bức ảnh dưới đây là minh họa điều này (ảnh bên trên là ảnh gốc với kích thước 332x480 và ảnh bên dưới là ảnh sau khi áp dụng thuật toán seam carving đẻ thu nhỏ còn lại kích thước là 272x400).

Thuật toán này khá phổ biến nên có thể dễ dàng tìm thấy rất nhiều bài viết nói về nó. Tuy nhiên hầu hết đa số các tác giả đã không đọc bài báo cáo ban đầu và chỉ cung cấp cách cài đặt thuật toán khá cơ bản. Trong bài viết này tôi sẽ mô tả thuật toán đầy đủ các chi tiết như trong bài viết của Avidan & Shamir, dưới góc nhìn của một lập trình viên. Ở đây ta sẽ sử dụng matlab để cài đặt thuật toán. Phần chứng minh cụ thể các bạn xem ở phần tham khảo.

Năng lượng (Energy)

Để đơn giản, bài viết này chỉ tập trung nói về việc làm giảm kích thước hình ảnh. Tuy nhiên việc làm tăng kích thước hình ảnh cũng có thể làm tương tự, và sẽ được mô tả sơ qua ở phần sau. Ý tưởng chính của thuật toán là việc loại bỏ các nội dung có ít ý nghĩa đối với người sử dụng (chứa ít thông tin). Ta gọi thông tin này là **Năng lượng** (Energy). Vì vậy ta cần định nghĩa hàm năng lượng để tính năng lượng điểm ảnh từ các điểm ảnh của ảnh gốc. Ví dụ, ở đây ta có thể tính năng lượng của ảnh thông qua đạo hàm của từng điểm ảnh theo các hướng:

$$e_1 = \left|\frac{\delta I}{\delta x}\right| + \left|\frac{\delta I}{\delta y}\right|.$$

Nếu như ảnh có 3 kênh màu thì ta lấy tổng giá trị năng lượng của 3 kênh này lại với nhau. Đoạn code Matlab dưới đây sẽ mô tả quá trình tính. Hàm `imfilter` được áp dụng cho các điểm ảnh được đánh dấu, do đó kết quả là

$$dI(i,j)/dx = I(i+1) - I(i-1)/dx \text{ với } dx = 1.$$

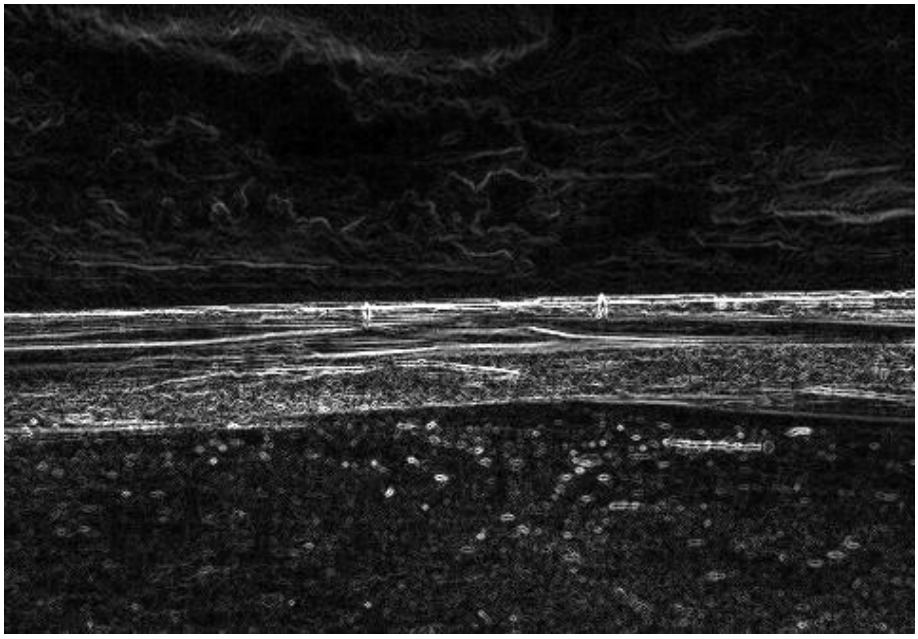
Tương tự cho $dI(i,j)/dy$:

