Al Akhawayn University

Capstone Project

Content Aware Image Resizing Using Seam Carving

Ayman Bentourki

supervised by

Dr. Naeem Sheikh

May 3, 2016

Contents

Approved by The Supervisor: Dr. Naeem Seikh

Acknowledgment

First of all, I would like to express my sincere gratitude Dr. Naeem Sheikh for his enormous help and his availability throughout this capstone project. Also, I would like to thank my family and friends for their continuous support.

My gratitude also goes to the school of science and engineering, as well as most of my professors who contributed to making me a well educated person.

1 Abstract

One of the most successful algorithms in the image resizing domain is Seam Carving introduced by Shai Avidan and Ariel Shamir. The main purpose of this capstone project is to analyse the performance of this algorithm and its effectiveness for different types of images and propose optimisations to it.

Furthermore, this project will explore energy functions, whose purpose is to evaluate the importance of images' content, and find out which energy functions perform better on which types of pictures and features.

Finally, we will discuss the possibility of incorporating the algorithm within an application and taking other new variables such as user input into consideration.

2 Introduction

Digital image processing is a very significant and important component in the field of computer vision. Whether it is used to improve pictorial data for human interpretation, or used to optimize computer processing of images, image processing can be used in different other fields such as mathematics, optics, visual physics etc. Applications of image processing are numerous and vary from object recognition to remote sensing and optical sorting etc. Images, which are definitely the main variable in image processing, come in different sizes. Therefore, resizing them is a very important issue. Resizing pictures can be useful for human interpretation in order to keep the important information of a certain picture intact across different sizes. Also, it can be used for computer processing, through compression for example, since this process using some wavelets can only work on exact sizes.

3 STEEPLE Analysis

In the wake of recent technological advancements, devices have started to come in different sizes and different shapes. Different shapes and sizes means different screens, and one of the issues web developers find with this variation of screen sizes, is responsiveness. Many technologies and frameworks are created nowadays to allow responsive design. However, pictures are much harder to adjust to the size of different screens, as resizing them while taking into consideration only the size and ignoring the content yields unpleasant pictures. That is why the seam carving algorithm is useful in the technological aspect, for it helps in producing these dynamic and responsive web pages by resizing pictures in a content-aware

fashion.

In the social aspect, this technology will allow people browsing the internet on small screens, to see the important details in many pictures. Zooming in on pictures to catch a specific detail may result in a blurry picture and may hide what the user wanted to see in the first place. Also, since the application is accessible through the web, it can make the job easier for users who only want to resize pictures and do not need to download a whole application that has a huge number of unneeded functionalities.

Finally, in the economic aspect, the algorithm and other technologies used in the project are open source. Therefore, It might help startups who just began in the IT world, or even outside it, since they can use it and modify it freely.

4 Methodology

The first step in the seam carving project is to think about the algorithm. Dynamic programming (DP), as a concept, is very simple to understand, considering that it just consists of not computing the same operations multiple times. However, every situation that requires a DP solution is different, and deriving an algorithm for that specific situation might be difficult at times. Fortunately, the algorithm is already created, and what is left is to implement it, and also understand all the details about it, in case any improvements need to be done for this specific project.

Furthermore, this algorithm works on the energy of the pixels to cut down the least important pixels of the picture. Here, the energy of a pixel is a measure of its importance in the picture. This idea can be captured through various different mathematical functions. I

will start by using the most basic one which deals with computing gradients. Whether it is the magnitude or the orientation that we are concerned about, there are many variation to compare gradients and many energy functions combine these variations in their own way. It is very important to compare all major energy functions and find out which ones are good for which kind of pictures.

Python is the main programming language in which the algorithm is going to be implemented. With the help of Pillow which is a fork of PIL (Python Imaging Library), the task of dealing with images and manipulating the pixels is made a lot easier and more efficient.

