Content-Aware Image Resizing Using a Greedy Seam Carving Algorithm

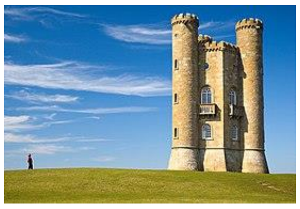
Zayd Muhammad Kawakibi Zuhri - 13520144
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13520144@std.stei.itb.ac.id

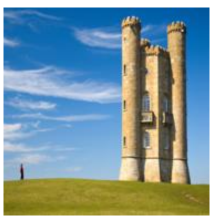
***Abstract*—The need of flexible digital images is ever-growing in these times of advancement. In 2007, Avidan and Shamir proposed a simple, yet effective method of image retargeting named “Seam Carving”, making use of a dynamic programming algorithm. This paper aims to implement a more straightforward, simple, and lightweight alternative to Seam Carving by using a greedy algorithm for seam generation. The resulting images are resized in a content-aware fashion, preserving essential features of an image.**

***Keywords*—image resizing; content-aware image retargeting; seam carving; greedy algorithm**

I. INTRODUCTION

With the advancement of technology comes along the ever-growing need of even better digital media. The wide variety of devices and displays demands more flexible media to adjust accordingly in various situations. Not to mention the wide-spread use of videos and photos in everyday lives of the average person requires accessible ways of manipulating media to one’s needs. Web-based content is becoming more diverse and dynamic, with various platforms adapting different layouts to deliver the optimal user experience. Yet even though images are the most used media in recent times, they remain static and are limited to the ways they can be scaled and sized.


(a)


(b)

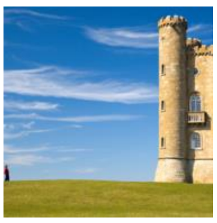

(c)

Figure 1: Classic Image Resizing, (a) Original Image, (b) Image Scaling, (c) Image Cropping

In cases where the size or aspect ratio of an image must be changed, classical image resizing techniques lack finesse. Normal image rescaling only stretches and deforms images to a desired size and does not take into consideration the quality of the image thereafter, resulting in weird proportions of objects and various distortions of features. Image cropping gets rid of pixels in images completely, which might work fine on images with a single object or feature, but more intricate images might contain multiple important features that are harder to consider when cropping, especially if those features reside on the edges of an image, resulting in a loss of information. To achieve better image resizing in terms of preservation qualities, a technique is needed that takes the contents of the image into consideration.

Enter image retargeting, which aims to achieve image resizing whilst keeping the important features intact. Several approaches have been explored, such as top-down methods by Viola and Jones (2001) using face detectors to ascertain the inclusion of people as important features of the image, then using scaling and cropping to resize without losing those features. Other methods may be bottom-up, such as the one developed by Itti et al. (1999) that makes use of the construction of a visual saliency map of the image in question to detect important features. Although these methods can achieve significant results in some cases, they still utilize classical image resizing techniques and therefor are also limited by them. Other methods such as the one by Setlur et al. in 2005, utilize an automatic algorithm to decompose an image to a background layer and foreground objects, segmenting them, resizing the background, filling in the gaps, then reinserting the important objects back into the resized image. This also achieves impressive results, but also needs plenty of overhead processing time.

In 2007, Avidan and Shamir proposed a much simpler method of image resizing, giving it the term “seam-carving”. The operation enables the changing of the size of an image by carving out lines of pixels in the image, so-called “seams”. These seams are connected paths of pixels that have the least energy in the image, with energy defined by an energy function, that can be chosen at will. Possible energy functions are gradient magnitudes, image entropy, visual saliency, eye-gaze data, and the like. Connected pixels are chosen from top to bottom or left to right depending on whether one needs to resize horizontally or vertically. The algorithm for choosing this low-energy seam can be any shortest-path type algorithm such as Djikstra's algorithm, greedy algorithm, graph cuts, or, as chosen by Avidan and Shamir, a dynamic programming algorithm. These seams are then successively removed or inserted to reduce or increase the size of an image. By following an energy function while generating seams, carving them out effectively results in content-aware resizing of the image.

