Content-Aware Image Resizing

Figure 1: Original image overlaid with seams to be removed (left); width contracted by 50% (right).

1 Introduction

With rapid advances in technology, there is now great diversity in the displays available to view digital media content. This poses a challenge for application designers who must make their content readable and appealing across a wide range of sizes and resolutions. This is especially challenging when the media contains both images and text, neither of which a designer wants to be distorted or cropped to fit a smaller screen or different aspect ratio.

Many web browsers and software applications already handle text resizing in an efficient, high-quality manner. However, standard image scaling today is not mindful of important image content that one often wants to preserve. Resizing an image is usually done uniformly or by cutting out a portion of the content (some of which may be essential to the image).

This paper implements a powerful algorithm described thoroughly in the excellent paper by Avidan and Shamir [2007]. We use seam carving to find and remove the parts of the image that contain the least important content first. This is done by calculating an energy function over the image and using a dynamic programming approach to find the lowest-energy paths from one side of the image to the other.

This simple algorithm can be extended nicely to several interesting applications. As described above, we can remove either horizontal seams or vertical seams to contract the image in either direction. We can also expand images by inserting seams of similar pixels next to low-energy seams we find, instead of removing them. This allows us to create larger images without changing the proportions of the important image content.

Sometimes users do not know beforehand what size they want to reduce their image to. In the simple approach where we re-compute seams each time the user wants to change size, this could take on the order of tens of seconds per attempt and thus become frustratingly impractical. By pre-computing all of the seams in each direction which a user may want to remove, we can allow the user to dynamically resize their image smoothly from any size in the range 0% to 200% the original dimension. (Of course, the upper limit could be extended even further, but serious artifacts can begin to form even at changes in size by 50%.)

In addition, since this algorithm is not perfect and can often remove parts of the image that the user wants to preserve or vice versa, we provide a method for selecting certain regions to be protected or erased. We allow the

user to select bounding boxes for part of the image that they think are particularly important or unimportant. By artificially increasing or decreasing the energy assigned to the pixels in these regions, we can change the priority with which seams through them will be selected.

We created a Graphical User Interface (GUI) that allows users to dynamically resize their images either horizontally or vertically to any size (between 0% and 200%). They are also allowed to select a set of bounding boxes for parts of the image that they want to protect or erase. Finally, there are adjustable parameters (e.g. energy function used) that can affect the pre-computation time, the types of seams removed, and the quality of the resulting image which we will discuss in detail later. The graphical user interface and varying the combination of parameters used are the main original contributions of this paper.

2 Previous Work

There has been growing interest in image retargeting that resizes the image while preserving important, salient features. Viola and Jones [2001] have used top-down approaches (e.g. face detectors) to detect essential regions of the image. On the other hand, Itti et al. [1999] have used bottom-up approaches that construct visual saliency maps of the image and subsequently crop the image to attain a certain size. Santella et al. [2006] use eye tracking of human subjects to determine which images are most viewed and thus most important to crop images intelligently. Many of these methods achieve great results, but are limited by their dependence on cropping or uniform scaling to ultimately change the image size.

Additional related works can be found in the thorough list written by Avidan and Shamir [2007].

3 Algorithm

Our seam carving algorithm finds and removes seams from the image in order from least important to most important. How do we determine which pixels are essential to the overall character of the image? There are many ways to do this, and we explore a couple different energy functions that weight certain features of the image differently. For now, we will describe what we will call e_2 energy, which we use for the images in this paper.

In discussing the algorithm, we will always assume that we want to remove vertical seams in order to reduce the width of the image. However, we can easily apply the same functions to the transpose of the image to remove horizontal seams and reduce the height. No additional special cases are necessary.

3.1 Energy Function

Because regions of interest (and importance) often have several different colors and/or many edges and corners, while unimportant regions (background, sky, water, grass, etc.) are often characterized by a fairly uniform texture and less sharp edges, we attempt to measure importance by using an energy function that calculates the magnitude of the gradient at each pixel, e_2.

$$e_2(\boldsymbol{I}) = \sum_c \left|\frac{\partial}{\partial x}\boldsymbol{I}_c\right|^2 + \left|\frac{\partial}{\partial y}\boldsymbol{I}_c\right|^2$$

where c represents a color channel (either R, G, or B).

