10/01/2018

# *Project Report*

# *Seam Carving for Content Aware Resizing*

Image Processing I

MAIA 2017/2018

University of Burgundy

Author(s):

Ali BERRADA

Basel ALYAFI

Kenechukwu Henry NGIGE

Supervisor(s):

Dr. Désiré Sidibé

Ms. Mojdeh Rastgoo

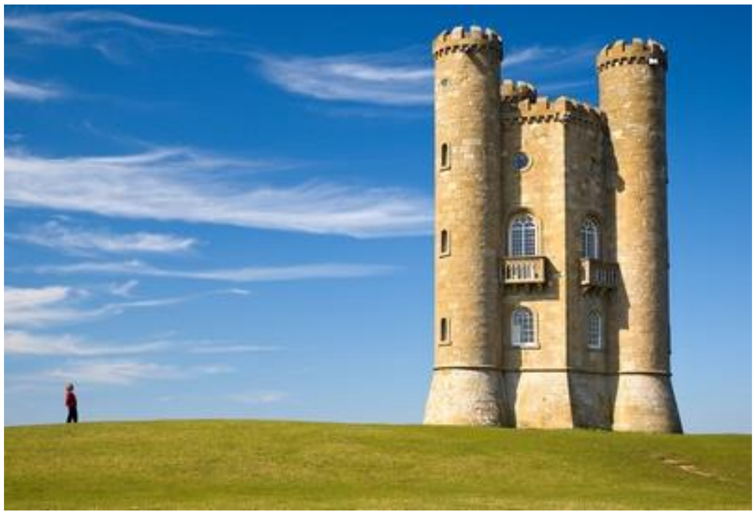
## Introduction

In 2007, Shai Avidan and Ariel Shamir proponed a novel method for intelligently resizing images without deteriorating its highly valued content, baptized “seam-carving”. Their algorithm, as described in their paper “Seam Carving for Content-Aware Imaging”, shows how interesting elements are being preserved while the input image is being reduced or expanded. The result is a user-defined resized image that does not deform the regions of interest. This method finds particularly useful applications in today’s versatile display devices which create the need for effective image retargeting. Beyond that, many image editing software saw an advantage from the attractive capabilities and results of this algorithm and implemented it as part of their out-of-the-box functionalities.

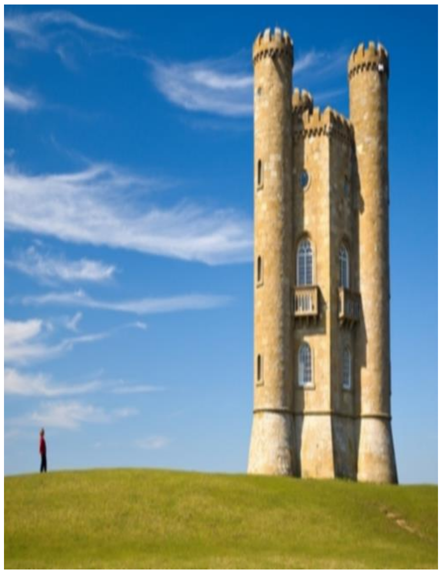
In this report, we will further describe the idea behind seam carving and how we can achieve an implementation of the algorithm as well as our proposed user-friendly software that displays the capabilities of the method. Some small but subtle details concerning the implementation will not be explicitly mentioned in this report, but the actual code is well commented and include these kind of information as needed.

## Problem Description

To appreciate the advent of seam carving, we need to look at why we seek to preserve the image content while resizing. Let us consider for instance the following 400 x 271 pixels image:



Assume there exists a device commonly found in the market and owned by a wide population in the world that has a screen size of 240 x 320. In full screen mode, the image in the device will look like this:



We can notice how the areas that intuitively carry minimal importance such as the sky and the grass look rather OK while the elements that catch the most attention, i.e. the castle and the human, have been heavily affected and shows unrealistic dimensions. In fact, any reasonable owner of that device would want to see the image rather close to the following:

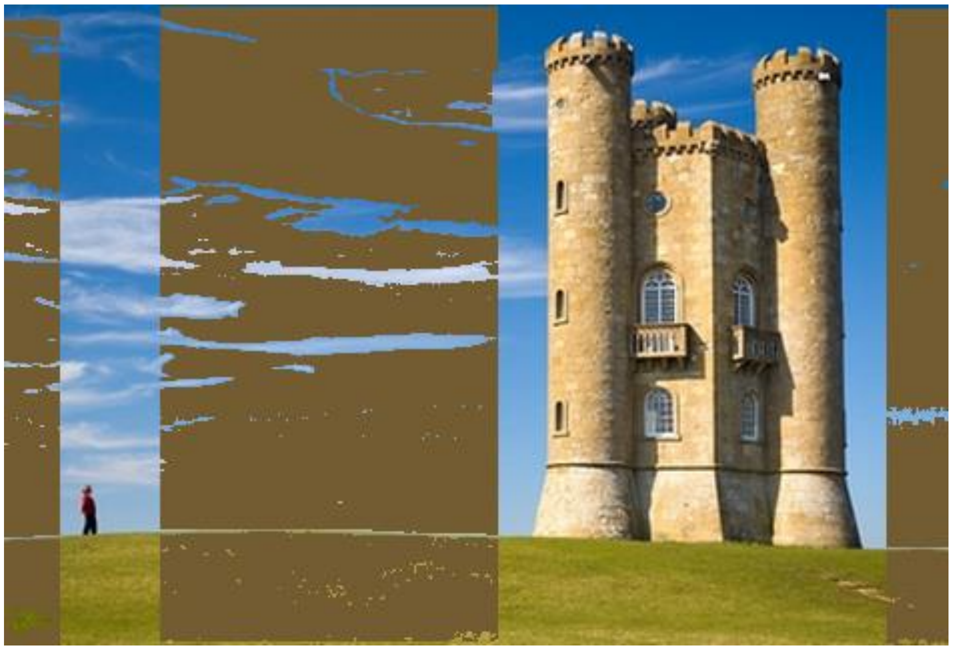
This simple example depicts the usefulness of content aware resizing and this is why we are interested in implementing seam carving.

## Strategy

By now, we understood why we want to keep the important elements of an image untouched when resizing an image. We can also realize that this is done at the expense of less important elements in the image. So, the goal here is to in fact be able to find or determine these elements such that they can be removed, or as the authors would call, carved. Since an image, at the microlevel, is basically a bunch of pixels, the aim is to find paths through these pixels that are drawing the elements or regions we want to remove. These paths are termed in the literature as seams. Therefore, by finding and removing these less interesting seams, we can achieve seam carving. Yet, this algorithm is not meant to solely remove seams, and by that reducing the width and/or height of an image, as its name suggests but it can also be used to expand an image by adding seams horizontally and/or vertically.

Since the approach of reducing and expanding the dimensions of an image is similar, we can first describe the removal of seams in more details before extending the concept to duplicating seams.

Looking back at our sample image, we can visually delimit the regions of less importance we may afford to lose for the sake of reducing the width of the image without having our castle and human ending up skewed:



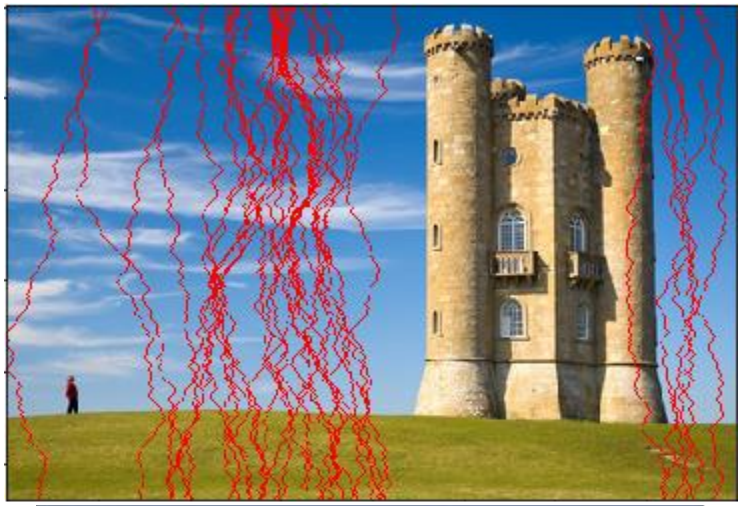
Our algorithm should therefore be able to find a path of connected pixels that starts from the first row and ends at the last row. Each top-to-bottom path represents a vertical seam, and removing a vertical seam comes back to reducing the width dimension by 1. Similarly, a horizontal seam can be defined as a path of connected pixels that extends from the first column to the last column.