This paper will attempt to implement an alternative version to the original seam-carving algorithm. Where the original uses a dynamic programming approach in seam generation, I will implement a more straight-forward, lightweight, and intuitive approach by using a simple greedy algorithm. This approach is expected to be much more memory-friendly since it does not require the storing of tables and potential seams that are crucial in the dynamic programming algorithm. This may result in a trade-off with the quality of the chosen seam, but the memory and time saved in comparison should be favorable. Do note that this implementation will be limited to reducing image sizes and will not support the image enlargement by seam insertion possible in the original. The energy function chosen in this implementation is the Sobel operator, which acts as an approximation of the gradient of the image intensity function. The hope is to obtain results as effective as the original in terms of image size reduction quality, delivered in a more compact and intuitive package.

II. THEORETICAL BASIS

A. *Digital Images and the Sobel Operator*

A digital image is made up of picture elements, so-called pixels, that each contain discrete digital values. Raster images contain a fixed number of rows and columns of these pixels, that are the smallest elements of an image, containing finite values that represent the brightness of a given color. Typically, pixels are arranged in an ordered rectangular array, thus, the size of an image is determined by the dimensions of this array, with the width being the number of columns and the height being the number of rows of pixels. When unpacked, each array element contains an array of integer values, e.g., 3 values in an RGB image represent the intensity of the colors Red, Green, and Blue. Some file formats such as PNG also support an alpha value, which translates to the transparency of the particular pixel. Single value pixels can be interpreted as black and white images in grayscale.

The Sobel operator, also known as the Sobel–Feldman operator or the Sobel filter, is used in image processing and computer vision to emphasize edges in images. Most useful in edge detection algorithms, it is named after and first presented by Irwin Sobel and Gary Feldman in 1968. The filter effectively approximates the gradient of the image intensity function, resulting in the norm of the gradient vector at each point of the image. This operator involves convolving the image in the horizontal and vertical directions using a tiny, separable, integer-valued filter, and so is computationally inexpensive, but provides a rough estimation especially for high frequency fluctuations and features in the image.

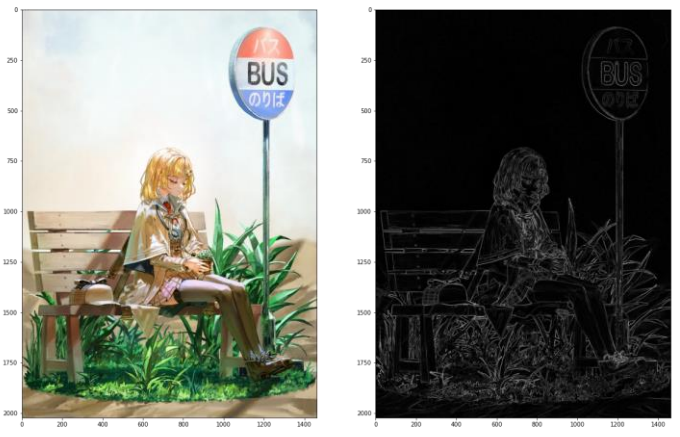


Figure 2: Left: Original image, Right: Applied Sobel filter to grayscale version of the original

B. *Greedy Algorithms*

A greedy algorithm is any algorithm that solves problems step-by-step by choosing the best choice at every step of the way without taking into consideration the consequences of that choice. Thus, it does not care for the future and stands by the "take what you can get now!" principle. Every step of the way the algorithm can only hope to achieve the global optimum by always choosing the local optimum. Greedy algorithms do not guarantee an optimal solution but can approximate a globally optimal solution in a fraction of the time of more complex algorithms.

