Seam Carving for Content-Aware Image Resizing

Daniel-Ioan Mlesnita

October 2, 2024

Abstract

This report presents the implementation and analysis of the seam carving algorithm for intelligent image resizing. The project applies some techniques, such as gradient functions and Gaussian smoothening, learned in the "Introduction to the Mathematics of Photographs" course at the University of Helsinki to perform content-aware image resizing using MATLAB. The project was inspired by the paper "Seam Carving for Content-Aware Image Resizing" by Shai Avidan and Ariel Shamir. [1]

Contents

1 Brief Introduction **1**

2 Theoretical aspects **2**
2.1 Energy function in Seam Carving 2
2.2 Algorithm Logic 3

3 User Guide **4**

4 Methodology **5**
4.1 Data Collection 5

5 Results and Analysis **6**
5.1 Visual Results and Discussion 6
5.2 Performance Analysis 7
5.3 Challenges and Solutions 8

6 Conclusion and Future Work **8**

1 Brief Introduction

Seam carving is a content-aware image resizing technique that adjusts the dimensions of an image while preserving its important features. Unlike traditional resizing methods, such as scaling or cropping, seam carving removes or inserts "seams" — paths of least importance in the image — to alter its width or height.

A seam is defined as a connected sequence of pixels, one in each row or column, from top to bottom (for vertical seams) or left to right (for horizontal seams). The importance of each pixel is determined by an energy function, often based on edge detection, which highlights areas with high contrast or sharpness. The seams are chosen to pass through low-energy areas, where the image can be altered with minimal noticeable distortion.

Seam carving has several applications in image processing: [7] [3]

- Resizing images without distorting important content, keeping aspect ratios consistent.
- Removing objects by finding seams that pass through the object and eliminating them.
- Expanding images by inserting seams that follow the existing content, ensuring the new pixels blend naturally.

Additionally, seam carving supports the option to manually define areas where pixels should remain unchanged, and, as mentioned in the bullet points, it can even remove entire objects from photographs. Although the current project does not yet implement the feature of defining specific areas, it would be an interesting addition for a future version.

2 Theoretical aspects

2.1 Energy function in Seam Carving

In this implementation, we get the energy of an image in the following way:

1. The gradient of the image in both x and y directions is computed using the `imgradientxy` function. This function computes the directional gradients using the Sobel operator.
2. The magnitude of the gradient is then calculated using the formula $\sqrt{G_x^2 + G_y^2}$. This provides a measure of how rapidly the image is changing at each pixel.
3. Areas with high gradient magnitude (i.e., high energy) correspond to edges or textures in the image. These are considered important features and are less likely to be included in a seam.
4. Conversely, areas with low gradient magnitude (low energy) are more likely to be included in a seam, as they represent regions of the image with less noticeable change.

This energy function is effective because it preserves the structural elements of the image while allowing for removal of pixels in smoother, less noticeable areas.

(a) Original Image (b) Energy Map

Figure 1: Comparison of an Image and Its Energy Map

It's worth noting that while this gradient-magnitude energy function is effective, the original paper on the process mention other energy functions that could be employed, such as entropy-based functions or functions that incorporate saliency maps to better preserve visually important regions. ([1] sections 3.1 and 3.2) The choice of energy function can significantly impact the results of the seam carving algorithm and may be tailored to specific types of images or desired outcomes.

2.2 Algorithm Logic

The seam carving algorithm operates through several key steps (all relevant parts of the functions are in the appendix):

1. **Choice of Action**

   Before starting the computation, depending on the user's choice of the scale factor, the algorithm can either expand or reduce the image. If the user decides that they want to reduce an image, they have to choose a scale_factor that is between 0 and 1. For expansion, the scale_factor has to be between 1 and 2. A factor of 1 will keep the image unchanged.

2. **Energy Calculation:** The first step of the algorithm begins by calculating the energy of each pixel in the image. In the MATLAB implementation, this is accomplished by the **energyImage** function.

   This function computes the gradient in both x and y directions and then calculates the magnitude of the gradient as the energy. Providing a good distinction of edges, which is needed to keep the aspect ratios and shapes consistent.

   Gaussian smoothing is applied for edges to not achieve artifacts from noise or abrupt intensity changes. With smoothing, the performance is slightly increased, but results in quality image can vary. Experimenting with different sigma values for the same picture is recommended.

