Content-Aware Image Resizing using Dynamic Programming

Michel Fang 13518137[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
[1]mfang555@gmail.com

***Abstract*—To display an image on media with various sizes, more often than not we will need to resize the image accordingly. Conventional image resizing techniques often distort the image or crude the image details, and cropping the image may trifle away the image novelty. In this paper, we will take a look at content-aware resizing that considers the image content, eliminating or adding part of the image that is of least interest. This method is made possible using seam carving which presents an interesting application of dynamic programming.**

Keywords—image resizing; content-aware image resizing; image manipulation; image retargeting; seam carving; dynamic programming

I. Introduction

Today, devices have various display sizes, this poses a challenge on how to display media content responsively. Websites and mobile applications support dynamically changing the property of each component, this is contrast to the rigidness of image resizing.

Image resizing is a standard procedure that can be done by many image processing applications. Usually, an image is uniformly shrunk down or enlarged to fit a target size. Standard image scaling methods diminish the quality of the image, and traditionally cropping the image can only be done on the border of the image. A better approach is to eliminate parts of the image that have least details such that it does not diminish the quality of the image when its size is altered.

In this paper, we will take a look on content-aware image resizing using seam carving, proposed by Avidan, S and Shamir, A [1]. This approach allows resizing without losing significant quality and meaning of the image, this is done by assigning an energy value to each pixel and locating optimal seams, a path of pixels going from one edge of the image to the other. Calculation of such seams utilize dynamic programming, a versatile concept that is utilized elegantly in this problem. We will also take a look at an improvement of this algorithm using formard energy calculation, which takes into account the new energy values of removed or added seams [2].

II. Image Resizing

A. Interpolation Algorithms

In mathematics, interpolation is a form of constructing a new value using an already known set of values. An image interpolation occurs when an image is resized, distorting the image from one pixel grid to the other. Image interpolation tries to achieve the best approximation of a pixel's density by using values on the surrounding pixel.

One of the simplest interpolation algorithms is nearest-neighbor interpolation, this algorithm replaces every pixel with the nearest pixel in the target output. This algorithm preserves fine details in pixel art images, but introduces artifacts in previously smooth images called jaggedness or 'jaggies', jaggies are stair-like lines that appear where a smooth curve should be.

Another interpolation technique is bilinear interpolation, this algorithm interpolates pixel color values, thus creating a continuous transition in the output image. This algorithm reduces contrast because of the introduced continuous transition, softening the details. Jaggedness may still be present in the output image.

Figure 1: An image of a green shell interpolated by nearest-neighbour interpolation (left) and bilinear interpolation (right)

B. *Scaling and Cropping*

Scaling a picture means applying scaling algorithms, some of which are explained before, to an image. Scaling causes distortion of the image which causes some objects to be widened or thinned.

Cropping is the removal of unwanted outer areas from the picture. Cropping is used to remove unwanted outer parts of the picture, alter the aspect ratio, or to isolate the subject of the picture from the background. Resizing an image can also be done with cropping, this maintains the quality of the image but removes part of the image.

Figure 2: Picture of a broadway tower

In the following example, we are trying to narrow the picture of a broadway tower. We can see that scaling results in the tower being distorted, and cropping causes part of the castle to be removed.

Figure 3: Undesirable result of scaling (left) and cropping (right)

III. Content-Aware Resizing

A. *Overview*

Content-aware resizing is a resizing technique that resizes the image without changing important visual content such as the objects in the image. The resizing process only affects pixels in areas that are not too important. This technique consists of mapping energy values, finding optimal seams, and removing or inserting seams.

B. *Calculating Energy Map*

By intuition, we should remove pixels that blend with the surrounding, the problem is how to pick such pixels? We can use the following energy function to calculate the value for each pixel, which is simply the absolute value of the image gradient in both x and y directions.

$$e(I) = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| \quad (1)$$

Calculating the actual gradient of the image can be done using several operators, such as Sobel Operator or Scharr Operator.