In terms of seam-carving, the elements of a greedy algorithm can be broken down as follows:

- The candidate set: Contains the candidates to be chosen at every step, i.e. all pixels in the image.
- The solution set: Contains chosen candidates that constitute the solution, i.e. pixels at the seam line.
- The selection function: Decides the local optimum at every step, i.e. pick the minimum energy pixel
- The feasibility function: Decides whether a candidate can be selected or not, i.e. test for neighbor pixels
- The objective function: Decides optimality for selection, i.e. minimum of energy function

A greedy algorithm's decision may be influenced by previous decisions, but not by future decisions or all possible solutions to the subproblem. In other words, a greedy algorithm never reconsiders its choices. This is the primary distinction between it and dynamic programming, which is exhaustive and thus guarantees the globally optimal solution. At every step of the way, dynamic programming makes choices with all the previous choices in consideration and may result in a reconsideration of the optimal path given by the algorithm. This is where the overhead of memory and time comes into play, because of this all the possibly optimal paths are stored to reconsider in the future. This is not the case with a greedy algorithm, where it does not need any more memory than that needed to store the solution, nor extra time to reevaluate. This trade-off is not a problem in the case of seam carving.

III. Implementation

A. *Image Preparation*

Before resizing, an image will have go through a couple of steps as preparation. We will keep track of two versions of an image: its full image, and an energy matrix. The energy matrix is the result of the Sobel filter and is essentially a two-dimensional array the size of the image, containing energy values given by the Sobel operator. This will be the matrix that is used by the greedy algorithm. To create this energy matrix, the image must first be converted to a grayscale image. Then, to amplify the effect of the Sobel filter and compensate for images with sub-optimal lighting, exposure, and the like, we will rescale the intensity of the image. Also, to correct the exposure, we equalize the image histogram. Only then will the Sobel filter be applied on the grayscale image, thus resulting in the energy matrix.

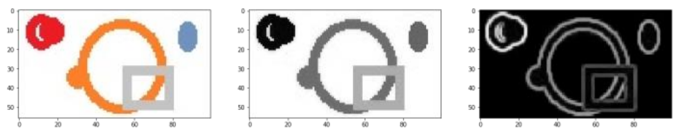


Figure 3: Left to right: Original image, grayscale image, and energy matrix

B. *Seam Generation*

With the energy matrix in hand, we can start generating seams to later remove. In this implementation of seam carving, seams are only generated one-by-one and not multiple at once, unlike the original implementation. A seam will be generated and removed, and only after the removal will another seam be generated and removed, so on until the desired size is reached. This enables much faster and efficient seam generation.

Seams will be created using a greedy algorithm. We will need a starting pixel in order to start carving out a seam, since we will only be generating one seam at a time. The starting pixel will be on the first row or column, in other words at the top or on the left side of the image, depending on whether we are currently resizing in the horizontal or vertical direction. The first intuitive option is to randomly pick a starting pixel and generate a greedy seam from there, but this will result in seams that most likely interfere with important features of the image. Thus, a heuristic approach is taken here to determine the starting pixel, by first calculating the sum of all energies in all columns when choosing from the first row, or from all rows when choosing from the first column. This will give an approximation on how important that particular row or column is in terms of the containing features of an image. By choosing the minimum from this approximation, we essentially choose the starting pixel and pixels below, or to the side of it, with potentially the lowest energy seam in the entire image. In other words, this is our heuristically greedy shortcut or alternative to exhaustively comparing multiple seams. Determining this starting pixel is essential to generating an optimal seam that, hopefully, does not interfere with important features of the image.

After choosing a starting pixel, we can begin generating a seam originating from it. We define connected pixels as three neighboring pixels in the next row or column. When constructing a seam horizontally, that would be the pixels on the next column, one in the row above, one in the same row, and one in the row below. When constructing a seam vertically, the connected pixels would be the pixels in the row below, one in the column to the left, one in the same column, and one in the column to the right. In other words, we shall view the energy matrix as a matrix of 8-connected pixels, but only consider the connected pixels in the direction of the seam being constructed.