5 Image Resizing

As mentioned in the previous sections, picture resizing is very important for various purposes. In order to resize a picture, the first idea is to do it through scaling. Scaling may be a good idea if the aspect ratio change is not very big. There are also many ways in which scaling can be done such as linear or cubic interpolation. Depending on the picture and the variation of scaling, it could be an enough process. However, in the majority of cases, and especially when the change in the aspect ratio of the image is noticeable, scaling is undesirable. It may produce a squish effect as shown in Figure 1.2. This latter is eliminated by seam carving as it removes pixel with low importance rather than keeping all them and trying to fit them all within a different size picture. Scaling can be seen as trying to move a set of items from your big bag to your small backpack for a trip even if you know that it can't hold that huge number of items. What happens to your items? They get deformed and distorted because they don't have enough room.

An alternative to scaling for resizing that is also very easy and straightforward is cropping. Cropping is a technique that deletes entire regions of the original picture to reach the desired size. As shown in Figure 1.4, this technique can also lead to undesirable result, and elimination of some important content of the image. Cropping can also be applied to our example, and it just throws away the set of times that are next to each other and easier to get rid of. It can even cut out the backpack.

(a) Original image

(c) Cropping

(b) Scaling

(d) Seam Carving

Figure 1: Comparison of three different ways of resizing

How do you fit all the items in the backpack? The solution is to remove the items that are least important to you, and remove enough so that the remaining items can fit. However, for images it is a little bit more complicated than that, as pixels are related to their neighbours.

First, we have to find some way to calculate the importance of a pixel for the picture. We will start to refer to this quantity as energy. The higher the energy of a pixel the more important this pixel is, and removing it will have more impact on the picture than removing a lower energy one. Energy functions are a crucial part of seam carving. The algorithm is standard for all kinds of pictures, yet energy functions differ, and choosing the right function for the right picture can result in very good results. We will explore many energy functions in the subsequent sections and compare and analyse them.

Let us say we have a picture that is 100×100 and you want to retarget it to a 100×80 picture, and we have each pixel's energy calculated. How do we proceed with the elimination of pixels?

5.1 Optimal Deletion

First idea that comes to mind is deleting set of least important pixels(20×20 in this case) and shifting your picture left or tight every time a deletion is made. This way we will preserve the maximum energy possible on the expense of the shape of the image, because this technique will end up deleting different numbers of pixels in every row. It is safe to say that preserving the shape is more important as we can see in the next picture.

Figure 2: Globally deleting pixels

5.2 Local Row Deletion

An alternative to the optimal measure is to consider each row on its own. This measure preserves the shape which is an improvement in consideration to the previous technique, since the same number of pixels gets deleted from every row. However, the content still gets distorted to a great extent because some pixels from very different and far apart columns get joint up in many cases, as we can see in the following figure:

Figure 3: deleting the lowest energy pixels from every row

5.3 Entire Column Deletion

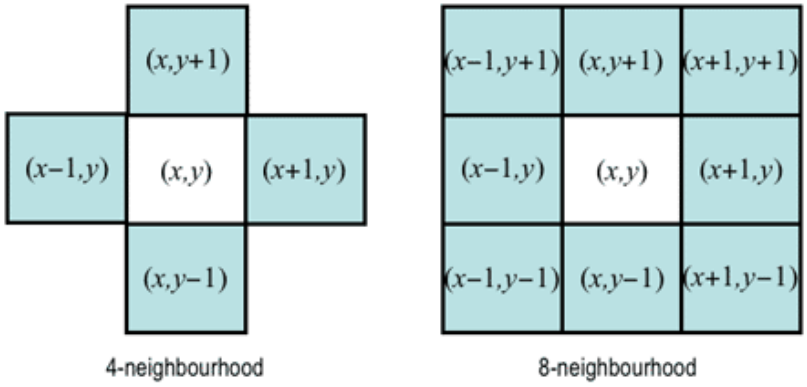
Column deletion is the closest one to seam carving, and the simplest essentially. In column deletion we delete the columns that have the lowest energy total. It distorts less the content of pictures, but it does not allow you to be very selective on the content you want to delete. Deleting only columns does not allow, for example, to avoid edges which are not linear or neither horizontal nor vertical. Also, deletion of columns can produce unsightly seams (where two different color tones meet discontinuously).