3. **Cumulative Energy Calculation:** The next step involves calculating the cumulative minimum energy for all possible connected seams for each entry. This is performed in the **findVerticalSeam** function in which dynamic programming is used.

   This creates a matrix M where M(i,j) represents the minimum cumulative energy to reach the pixel at (i,j) from the top of the image.

4. **Seam Identification:** After calculating the cumulative energy, the algorithm identifies the optimal seam by backtracking from the bottom row. This is also performed in the **findVerticalSeam** function:

5. **Technique:**

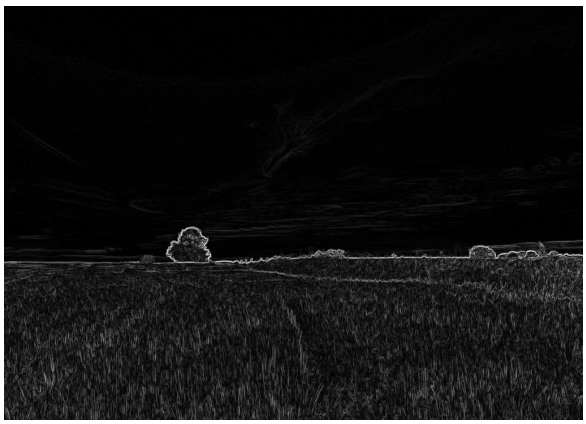
   The choice of scale_factor will determine if the process would be to **expand** an image or to **reduce** an image by a certain percentage. Here we have a "two-phase approach" for both seam removal and addition: first, all seams are precomputed iteratively on a temporary image, and then they are applied (removed or inserted) to the original image. This method allows for the seams to be influenced by previous seam locations while still operating on the original image dimensions until the final step.

   In **seam removal**, the identified seam is detached from the image by shifting pixels to fill the gap left by the removed seam, realised in the **removeSeam** function and in **seam insertion**, the identified seam is inserted into the image by shifting pixels and interpolating new pixel values along the seam path in the function **insertSeam**.

6. **Seam Highlighting:**

   For better communication of performance and algorithm behaivour given different parameters and images, function **highlightSeam** is used to to highlight individual added or removed seams over the original image.

(a) Original Image



(b) Energy Map

(c) Seam Visualization

(d) Resised Image

Figure 2: Comparison of Different Images, with scale factor = 0.85

3 User Guide

Before running of the program:

1. On line 56 of the code `seam_carving.m`, the user has to provide the name of the image that they want to perform the computation on. The image has to be in the same directory as the code file.

   e.g.

```
56 image = imread('prague.jpg');
```

Optional: On line 286, at the start of `energyFunction`, the user can change the `sigma` parameter to affect the behavior of the Gaussian smoothing.

```
285     % Recommended range: 0.25 - 1.5
286     sigma = 0.75;
```

After these steps, the user can run the program and will be met with the following prompt:

Since the processing of the seams can take a significant amount of time depending on image size and scale_factor, the code provides constant feedback of the computation process in the command window:

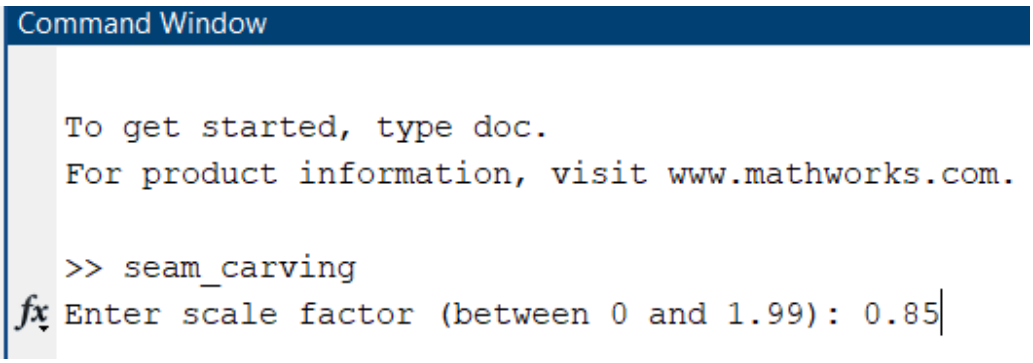


Figure 3: Scale factor

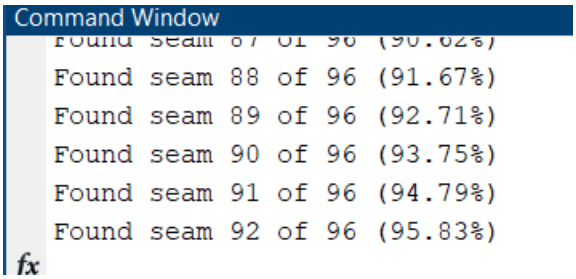


Figure 4: Feedback

At the end of the run, the user will be provided with the following images, similar as in Figure 2: Original Image, Energy Map of Original Image, Seam Visualisation over Original Image and the Resised Image.

