Seam Carving High Frequency Textures

Terrence Anderson

Supervisor:
David Mould
Acknowledgments

First and foremost, I’d like to thank David Mould for his unending patience and incredible guidance. He got me through a lot of undergrad.

Secondly, my wife Rachel, who kept me motivated in the final semester, where motivation was spent overwhelmingly at the workplace.

Lastly, the researchers who worked tirelessly before me to get these methods working.

Table of Contents:

Abstract

Seam carving is a technique that’s been fundamental to image processing in the last decade. This technique allows content-aware modifications to an image’s size. The key part of “content-aware” is that it allows a path to modifying an image without affecting important parts of the image. These ‘seams’ are 1 pixel wide paths that run from one side of an image to another according to a chosen energy function [1].

However, there is no research into resizing high frequency content in images, which is a weakness of standard seam carving methods. Some images are defined by the regularity of their recognizable content, sometimes referred to as ‘textons’, and so modifying them becomes very apparent to the observer’s brain[2]. The seam carving method assumes that important content is largely separated, and that removing seams is statistically unlikely to affect vital image content. This assumption obviously breaks down once you have repeated or regular content, since the seams are unidirectional they are likely to collide with many important sections of the image, thus the changes are easily perceptible[3].

This problem is an important one for texture-mapping onto planes or 3D models, because of the overhead in handling surfaces with different aspect ratios than the one source image. Simple methods involve scaling or cropping the source image, other methods are to create many different textures for different aspect ratios (via a tool like photoshop). These are not ideal since they are fundamentally not suited for this task, or are too much extra work, poor scalability, and storage footprint for the multiple-images approach.
The goal of this project is to outline an approach for resizing images with a high frequency of textons.

Introduction

Image manipulation is key to many industries, such as film, games, and a lot of web content. Websites need to crop or resize images, games need highly malleable textures for rendering, and films use a wide variety of methods to achieve immersion and realism. Any methods or solutions to making image manipulation easier, more efficient, or more accurate are fundamentally important for innovation in these industries.

One such problem is resizing a static image such that the aspect ratio has changed – that is, there are more lines removed in one dimension than another. Strictly cropping or scaling the image could undo the work put into the picture by obstructing or destroying the intended look of the image.

A method to repurpose images was published in 2005, which started the whole concept of algorithmically modifying images to allow reuse in broader contexts[5]. The authors of this article suggested marking key parts of an image, and modifying the rest of the image separately, achieving a level of simplistic restructuring of an images layout. This would be proven to be handy for situations such as an image that could be used for a website, as well as a mobile phone or tablet (which could have very different resolutions).

In general, image resizing is complicated because authors or photographers of images have to spend a great deal of creative energy to get it to look aesthetically optimal. When the image needs to be reused in a new context, or within a specific margin, it can be difficult to accommodate artistic vision with ever-changing goals or intentions. This shows up a lot in businesses that deal with visual aspects.

Image retargeting had flaws though. Non-important regions could easily become highly distorted, thus rendering most of the image sensitive to the algorithm applied to reapply key regions. A method outlined below called “Seam Carving” was introduced that would allow more flexibility in determining important regions. It allowed low-impact ‘seams’ to be removed, based on a definition of low-impact that suited the image.

But the above methods have researched how to retarget/manipulate image dimensions with sparsely populated perceptually distinct regions, because generally images are structured that way: A kid in a field, a pleasing landscape, or a line of trees in the distance (figure 1).
The aforementioned image retargeting is not designed for repeated, high frequency content. Sometimes pictures contained perceptually clear parts of the image which are easy to distinguish by the human brain, but not necessarily easily defined mathematically. These have been referred to as ‘textons’ in some cases, meaning a piece of an image that is easily distinguished by human observers - and the word is helpful in this case because the goal is to apply an image manipulation method to maintain perceptual continuity.

Some pictures taken could be of gravel, or of honeycombs, which are desired for their visual impact:

![Honeycombs and rocks](image1.png)

Figure 2. Some images whose content is clearly discernible. Honeycombs, and rocks.

But these are notably different from noise, which has not such perceptual or unique pattern that must be maintained. You can define each of the parts of the above images, but you cannot define any such shape in (3). This is because random noise is not the goal for image resizing or modification, since it’s likely easier to just recreate that noise, giving a unique image.

![Fractal noise pattern](image2.png)

Figure 3. A fractal noise pattern. Nothing discernible.
So seam carving won’t be enough, on its own. Seam carving can be repurposed by changing the parameters for how to carve a seam, to include components of an image. So instead of carving a seam of 1 pixel, you actually carve out ‘textons’. This carving would remove constituent pieces of the image, and keep the remaining textons intact. Additionally, some seam carving may be done to fill in some gaps, or other postprocessing operations.