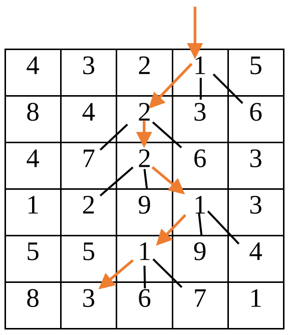


Figure 4: Visualization of seam generation, with the numbers in the cells being the pixel energies

We iterate from the starting pixel the same amount of times as the height of the image when generating vertically, or the width of the image when generating horizontally. As specified in the greedy algorithm, the pixels will be chosen from the next neighboring pixels by their energies, choosing the pixel with the minimum energy at every iteration, until the seam reaches the other end of the image. This selection of pixels is stored in an array of indices, where every element with index i, containing an index j, represents a pixel at the seam at the row i and column j when generating vertically, or row j and column i when generating horizontally. This process effectively creates a line the width of 1 pixel along the axis of resizing, containing pixels of little importance to the overall image. When compared to the process of image cropping, it essentially does the same thing. Cropping gets rid of a straight line of either vertical or horizontal pixels, most probably at the edges. Seams are, in other words, lines of dynamically cropped pixels, since the amount of pixels removed at every row or column is the exact same. By dynamically doing this with respect to the energies of the pixels, these seams should avoid important features and preserve objects as seen by the Sobel filter.

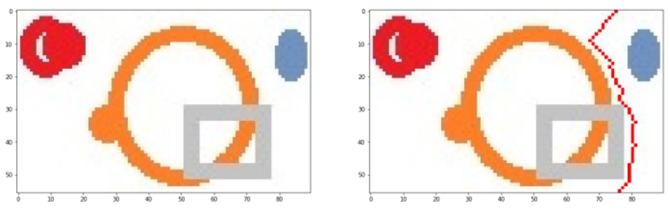


Figure 5: Original image on the left, Generated vertical seam illustrated in red on the right.

C. *Seam Removal*

After generating a full seam from edge to edge, we can finally remove that seam to reduce the width or height of the image by one pixel. An important thing to remember is that we have to remove the seam both in the image and in the energy matrix, so that we don't have to recalculate the energy matrix after every seam removal, so all removal operations will be done both on the image and the energy matrix to pass on to the next iteration. First we initialize new two-dimensional arrays that are one pixel shorter in the direction we are resizing. These will be filled in row-by-row with the original rows, but with the single seam pixel deleted. To simplify the deletion pixels in columns when resizing the height of an image, the rows and columns will be swapped in the case of horizontal seams, since deleting column-wise is mostly unsupported. Vertical seams are already in the ideal form, so we can begin deleting and inserting pixels. We iterate through the height/width of the image, deleting the pixel at the index specified in the seam array, then inserting that row into the new matrix. After every row has been copied without the seam, we can swap back the rows and columns for horizontal seams, and return the resized image and energy matrix.

From testing both seam generation and seam removal, there seems to be a great discrepancy between the two of them in terms of computation time. This difference can reach a whole order of magnitude slower. The time needed to remove a single seam can be 10 times slower than the time needed to generate that very seam. It seems like the problem lie not with the greedy algorithm, but a major bottleneck of the seam removal algorithm in this paper's implementation. Alternative solutions to this problem will be discussed further in the conclusion section.

D. *Resizing Down to A Specified Size*

Now that we know how to generate a seam and remove it from an image, we can dynamically resize images down to any size or, in other words, resolution. This can be done in a number of ways, depending on the current application and implementation of the seam-carving algorithm. In the case of a more manual approach, the user will input an image, and the desired size containing the width and height of the output. Given these variables, we can start the process of seam-carving by first generating an energy matrix from the image. This only has to be done once. After that, we can loop through seam-generation and seam-removal in the horizontal direction x amount of times, with x being the difference between the original height of the image and the desired height. We can do the same for the vertical direction, reducing the original width of the image to the desired width. After these two loops are done, the resulting image will be in the desired width and height.

In the case of web applications, images should be able to change dynamically according to the view the browser currently finds itself in. Since the algorithm itself is lightweight in nature, it should be able to be implemented on the client and displayed through HTML/CSS. I have not implemented this application, and it is of further interest to expand it.