It is reasonable to suspect that this energy function might not be the best one to use because human color perception is not linear in the RGB color space. For example, red and yellow can seem pretty distinct, but are not actually that far away in RGB color space compared to green and purple, which are much farther away in RGB color space but perhaps not any more perceptually different. For example,

humans might think of a dark red (100, 0, 0) and black (0, 0, 0) as perceptually very different and a medium grey (100, 100, 100) and black as less perceptually different, yet the distance between grey and black in RGB color space is much higher.

Nevertheless, e_2 is computationally inexpensive and in many cases performs well enough. We can easily compute e_2 over the entire image by convolving a Derivative of Gaussian (DoG) Kernel with the image. We first smooth the image with a Gaussian in order to reduce noise. We can adjust the sigma of the Gaussian used, and upon experimentation found that a sigma value of approximately 1 works well.

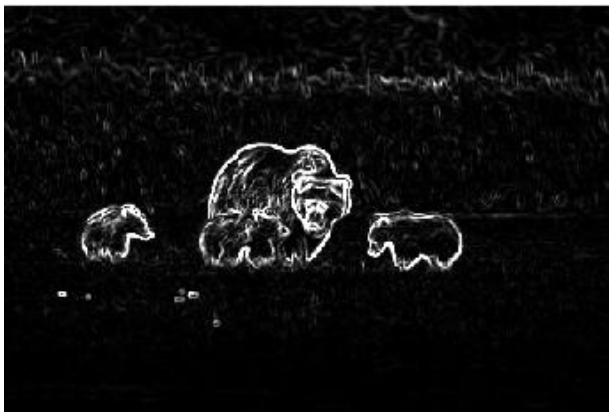

Figure 2: Output of the energy function, which convolves a DoG filter with the image.

In figure (2), we show the output of the e_2 energy function using the image of bears. We can see that the outlines of the bears and the border between the brush and the background trees have high energies while pixels within a uniform region have low energies.

3.2 Dynamic Programming

Once we have the energy image, we want to calculate vertical (horizontal) seams by finding the cheapest paths from the top (left) to the bottom (right). Pixels with high energy are treated as high cost. A naive approach might try to calculate every possible path from one side to the other and take the cheapest one. On an M x N pixel image, this would take exponential time in the size of M and N (proportional to $N*K^M$ for vertical seams where K is the number of choices at each step in the path construction). However, a dynamic programming (DP) approach allows us to calculate the cheapest path in linear time with respect to M, N, and K.

To calculate a vertical seam, we construct a new M x N matrix row by row from top to bottom, where each energy in this matrix $D_{i,j}$ is the total cost of the cheapest path from the top of the image to the pixel at (i, j). If K = 3, this is the equation to calculate each row after the first:

$$D_{i,j} = C_{i,j} + \min\left(D_{i-1,j-1}, D_{i-1,j}, D_{i-1,j+1}\right)$$

where $C_{i,j}$ is the cost of the pixel at (i, j). (If K = 1, then the vertical seams are simply columns. If K >= 3, then each seam does not have to be connected, i.e. they can skip across columns as they go from top to bottom.) This makes use of memoization to avoid redoing calculations that we have already done for subcomponents of the larger paths.

Finally, we can find the cheapest overall path by finding the lowest value in the bottom row of *D* and then tracing the cheapest path associated with that pixel back up to the top row of *D*. To reduce the width of the image by one pixel, we simply delete the pixels in this seam (and slide the neighboring pixels over to fill the gap).

By the nature of our DP approach, if we calculate all the cheapest paths associated with each of the bottom pixels, we will get many clusters where the paths back to the top heavily overlap. Because we can only remove each pixel at most once, if we want to pre-compute all seams, then we cannot simply run one iteration of the DP calculation. With only one iteration, we often only find a few completely distinct paths from top to bottom (as shown in Figure 3).

