SEAM CARVING FOR IMAGE RESIZING

Abstract

Objective of this project is to understand the paper by Shamir provided to us in the Image processing course, implement the project and if possible propose some changes

VAISHNAV Mohit and BATERIWALA Malav

Under the Supervision of
DÉSIRÉ Sidibé
RASTGOO Mojdeh

Table of Contents

Table of figures:

Objective

Objective of this project is to understand the paper by Shamir provided to us in the Image processing course, implement the project and if possible propose some changes. Basic idea of this seam carving is to have the content aware image resizing technique. Here the author had proposed to change the image dimension by just removing the parts which are not required or are of less importance. Hence even after the resizing, not all the section of the image is distorted but just some. Criterion for selection and method are described in the sections below.

Introduction

Images have always been one of the major elements of the media world, typically remaining of rigid size and it cannot resize to fit different layouts automatically. Traditional image resizing, or scaling is not sufficient because it's not dependent on the image content and typically can be only applied in a specific manner or uniformly. Cropping is only limited to change or remove pixels from the periphery of the image. Better and effective resizing can only be achieved by considering the content of the image and not boundary limits. Seams can be either horizontal or vertical and it is a connected path of low energy pixel of any image. A vertical seam is basically the path of pixels connected from top to bottom in an image with one pixel in each row. A horizontal seam is also similar to the vertical seam but with the difference of the connection being from one column to another. The importance or energy functions values of each pixel are measured with it contrast to its adjacent pixels.

For any simple image the term seam carving means to change the size of an image by using an algorithm for carving out or inserting pixels in different parts of the image. For the method of image reduction, seam selection makes sure that for preserving the image structure, we only remove the low energy pixels and less of the higher energy ones. For increasing the size of image or image enlarging, the order of seam insertion ensures that the balance between the original image content and the artificially and duplicated inserted or added pixels

Our optimized content-aware image resizing algorithm starts from getting the energy function on the original image. Then we use our algorithm to make the seams which needs to be carved from the image. After each seam is removed or added according to their energy in the rows and columns, we scale the current image to the target size and then we compute the size of the original image. The resized image with the given input size is the final result. Combining seam carving algorithm with scaling can protect the complete image visual effectively and some particular structures or shapes of the original image, even when the output size or resolution is much lower than the input one. For image size enlarging, we followed an automatic seam selection algorithm to replace the manual method increasing the size pattern. Rather than manually specifying the number of seams to be duplicated, our algorithm can automatically find the feasible seams to insert into the output image. We then show the final output by applying proposed initial size for indirect image resizing. Our input function only takes a few parameters which can be given according to the structure of the image for most examples and which produces less visible change to the result quality.

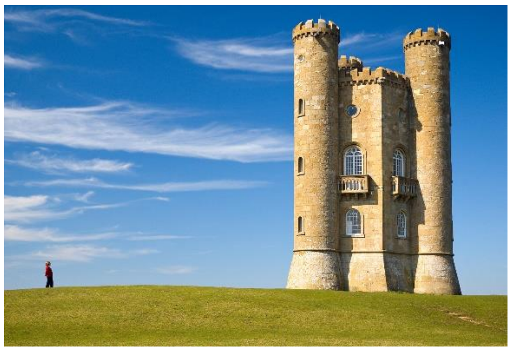


Figure 1: Universally famous image

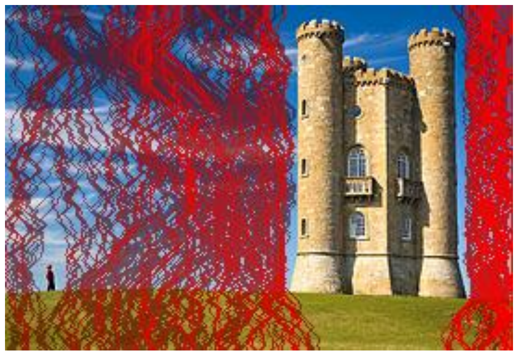


Figure 2: Various seams of that image

Seam carving till now remains important tool for image resizing, but it has its limitations. High level features such as face detection is a way to improve the seam carving algorithm, but only some work has been done to improve the use of low-level features of the image. Moreover, the detection of such high-level features may fail in practice examples and we thus cause the entire image to fail its content. Still, high energy features can be determined with any seam carving algorithm and the one we used to be no exception. So, as we know the energy value of the middle part of an image is not usually as high as the edges and it might be carved after some seam carving seams. So, by doing this an important feature of the whole image may be disturbed, changed or distorted. Here are some examples of it.

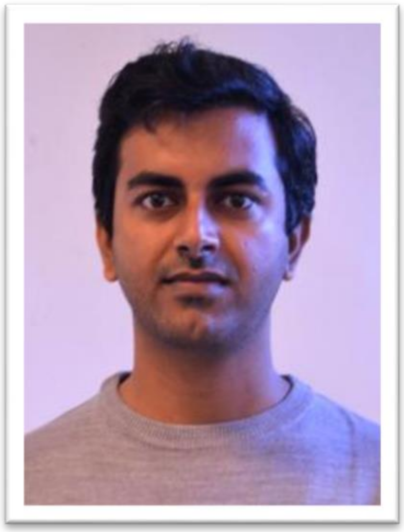


Figure 3: Size 240 x 320

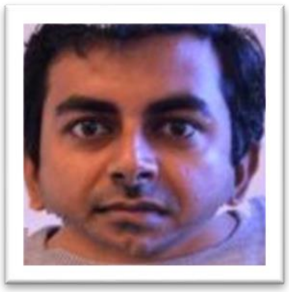


Figure 4: Size 150 x 150 Distorted Image

Application of seam carving are:

- Aspect ratio change,
- Image retargeting,
- Image content enhancement,
- Object removal.

Methodology

Basic algorithm follows the following procedure:

- Calculation of the weight/density/energy of each pixel.
    - Various techniques which could be used are: gradient magnitude, entropy, visual saliency, eye-gaze movement.
    - In our implementation we have used gradient magnitude which gives us good results.
- After the energy gradient is estimated we have to generate the seams which are known to be the part of image less important. To calculate the seams, we have used the dynamic programing.
- After we have found seams it could be used for various applications such as decrement, increment, object removal, image amplification.