IV. Experimentation

In this section we will explore the capabilities of the seam-carving algorithm, but also its limitations. Since it is a relatively simple algorithm, the results given by it will vary drastically on a case-by-case basis. The resulting sizes are chosen to demonstrate the algorithm at various edge cases, and are not random. To start off, we will first be looking at an extremely simple image of shapes on a white background. The image is very small at only 56 x 100 pixels, and will be a great look into how the algorithm works at the smallest scale.

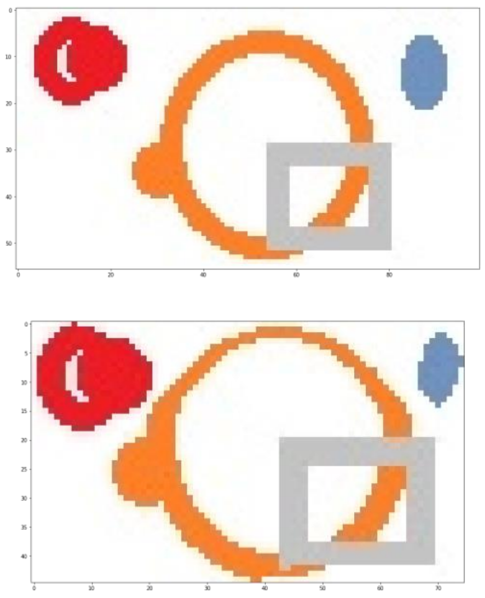
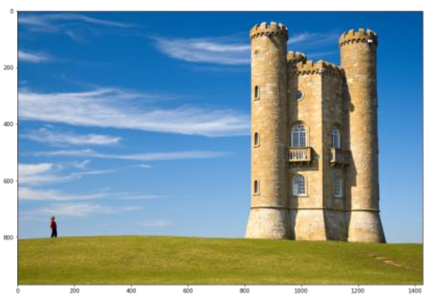


Figure 6: Simple shapes, Top: 56x100, Bottom: 45x75

As seen in Figure 6, a reduction of almost 20% of the original height and 25% of the original height gives us an interesting result. First of all, the blank spaces at every edge of the image are, expectedly, gone, since they are trivial in the context of size reduction and *should* be removed first and foremost. But those reductions are not enough, and that is where the algorithm steps in to create seams *in-between* the objects, or in this case, shapes. Intuitively, we would want to shrink the image so that the objects are closer together and will not be deformed. The algorithm achieves this relatively well. Looking at the distance between the red shape and the orang shape, we can see clearly that they are almost touching one another in the final image, a result of trying to squeeze them together in a limited space as much as possible. However, we can also see the orange shape deforming at the top. This is because of the limitations of the seam itself, which cannot go sideways at an angle more than 45 degrees. This is why as it carves a seam between the orange and red shapes, it 'shaves off' the top of the orange shape. other than that, we can also see a deformation of the blue shape on the right. It seems like the algorithm could not shove it further to the right, so it had no choice but to shave off the side of the blue oval. This seems to be cause by the same limitation as the deformation happening at the top of the orange shape. The blue oval cannot be further moved to the left since the angle of its sides relative to the orange shape and grey rectangle is too steep to be carved by a seam.



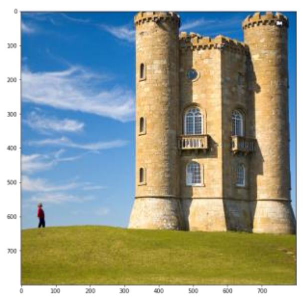
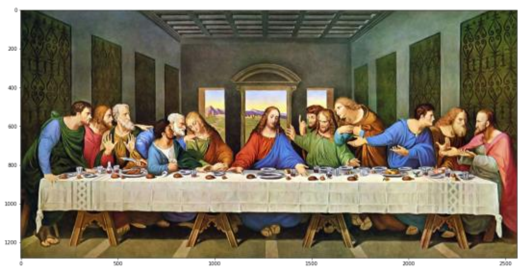


Figure 7: Man and Tower, Top: 968x1428, Bottom: 800x800

The image in Figure 7 is truly the ideal use case of this algorithm. A clear background and foreground, with easily defined objects, a man and a building. These two features are separated in a way that would be hard to resize using traditional image resizing techniques like cropping, since they would be cut off. But using the seam carving algorithm, it manages to preserve these features extraordinarily well. The main places where the algorithm chose to cut off were the trivial edges, and intuitively the space between the man and the tower. Oddly enough. the algorithm carved seams that are sparse enough to the point where the grassy ground and the sky with the clouds does not look distorted at all. But, the algorithm may have shortened the top of the tower just a little bit, where it could have just cropped the bottom of the image. This is likely because of the Sobel filter registering the grass as a more energy-heavy feature than the top of the tower.