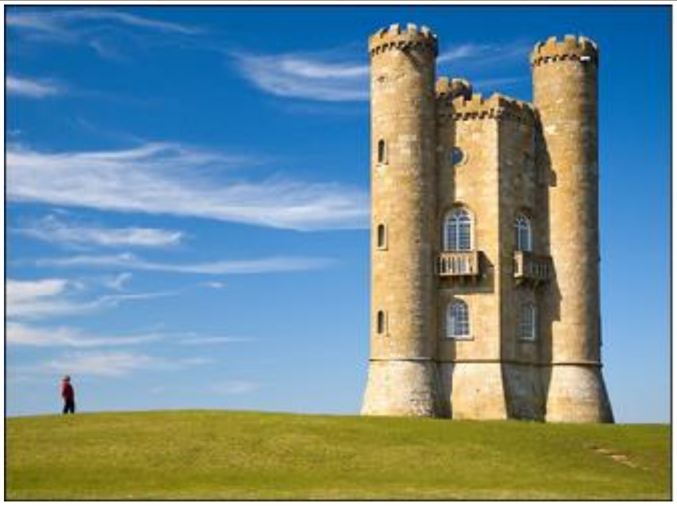
The crucial issue left to address is what mathematical abstraction can our algorithm exploits to “see” these regions of less importance like humans do using their cognitive abilities.

We can achieve this programmatically by computing an energy map that determines the relative energy strength between pixels of the image. Pixels with higher energy should convey that they are pixels of higher importance. Inversely, pixels with lower energy represents regions of lower importance and therefore the seam can be computed by following a path of lowest energy pixels. From mathematics, we know that derivatives represent rates of change. In the field of image processing, the derivative of an image represents the rate of change in intensity. Therefore, the higher the intensity change, the higher the importance, and thus the higher the energy. The Sobel filter is a simple operator that finds the gradient approximation of an image. It has been tested to give good enough results. The Histogram of Oriented Gradient (HOG) can also be used as an alternative energy function.

For computing the seams optimally which would otherwise be computationally expensive, we use dynamic programming because we need to be able to easily backtrack the pixels along the path from bottom-to-top after we found out which vertical seam downwardly accumulates to the lowest energy. Practically, this happens at the expense of memory as we need to maintain a matrix that stores the lowest path to a pixel coming from one of the 3 pixels above it.

Using our image, we can detect a few seams and remove them to get an image with reduced width:

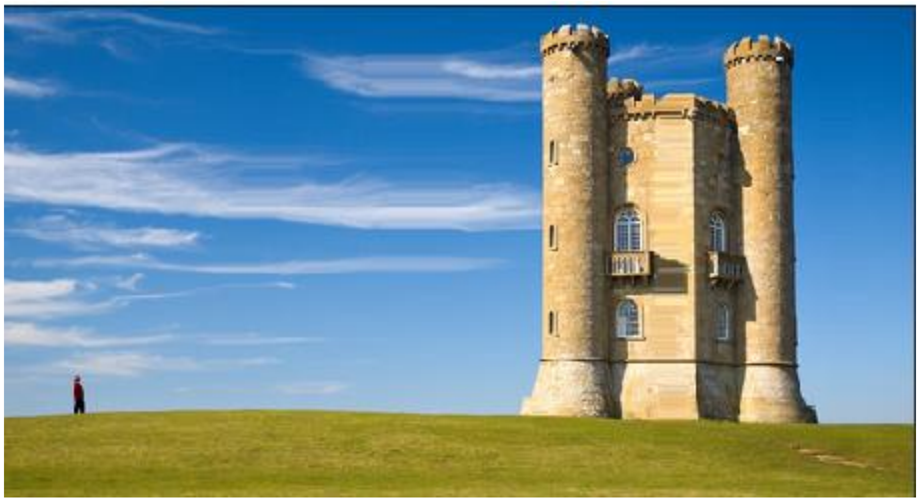




A seam is practically removed by shifting, for every pixel in the seam, all pixels on the right by 1 position to the left. The last column of the image can be then discarded to get the reduced width.

To be noted that we compute each seam at a time, which means that the energy matrix is recomputed as needed. This is meant to achieve the most accurate lowest energy seam computation since after removing a seam from the image, the energy of the image naturally changes and thus the next lowest energy seam may be present at a totally different location.

If instead we duplicate the set of vertical pixels along the seam's path, we will increase the width of the image by 1; therefore, achieving image expansion with the same approach. The below image is an example of image expansion with 100 seams:



It is worth to mention that by adding multiple seams, we may notice artifacts as a result of discrepancy in pixel intensities. In this case, we take the average intensity of the left and right pixels and insert it as a pixel between the two. Also, repeating the steps of computing a seam then inserting it will end up in finding almost the same seam each time because of the averaging and the always present seam (the seam is never removed from the image, it just gets added with some averaging). We solved this issue by not always taking the lowest seam at each iteration but rather taking some $k^{th}$ lowest seam where the k value is closely determined by the loop iteration index.

Finally, we can achieve the computation of horizontal seams – to reduce or increase the height of an image – with the same strategy but this time moving from left-to-right instead of top-to-bottom.

In our program, we used a simple trick which is to flip the image by 90° before feeding it to the algorithm since carving horizontal seams from an image is same to carving vertical seams from the same image rotated by 90°.

Now that we are able to carve and add seams, we can eventually go a step further and achieve object removal. This however, needs user input to delimit the area of the object to be removed. Once done, we know which specific pixels to target for seam carving (to remove the object). The computed energy matrix will then be set to 0 for the pixels of the region selected and so the computed horizontal seams will be certainly passing through this area of extremely low energy. After seams removal, we need to add enough seams to return to the original image size. The result is an image similar to the original one but with the selected object missing.