Figure 5: Output

4 Methodology

4.1 Data Collection

The images selected for this project were random pictures with creative commons or open licenses. The camera settings were not taken into consideration for the purposes of this project.

The principal objective of selecting images was to get different sorts of sceneries, image subjects and image sizes to test how the algorithm behaves in general contexts. As such, the chosen images were mainly depicting landscapes, crowds of people, subjects in motion in relative to stationary objects of different geometries and art paintings. Here are the main pictures of reference and all their computations can be found in the directory **processed_images**:

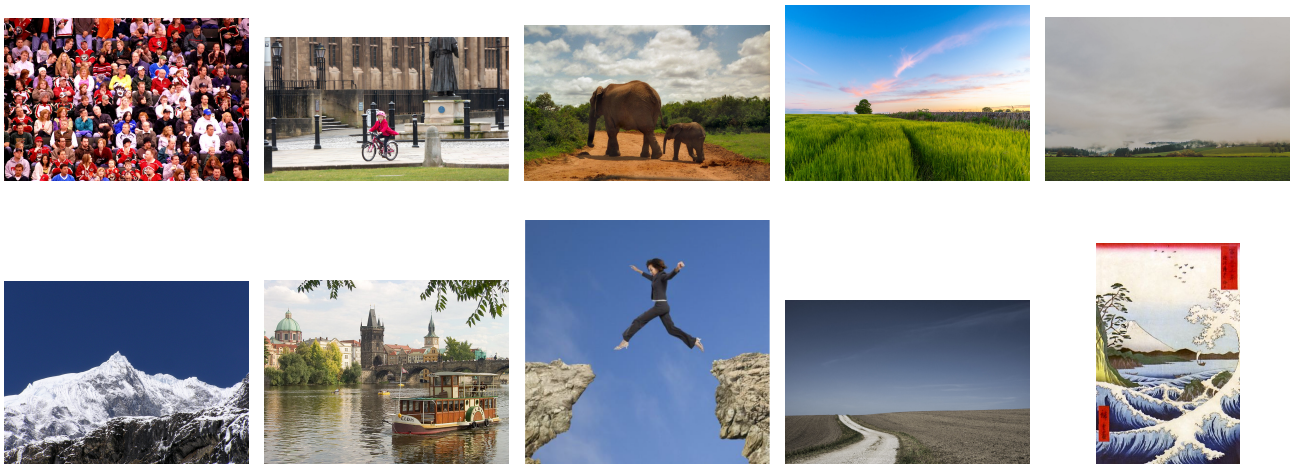


Figure 6: Collection of various images used in testing

5 Results and Analysis

It is known ([4] Results and Failures, [3]) that seam carving algorithm should excel at working with images with predictible patterns and big sceneries such as landscapes. The algorithm should have a harder time dealing with more detailed images, but even though the script presented works fairly decent, we cannot help but notice that it still produces visible artifacts and that it doesn't work very consistent, at first glance, with even similar pictures.

Once we analyse the energy maps of those images, we can come up with explanations on why this happens.

To note, that other implementations of this algorithm do not seem to solve this issue directly with gradient mapping, other techniques such as carving in the gradient domain or using different energy functions are needed. ([1] Figure 4)

5.1 Visual Results and Discussion

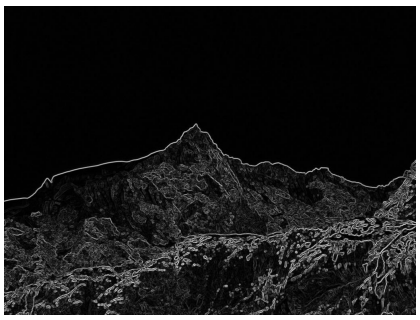
For example, even for "ideal" cases such as the mountain picture, we can still see uneven edges or weird warping of background objects in the 2nd picture, because their energy is lower overall in those columns. In the left area there is more of a constant white snow, whereas in the middle of the mountain, the rocky areas under the snow are more distinct.