$$dI(i,j)/dy = I(j+1) - I(j-1)/dy \text{ với } dy = 1.$$

```
function res = energyRGB(I)
% Input: Ảnh màu (3 kênh màu)
% Output: Một mảng 2 chiều thể hiện năng lượng của các điểm trong ảnh.
% e = |dI/dx| + |dI/dy|
% Vì ảnh có 3 kênh màu nên ta trả ra tổng năng lượng theo 3 kênh màu
    res = energyGrey(I(:, :, 1)) + energyGrey(I(:, :, 2)) + energyGrey(I(:, :, 3));

function res = energyGrey(I)
% Input: Ảnh đen trắng
% Output: Một mảng 2 chiều thể hiện năng lượng của các điểm trong ảnh.
% e = |dI/dx| + |dI/dy|
    res = abs(imfilter(I, [-1,0,1], 'replicate')) + abs(imfilter(I, [-1;0;1], 'replicate'));
end
```

Năng lượng thu được:



Seam

Nếu chúng ta xóa đi các điểm ảnh có nặng lượng thấp nhất ở các vị trí ngẫu nhiên, ta sẽ ra một hình ảnh méo mó. Nếu chúng ta xóa theo cột hoặc hàng với năng lượng tối thiểu, ta sẽ nhận được một bức ảnh hoàn chỉnh được thu nhỏ kích thước lại. Ở đây cột j nghĩa là tập hợp *{(i, j) với j cố định}* và một hàng i nghĩa là tập hợp *{(i, j) với i cố định}*.

Thuật toán Seam Carving xóa các hàng và cột tổng quát (được gọi là đường seam). Cụ thể hơn, gọi I là một bức ảnh có kích thước $n * m$, một đường seam dọc là $(s^x)i = (i, x(i))s.t. \forall i, |x(i) - x(i-1)| \leq 1$ trong đó $x[1..n] \rightarrow [1..m]$. Nói một cách dễ hiểu hơn, một đường seam dọc (**vertical seam**) là một đường đi từ biên trên của bức ảnh xuống biên dưới của bức ảnh với độ dài đường đi bằng chiều cao của bức ảnh, và với mỗi phần vị trí (i, j) của đường seam, ta có thể đi tiếp đến các phần tử $(i+1, j-1)$, $(i+1, j)$, $(i+1, j+1)$. Tương tự ta cũng có thể định nghĩa cho đường seam ngang (**horizontal seam**). Ví dụ về các đường màu đen là các đường seam trong hình dưới đây.

Chúng ta sẽ tìm kiếm một đường seam sao cho có tổng giá trị năng lượng là nhỏ nhất (theo chiều chúng ta chọn): $s^* = [\min_s \sum_{i=1}^{n} e(I(s_i))]$. Cách để tìm được kết quả tối ưu cho bài toàn là sử dụng phương pháp quy hoạch động.

1. Tìm đường seam tối ưu từ biên trên của ảnh đến mỗi điểm ảnh (i, j).
    - Gọi $M[i, j]$ là giá trị năng lượng nhỏ nhất đi từ biên trên của ảnh đến điểm ảnh (i, j).
    - $M[1, j] = e(1, j)$ với $e(i, j)$ là năng lượng điểm ảnh tại (i, j).
    - $M[i, j] = min(M[i - 1, j - 1], M[i - 1, j], M[i - 1, j + 1]) + e(i, j)$.
2. Ở biên dưới của ảnh, ta tìm điểm đường seam tối ưu (tổng giá trị năng lượng thấp nhất thông qua bảng phương án M) và đi ngược về để tìm đường đi tối ưu.