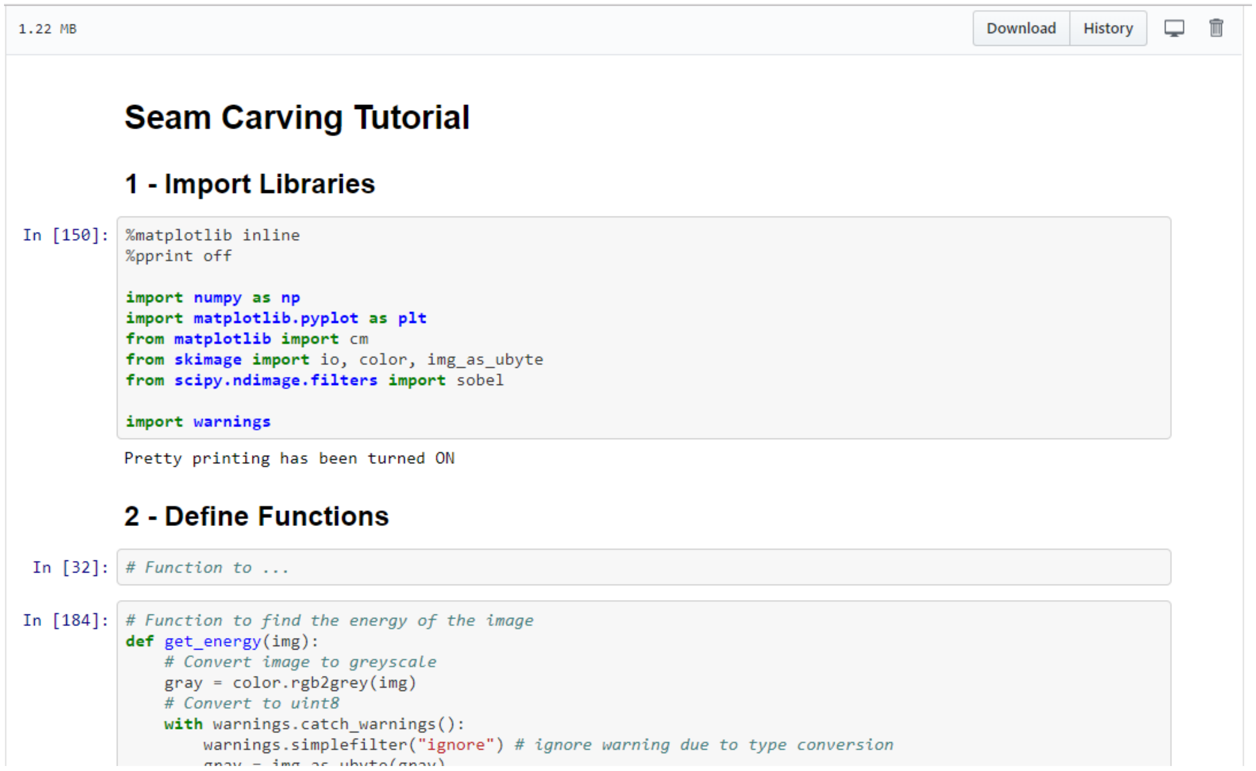
## Software

The program was initially implemented as a script using Python and Skimage. It has been made available as a Jupyter notebook and can therefore be seen with the actual output from the browser. This is more for those who want to learn about the algorithm and see the core of the implementation; therefore, it is clean, neat and well documented. It is located at the following address:

https://github.com/ali-yar/maia-imageprocessing/blob/master/Project/seam_carving.ipynb

Additionally, we provide a GUI application for the user to perform content aware resizing of images by simply providing the input image and then entering the desired new height and width to obtain a reduced or expanded image.

## Notebook Screenshot



1.22 MB

Download History

# Seam Carving Tutorial

## 1 - Import Libraries

In [150]:
```
%matplotlib inline
%pprint off

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from skimage import io, color, img_as_ubyte
from scipy.ndimage.filters import sobel

import warnings
```

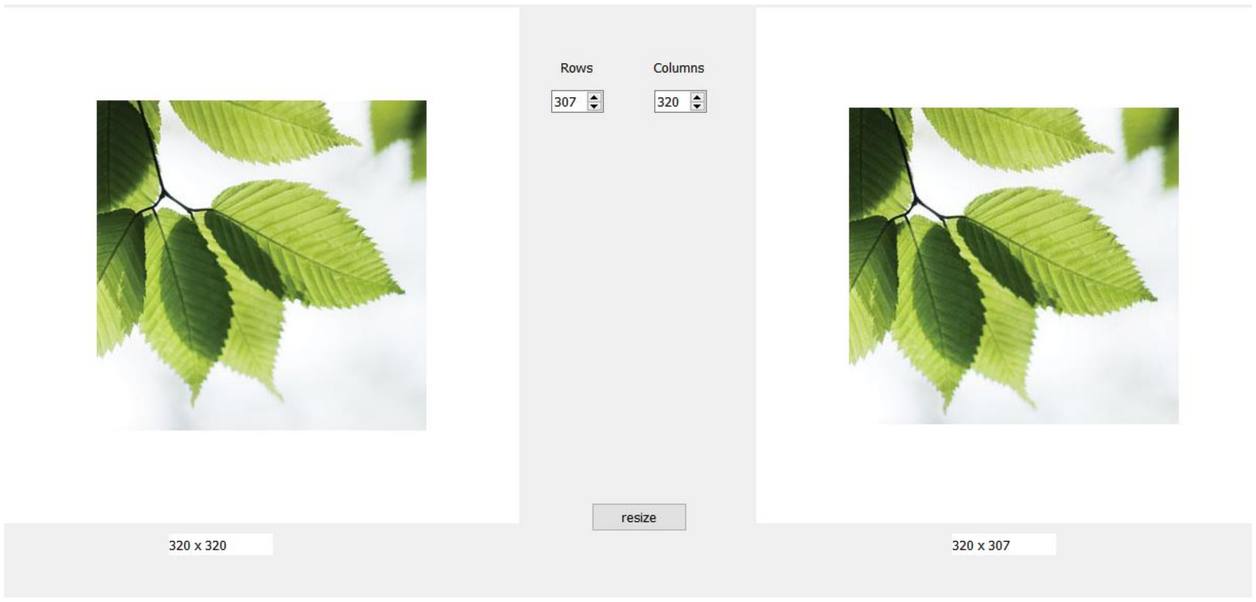
```
Pretty printing has been turned ON
```