Now we are describing each step in the following section:

Energy Map

For carrying out any operation of seam carving technique basically we require “energy” value i.e. each pixel value has to be converted into energy value. To find the energy map, gradient operator is the simplest way which calculates considering both perpendicular and horizontal direction. The energy function is defined for each pixel as follows:

We have used imfilter to implement the gradient operator working both ways in horizontal and vertical direction then adding all those value to get the energy map. Figure has been shown below to represent the output of the algorithm used. If the image is of three channels we just used the same strategy for each of the channel and add them.

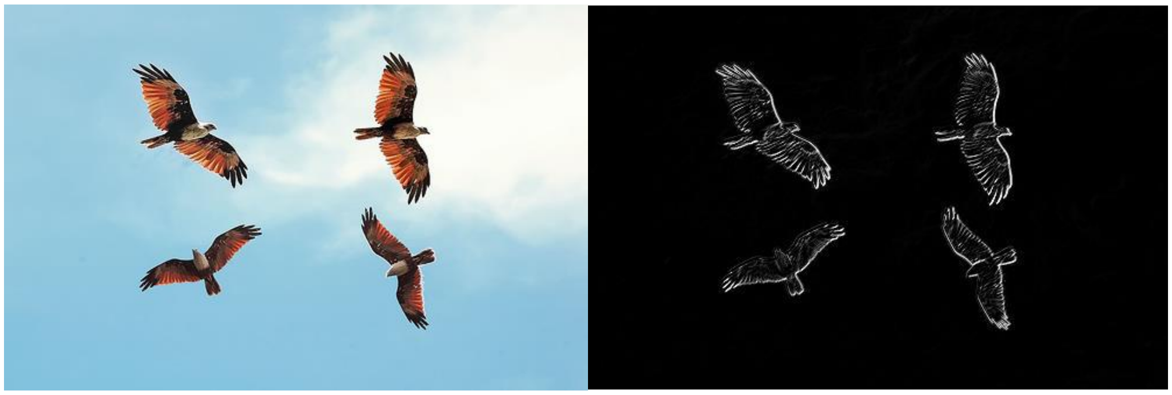


Figure 5: Energy of the input image

We use the following simple energy function which was introduced in the paper:

$$e_1(\mathbf{I}) = |\frac{\partial}{\partial x}\mathbf{I}| + |\frac{\partial}{\partial y}\mathbf{I}|$$

Seam:

It is senseless to delete the pixels at random location thereby distorting the whole of image so for overcoming this barrier technique was named as seam carving. A seam is the line having the minimum energy values. To find the seam dynamic approach is widely prevalent which looks for either horizontal or vertical path.

With the estimated energy map of an image, one can start from the second row in the image and move row-by-row in a dynamic programming fashion updating a cumulative energy matrix M with the same size as the image. In particular, we compute each (i, j) entry in M as follows for vertical and horizontal seams:

- Vertical Seams

$$\mathbf{s}^{\mathbf{x}} = \{s_i^x\}_{i=1}^{n} = \{(x(i), i)\}_{i=1}^{n}, \text{ s.t. } \forall i, |x(i) - x(i-1)| \leq 1,$$

- Horizontal Seams

$$\mathbf{s}^{\mathbf{y}} = \{s_j^y\}_{j=1}^{m} = \{(j, y(j))\}_{j=1}^{m}, \text{ s.t. } \forall j |y(j) - y(j-1)| \leq 1$$

After doing the same for whole of the image where the values are saved in a separate matrix, we have to look for the least value found in the energy function. From this location we have to traverse the reverse path enlisting each separate seam. Starting from the last stop in the previous step we have to look for above three neighbors following equation above.

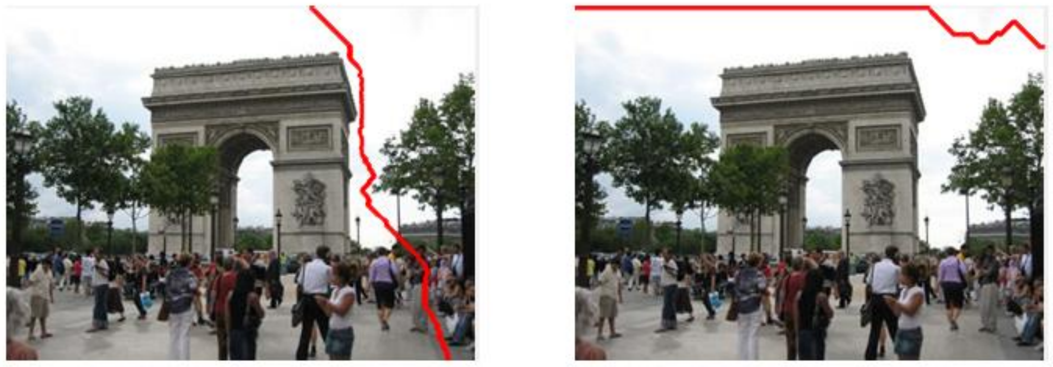


Figure 6: Example of vertical and horizontal seams

In terms of coding:

“S” is the seam matrix, “M” is the energy matrix. Seam matrix is a bit matrix where if the pixel is in the seam it is represented as 0 otherwise 1. After this to find the optimal seam, function “findoptimalSeam” has been created.

Vertical and Horizontal Seam Removal

Now that we know what a seam looks like, and given that we have the energy of each pixel calculated, we already find the seam with the lowest energy to remove. We will only tackle vertical seam removal, and everything mentioned in this section can be applied to the horizontal approach. Initially, we have to consider every possible path (that takes one pixel from each row) from the starting row of pixels to the ending row. In order to perform this, a backtracking approach is not practical (because of the huge number of such paths possible) and we have to go through a dynamic programming approach.

Removing the pixels of a seam from an image has only a local effect and all the pixels of the image are shifted left (or up) to compensate for the missing path. One can replace the constraint $|x(i)−x(i−1)| \leqq 1$ with $|x(i)−x(i−1)| \leqq k$, and get either a simple column (or row) for k = 0, a piecewise connected or even completely disconnected set of pixels for any value $1 \leqq k \leqq m$.