Lưu ý: trong đoạn code dưới đây trả về một ma trận $n * m$ chỉ gồm 0 và 1 với các điểm ảnh trên đường đi seam sẽ có giá trị là 0 và ngược lại. Để tìm đường seam ngang, ta chỉ cần chuyển vị ma trận năng lượng lại.

```
function [optSeamMask, seamEnergy] = findOptSeam(energy)
% Input: mảng 2 chiều là năng lượng của các điểm ảnh
% Output: Đường seam dọc tối ưu & năng lượng
% Mảng optSeamMask gồm 0/1, với 0 thể hiện điểm đó thuộc đường seam
% Để tìm đường seam ngang tối ưu, cho Input là ma trận chuyển vị

    % Tính mảng quy hoạch động M cho các đường seam dọc
    M = padarray(energy, [0 1], realmax('double')); % M = mảng energy thêm 2 cột có giá trị cực đại ở đầu và

    sz = size(M);
    for i = 2 : sz(1)
        for j = 2 : (sz(2) - 1)
            neighbors = [M(i - 1, j - 1) M(i - 1, j) M(i - 1, j + 1)];
            M(i, j) = M(i, j) + min(neighbors);
        end
    end

    % Tìm phần tử nhỏ nhất hàng cuối
    [val, indJ] = min(M(sz(1), :));
    seamEnergy = val;

    optSeamMask = zeros(size(energy), 'uint8');

    % Đi ngược lại và truy vết
    for i = sz(1) : -1 : 2
        % optSeam(i) = indJ - 1;
        optSeamMask(i, indJ - 1) = 1; % -1 vì lúc đầu ta thêm một cột 0 vào bên trái
        neighbors = [M(i - 1, indJ - 1) M(i - 1, indJ) M(i - 1, indJ + 1)];
        [val, indIncr] = min(neighbors);

        seamEnergy = seamEnergy + val;

        indJ = indJ + (indIncr - 2); % (x - 2): [1,2]->[-1,1]]
    end

    optSeamMask(1, indJ - 1) = 1; % -1 vì lúc đầu ta thêm một cột 0 vào bên trái
```

```
    optSeamMask = ~optSeamMask;
end
```

Tìm phương án tối ưu để xóa đường seam

Bây giờ ta có thể tính toán ra được đường seam và sử dụng đoạn code dưới đây, ta có thể loại bỏ đường seam ra khỏi bức ảnh.

```
function imageReduced = reduceImageByMaskVertical(image, seamMask)
    % Input: Ảnh & mask của đường seam
    % Output: Ảnh sau khi xóa đường seam dọc
    imageReduced = zeros(size(image, 1), size(image, 2) - 1, size(image, 3));
    for i = 1 : size(seamMask, 1)
        imageReduced(i, :, 1) = image(i, seamMask(i, :), 1);
        imageReduced(i, :, 2) = image(i, seamMask(i, :), 2);
        imageReduced(i, :, 3) = image(i, seamMask(i, :), 3);
    end
end

function imageReduced = reduceImageByMaskHorizontal(image, seamMask)
    % Input: Ảnh & mask của đường seam
    % Output: Ảnh sau khi xóa đường seam ngang
    imageReduced = zeros(size(image, 1) - 1, size(image, 2), size(image, 3));
    for j = 1 : size(seamMask, 2)
        imageReduced(:, j, 1) = image(seamMask(:, j), j, 1);
        imageReduced(:, j, 2) = image(seamMask(:, j), j, 2);
        imageReduced(:, j, 3) = image(seamMask(:, j), j, 3);
    end
end
```

Đây là một thuật toán hiệu quả để làm giám kích thước ảnh theo một chiều - chỉ cần việc tìm và xóa các đường seam nhiều lần như bạn cần. Nhưng nếu làm giảm kích thước theo cả hai chiều, ta cần phải làm như thế nào? Làm sao để quyết định rằng ở mỗi lần lắp đưa ra quyết định là xóa theo dòng hay cột sẽ tốt hơn? Vấn đề này một lần nữa được giải quyết bằng quy hoạch động. Ta gọi $T(i, j)$ là giá trị năng lượng thấp nhất khi ta loại bỏ i đường seam theo chiều dọc và j đường seam theo chiều ngang. Cụ thể:
$T(i, j) = min(T(i, j - 1) + E(seamVertical), T(i - 1, j) + E(seamHorizontal))$. Trong đó $E(seamVertical)$ là giá trị nhỏ nhất (tối ưu) đường seam dọc loại bỏ đi, $E(seamHorizontal)$ là giá trị nhỏ nhất (tối ưu) đường seam ngang loại bỏ đi. Ta sử dụng thêm một mảng $transBitMask(i, j)$ lưu truy vết đường đi cho bản phương án $T(i, j)$. $transBitMask(i, j) = 1$ bỏ đi đường seam dọc, $transBitMask(i, j) = 0$ bỏ đi đường seam ngang. Nhìn một đoạn code giả dưới đây để có thể dễ hình dung.

```
1) T(0, 0) = 0;
2) Khởi tạo T:
   for all j {
       T(0, j) = T(0, j - 1) + E(seamVertical);
   }
   for all i {
```

```
        T(i, 0) = T(j - 1, 0) + E(seamHorizontal);
    }
3) Initialize borders of TransBitMask (TBM):
   for all j { TBM(0, j) = 1; }
   for all i { TBM(0, i) = 0; }
4) Tính T và TBM:
   for i = 2 to r {
       imageWithoutRow = image;
       for j = 2 to c {
            energy = computeEnergy(imageWithoutRow);

            horizontalSeamEnergy = findHorizontalSeamEnergy(energy);
            verticalSeamEnergy = findVerticalSeamEnergy(energy);
            tVertical = T(i - 1, j) + verticalSeamEnergy;
            tHorizontal = T(i, j - 1) _ horizontalSeamEnergy;
            if (tVertical < tHorizontal) {
                T(i, j) = tVertical;
                TBM(i, j) = 1
            } else {
                T(i, j) = tHorizontal;
                TBM(i, j) = 0
            }
            // Xóa đường seam dọc
            imageWithoutRow = removeVerticalSeam(energy);
        }

        energy = computeEnergy(image);
        image = removeHorizontalSeam(energy);
    }

5) Truy vết theo T và TBM.
```