In the mountain picture, seams do not take the path of the middle of the mountain because of the different shifts in colour and texture, resulting in a preference of going to the left side of the picture to cut it.

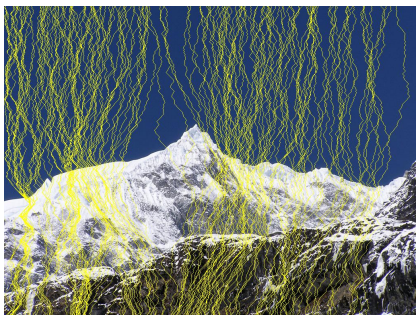
As such, we'll have an uneven surface of the mountain edge and the final picture will not have the same aspect ratio and constant shape. The lower part of the mountain seems to have been changed better.

In the boat picture, there seems to be a preference in the middle part of the image. This could be because of the low energy in that area: the dark building and calm lake area do not have a lot of changes. Over time, as the picture increases, no other areas seem to grow in complexity very much and the bias will continue to increase to the middle of the picture.

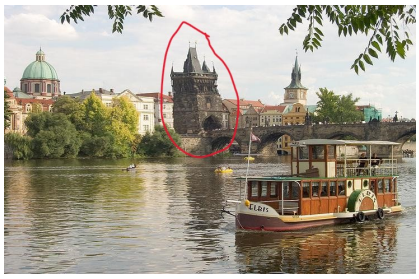
(a) 115% of the Boat picture with artifact



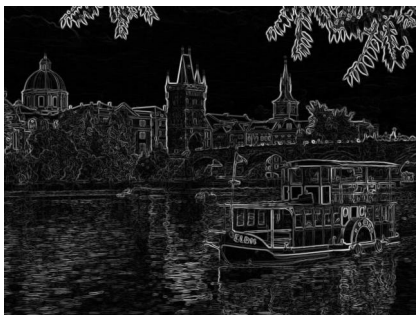
(b) Energy Map



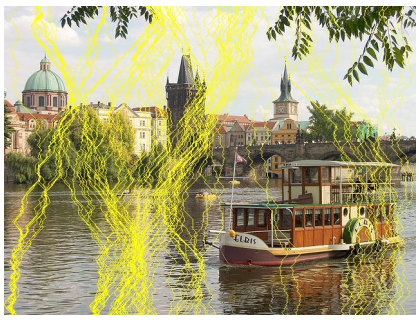
(c) Original Image with seams



(a) 85% of the Mountain picture with artifact



(b) Energy Map



(c) Original Image with seams

We can suppose that the seams seem to not go through the right part of the picture because the boat is there and looking at the energy map, the different shapes of the boat, seem to add significant energy to the right side columns.

The boat itself has been changed quite satisfactory.

(a) 85% of the Crowd picture with artifact

(b) Energy Map

(c) Original Image with seams

Surprisingly, in this image with a lot of detail and information, the algorithm did fairly well. And kept the faces mostly unwrapped. Usually, because parts of human faces, such as the forehead, do not contain a lot of energy, they can be taken as preferred paths to the seams. In this case it seems that the algorithm found better ways than to cross the edges.

(For full resolution of pictures, please check the added files.)

5.2 Performance Analysis

The computational complexity of the seam carving process is approximately

$$O(w \cdot h \cdot |1 - \text{scale\_factor}|) \tag{1}$$

where w and h are the width and height of the image, respectively. (the time complexity seems to be similar to other implementations such as [6]). This indicates that the processing time increases linearly with the image resolution and the extent of resizing required.

The testing data, with computational times for different scale factors, saved in the file **processing_times.txt**, supports the formula of the time complexity.

The biggest factors that affected computation time were drastically the size of the picture, and the second one was the scale factor. The amount of energy in an image also seemed to impact the computation, but the influence wasn't impactful.

We could argue that the energy distribution, more precisely, how many possible paths are possible to be taken by a seam, could impact the time complexity. But there is no mention of that in any sources, and testing did not seem to support this.

5.3 Challenges and Solutions

The project itself was fairly straight forward, but there was a set of options when choosing multiple techniques at different stages of the computation process.