As such, in order to compute the *n* cheapest vertical seams (to reduce the image

width by *n* pixels) that do not share any pixels, we need to do the DP computation *n* times, after removing the single cheapest seam each time. This is more computationally expensive, but ensures that we are removing only the cheapest seams. You could potentially remove more distinct seams each time, but this might cause you to remove a very expensive seam unless you set a threshold, which is another parameter to worry about.

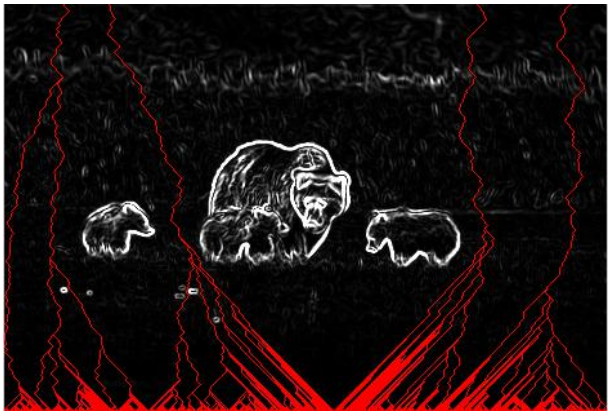

Figure 3: Back-traced cheapest paths from each pixel in the bottom row, as a result of one dynamic programming calculation.

3.3 Expanding the Image

Once we know which low-energy seams we would want to remove in order to contract the image, it is simple enough to extend the algorithm to expand the image. To increase the width of an image by 1 pixel, we can find the cheapest seam and average that seam with the set of pixels just to the left and the set of pixels just to the right. We will replace the original seam with these two new seams, thereby increasing the width by 1. To increase the width of an image by *n* pixels, we can simply repeat this process *n* times. Figure 4 includes an image expanded by 20% in the vertical direction.

When we wish to expand the image by large amounts (e.g. greater than 50%), using the above method can cause noticeable distortions in the image. For example, expanding by 100%

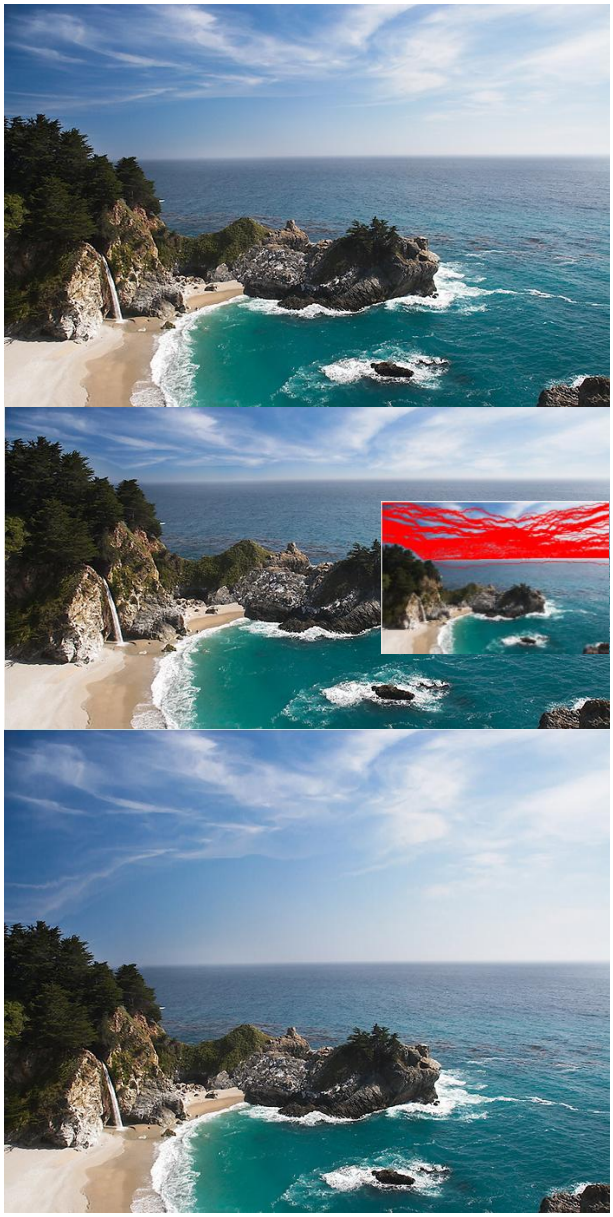

Figure 4: Original image of cove (top), image with height contracted by 20% and inset of original image with seams overlaid (middle), image with height expanded by 20% (bottom).