Đoạn code bằng matlab. Chú ý ở pseduocode dùng zerobased index, do matlab sử dụng onebased index nên cần phải đẩy index lên 1 đơn vị.

```
function [T, transBitMask] = findTransportMatrix(sizeReduction, image)
% Input: Kích thước cần giảm & ảnh gốc
% Output: T, TBM định nghĩa ở trên

    T = zeros(sizeReduction(1) + 1, sizeReduction(2) + 1, 'double');
    transBitMask = ones(size(T)) * -1;

    % Khởi tạo T(i, 1), T(1, i), TBM(i, 1), TBM(1, i)
    imageNoRow = image;
    for i = 2 : size(T, 1)
        % Tính năng lượng
        energy = energyRGB(imageNoRow);
        % Tìm đường seam ngang tối ưu
        [optSeamMask, seamEnergyRow] = findOptSeam(energy');
        % Xóa đường seam
        imageNoRow = reduceImageByMask(imageNoRow, optSeamMask, 0);

        % Tính T và TBM
        T(i, 1) = T(i - 1, 1) + seamEnergyRow;
```

```
    transBitMask(i, 1) = 0;
end;

imageNoColumn = image;
for j = 2 : size(T, 2)
    % Tính năng lượng
    energy = energyRGB(imageNoColumn);
    % Tìm đường seam dọc
    [optSeamMask, seamEnergyColumn] = findOptSeam(energy);
    % Xóa đường seam dọc
    imageNoColumn = reduceImageByMask(imageNoColumn, optSeamMask, 1);

    % Tính TBM & T
    transBitMask(1, j) = 1;
    T(1, j) = T(1, j - 1) + seamEnergyColumn;
end;

% Xóa 1 hàng và 1 cột
energy = energyRGB(image);
[optSeamMask, seamEnergyRow] = findOptSeam(energy');
image = reduceImageByMask(image, optSeamMask, 0);

energy = energyRGB(image);
[optSeamMask, seamEnergyColumn] = findOptSeam(energy);
image = reduceImageByMask(image, optSeamMask, 1);

% fill in internal part
for i = 2 : size(T, 1)

    imageWithoutRow = image; % Ta sẽ xóa 1 hàng của imageWithoutRow

    for j = 2 : size(T, 2)
        energy = energyRGB(imageWithoutRow);

        [optSeamMaskRow, seamEnergyRow] = findOptSeam(energy');
        imageNoRow = reduceImageByMask(imageWithoutRow, optSeamMaskRow, 0);

        [optSeamMaskColumn, seamEnergyColumn] = findOptSeam(energy);
        imageNoColumn = reduceImageByMask(imageWithoutRow, optSeamMaskColumn, 1);

        neighbors = [(T(i - 1, j) + seamEnergyRow) (T(i, j - 1) + seamEnergyColumn)];
        [val, ind] = min(neighbors);

        T(i, j) = val;
        transBitMask(i, j) = ind - 1;

        % Ta xóa lần lượt từng cột
        imageWithoutRow = imageNoColumn;
    end;

    energy = energyRGB(image);
    [optSeamMaskRow, seamEnergyRow] = findOptSeam(energy');
     % move from top to bottom
    image = reduceImageByMask(image, optSeamMaskRow, 0);
end;
```

```
end
```