Texton detection can be achieved with a graph-based image segmentation provided by Felzenszwalb, and Huttenlocher. This algorithm selects similarly coloured pixels and places them in a set, thus breaking the image down into multiple components. Generally, this algorithm’s choice of weighting selects for the textons we’re aiming for. With the pixels now associated to their visually recognizable segment, we can begin carving them away to create a new image that closely resembles the original appearance (because it leaves much of the image unaffected.

**Seam Carving**

In 2007, an approach to aspect ratio modification was developed. This method was published in the paper “Seam carving for content-aware image resizing”[1] which outlined a method for resizing a source image while maintaining the structure of region that were more discernible by the human brain. Some regions in images are vital to maintain their shape or structure upon resizing.

The key to this method is the energy function, which is what is parsed to track seams. Any function can be used, as long as the final output is 1D, for the seam calculation algorithm. The energy function is $F(s) \in C^n \Rightarrow D(y) \in C^1$ where:
- $F(s)$ maps from $N$ dimensions to $n$ dimensions, processing the original image to an energy set $Y$
- $M(x)$ maps from $m$ dimensions to 1 dimensions, where the input is the processed energy set $Y$

$M(x)$ is the function used for the seam weightings. Seam carving uses the function $M$ in a least-path algorithm, summing up the smallest weighted path from one side to another (creating a continuous ‘seam’).

Many photos have ‘noise’ in them - such as grass, leaves, flowers - which populate the bulk of the frame. This noise is then often punctuated by structurally unique figures, which pull the human perception to that section of the photo. The human brain is attuned to quickly focus on those unique regions, and so they are highly sensitive to modifications. You can easily change that region too much and it will look wildly different, wrong, or fall into the noise[3]. An example:
As you can see, the sunflowers are the first thing you’re likely to notice, they stand out in the image. The flower on the left is more easily recognizable than the right. Seam carving can shrink an image, using an appropriate energy function. Here’s a seam carving with a simple gradient energy calculation:

![Sunflowers image](image-url)

and it’s associated energy map:

![Energy map image](image-url)

So this method is useful in maintaining the essential perceptual pattern in the photo or image.

Once you’ve calculated the energy, you can determine the seams algorithmically by dynamic programming. Here’s pseudocode for a horizontal seam:

```plaintext
text

foreach pixel in row:
    foreach pixel in column:
        middle = previousColumn@row;
        upper = previousColumn@(row – 1);
        bottom = previousColumn@(row + 1);
        seamPoint(row, column) = min(upper, middle, bottom) + currentPixel(row, column);
```

This will populate a 2D array which will contain the seams. To run along a seam, start from the opposite side from where you starting, and travel along the minimum path to the starting side. I.E. if you start from \( x = 0 \), then check your seam starting from \( x = \text{width} - 1 \).

To perform seam carving horizontally, you can simply remove the lowest-weighted horizontal seam to reduce the height by 1. Same exercise for vertical operations. The lowest-weighted horizontal seam can be found by finding the smallest value in the side with the terminal points, since you know that these values are assigned by a summation (and thus only decrease from that point).

Sometimes there are regions where the sensitivity to modification is so high that and path intersecting with them should be treated as extremely high weighted, such as faces.

One important part of this algorithm is that each seam is a continuous path from one end of the image to another, so that you can reduce an image in size line by line. This means that images with a high degree of perceptually distinct components will likely have the seam intersect with them, which would disrupt the intended appearance of the image. For example, this image of rocks:

![Image of rocks](image.png)

No seam can effectively traverse from left to right, or top to bottom that won’t intersect with the perceptually vital parts of the image. So the seams would eventually cause artifacts, because they would be almost equivalent to random lines.

If we could break down an image into visually distinct pieces, we would have a much clearer idea of what to remove.

### Image Segmentation

One such approach that I investigated was to segment the image into smaller components, such that the pixels in a component were more alike to each other than to pixels in a neighbouring set [4]. This proved to be pretty good when you have many distinct pieces contained in an image. The assumption is that the repeated texton are unique enough from each other according to a boundary. This would likely not work for a picture of waves.
This component segmentation method treats the image like a NxM graph where the vertices are pixels, and the edges are the difference in magnitude between the R, G, B channels. Given this set of points and edges, we must group them according to likeness. The first step is to set each pixel as its own component of size 1.

When comparing 2 components (of any size), you can merge them if the difference between the smallest edge connecting 2 components is smaller than the internal difference within the components. To compare these efficiently, we can calculate all the edge weights between each pixel, sort the edges, and then run over them to construct and calculate the components and their internal differences. Therefore the algorithm can be completed in \(O((NxM)\log(NxM))\) time, since the edges are of \(O(NxM)\) in number.

Refer to Appendix 1 for calculating the internal difference and further algorithmic information. Each component is compared and merged until clear boundaries exist between each component.

**Method**

Combining these two approaches, we can begin an approach to carving out many textons.

First, we process the energy on a picture. This energy is used for the Seam Carving portion, so that we know the lowest impact lines of the image. Low impact seams give a small amount of assurance that the removed lines won’t be too large in effect.