Figure 4: Column Deletion

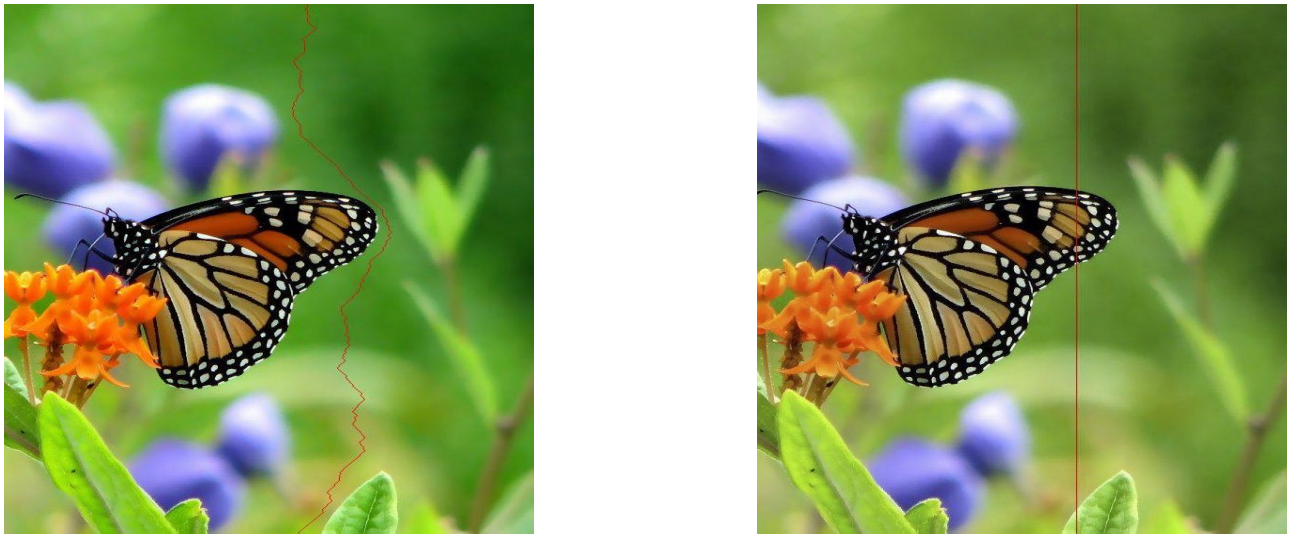
Figure 5: Seam Carving

5.4 Seam carving

A seam is an 8 connected path of pixels in a single direction of lowest energy. As shown in the picture, a pixel in a 8 connected path can be connected to any of its 8 neighbours. However, since the seam is going in a single direction, the algorithm, when computing the seams, has only the choice between three pixels, at each pixel. These, in the case of a vertical seam going downwards for example, are: $(x-1, y-1), (x, y-1), (x+1, y-1)$.



Therefore, the objective of the algorithm is to cut down as many seams of lowest energy as possible to reach the desired size. After deleting the seam, the pixels have to be shifted left or right (or up or down) exactly as we mentioned before except that the shape, and the content to a certain extent, is still preserved. Furthermore, as shown in the next figure a seam allows you to avoid edges in the picture, which is often not allowed by column removal. The set of pixels to be deleted is highlighted in red.



(a) Seam-removal (b) columnremoval

Figure 6: difference between column and seam removal

6 Vertical Seam Removal

6.1 Process

Now that we know what a seam looks like, and given that we have the energy of each pixel calculated, how do we find the seam with the lowest energy to remove? In this section we will only tackle vertical seam removal, and everything mentioned in this section can be applied to the horizontal approach.

Initially, we have to consider every possible path (that takes one pixel from each row) from the starting row of pixels to the ending row. In order to perform this, a backtracking approach is not practical (because of the huge number of such paths possible) and we have to go through a dynamic programming approach.

The idea of this approach is to go through each pixel, and record in that pixel the energy of the optimal seam going through that pixel. This way, when moving on to the next row of pixels, we'll be able to use the values for the previous row to find this row's values. The flow of the algorithm is explained in this next example.

Here, we have a 4×3 picture. Every entry in the matrix represents a single pixel in the form of a tuple (x, y, z) where x is the energy of the pixel, y is the cumulative energy leading up to that pixel, and z represents which parent leads to that pixel, $z \in \{-1, 0, 1\}$, -1 for the left parent, 0 for the middle parent, 1 for the right parent.