.Figure 4: Edge detection of an image with Sobel filters [4]

C. *Seam*

Seam is an 8-connected path of pixels in the image from top to bottom, containing one, and only one, pixel in each row of the image [1]. Removing seams instead of removing disjoint pixels from each row preverses the rectangular nature of the image. Formally, a vertical seam for an image **I** of size $n \times m$ is

$$s^x = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n \quad \forall i, |x(i) - x(i-1)| \leq 1 \quad (2)$$

where x is a mapping

$$x : [1, ..., n] \rightarrow [1, ..., m]$$

we can also define a horizontal seam in a similar manner.

The coordinate for the vertical seam pixels are

$$I_s = \{I(s_i)\}_{i=1}^n = \{I(x(i), i)\}_{i=1}^n \quad (3)$$

When we remove a seam, all the pixels are shifted by one position (horizontally or vertically, respective to the seams) to compensate for the lost seam. Now, we need to find the optimal seam to be removed, let $E(s)$ be the cost of the seam, then we can find the optimal s^* :

$$E(s) = E(I_s) = \sum_{i=1}^{n} e(I(s_i)) \quad (4)$$

$$s^* = min_s E(s) = min_s \sum_{i=1}^{n} e(I(s_i)) \quad (5)$$

The optimal seam can be found using dynamic programming, as we will later see in IV.

D. *Seam Removal and Insertion*

To remove a seam, we can just loop for each pixel in s^* and shift the image matrix left or up respectively. Seam insertion is simply an inversion of seam removal, since the optimal seam consists of pixels that blend with their

neighbouring pixels, we can insert another seam with similar pixel values to their neighbours. This inserted pixel value can be the average of the left and right pixel neighbours.

Figure 5: Original picture (left), seam visualization (center), carved picture (right)

IV. Dynamic Programming

A. Overview

Dynamic programming is a problem solving paradigm by decomposing solutions into several stages [5]. The solution can be viewed as a solution of smaller problems (state) that is achieved through a series of decisions (transitions).

B. Characteristics

In dynamic programming, decisions are made based on the *optimality principle*; if the final solution is optimal, then the solution for the *k*-th step is also optimal.

According to [5], these are the characteristics of a dynamic programming problem:

1. A problem can de defined as a set of stages, where in every stage a decision must be made
2. Each stage consists of a set of states that is connected. A state is a possible decision that can be made during a certain step and can be modelled as a graph
3. Result of the decision taken within each step is used to transform the state to be used for the next stage
4. The cost in each step increases steadily and depends

   on the cost of previous steps

A dynamic programming solution can be done bottom-up or top-down. In bottom-up, we construct the solution table from the base case to the full solution, usually populating the entire table. In top-down, we make a recursive call to a subproblem, then eventually reaching a base case, top-down usually only populates the solution table as necessary. In either approach, we similarly define a state as a function, as seen in the fibonacci example

$$f(0) = 0 \text{ (base case)}$$

$$f(1) = 1 \text{ (base case)}$$

$$f(i) = f(i-1) + f(i-2),\ i > 1$$

C. Seam Finding

Let the image be an $n \times m$ grid with each tile representing a pixel with energy value $e(i, j)$

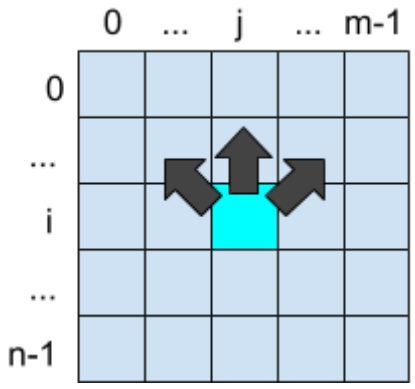

Figure 6: Image represented as a grid

Then we can define $f(i,j)$ to be the minimum cost of a seam path, starting from $(0, l)$ to (i,j), where l is some column in the first row.

$$f(0,j) = e(0,j),\ \forall j,\ 0 \le j < m \text{ (base case)}$$

$$f(i,j) = e(i,j) + min(f(i-1,j-1), f(i-1,j), f(i-1,j+1))$$

We also need to be careful when taking care of a pixel that is on the edge of the grid.