Phóng to hình ảnh

Để phóng to hình ảnh, thay vì ta loại bỏ đường seam ra khỏi ảnh, thì ta thêm một đường seam mới vào với giá trị trung bình của các điểm ảnh lận cận.

```
function imageEnlarged = enlargeImageByMaskVertical(image, seamMask)
% Input: Ảnh và đường seam
% Output: Ảnh đã được phóng to thêm 1 cột

    avg = @(image, i, j, k) (image(i, j-1, k) + image(i, j+1, k))/2;

    imageEnlarged = zeros(size(image, 1), size(image, 2) + 1, size(image, 3));
    for i = 1 : size(seamMask, 1)
        j = find(seamMask(i, :) ~= 1);
        imageEnlarged(i, :, 1) = [image(i, 1:j, 1), avg(image, i, j, 1), image(i, j+1:end, 1)];
        imageEnlarged(i, :, 2) = [image(i, 1:j, 2), avg(image, i, j, 2), image(i, j+1:end, 2)];
        imageEnlarged(i, :, 3) = [image(i, 1:j, 3), avg(image, i, j, 3), image(i, j+1:end, 3)];
    end;
end

function imageEnlarged = enlargeImageByMaskHorizontal(image, seamMask)
% Input: Ảnh và đường seam
% Output: Ảnh đã được phóng to thêm 1 hàng

    avg = @(image, i, j, k) (image(i-1, j, k) + image(i+1, j, k))/2;

    imageEnlarged = zeros(size(image, 1) + 1, size(image, 2), size(image, 3));
    for j = 1 : size(seamMask, 2)
        i = find(seamMask(:, j) ~= 1);
        imageEnlarged(:, j, 1) = [image(1:i, j, 1); avg(image, i, j, 1); image(i+1:end, j, 1)];
        imageEnlarged(:, j, 2) = [image(1:i, j, 2); avg(image, i, j, 2); image(i+1:end, j, 2)];
        imageEnlarged(:, j, 3) = [image(1:i, j, 3); avg(image, i, j, 3); image(i+1:end, j, 3)];
    end;
end
```