is similar to just uniformly increasing the width, since every pixel is eventually part of a seam. Suppose we want to limit the distortion of important regions of the image, while expanding less salient parts of the image (e.g. background). Unlike contracting the image, we are not restricted to using each unique seam once. We can instead set some breakpoint (e.g. at 50%)

such that for increases above 50%, we re-use the seams that we used below 50%. In this way, we only expand the image around seams which are below the median energy and we avoid distorting the high-energy, important parts of the image. This breakpoint is an adjustable parameter in our implementation and we discuss it further in section 4.2.

3.4 Dynamic Resizing

This algorithm works well enough for users who want to take an input image and change its size once. However, users don't always know the dimensions that they want for their resized image. As a result, it is useful to allow users to dynamically resize their image, decreasing and increasing the dimension until they achieve a desirable result. Ordinarily, this type of experimentation would be painfully slow. Re-running the entire program with each change might take tens of seconds.

Instead, we pre-compute all N (distinct) vertical seams that could potentially be removed by the user if they wanted to reduce the width of the image to anywhere from 0% to 100%. We store information about which pixels should be removed by the *n*th cheapest seam in an M x N matrix. Each entry (i, j) contains the number *n* of the seam that would remove the pixel in that position. In this way, we have a space efficient way of storing all necessary information to quickly know which pixels to remove when the user asks to reduce or increase a dimension of the image. We create a separate index map V and H for vertical seams and horizontal seams, respectively. Actually removing the pixels is quick enough that we can do this in a fraction of a second each time the user changes the desired image size.

This method can also support enlarging of images if we want to increase the width of our image to M' > M. However, instead of averaging neighboring pixels to increase the width of an image as before, we will insert a new seam of pixels to the left (or right) of the seam that is the average of the original seam and the pixels to its left (or right). In this way, we do not modify the original pixels of the image, but simply add new ones. For the *kth* cheapest seam, we create a new seam to add which we will index in the negative direction and call the *-k*th seam. So in our index maps (V' or H', each of size M' x N), the entries for the newly added pixels (for the *kth* cheapest seam) will be *-k*, as shown in Figure 5.

	250	50	181	186	283	
	224	250	50	181	186	
	250	224	181	50	186	
	250	224	50	181	186	
	224	250	50	186	181	

	250	50	-50	181	186	283	
	224	250	50	-50	181	186	
	250	224	181	50	-50	186	
	250	224	50	-50	181	186	
	224	250	50	-50	186	181	

Figure 5: Example V index map (top), Example V' index map after inserting a seam to the right of the 283rd cheapest seam in the original image (bottom).

4 Results

For minor size changes, this algorithm tends to perform fairly well, maintaining the important features, while removing/adding less-noticeable components. The more extensive the background, fairly homogeneous regions of the picture are, the greater the magnitude of the changes in size this algorithm can perform well. And the higher the energy differences between truly important and truly unimportant regions are, the better this algorithm chooses what to keep.

4.1 Varying the frequency of re-calculating energies

Although we likely need to re-calculate the cumulative path costs with dynamic programming each time we remove a seam, we do not necessarily need to re-calculate the energies (gradient magnitudes) at each pixel every time. This is not something we explicitly tested, but we hypothesize that re-calculating the energies only 10% of the time would not decrease performance by much. Arguably, you might not even want to re-calculate the energies at all if you want to base your seams solely off of the original image. Not re-calculating the energy would probably reduce the runtime by a nontrivial amount, but the bulk of the runtime is probably from re-doing the DP calculation.