Given an energy function E, we can define the cost of a seam as:

$$E(\mathbf{s}) = E(\mathbf{I_s}) = \sum_{i=1}^{n} e(\mathbf{I}(s_i))$$

Now we look for the optimal seam s* that minimizes this seam cost:

$$s^* = \min_{\mathbf{s}} E(\mathbf{s}) = \min_{\mathbf{s}} \sum_{i=1}^{n} e(\mathbf{I}(s_i))$$

The above optimal seam can be found using dynamic programming.

Object Removal Method

To either preserve or remove the object from the image we have to increase or decrease the energy value of the particular region. We have created user interface to allow user to select the rectangle region which has to be preserved or deleted.

The use of an energy function allows one to create custom metrics for selecting the seams. So, suppose that we'd like to protect a region of an image from being carved, or if we'd like to specify a region for removal. This can easily be done by altering the energy function of those specific regions. Recall that the seam carving algorithm selects the seam with the lowest energy. Thus, in the case of object removal, we just modify the energy of the object pixels to be very low (large negative number). For object protection, we just modify the energy of the object pixels to be very large (large positive number). This way, the algorithm will select/avoid the appropriate seams for the job. Figure 8 showcases the result of seam carving on two regions; green is protected and red is removal.

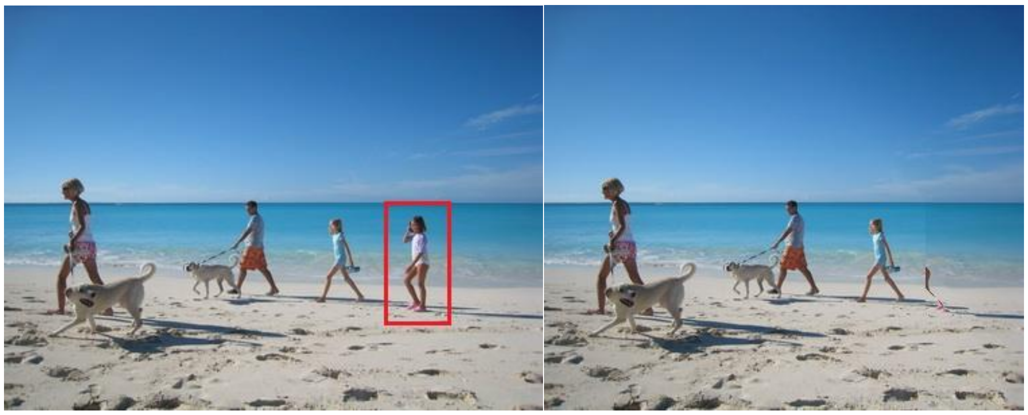


Figure 7: Object removal via decreasing energy of the selected region

Enlarging:

Enlarging an image is a simple process. Consider the case where we would like to increase the width or height by one. First, we identify the optimal seam as discussed above (vertical or horizontal). Then we duplicate this seam or do the averaging of its left or right neighbors. To increase the size by more than one, we just identify the k optimal seams for removal and duplicate them by averaging.

Amplification:

To implement the amplification part we first enlarged the image to some particular value let's say 150% then using the technique of seam carving we again got back to the original shape of the image. This method is simpler to implement along with retaining most of the information saved.

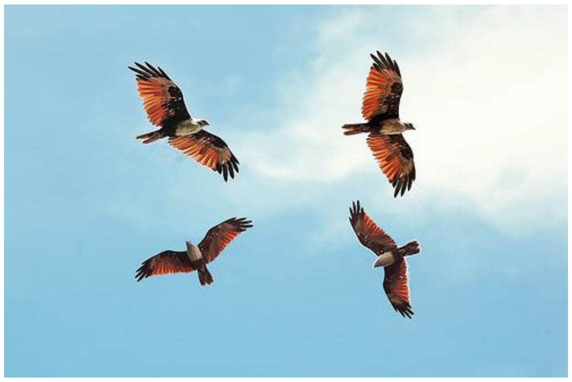


Figure 8: Original Image

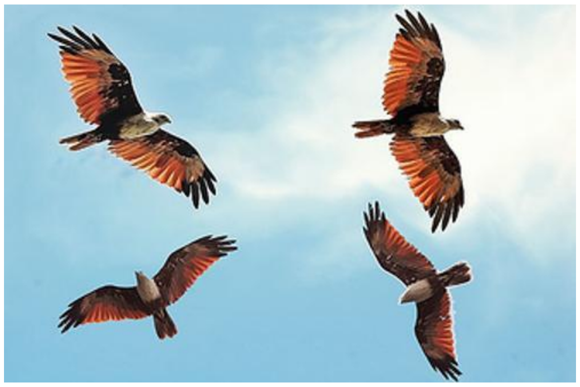


Figure 9: Amplified Image

Results

Artistic examples for image resizing:

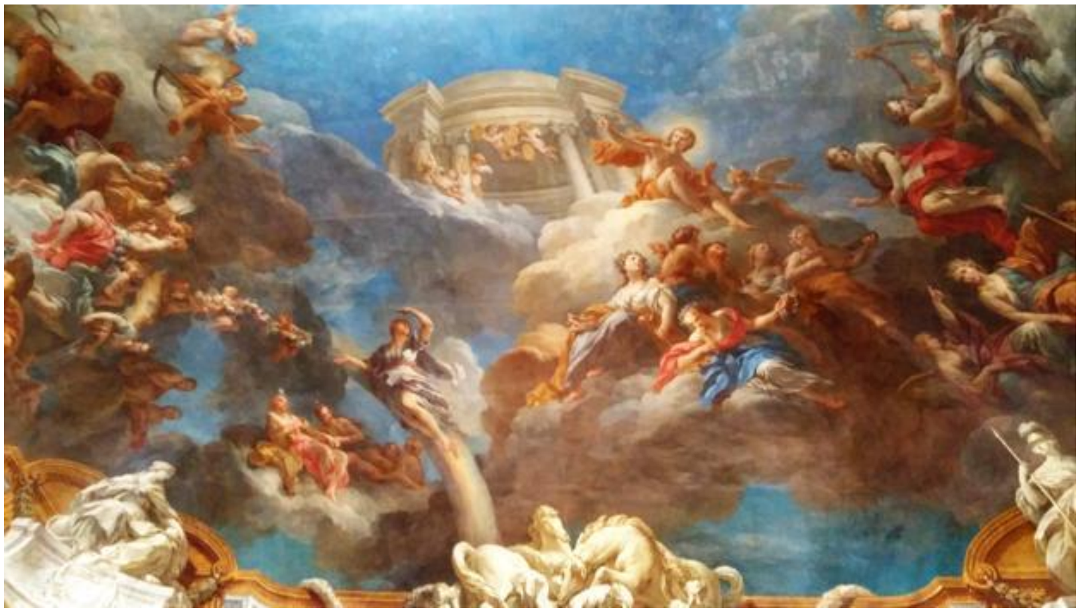


Figure 10: Versailles Roof painting

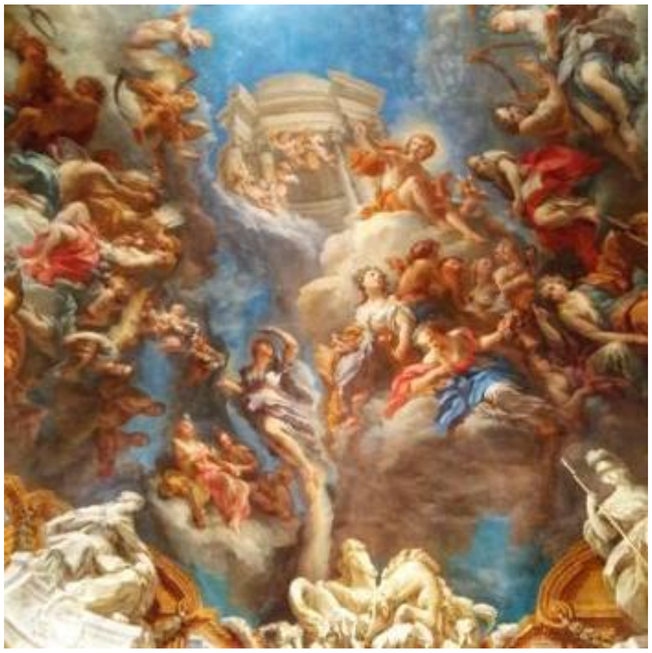


Figure 11: Resizing it to 350 x 350

Image Deformation example:

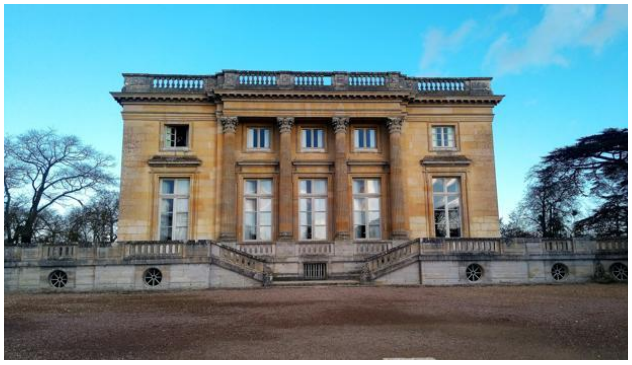


Figure 12: Another Versaillies Example

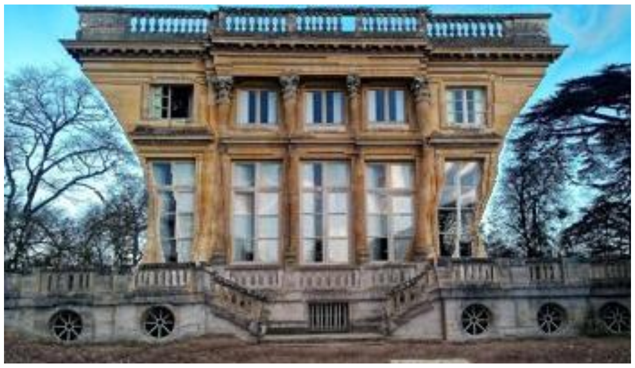


Figure 13: Size of 200 x 350

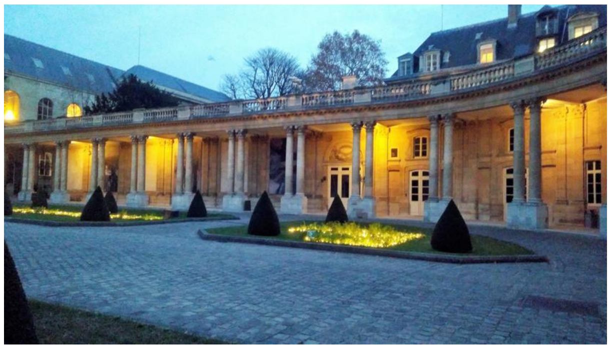


Figure 14: Architecture Museum of Paris

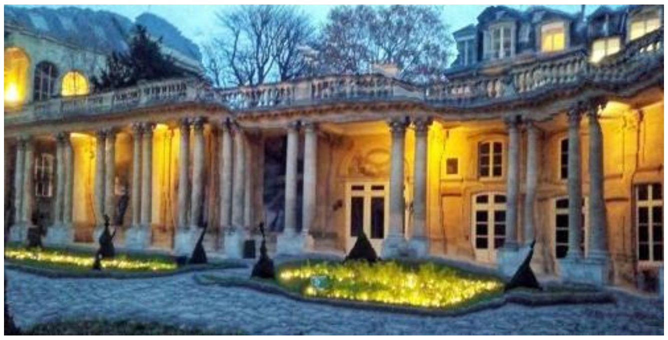


Figure 15: Size changes to 500 x 250

Blurring of High Information Image:

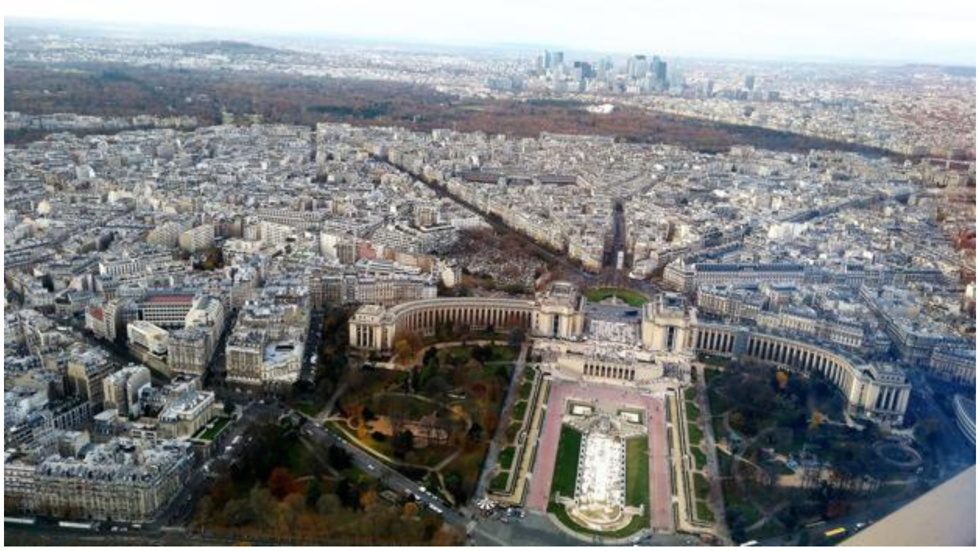


Figure 16: View from top of Eiffel Tower

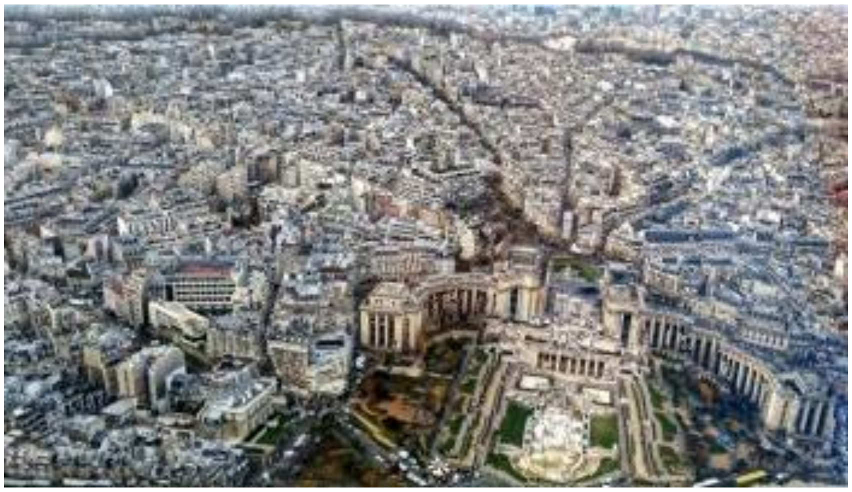
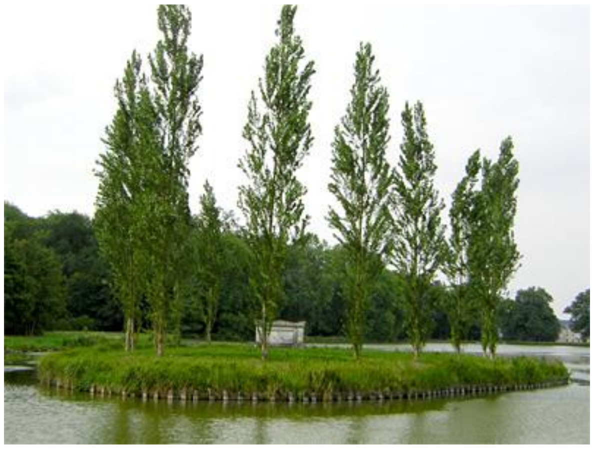
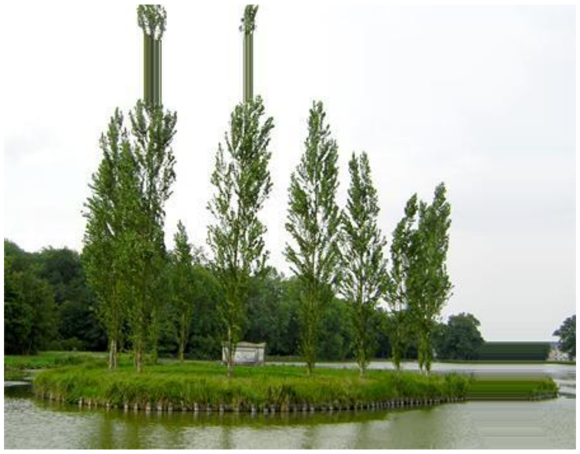