Initially, our DP matrix looks like this:

$$\begin{bmatrix} (4,0,0) & (5,0,0) & (1,0,0) \\ (7,0,0) & (5,0,0) & (8,0,0) \\ (3,0,0) & (4,0,0) & (6,0,0) \end{bmatrix}$$

For the first row, the cumulative sum is equal to the energy of the pixels, and every pixel has no predecessor therefore $z = 0$

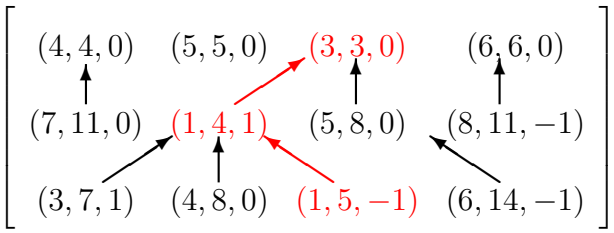
$$\begin{bmatrix} (4,4,0) & (5,5,0) & (3,3,0) & (1,1,0) \\ (7,0,0) & (1,0,0) & (5,0,0) & (8,0,0) \\ (3,0,0) & (4,0,0) & (1,0,0) & (6,0,0) \end{bmatrix}$$

For the next rows, we choose the minimum cumulative sum between the three belonging to each possible predecessor, and add it to the energy of the current pixel to have our new cumulative sum. After running the algorithm on the second row, the result is the following:

$$\begin{bmatrix} (4,4,0) & (5,5,0) & (3,3,0) & (1,1,0) \\ (7,11,0) & (1,4,1) & (5,8,0) & (8,9,0) \\ (3,0,0) & (4,0,0) & (1,0,0) & (6,0,0) \end{bmatrix}$$

After reaching the last row, we go through it and pick the cell with the least accumulated sum. Backtracking through the parents of that pixel yields the seam with the least energy

possible. In this example, the seam is highlighted in red:



Given an energy *energy* matrix where the energy of each pixel is stored, we compute an entry (i, j) in our dynamic programming matrix $Matrix$ as the following:

$$Matrix(i,j) = energy(i,j) + min[Matrix(i-1,j), Matrix(i-1,j+1), Matrix(i-1,j-1)][5]$$

Of course the exceptions for this statement are the first row of the matrix which needs to be initialised to the exact energy of the pixels, and the edge pixels that only have two choices of predecessors. In our Python code, the function that performs this algorithm is the following:

```
def find_all_vertical_seams(im, energy, matrix):
    #the width of the image
    xlen = im.size[0]
    ylen = im.size[1]

    #first row of the matrix
    for x in xrange(0, xlen):
        matrix[x][0] = (energy[x][0], 0)

    for y in xrange(1, ylen):
        for x in xrange(0, xlen):
            #adjusting the statement for edge pixels
            if x == 0:
                predecessors = list([matrix[x][y - 1][0], matrix[x + 1][y -
                    1][0]])

            elif x == xlen - 1:
                predecessors = list([matrix[x - 1][y - 1][0], matrix[x][y -
                    1][0]])
```

```
            else:
                predecessors = list([matrix[x - 1][y - 1][0], matrix[x][y -
                    1][0], matrix[x + 1][y - 1][0]])

            min_energy = min(predecessors)
            index_of_min = predecessors.index(min_energy)

            if x == 0:
                index_of_min = index_of_min + 1

            matrix[x][y] = (energy[x][y] + min_energy, index_of_min)

    return matrix;
```

6.2 Performance

The removal of a single seam from an $n \times m$ picture using an $e1$ energy function has algorithm complexity of $O(n \times m)$, mainly due to the computation of the dynamic programming matrix. Shifting the pixels left or right and energy computations have the same complexity and they run in parallel [1].

The most important problem in the method is when we have to remove multiple seams. That is the aspect of seam carving that makes it a costly algorithm. Indeed, energy computations can take as much time, but the difference is that the energy process only needs to be performed as a preprocessing measure. Once the seam removal starts, energy has to be updated only in the neighbouring region of the seam, which mainly is two pixels wide. The update process for the matrix that stores the cumulative energy is much more complicated than that. We can choose to update this matrix in the same fashion as energy, or ignore updating it as a last resort. However, this approach can easily lead to very distorted resulting picture when the content is substantially changed after many seam removals.