In addition, although we need to do a new DP calculation for each new seam, we may be able to use prediction heuristics to reduce the search space. Low-energy seams are likely to be near other low-energy seams while high-energy seams are likely to be near other high-energy seams. For example, we could only run the DP on a 20-column-wide section of the image around the previous seam when finding the next seam. This may reduce our accuracy to some extent, but this could also significantly reduce the runtime (by possibly an order of magnitude).

4.2 Varying the breakpoints

When greatly expanding an image, we might not want to add seams based on the seams that were high energy. If we do, we may end up producing results that are similar to uniformly growing the image, which distorts important figures in the image (as well as unimportant ones). To combat this effect, we can instead set a breakpoint at say 50%, which would cause us to only use the seams in the lower half of energy when trying to expand the image. For example, when looking for the last seam to expand our image to 160% of its original size, we will use a seam near the 10th (60 mod 50) percentile of energy. In this way, we will attempt to add only lower-energy seams to the image.

We can try even lower breakpoints (e.g. 25% or 5% as in Figure 6). This may work well for some images if, for example, there isn't much water and we want to increase the width of the image by 50% but only want to increase the amount of sky. Figure 6 shows that even a breakpoint of 5% doesn't perform terribly for some images, but it does demonstrate some tradeoffs. Lower breakpoints allow us to preserve the size of important features in the image (islands in the case of Figure 6). We can see that the skinny island in the lower right hand corner gets expanded with breakpoints of 25% and 50%, but doesn't get expanded at all at 5%. By contrast, most of the other islands must have been above the median energy and received no expansion at any breakpoint. However, lower breakpoints (for the same expansion) clearly require more repetition. This can sometimes be noticeable, as in the border between darker and lighter water in the top third of the image. The lower the breakpoint, the more streaky and noticeably doctored the image might be.

Figure 6: (left to right, top to bottom) [height increase, breakpoint] of island image:
a) [50%, 50%], b) [50%, 25%], c) [50%, 5%],
d) [100%, 50%], e) [100%, 25%], f) [100%, 5%]