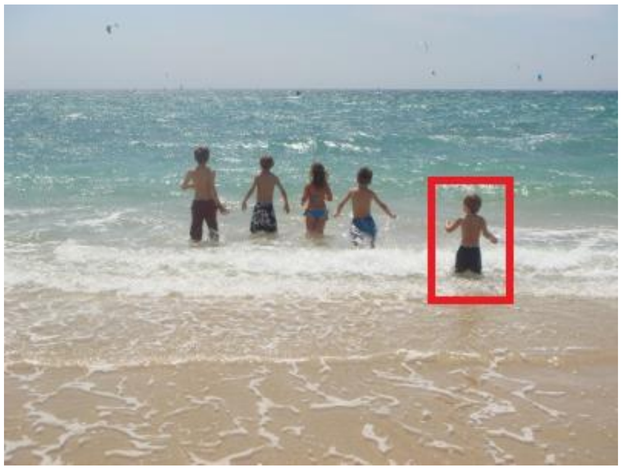
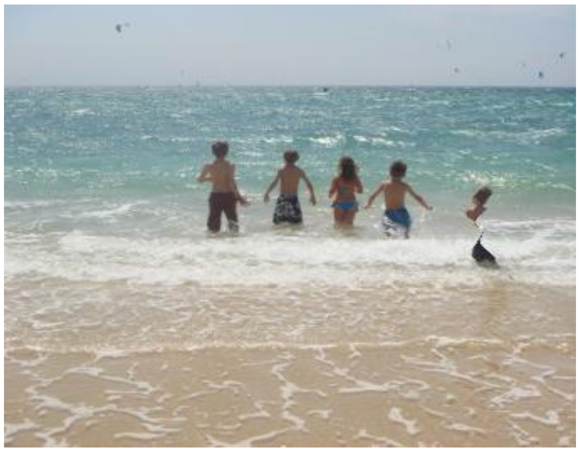
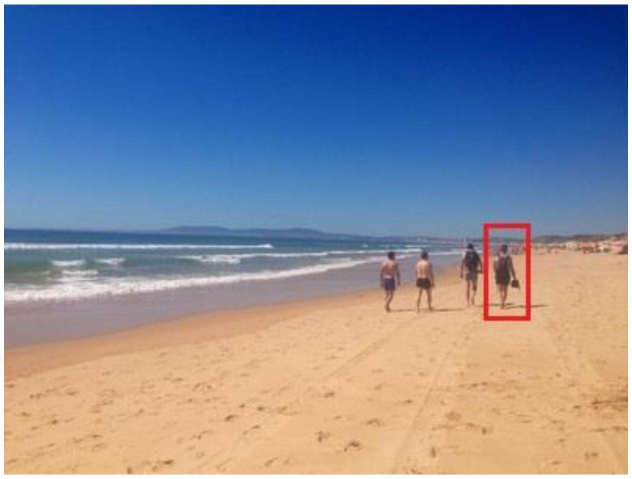
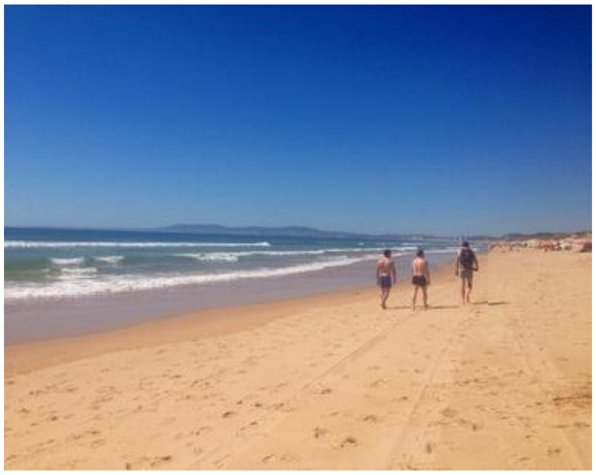
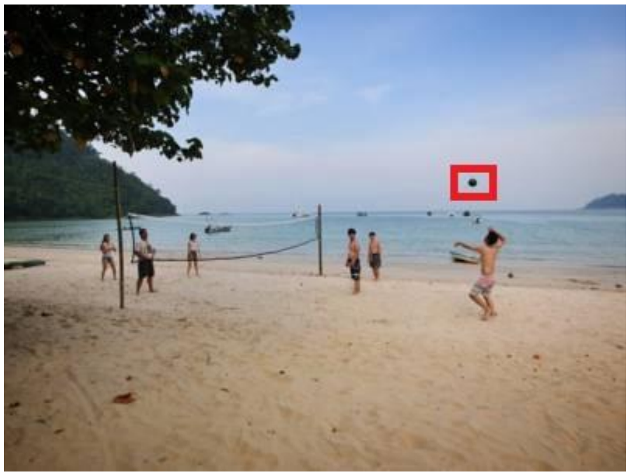
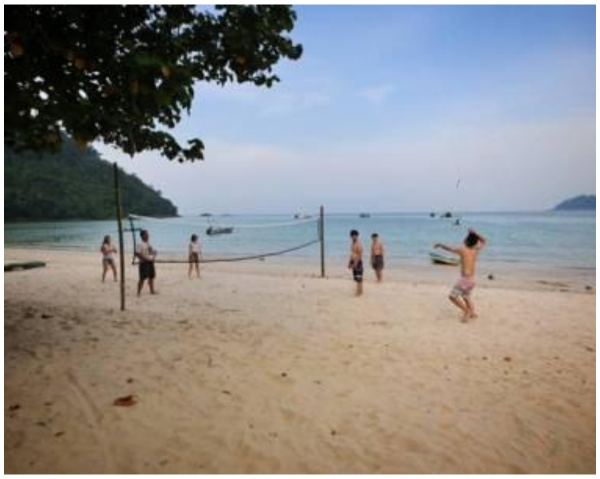
Figure 17: Image resized to 200 x 350

Size increment:





Object removal examples:

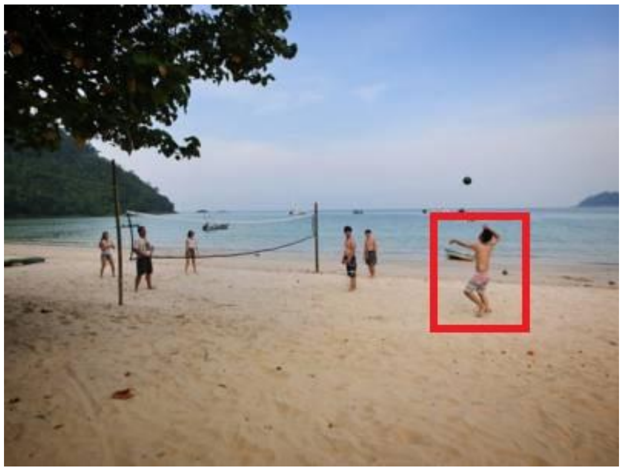
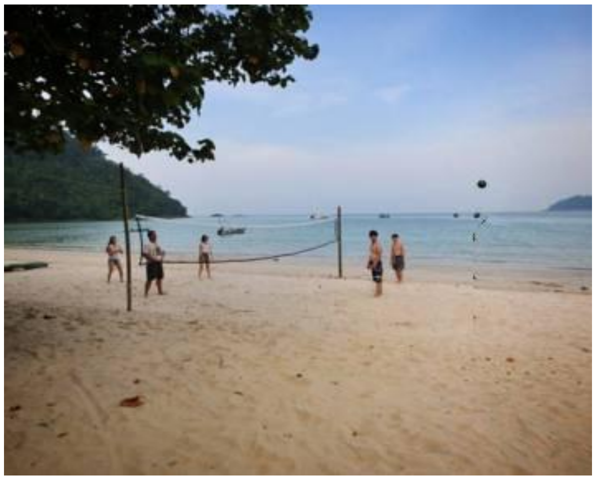



Image Amplification:

Conclusion and Contributions:

Our main task in this project was to create a fully functional implementation of the Seam Carving technique. Though there are many codes available on internet, but we believe that neither all the functionality has been implemented nor the code is optimized. Our main contribution to the code was in terms of boosting the runtime of the code by dividing the seam calculations separately into horizontal and vertical directions.