Source code

Dưới đây là toàn bộ code của tác giả (giữ nguyên lại comment gốc bằng tiếng Anh)

```
% (C) Copyright Kirill Lykov 2013.
%
% Distributed under the FreeBSD Software License (See accompanying file license.txt)

function image = seamCarving(newSize, image)
% apply seam carving to the image
% following paper by Avidan and Shamir '07
```

```
    sizeReductionX = size(image, 1) - newSize(1);
    sizeReductionY = size(image, 2) - newSize(2);

    mmax = @(left, right) max([left right]);

    image = seamCarvingReduce([mmax(0, sizeReductionX), mmax(0, sizeReductionY)], image);

    image = seamCarvingEnlarge([mmax(0, -sizeReductionX), mmax(0, -sizeReductionY)], image);
end

function image = seamCarvingReduce(sizeReduction, image)
    if (sizeReduction == 0)
        return;
    end;
    [T, transBitMask] = findTransportMatrix(sizeReduction, image);
    image = addOrDeleteSeams(transBitMask, sizeReduction, image, @reduceImageByMask);
end

% TODO Bug: enlarge gives artifacts althout I chouse different seams as described
% in 4.3 in the paper
function image = seamCarvingEnlarge(sizeEnlarge, image)
    if (sizeEnlarge == 0)
        return;
    end;
    [T, transBitMask] = findTransportMatrix(sizeEnlarge, image);
    image = addOrDeleteSeams(transBitMask, sizeEnlarge, image, @enlargeImageByMask);
end

function [T, transBitMask] = findTransportMatrix(sizeReduction, image)
% find optimal order of removing raws and columns

    T = zeros(sizeReduction(1) + 1, sizeReduction(2) + 1, 'double');
    transBitMask = ones(size(T)) * -1;

    % fill in borders
    imageNoRow = image;
    for i = 2 : size(T, 1)
        energy = energyRGB(imageNoRow);
        [optSeamMask, seamEnergyRow] = findOptSeam(energy');
        imageNoRow = reduceImageByMask(imageNoRow, optSeamMask, 0);
        transBitMask(i, 1) = 0;

        T(i, 1) = T(i - 1, 1) + seamEnergyRow;
    end;

    imageNoColumn = image;
    for j = 2 : size(T, 2)
        energy = energyRGB(imageNoColumn);
        [optSeamMask, seamEnergyColumn] = findOptSeam(energy);
        imageNoColumn = reduceImageByMask(imageNoColumn, optSeamMask, 1);
        transBitMask(1, j) = 1;

        T(1, j) = T(1, j - 1) + seamEnergyColumn;
    end;
```

```
    % on the borders, just remove one column and one row before proceeding
    energy = energyRGB(image);
    [optSeamMask, seamEnergyRow] = findOptSeam(energy');
    image = reduceImageByMask(image, optSeamMask, 0);

    energy = energyRGB(image);
    [optSeamMask, seamEnergyColumn] = findOptSeam(energy);
    image = reduceImageByMask(image, optSeamMask, 1);

    % fill in internal part
    for i = 2 : size(T, 1)

        imageWithoutRow = image; % copy for deleting columns

        for j = 2 : size(T, 2)
            energy = energyRGB(imageWithoutRow);

            [optSeamMaskRow, seamEnergyRow] = findOptSeam(energy');
            imageNoRow = reduceImageByMask(imageWithoutRow, optSeamMaskRow, 0);

            [optSeamMaskColumn, seamEnergyColumn] = findOptSeam(energy);
            imageNoColumn = reduceImageByMask(imageWithoutRow, optSeamMaskColumn, 1);

            neighbors = [(T(i - 1, j) + seamEnergyRow) (T(i, j - 1) + seamEnergyColumn)];
            [val, ind] = min(neighbors);

            T(i, j) = val;
            transBitMask(i, j) = ind - 1;

            % move from left to right
            imageWithoutRow = imageNoColumn;
        end;

        energy = energyRGB(image);
        [optSeamMaskRow, seamEnergyRow] = findOptSeam(energy');
         % move from top to bottom
        image = reduceImageByMask(image, optSeamMaskRow, 0);
    end;

end

function image = addOrDeleteSeams(transBitMask, sizeReduction, image, operation)
% delete seams following optimal way
    i = size(transBitMask, 1);
    j = size(transBitMask, 2);

    for it = 1 : (sizeReduction(1) + sizeReduction(2))

        energy = energyRGB(image);
        if (transBitMask(i, j) == 0)
            [optSeamMask, seamEnergyRaw] = findOptSeam(energy');
            image = operation(image, optSeamMask, 0);
            i = i - 1;
        else
            [optSeamMask, seamEnergyColumn] = findOptSeam(energy);
```

```
            image = operation(image, optSeamMask, 1);
            j = j - 1;
        end;

    end;
end

function [optSeamMask, seamEnergy] = findOptSeam(energy)
% following paper by Avidan and Shamir `07
% finds optimal seam
% returns mask with 0 mean a pixel is in the seam

    % find M for vertical seams
    % for vertical - use I`