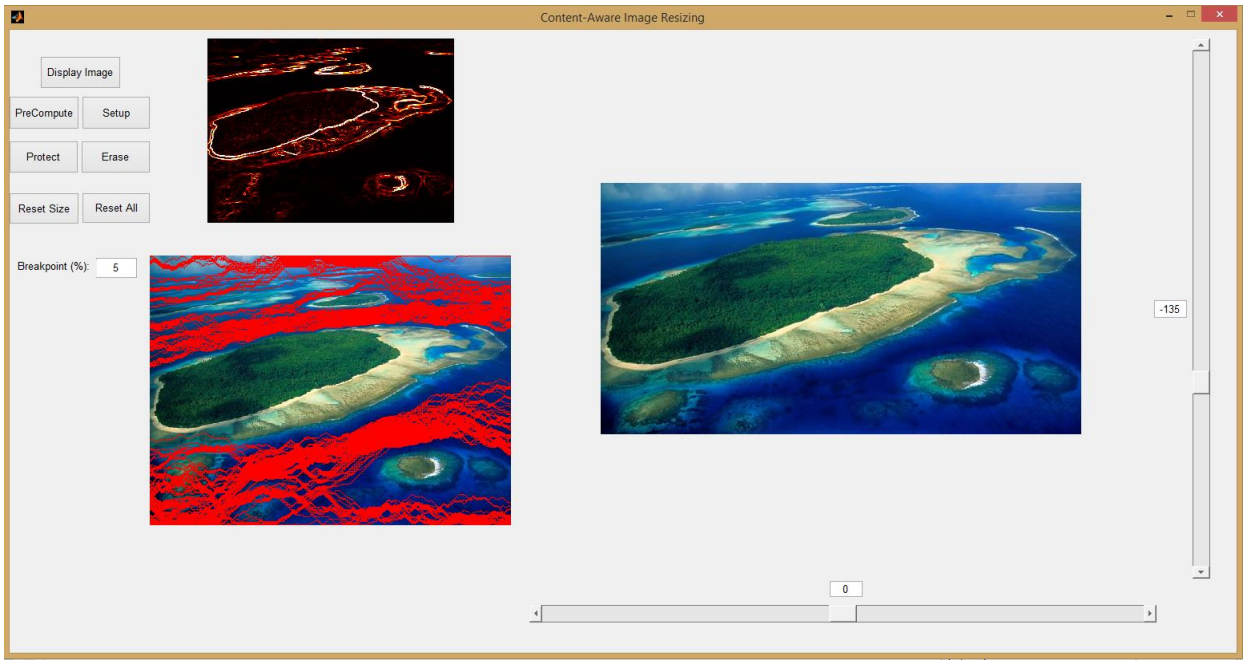

Figure 7: Our GUI which contains scroll bars to dynamically resize the image, buttons to protect or erase parts of the image, buttons to reset the image, and an input box to change the breakpoint. The top left shows the energy map of the image, the bottom left shows the original (shrunken) image with the seams removed overlaid, and the right shows the resized image.

5 Extensions

The main contribution of this work is (1) the exploration of effects of varying some parameters on the computation speed, seam removal priority, and resulting image quality and (2) the creation of a GUI that allows users to adjust these parameters on their own.

5.1 Erasing and Protecting

Sometimes, the seam carving algorithm removes important aspects of the image and preserves parts of the image that are unimportant. In Figure 10, when we try to reduce the width of the original image (Figure 10a) by 108 pixels we see that using our basic seam carving algorithm egregiously removes parts of the man in the foreground who is an important subject of the photo (Figure 10c). On the other hand, the man in the blue patterned shirt on the right is largely unaffected by the seam removal.

To address this, we allow users to specify parts of the image to protect and parts of the image to prioritize removal.

Figure 8: Original image with protection of purple shoe and erasure of pink shoe (top-left); width reduced by 70%, pink shoe is erased while purple shoe remains intact (top-right); width increased by 30%, pink shoe is expanded while purple shoe remains intact (bottom).

In our algorithm, the protected areas (boxed in green in Figure 10b) are given artificially higher

weights, such that anything in a protected region will definitely be removed only after every non-protected region is. The areas selected for erasing (boxed in green) are given artificially lower weights such that the algorithm will prioritize their removal. While a potentially viable solution, we decided not to make the "erased" weights so negative that all of the first seams would be forced to go through the erased regions, but to simply make them 0 so that seams prefer to take a path through the erased regions.

Figure 10d shows the result of having protected the man with the black shirt and erased the man with the blue shirt. While the protection worked well, we see that strongly encouraging some seams to go through the man in the blue shirt caused several artifacts in the lighter grey tiles on the ground sooner than they might have occurred otherwise. The final pair of images (Figures 10e and 10f) show that the man is still very well-protected even with extensive contraction.

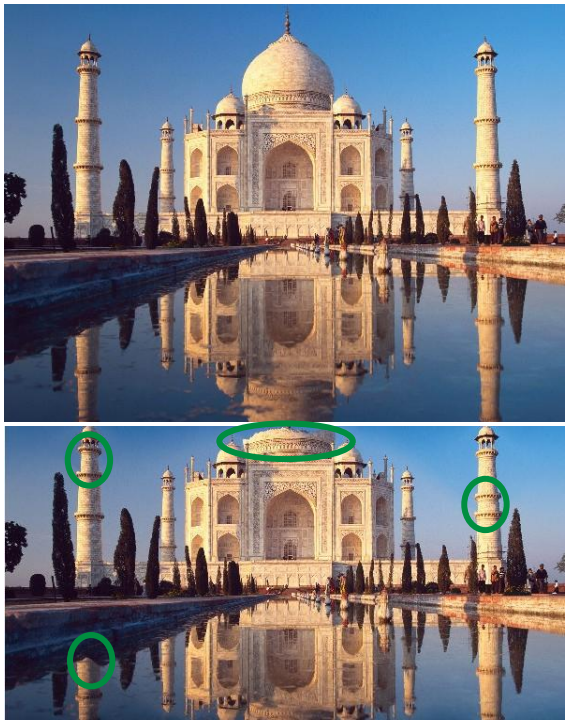

Figure 9: Original image (top); Height reduced by 30%, has noticeable artifacts (bottom)

Figure 10: (left to right, top to bottom); a) First 108 seams overlaid on image; b) Basic algorithm to remove seams; c) User-specified bounding boxes to protect (green) and to erase (orange); d) with protection/erasure; e) Seams overlaid on image with protection/erasure; f) Extensive seam removal based on (e).