On the part of object removal, we tried to create an interface where the user can himself/herself select the object to be removed and then save the result in the chosen directory. For the image of dimension ranging from ~[200 300] our code takes even less than a minute while working on the core2duo processor with MATLAB version 17.

We have also improved the readability of the code by creating different function for each of the usable forms which can be called anytime easing the debugging too.

One major part which we feel worth mentioning is that about the documentation of code, we have clearly mentioned comments for each of the codes and their usability for the future reference and implementation. Command line input was asked from the user informing him about the present scenario of the dimension of the image was done in terms of creating the interactive nature of the program.

No one algorithm to resize images is the best option for every image. As we have seen in this paper, traditional seam carving works well on landscapes and images with enough background to carve out. The distortion from seam carving typically occurs in images with faces or images with too much foreground. One can think of a strategy to tackle this situation where they will segregate this information and pass on to the algorithm and suggest an optimal aspect ratio thereby preventing the distortion because of seam carving.

Bibliography

- AVIDAN, S., AND SHAMIR, A. 2007. Seam carving for content-aware image resizing. ACM Trans. Graph. 26, 3, 10.
- CHEN, L., XIE, X., FAN, X., MA, W., ZHANG, H., AND ZHOU, H. 2003. A visual attention model for adapting images on small displays.
- ACM Multimedia Systems Journal 9, 4, 353–364. CHO, T. S., BUTMAN, M., AVIDAN, S., AND FREEMAN, W. T. 2008.
- EL-ALFY, H., JACOBS, D., AND DAVIS, L. 2007. Multi-scale video cropping. In MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia, ACM, New York, NY, USA, 97–106.
- LI, J., AND LU, B.-L. 2009. An adaptive image euclidean distance. Pattern Recogn. 42, 3, 349–357. LIU, H., XIE, X., MA, W.-Y., AND ZHANG, H.-J. 2003. Automatic browsing of large pictures on mobile devices. In MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia, ACM, New York, NY, USA, 148– 155.
- MANJUNATH, B. S., OHM, J. R., VASUDEVAN, V. V., AND YAMADA, A. 2001. Color and texture descriptors. Circuits and Systems for Video Technology, IEEE Transactions on 11, 6, 703– 715.
- MANJUNATH, B., SALEMBIER, P., AND SIKORA, T. 2002. Multimedia Content Description Interface. Wiley, Chichester. MIN, R., AND CHENG, H. D. 2009. Effective image retrieval using dominant color descriptor and fuzzy support vector machine. Pattern Recogn. 42, 1, 147–157.
- PRITCH, Y., KAV-VENAKI, E., AND PELEG, S. 2009. Shift-map image editing. In ICCV 2009: Proceedings of the Twelfth IEEE International Conference on Computer Vision, 721.
- RUBINSTEIN, M., SHAMIR, A., AND AVIDAN, S. 2008. Improved seam carving for video retargeting. ACM Trans. Graph. 27, 3, 16. RUBINSTEIN, M., SHAMIR, A., AND AVIDAN, S. 2009. Multioperator media retargeting. ACM Trans. Graph. 28, 3, 23.
- http://kirilllykov.github.io/blog/2013/06/06/seam-carving-algorithm/

MATLAB Code:

```
clc
close all
clear
tic %For calculating the time of the program run

%Read the image from the user
image = imread('./images/1.jpg');

%Converting image to double
image = im2double(image);
fprintf('Size of original image is: ');
size(image) %Display the size of the image

%User input to enter the new size
% l = input('\n enter the length: ');
% b = input('\n enter the new bredth: ');
l = 250;
b = 500;

newSize = [l b 3]; % apply seam carving to the image
```

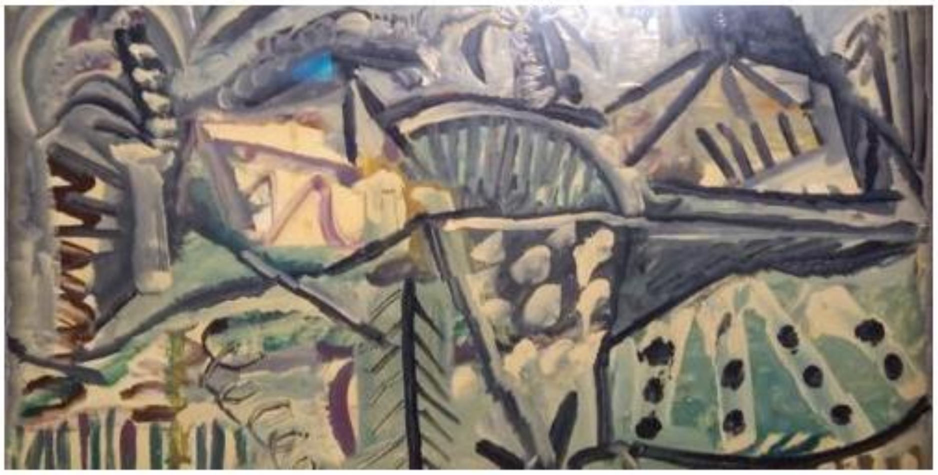
```
Size of original image is:
ans =

   390   693     3
```

```
%Calling the function
image_output = seamCarving(newSize, image);

%Displaying the output and the input for comparison.
figure()
imshow(image)
title('Original Image');

figure()
imshow(image_output)
title('Modified Image');

fprintf('Size of modified image is: ');
size(image_output)
```

```
Size of modified image is:
ans =

   250   500     3
```