We investigated the minimum region that needs to be updated after the removal and

found the following: the region that needs to be updated starts with the pixel to the right and the pixel to the left of the top-most deleted pixel and encompasses all the pixels inside the triangular region given by the two diagonals, the left one from the top left pixel and the right one from the top right pixel. This is illustrated below by two examples: red pixels are part of the seam, blue denotes the pixels whose energy needs to be updated, and green denotes the pixels whose cumulative sum needs to be updated.

$$\begin{bmatrix} 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \end{bmatrix}$$

Here is a different example with a different seam:

$$\begin{bmatrix} 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \\ 2 & 3 & 5 & 3 & 1 & 3 & 2 & 1 & 2 \end{bmatrix}$$

6.3 Aspect Ratio Retargeting

In the previous sections of this report, we only debated vertical seam, which is retargeting the aspect ratio across one dimension. In other words, if we want to reduce the size of an image from $x \times y$ to $x \times y'$ where $y' - y = k$, we just have to remove k vertical seams. Similarly, if

we had to retarget it from $x \times y$ to $x' \times y$ where $x' - x = k$, we have to remove k horizontal seams. Therefore until now we did not have to deal with retargeting an image through both dimensions. The issue with the latter is the order in which the seams have to be removed, do we remove horizontal seams, or vertical seams first? Or do we alternate both?

Dynamic Programming Approach

Ariel and Shamir proposed a DP approach to solve this problem of choosing whether to start with horizontal or vertical seams. The idea of this approach is simple and states that the cost to remove r horizontal seams and c vertical seams, is equal to the cost of removing r and $c-1$ plus the cost of the lowest vertical seam, or the cost of removing $r-1$ and c plus the cost of the lowest horizontal seam. We basically choose the minimum of the two alternatives, and that makes this solution a sort of a divide and conquer algorithm. Consequently, if we are to resize a picture from $x \times y$ to $x' \times y'$ where $x' - x = r$ and $y' - y = c$, we start building the dynamic programming matrix from the bottom up by making the initialization $M(0,0) = 0$. The update statement when moving from one cell to another is given by:

$$M(n,m) = min(M(n-1,m) + E(S_x(I_{x-n+1 \times y-m})), M(n,m-1) + E(S_y(I_{x-n \times y-m+1}))) \quad (1)$$

However, this algorithm is time and space consuming. For each cell in the matrix a seam calculation has to happen to determine the lowest cost seam. This results in the algorithm being $O(n^2 \times m^2)$. Moreover, to be able to calculate the lowest cost seam, we have to have an image with the same dimensions and with the same order of removed seams for each cell. This means we will have $n \times m$ as the number of matrices representing the images. Keeping

track of these matrices will be hard unless we are using a top down approach. Second, even if we use the top down approach, the space might not be enough and the process could result in an overflow. Therefore, we thought about an alternative solution, that can make the two dimensions problem much easier.

Reducing the Problem to One Dimension

As mentioned already, the dynamic programming approach for retargeting an image across two dimensions is too costly, for time and space. Therefore, we thought about a simple workaround using scaling. Scaling works particularly good, when both dimensions of an image are reduced by the same factor. For example, reducing an image from 100×50 to 50×25, can be done using scaling by the operating system or any web browser, without distorting or affecting the content.

Therefore, if we want to modify the size of a picture from $x \times y$ to $x' \times y'$, we can use scaling to match y with one dimension with the new dimension, and use seam carving to adjust the other dimension. For example, retargeting a picture from size 100×50 to 70×30, we use scaling by a factor of 0.7 to reach a 70×35 and then delete 5 vertical seam to reach the desired size. Formally, we can use scaling by a factor of $min(\frac{x'}{x}, \frac{y'}{y})$, and then delete $x - min(\frac{x'}{x}, \frac{y'}{y})$ horizontal seams, and $y - min(\frac{x'}{x}, \frac{y'}{y})$ vertical seams.