6 Discussion and Limitations

Seam Carving works well for images with a clear background and foreground where the background is either fairly homogeneous in texture/color or the features are so small that artifacts are unnoticeable. As we saw very clearly in Figure 1, the algorithm removes uninteresting gaps between foreground objects very well and allows us to change the size of such images without cropping out or distorting important features.

However, we see that this algorithm can also cause serious artifacts when applied to images that have significant lines/edges,

faces/people, geometric structures, and/or symmetry (as in Figure 9). Many of these failures stem from the fact that such structures can be important and have clearly defined boundaries to the human eye, but lack large enough gradients in pixel color intensity to cause the algorithm to see them as important.

In addition, this algorithm as I have implemented it above cannot do both horizontal and vertical dynamic image resizing. This is because vertical and horizontal seams might overlap in more than one pixel. Overlapping in one pixel is inevitable, but overlapping in more pixels causes the seam that is removed later to be missing at least one pixel. There are several ways to circumvent this issue, but most of them are way too computationally expensive. One of the most reasonable ways of dealing with this issue is described in the appendix of Avidan and Shamir's [2007] excellent paper

7 Future Work

We would like to be able to dynamically resize images in both the horizontal and vertical directions as described above. Ideally, this would not be very computationally expensive and would not reduce the quality of the resulting image.

We would also like to experiment with different energy functions (besides e_2) to see which functions work well for which types of images (structured vs. unstructured, distinct features vs. many blended components, etc.). Of course, it is likely the case that no energy function works well for all images, but it is helpful to have a toolbox to draw from when trying to resize a new image of unknown type.

In addition, the paper by Rubinstein et. al describes a method of using "forward" energy to insert seams that would *add* the least amount of energy to the resulting image rather than "backward" energy (insert seams that currently have the least total energy). They mention that this performs better than "backward" energy in some cases, but worse in others. We would like to investigate further the performance differences between these two methods.

Finally, we would like to explore how we might best combine the use of cropping, scaling, and content-aware image resizing to produce a better overall tool for more diverse situations. Sometimes cropping or scaling outperform resizing and it might make sense to do some combination of the three to achieve the best results. The difficult part is to program software that would know or learn when to combine them in how.

There is so much potential for future work on content-aware image resizing algorithms, and the wide range of possibilities for new applications is exciting.

8 References

Avidan, S., and Shamir A. 2007. *Seam Carving for Content-Aware Image Resizing*. In ACM Transactions on Graphics, Volume 26, Number 3, SIGGRAPH 2007

Itti, L., Koch, C., and Neibur, E. 1999. *A model of saliency-based visual attention for rapid scene analysis*. PAMI 20, 11, 1254–1259.

Karl. 2009. *Close all figures except those lised (cab.m)* http://www.mathworks.com/matlabcentral/fileexchange/24420-close-all-figures-except-those-listed

Loomis, J. 2001. *2D Convolution (add_border.m)* http://www.johnloomis.org/ece563/notes/filter/conv/convolution.html

Rubinstein, M., Shamir A., and Avidan, S. 2008. *Improved Seam Carving for Video Retargeting*. In ACM Transactions on Graphics, Volume 27, Number 3, SIGGRAPH 2008.

Santella, A., Agrawala, M., DeCarlo, D., Salesin, D., and Cohen, M. 2006. *Gaze-based interaction for semiautomatic photo cropping*. In ACM Human Factors in Computing Systems (CHI), 771–780.

Viola, P., and Jones, M. 2001. *Rapid object detection using a boosted cascade of simple features*. In Conference on Computer Vision and Pattern Recognition (CVPR).