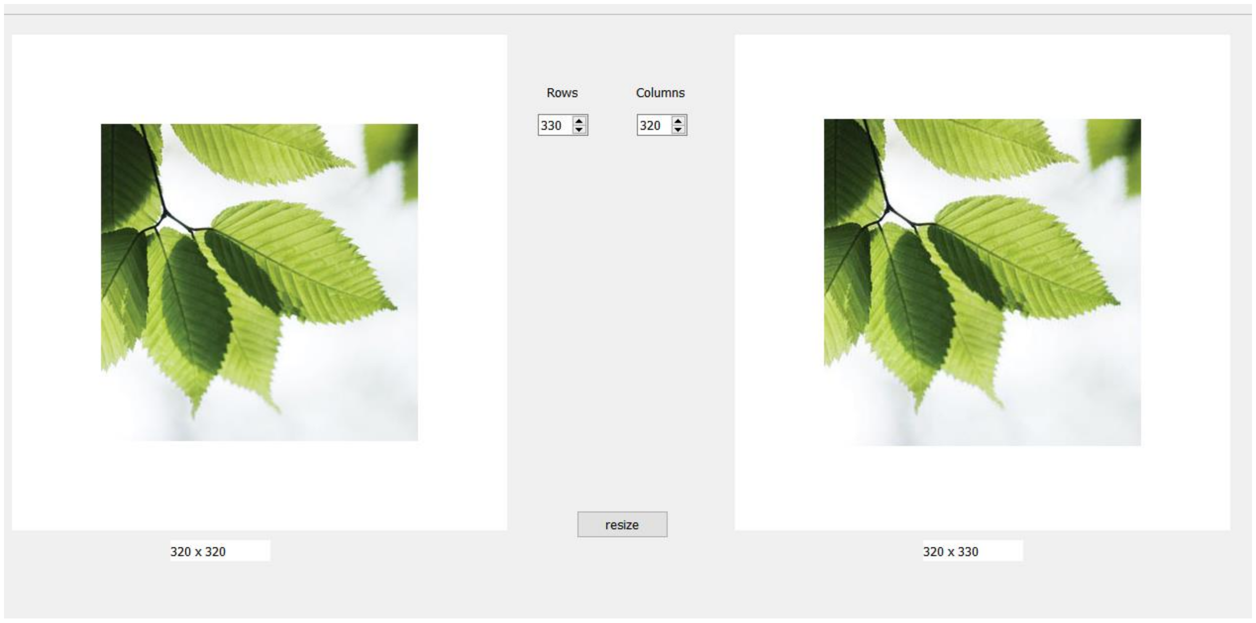
## 2 - Define Functions

In [32]:
```
# Function to ...
```

In [184]:
```
# Function to find the energy of the image
def get_energy(img):
    # Convert image to greyscale
    gray = color.rgb2grey(img)
    # Convert to uint8
    with warnings.catch_warnings():
        warnings.simplefilter("ignore") # ignore warning due to type conversion
```

## GUI Screenshots


Rows
Columns
307
320
resize
320 x 320
320 x 307


Rows
Columns
330
320
resize
320 x 320
320 x 330