7 Energy Functions

Considering that the seam carving algorithm is standard for all types of pictures. The main variable in this process is the energy functions. a good energy function can produce amazing

results in the resizing of some pictures. While a bad energy function can make the whole algorithm useless. In order to go through the energy functions, it is necessary to clarify some core concepts in image processing.

7.1 Image Derivatives and Gradient Vectors

A gradient is one of the fundamental concepts in image processing. It can be seen as the directional change of colour or intensity in an image. Numerically, the slope of a image function at every picture pixel is a 2D vector with ta magnitude and an orientation. At every picture point, the angle vector focuses towards biggest possible colour or intensity change, and the length of the gradient vector corresponds to the rate of change. The gradients are used as an energy function because they detect edges in some sort. Edges have high energy because the colour change across an edge is very noticeable, and that is very important for seam carving. One of the main issues of seam carving is when a seam crosses an edge and that creates a distortion effect as we will see in our analysis of the energy functions.

Calculating Gradients

In Mathematics, we know that the derivative of a function is given by:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x) \tag{2}$$

For discrete functions, the smallest Δx is 1, and since any image function is a discrete function. The derivative of an image at a point x is:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} f(x) - f(x-1) = f'(x) \tag{3}$$

Furthermore, images are functions of two variables, $f(x, y) = k$, where x and y are the pixel coordinates and k is the colour or intensity of said pixel. Therefore, we have to take the partial derivatives in both dimensions to obtain the gradient vector at a certain pixel. Accordingly, the gradient of an image can be given by the formula [4]:

$$\bigtriangledown f = \begin{bmatrix} g_x \\ \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\sigma f}{\sigma x} \\ \\ \frac{\sigma f}{\sigma y} \end{bmatrix} \tag{4}$$

g_x is the gradient along the x-direction and g_y is the gradient along the y-direction.

$$|\bigtriangledown f(x, y)| = \sqrt{g_x^2 + g_y^2} \tag{5}$$

equation (1.5) represents the magnitude according to the L2 norm.

The orientation of the gradient vector is given by:

$$\theta = tan^{-1}(\frac{g_x}{g_y}) \tag{6}$$

Considering that we are working with RGB pictures, we will have to deal with three different gradient vectors for each pixel. In order to have one single magnitude, we can choose to either select the vector with the greatest magnitude or compute a norm for the three gradient vector magnitudes.

Gradients as an Energy Function

Gradients are used in image processing generally to extract information for the image. A standout amongst the most well-known uses of gradients is in edge location. After they have been processed and calculated, pixels with great gradient magnitude have more chances and are more susceptible to be part of edges. The orientation is also used to detect the edges exactly as used in the HOG energy function.

Considering the $e1$ energy function, which is the L1 norm of the gradient, We can use only the magnitude to calculate the energy of each pixel. The magnitude in $e1$ does very well in most images when the number of seams to be removed is not very large. As soon as only edges are left in the picture, the horizontal and vertical edges in the picture start having considerably less energy than the inclined ones. This latter creates problems making the algorithm cut through the horizontal and vertical edges which is not always the best solution as we can see in Figure 1.7.

One solution that I have found is to take the gradients only in the direction that is perpendicular to the seam direction, or at least give it more weight than the parallel direction. Since the seams will cut through normal edges anyway, we have to make sure it does not cut through parallel edges, because in that case, it will only do it once and that will distort the shape of that edge. Taking gradients in only one direction will assure that the edges parallel to the seam have higher energy by a notably large amount. In Figure 1.7, we apply the $e1$ energy function to the same picture, except that in (b) we only consider horizontal gradients. This latter makes sure that vertical edges have more energy which we can see by the increase of contrast from (a) to (b). The result is clear as , after applying seam carving, in (a) the

shape of the minaret gets distorted, but in (b) it doesn't.

(a) Gradients in both directions

(b) Gradients in the horizontal direction

(c) After Seam Carving

(d) After Seam Carving

Figure 7: difference between two dimensional and one dimensional gradients

7.2 HOG

The essential thought of the Histogram of Oriented Gradients is that content and shape and especially edges can regularly be described somewhat well by the distribution of gradient magnitude and orientation, even without exact information of the pixel whose gradient we are computing. In order to get thorough with this, the algorithm performs a quantization of

the gradient vectors. The histogram of gradients is done over a square window of pixels. For each window, a histogram of 9 bins is created in which we fit the contribution of gradients.