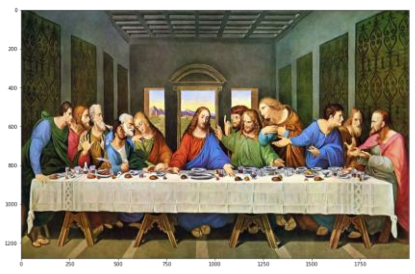
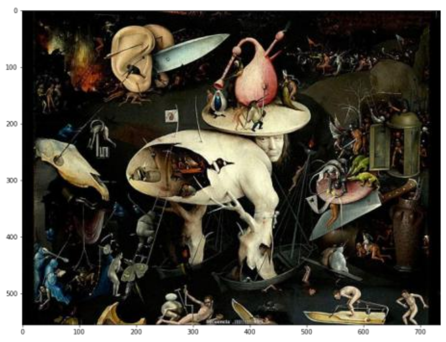


Figure 8: Last Supper, Top: 1280x2560, Bottom: 1280x2000

Figure 8 features the famous image of the Last Supper, going through a 28% reduction in width. This showcases the capability of the algorithm to compress a lot of features into the desired size. Jesus and the Twelve Apostles represent a challenge for the algorithm since there is barely any room to crop off width-wise. Yet the algorithm manages to squeeze them in in a way that is not too disruptive. A traditional scaling of the image would certainly result in a far more distorted final image. When looking at the rectangular wall ornaments at the background, it becomes clear where the algorithm chose to carve. The spaces between them are far closer compared to the original image, but the rectangles themselves have not been distorted by an observable amount. Of course, there is no ignoring the skewed result that is that the left side of the table is more compressed than the right side. This is most likely the result of a bias in the energy matrix from the Sobel filter, where the edges detected on the left are more vertical than on the right, hence resulting in a lower resistance to creating seams on the left. Notice that the faces of most of the apostles are mostly kept in tact, only distorted to a certain degree. This is most successfully done on the right side of the table, where it was not objected to the energy bias. Most notable are the apostle wearing an orange cape and the apostle with a red hood to the very right of the image. Their faces and body proportions are kept relatively in check. Also notable are the plates, food, and cutlery on the table. The spaces between them have obviously been carved off, and the resulting objects on the table are relatively safe and still look like the original. Of course, when looking at the image as a whole, it is easy to overlook the work of the algorithm and pass it off as traditional image scaling, the eyes are attracted to the distorted apostles on the left of the image. Nonetheless, this is quite the amusing result considering the challenge of compacting such an already compact image, with many objects spanning its entire width.



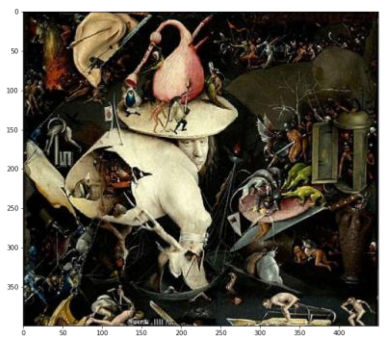
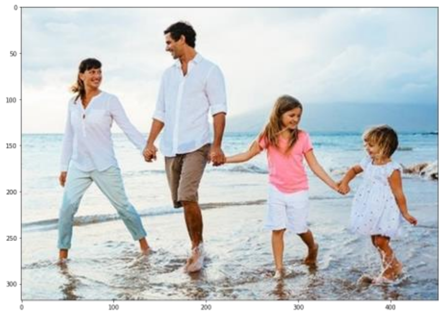