If we are to name a few, two key challenges when developing this project were:

1. Energy Computation Method
2. Path Finding Algorithm for Seam Choosing,
3. Method of Removing or Adding seams

We have explored different image energy computation methods, and we came to the conclusion that gradient energy function with Gaussian smoothening (for energy computation), and backtracking (for path finding) were the best compromise between results and performance.

Other methods that were taken into consideration were: entropy and visual saliency maps ([1], section 3.2). But each of them weren't satisfactory when added in the whole project.

When trying to come up with a better seam addition or removal, we have tried to use different iterative approaches in order to get rid of the artifacts, but the outcomes resulted in either: very long computational times or the seams were repeated for the same column of pixels like in the following figure:

(a) 150% Expanded Image

(b) Energy Map

6 Conclusion and Future Work

When compared to other similar implementations that can be found on the internet ([2], [5]), this project seems to be either on the same level or better in both computational time and satisfaction of processed images, but there are still significant limitations to this approach.

Firstly, the project is prone to produce artifacts, as discussed in section 5.1, which suggests that we should improve the path-finding algorithm. It might be a good idea to explore different

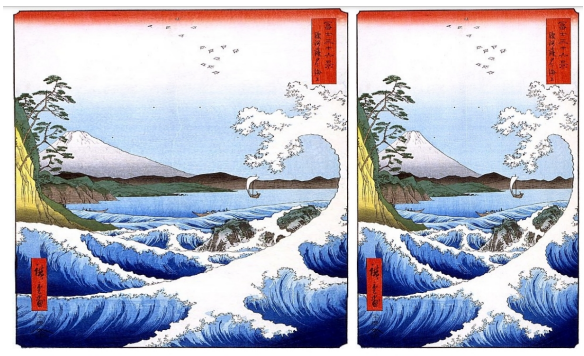
ways of how do "local areas" affect the energy computation and come up with other formulas or weights of how do we compute the energy of a pixel compared to its neighbours.

Another problematic aspect, is that fine details and large simple areas are prone to be damaged. They would usually include low energy areas such as human foreheads or, for example, the banner in the painting "Sea" of the Japanese master Utagawa Hiroshige:

(a) 115% Expanded Image

(b) Original Image with Seams

Whereas in the original study, the scaling of the same image, has been smoother and we can see that the red banners have remained unaffected by the algorithm



(a) 115% Expanded Image

One interesting method of solving such problems, seems to be the selection of areas of "higher importance" (INSERT SOURCE HERE OF PEOPLE THAT ALREADY IMPLEMENTED THAT). With this method, users can select areas over the image, to point pixels that should have **"higher priorities"**. In the path finding algorithm, the seams would try their hardest to avoid these higher priority routes in their ways.

A useful consequence of this implementation is that it allows for the feature of "object removal" which would work by defining areas to have **"lower priority"**. ([1], section 4.6)

Overall, it seems that the best use case for the current project would be images with a single main subject in a repetitive surrounding.

References

[1] Shai Avidan and Ariel Shamir. "Seam Carving for Content-Aware Image Resizing". In: *ACM Transactions on graphics (TOG)* (2007). DOI: 10.1145/1275808.1276390.

[2] Muhammet Balcilar. *Seam-Carving*. https://www.mathworks.com/matlabcentral/fileexchange/116698-seam-carving. MATLAB Central File Exchange. Retrieved October 1, 2024. 2022.

[3] Alan Edelman, David P. Sanders, and Charles E. Leiserson. *Introduction to Computational Thinking, Section 1.8 Lecture Video: Seam Carving*. https://computationalthinking.mit.edu/Fall24/images_abstractions/seamcarving/. Accessed: 2024-10-02. 2024.

[4] William Keys. *Seam Carving for Content-Aware Image Resizing*. 2018. URL: https://andrewdcampbell.github.io/.

[5] Shubham Mehta. *seam-carving: A Matlab implementation of an efficient algorithm for content-aware image resizing*. https://github.com/sumehta/seam-carving. GitHub repository. Accessed: 2024-10-02. 2014.

[6] Aditya Sharma. *Image Compression using Seam Carving and Clustering*. https://adityashrm21.github.io/Image-Compression/. Accessed: [Insert access date here]. 2019.

[7] William Wedler. *Project 2: Image Resizing by Seam Carving*. https://www.cs.cmu.edu/afs/andrew/scs/cs/15-463/f07/proj2/www/wwedler/. 15-463 Fall 2007, Carnegie Mellon University. Accessed: 2024-10-02. 2007.