Each bin represents an orientation of gradients(0 - 20 - 40 .. - 180) in the case of unsigned gradients. In the case of signed gradients, an 18 bin histogram is used instead. We compute each gradient for the pixels inside the window and add its magnitude to the closest bins. For example, if a vector in the window has a 20 degrees orientation, we add its magnitude to the 20 bin in the histogram. However, if a gradient orientation is between bins, we divide its magnitude across the two bins, giving the closest bin more weight. As an example, a 15 degrees gradient has $\frac{3}{4}^{th}$ of its magnitude added to the 20 bin and $\frac{1}{4}^{th}$ added to the 0 bin. This will allow any window of pixels of size n to be quantised as 9 vectors instead of $n \times n$.

In the original histogram of oriented gradients original, the histogram is further normalised with blocks of cells in order to reach the purpose of human recognition [3]. However, more normalization is not needed to be able to use HOG in an energy function. The advantage of HOG is that it is able to detect edges if they are within the window. Therefore, what Ariel and Shamir suggested is to compute the $e1$ energy of the pixel and then compute the histogram of oriented gradients around that pixel in an 11×11 window. After computing those components, the energy of the pixel is set to $e1$ divided by the maximum magnitude in the histogram. The maximum magnitude is basically used to designate the most dominant edge in that area. Therefore the energy is:

$$e_{HoG}(x, y) = \frac{e1(x, y)}{max(HoG(I(x, y)))} \tag{7}$$

This energy function makes sure that pixels close to the edge have lower energy. Therefore, it ensures that seam run very close to edges. It is still very hard for a seam it to cross the edge as we multiply the energy by the L1 norm of the gradient which is very high in an edge pixel.

7.3 User Input

Even after choosing a very good energy function, the effectiveness of seam carving can be questioned. For some reasons, it is just the nature of the picture that causes irregularities. An example would be this figure where the edge of the minaret have lower energy compared to other component of the pictures because it is very colourful and all gradients across the image have a considerably high magnitude. In this case we can resort to user input to tell us which content of the picture is truly important. After getting the user input, we can emphasize on the edges of the region the user specified, or just concentrate on the whole region. As a solution we can increase the energy value of every pixel in that region. Another solution would be to restrict the seams from penetrating the region. However, the latter solution might be impractical. At a point when many seams removed, and the pixels outside gain more energy, the algorithm might need to cut down from that region to preserve the content outside the region.

(a) original picture

(b) Resizing with user input help

(c) normal seam carving

Figure 8: Benefit of User Input for Seam Carving

8 Conclusion and Future Work

The field of image processing is a very interesting field, and I learned many concepts about it throughout this capstone project. I was able to come up with new improvements to the algorithm which mainly three different contributions: The new approach for aspect ratio retargeting, the performance optimisations, and the user input as part of an energy function. I would love to work on other projects related to this kind of algorithms. Now that we analysed the algorithm, and explored any optimisations possible and the use of some energy functions, we can incorporate the algorithm within a web or desktop application to allow any user to resize a picture to their desired size. We can also try to further develop or optimise the algorithm to allow for more responsiveness.

Bibliography

[1] H. Huang, T. Fu, P. L. Rosin, and C. Qi, Real-time content-aware image resizing, *Sci. China Ser. F-Inf. Sci. Science in China Series F: Information Sciences*, vol. 52, no. 2, pp. 172182, 2009.

[2] M. Rubinstein, A. Shamir, and S. Avidan, Improved seam carving for video retargeting, *TOG ACM Trans. Graph. ACM Transactions on Graphics*, vol. 27, no. 3, p. 1, Jan. 2008.

[3] N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*.

[4] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Upper Saddle River, NJ: Prentice Hall, 2008.

[5] S. Avidan and A. Shamir, Seam carving for content-aware image resizing, *ACM SIGGRAPH 2007 papers on - SIGGRAPH '07, 2007*.