Figure 9: Bosch’s Hell, Top: 555x736, Bottom: 400x450

Another famous painting in Figure 9 is the representation of hell by Hieronymus Bosch. An extremely busy image, and certainly an edge case for the seam-carving algorithm. Yet with all the business in it, it should still be recognizable in terms of the objects and features inside it Starting from the left side is where the algorithm chose to cut off the most content. The skull has been squeezed into the rear end of the main object in the image, the “witch tree”. Also notable is the ear and knife at the top of the image, where the knife has been carved off almost completely to make space for the pink bagpipe atop the witch’s head. Even so, the ear and the gate to the left of it have been preserved well enough. Aside from the quirkiness of the left side, the middle and right side of the image have been mostly untouched. The witch has been slightly distorted, in favor of the odd structure on the right side. This is most likely another product of bias on the energy matrix.



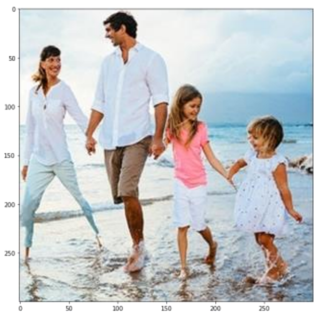
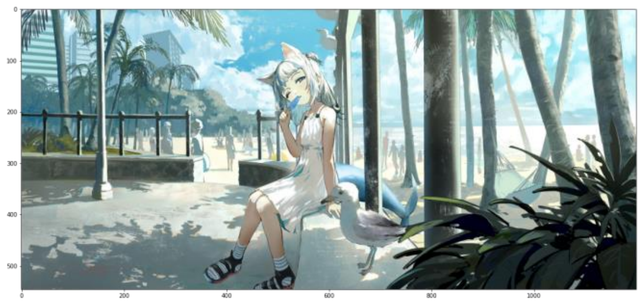


Figure 10: Family, Top: 318x452, Bottom: 300x300

This example in Figure 10 represents a more real-word case of image resizing. This family of four is spread out evenly length-wise across the image. How does the algorithm handle this if a more compact image is desired? A human eye would see the spaces between the family members as the most obvious place to save space, assuming the hands can be reduced in an unobtrusive way. As one can see, the algorithm seems to do the exact same thing. Most notably, the space between the father and the daughter has been reduced by a considerable amount. Even so, the resulting image looks relatively natural. Space between the other family members have been carved out too, as can be seen by the slightly distorted hands. This is quite impressive, and show the power of this lightweight algorithm. Of course, we cannot ignore the obvious distortions present in the resulting image. The legs of the mother have been considerably distorted, likely a result of a seam coming from the gaps above the legs. The algorithm also chose to distort the top of the image, the head of the father, rather than crop the bottom of the image.



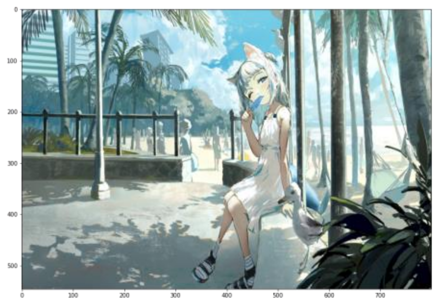


Figure 11: Sitting Girl, Top: 546x1200, Bottom: 546x800

The case presented in Figure 11 is an example of where this algorithm should not be applied to and fails to outperform traditional image resizing. The image of a girl is obviously a very easy image to crop down to a smaller size. Since there is only a singular object of interest in the middle of the image, the average person would just crop the left and right of the image containing relatively unimportant features, such as trees on the left and empty space on the right. This operation would leave the girl in the middle untouched and perfectly fine. Yet the algorithm sees the features on the left and right with the same energy and importance as the main subject of the images and chooses to apply a relatively even carving of multiple features in the image. Interestingly enough, the algorithm also chose to leave the right side of the image intact. The energy matrix seems to have weighted that side of the image heavily, compared to the middle and right side. The most obvious problem in this case is the distortion of the main subject of the image, the girl sitting in the middle. One wouldn’t mind the distortion on the trees, but seeing that if this image were to be further shortened and the algorithm distorts the girl even more, that would be a failure. The girl’s legs are severely distorted and the body unnecessarily compacted by the seam carving.