In my tests, I went with a simple gradient measurement. Where the energy of a pixel on an image is determined by the partial differential of each color channel, in both the x and y directions. Appendix 2 outlines this.

This will create a 2D energy map, which is used to construct the seams via dynamic programming.

With the energy calculated, we now need to run the image segmentation algorithm to determine the visually distinct sections of the image. Since there are a few variables selected by the user for the image segmentation, it’s best to play with values until you get a segmentation that works well for your case, such as in (4).
Figure 4. These bricks are segmented using a value of $\sigma = 0.8$, $k = 200$, $\text{min}_\text{size} = 150$

The seams are helpful for this, because they will show a path that has the lowest impact, which helps incur the lowest impact on removal.

In my analysis, I took the number of lines I wanted to remove, and remove them via seam carving, until I hit a threshold amount of lines. Once at this number, I use the current seam as a guide, and run along the seam and marking which components are intersected. The intersected components are remembered, and then on a second pass are removed entirely. The threshold chosen for the removal of a component was:

\[
\text{threshold} = \frac{\text{(average size of component)}}{2.5}
\]

This proved to be a good measure, but can be tweaked per image.

To remove a component, only a simple removal was used, where all the pixels are shifted to fill the hole. Since there’s no guarantee a component’s size, we need to perform postprocessing to keep the image as a rectangle.

Depending on seam direction, you have to shift the pixels back towards a rectangle. The method chosen was to fill in the holes by inserting them at regular intervals in the image, and inserting blurred pixels in their place. The pixels inserted are just an averaging of the surrounding pixels. For example, in a horizontal seam you insert the holes in a column at regular intervals. The number of holes to insert will be the (number of pixels removed due to component) – (number of seams removed).

References

1. Shai Avidan, Ariel Shamir
   “Seam carving for content-aware image resizing”

2. Julesz, B.
   “Textons, the elements of texture perception and their interactions.”

3. Florian Röhrbein, Peter Goddard, Michael Schneider, Georgina James, Kun Guo.
   “How does image noise affect actual and predicted human gaze allocation in assessing image quality?”

4. P. Felzenszwalb, D. Huttenlocher
   “Efficient Graph-Based Image Segmentation”

5. Vidya Setlur, Saeko Takagi
Appendix

1. For determining components, we must ensure that the pixels already in the component are valid. This is achieved through induction. If you start with each pixel as its own component, you can begin merging components whose internal difference is similar.

“Internal difference” is the maximum weight of the minimum spanning tree of the components

\[
\text{Internal}\_\text{difference}(C) = \max_{(\text{MST}(C,E))} (w(\text{edge}))
\]

This is a good metric to define the similarity of a graph, by the properties of a minimum spanning tree.

Then, we need to define the difference between components, the most rational measure is the smallest edge connecting two different components:

\[
\text{Dif}(C_1, C_2) = \min_{vi \in C_1, vj \in C_2} (w(vi, vj))
\]

This difference defines the boundary between two components, so while checking all edges you can compare the vertices on each point of the edge (in this case, the pixels) and see if the edge weight is less than, equal, or greater than the internal difference of the components. If the edge weight is smaller, than the two components belong together as a single component, and then merging the components with that edge will form the minimum spanning tree of the new component (since it was the minimum edge between the two original ones).

To perform this check, simply sort the edges by weight, and then check them iteratively. This will guarantee you’re always checking the minimum edge between components.

One can make the components have a unique merging behaviour by defining a threshold for when to merge:

\[
\text{Modified}\_\text{internal}\_\text{difference}(C_1, C_2) = \min(\text{Internal}\_\text{difference}(C_1) + \tau \cdot (C_1), \text{Internal}\_\text{difference}(C_2) + \tau \cdot (C_2))
\]

The threshold suggested is \(\tau = k / |C|\). That way the threshold for merging becomes smaller as components grow, but initially it’s at a maximum.
The approach suggested by the paper is to filter the image with a gaussian filter prior to segmentation analysis. Additionally, the source I used had a min_size parameter[6], which I thought was a great thing to include, size we can guesstimate the size of a texton visually.

2. Performing a gradient energy calculation, you treat the source image like a 3 dimensional domain with a 2 dimensional output (RGB to XY). Then measure the rate of change of these channels wrt x and y (perform partial differentiation).

This gets you $dr/dx$, $dg/dx$, $db/dx$, and $dr/dy$, $dg/dy$, $db/dy$

In the default case, I simply weighted each partial differential equally. 1/3 contribution for each. Then the energy is the sum of these values from the center pixel.

Code segment:

```c
float didx(float center, float left, float right)
{
    return (abs(center - left) + abs(center - right)) / 2;
}
float didy(float center, float top, float bottom)
{
    return (abs(center - top) + abs(center - bottom)) / 2;
}
```