    M = padarray(energy, [0 1], realmax('double')); % to avoid handling border elements

    sz = size(M);
    for i = 2 : sz(1)
        for j = 2 : (sz(2) - 1)
            neighbors = [M(i - 1, j - 1) M(i - 1, j) M(i - 1, j + 1)];
            M(i, j) = M(i, j) + min(neighbors);
        end
    end

    % find the min element in the last raw
    [val, indJ] = min(M(sz(1), :));
    seamEnergy = val;

    %optSeam = zeros(sz(1), 1, 'int32');
    optSeamMask = zeros(size(energy), 'uint8');

    %indJ = indJ - 1; % because of padding on 1 element from left

    %go backward and save (i, j)
    for i = sz(1) : -1 : 2
        %optSeam(i) = indJ - 1;
        optSeamMask(i, indJ - 1) = 1; % -1 because of padding on 1 element from left
        neighbors = [M(i - 1, indJ - 1) M(i - 1, indJ) M(i - 1, indJ + 1)];
        [val, indIncr] = min(neighbors);

        seamEnergy = seamEnergy + val;

        indJ = indJ + (indIncr - 2); % (x - 2): [1,2]->[-1,1]]
    end
    %optSeam(1) = indJ; % to avoid if in loop becuase matlab is slow as hell

    optSeamMask(1, indJ - 1) = 1; % -1 because of padding on 1 element from left
    optSeamMask = ~optSeamMask;

end

function imageReduced = reduceImageByMask( image, seamMask, isVerical )
% removes pixels by input mask
% removes vertical line if isVerical == 1, otherwise horizontal
    if (isVerical)
```

```
        imageReduced = reduceImageByMaskVertical(image, seamMask);
    else
        imageReduced = reduceImageByMaskHorizontal(image, seamMask');
    end;
end

% could not find a more elegant way to do it
function imageReduced = reduceImageByMaskVertical(image, seamMask)
    imageReduced = zeros(size(image, 1), size(image, 2) - 1, size(image, 3));
    for i = 1 : size(seamMask, 1)
        imageReduced(i, :, 1) = image(i, seamMask(i, :), 1);
        imageReduced(i, :, 2) = image(i, seamMask(i, :), 2);
        imageReduced(i, :, 3) = image(i, seamMask(i, :), 3);
    end
end

function imageReduced = reduceImageByMaskHorizontal(image, seamMask)
    imageReduced = zeros(size(image, 1) - 1, size(image, 2), size(image, 3));
    for j = 1 : size(seamMask, 2)
        imageReduced(:, j, 1) = image(seamMask(:, j), j, 1);
        imageReduced(:, j, 2) = image(seamMask(:, j), j, 2);
        imageReduced(:, j, 3) = image(seamMask(:, j), j, 3);
    end
end

function imageEnlarged = enlargeImageByMask(image, seamMask, isVerical)
% removes pixels by input mask
% removes vertical line if isVerical == 1, otherwise horizontal
    if (isVerical)
        imageEnlarged = enlargeImageByMaskVertical(image, seamMask);
    else
        imageEnlarged = enlargeImageByMaskHorizontal(image, seamMask');
    end;
end

function imageEnlarged = enlargeImageByMaskVertical(image, seamMask)

    avg = @(image, i, j, k) (image(i, j-1, k) + image(i, j+1, k))/2;

    imageEnlarged = zeros(size(image, 1), size(image, 2) + 1, size(image, 3));
    for i = 1 : size(seamMask, 1)
        j = find(seamMask(i, :) ~= 1);
        imageEnlarged(i, :, 1) = [image(i, 1:j, 1), avg(image, i, j, 1), image(i, j+1:end, 1)];
        imageEnlarged(i, :, 2) = [image(i, 1:j, 2), avg(image, i, j, 2), image(i, j+1:end, 2)];
        imageEnlarged(i, :, 3) = [image(i, 1:j, 3), avg(image, i, j, 3), image(i, j+1:end, 3)];
    end;
end

function imageEnlarged = enlargeImageByMaskHorizontal(image, seamMask)

    avg = @(image, i, j, k) (image(i-1, j, k) + image(i+1, j, k))/2;

    imageEnlarged = zeros(size(image, 1) + 1, size(image, 2), size(image, 3));
    for j = 1 : size(seamMask, 2)
        i = find(seamMask(:, j) ~= 1);
```

```
        imageEnlarged(:, j, 1) = [image(1:i, j, 1); avg(image, i, j, 1); image(i+1:end, j, 1)];
        imageEnlarged(:, j, 2) = [image(1:i, j, 2); avg(image, i, j, 2); image(i+1:end, j, 2)];
        imageEnlarged(:, j, 3) = [image(1:i, j, 3); avg(image, i, j, 3); image(i+1:end, j, 3)];
    end;
end

function res = energyRGB(I)
% returns energy of all pixelels
% e = |dI/dx| + |dI/dy|
    res = energyGrey(I(:, :, 1)) + energyGrey(I(:, :, 2)) + energyGrey(I(:, :, 3));
end

function res = energyGrey(I)
% returns energy of all pixelels
% e = |dI/dx| + |dI/dy|
    res = abs(imfilter(I, [-1,0,1], 'replicate')) + abs(imfilter(I, [-1;0;1], 'replicate'));
end
```

Tham khảo

Wikipedia

Seam Carving for Content-Aware Image Scaling

https://jeremykun.com/2013/03/04/seam-carving-for-content-aware-image-scaling/

http://kirilllykov.github.io/blog/2013/06/06/seam-carving-algorithm/

Last edited by **Trương Thành Đạt**, 2016-08-19 12:03:48