These examples go to show the wide range of results the algorithm can manage to achieve. Although most of these are heavily cropped at their widths, the same also apply to taller images that are compacted into shorter resized images. Most of the undesirable artifacts created through seam-carving, such as distortion and uneven resizing, seem to come from the same place. There is a certain bias and amount of error in the energy function, that is very limited by the way the Sobel filter functions. If we look further into the algorithm, the process of choosing the starting pixel is also very biased towards the left or top side, since the first instance of potentially multiple minimum energies is taken, and are not chosen by random. This causes heavier carving on the left or top side of relatively symmetric images.

V. Conclusion

The implementation of a greedy seam-carving algorithm demonstrated in this paper is quite variative in terms of the quality of resizing it does on an image. It really shines in cases where objects and features are seperated in front of a clear background, but struggles when the edge separation is unclear, whether an object belongs in the foreground or is in one with the background. This is of course a limitation caused by the Sobel filter, since it really only can go so far in detecting edges, and is easily confused by more obscure images.

Other than the quality of the resulting images, this implementation is heavily bottlenecked by several factors, hence why computing time is not heavily discussed in this paper. The implementation I made was created using Python, a relatively slow language. Adding to that, the algorithm for seam removal is abyssmal in terms of speed, and is a whole magnitude slower than generating the seam itself, as mentioned in the implementation section.

Putting aside the poor qualities of this specific implementation, I believe the attempt at a greedy seam-carving algorithm itself was relatively successful. It achieved surprisingly smart results using a very straightforward algorithm, that has the potential to be implemented even better. Improvements can be made in several areas, such as the energy function used, the heuristic determining of the starting pixel, a randomized selection from multiple minimum pixels, and of course a better seam removal algorithm.

Acknowledgment

I would like to thank Dr. Ir. Rinaldi, M.T. as my lecturer in class IF2211 for Algorithmic Strategies, for giving me this chance to write on this topic, and constantly pushing and supporting us students. Thanks to his guidance, I have gained a much better understanding on algorithms and the like, essential to the writing of this paper. An appreciation also goes to his writings that were very helpful in understanding these topics, and his diligence in keeping up his website, full of resources and references.

Video Link on YouTube

https://youtu.be/mklKpaamdrE

References

[1] Avidan, Shai; Shamir, Ariel (July 2007). "Seam carving for content-aware image resizing | ACM SIGGRAPH 2007 papers". Siggraph 2007: 10. doi:10.1145/1275808.1276390.

[2] Rubinstein, Michael; Gutierrez, Diego; Sorkine, Olga; Shamir, Ariel (2010). "A Comparative Study of Image Retargeting" (PDF). ACM Transactions on Graphics. 29 (5): 1–10. doi:10.1145/1882261.1866186.

[3] Munir, R., 2021. Algoritma Greedy (Bagian 1). [online] Informatika.stei.itb.ac.id. Available at: <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf> [Accessed 20 May 2022].

[4] VIOLA, P., AND JONES, M. 2001. Rapid object detection using a boosted cascade of simple features. In *Coference on Computer Vision and Pattern Recognition (CVPR)*

[5] ITTI, L., KOCH, C., AND NEIBUR, E. 1999. A model of saliencybased visual attention for rapid scene analysis. PAMI 20, 11, 1254–1259.

[6] SETLUR, V., TAKAGI, S., RASKAR, R., GLEICHER, M., AND GOOCH, B. 2005. Automatic Image Retargeting. In In the Mobile and Ubiquitous Multimedia (MUM), ACM Press.

Declaration

I hereby declare this paper as my own writing, by my own hands, and not adapted, translated, nor plagiarized from any other existing works.

Bandung, 23rd May 2022

[illegible]

Zayd Muhammad Kawakibi Zuhri, 13520144