Original Image

Modified Image



```
%Time calculation for computing seams
toc
```

```
Elapsed time is 500.922066 seconds.
```

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

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

```
function image = seamCarvingReduce(sizeReduction, image)
    if (sizeReduction == 0)
        return;
    end
    [~, transBitMask] = findTransportMatrix(sizeReduction, image); % find optimal order of
removing rows and columns

    image = addOrDeleteSeams(transBitMask, sizeReduction, image, @reduceImageByMask);
end
```

```
function image = seamCarvingEnlarge(sizeEnlarge, image)
    if (sizeEnlarge == 0)
        return;
    end
    [~, 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, ~] = findOptSeam(energy');
    image = reduceImageByMask(image, optSeamMask, 0);

    energy = energyRGB(image);
    [optSeamMask, ~] = 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, ~] = 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, ~] = findOptSeam(energy');
            image = operation(image, optSeamMask, 0);
            i = i - 1;
        else
```

```
            [optSeamMask, ~] = findOptSeam(energy);
            image = operation(image, optSeamMask, 1);
            j = j - 1;
        end

    end
end
```

```
function [optSeamMask, seamEnergy] = findOptSeam(energy)
% finds optimal seam
% returns mask with 0 mean a pixel is in the seam
    % find M for vertical seams
    % for vertical - use I` ie transpose of the matrix
    M = zeros(size(energy,1),size(energy,2)+2);
    M(:,2:size(energy,2)+1) = energy;
    M(:,1)=1000;
    M(:,size(energy,2)+2)=1000;

    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 row
    [val, indJ] = min(M(sz(1), :));
    seamEnergy = val;

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

    %go backward and save (i, j)
    for i = sz(1) : -1 : 2
        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); % To know the index of the above seam
    end

    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
```

```
function imageReduced = reduceImageByMaskVertical(image, seamMask)
    imageReduced = zeros(size(image, 1), size(image, 2) - 1, size(image, 3));
    %size(image, 2) - 1
    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));
    %size(image, 1) - 1
    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)

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

```
function imageEnlarged = enlargeImageByMaskHorizontal(image, seamMask)

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

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

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

Published with MATLAB® R2017a

Image Enlargement Changes:

```
%%
%Read the image from the user
image2 = imread('./images/16.jpg');
image = imresize(image2, 1.5);

%Converting image to double
image = im2double(image);
fprintf('Size of original image is: ');
size(image2) %Display the size of the image

newSize = size(image2); % apply seam carving to the image
%%
%Calling the function
image_output = seamCarving(newSize, image);
%%
%Displaying the output and the input for comparison.
figure()
imshow(image)
title('Original Image');

figure()
imshow(image_output)
title('Modified Image');

imwrite(image_output,'./images/image_output_16.jpg');
fprintf('Size of modified image is: ');
size(image_output)
```

Object Removal Code:

```
clc
close all
clear
tic
image = im2double(imread('./images/16.jpg'));
imshow(image)
```

```
title('Original Image');

size(image)

rect = getrect;       % get the area from the user which object is to be removed


global xmin
global ymin
global w
global h

xmin = (round( rect(1) ));
ymin = round( rect(2) );
w = round( rect(3) );
h = round( rect(4) );

[l,b,d]=size(image);

if w>h
    newSize = [l-h, b, d]; % apply seam carving to the image
else
    newSize = [l, b-w, d];
end

image_output = seamCarving(newSize, image);
imwrite(image_output,'./images/image_output_161.jpg');
figure()
imshow(image_output)
title('Modified Image');
```

```
toc

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

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

function image = seamCarvingReduce(sizeReduction, image)
    if (sizeReduction == 0)
        return;
    end
    [~, transBitMask] = findTransportMatrix(sizeReduction, image); % find optimal order of
removing rows and columns
```

```
    image = addOrDeleteSeams(transBitMask, sizeReduction, image, @reduceImageByMask);
end

function image = seamCarvingEnlarge(sizeEnlarge, image)
    if (sizeEnlarge == 0)
        return;
    end
    [~, 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, ~] = findOptSeam(energy');
    image = reduceImageByMask(image, optSeamMask, 0);

    energy = energyRGB(image);
    [optSeamMask, ~] = 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, ~] = 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, ~] = findOptSeam(energy');
            image = operation(image, optSeamMask, 0);
            i = i - 1;
        else
            [optSeamMask, ~] = findOptSeam(energy);
            image = operation(image, optSeamMask, 1);
            j = j - 1;
        end

    end
end

function [optSeamMask, seamEnergy] = findOptSeam(energy)
% finds optimal seam
% returns mask with 0 mean a pixel is in the seam
    % find M for vertical seams
    % for vertical - use I` ie transpose of the matrix
    M = zeros(size(energy,1),size(energy,2)+2);
    M(:,2:size(energy,2)+1) = energy;
    M(:,1)=1000;
    M(:,size(energy,2)+2)=1000;

    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 row
    [val, indJ] = min(M(sz(1), :));
    seamEnergy = val;

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

    %go backward and save (i, j)
    for i = sz(1) : -1 : 2
        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); % To know the index of the above seam
    end

    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

function imageReduced = reduceImageByMaskVertical(image, seamMask)
    imageReduced = zeros(size(image, 1), size(image, 2) - 1, size(image, 3));
    %size(image, 2) - 1
%     global w
%     w = w-1;
    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));
    %size(image, 1) - 1
%     global h
%     h = h-1;
    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)

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

function imageEnlarged = enlargeImageByMaskHorizontal(image, seamMask)

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

function res = energyRGB(I)
% returns energy of all pixels
% e = |dI/dx| + |dI/dy|
    global xmin
    global ymin
    global w
    global h

    res = energyGrey(I(:, :, 1)) + energyGrey(I(:, :, 2)) + energyGrey(I(:, :, 3));
    res(ymin:ymin+h,xmin:xmin+w,:)=-10000;

end
```

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

Published with MATLAB® R2017a