From (5), then the optimal seam path cost is

$$cost = min_j(f(n-1,\text{j})),\ \forall j\ 0 \le j < m$$

After we get the optimal cost of a seam, we can get all of the pixel coordinates by backtracking from the last row to the first row, and writing down the coordinates as we go along.

Algorithm 1 findSeam

```
1:  seam_{n-1} ← lastRowMinColumn
2:  for i = n − 2, n − 1 ... 0 do
3:      prev ← seam_{i+1}
4:      for j = max(0, prev − 1) ... min(m − 1, prev + 1) do
5:          if f_{i+1,prev} − e_{i+1,prev} = fi, j then
6:              seam_i ← j
7:              break
8:          end if
9:      end for
10: end for
11: return seam
```

V. Implementation

To test the result of this algorithm, I have made an implementation in Python with a simple image picker GUI made with PyQt5.

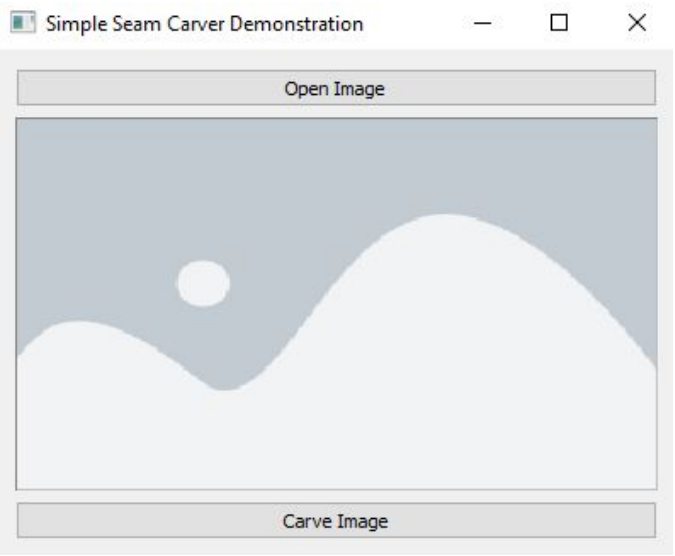

Figure 7: Application interface

To transform an image into a grid of numbers, OpenCV library was used along with NumPy for array heavy computations. Numba library was used to speed up the computations, as it translates the code into fast machine code using LLVM.

Below is the carve function that utilizes previously explained algorithms

```
def carve(self, r, c):
    # im is the original image
    ro, co = self.im.shape[:2]
    dc = c - co
    dr = r - ro

    if dc != 0:
        if dc < 0:
            self.seams_remove(-1 * dc, False)
        else:
            self.seams_insert(dc, False)

    if dr != 0:
        self.im = self.rotate_ccw(self.im, True)
        if dr < 0:
            self.seams_remove(-1 * dr, True)
        else:
            self.seams_insert(dr, True)
        self.im = self.rotate_ccw(self.im, False)
    cv2.imshow("Carver", self.im.astype(np.uint8))
    cv2.waitKey(1)
```

The energy calculation in the implementation uses Scharr Filter, which performs similarly to the Sobel Filter. To further improve the algorithm, I enforced forward energy calculation that was defined in [6] as a follow up to the original paper [5]. The forward energy calculation considers the resulting energy after a removal or insertion, since removing a low-energy pixel pushes two pixels together, and may increase the image energy greatly. The details of forward energy calculation can be found in [6].

To show the seams chosen for every changed width or height, a show function was implemented. This function colors each seam red as it is being removed from the image, it is called every time a seam deletion or insertion occurs to the image

```
def show(self, rotate, marker=None):
    tmp = self.im.astype(np.uint8)
    if marker is not None:
        r, c = tmp.shape[:2]
        for i in range(r):
            c = marker[i]
            tmp[i, c] = np.array([0, 0, 255])
    if rotate:
        tmp = self.rotate_ccw(tmp, False)
    cv2.imshow("Carver", tmp)
    cv2.waitKey(1)
```

Further demonstration of the application and other details can be seen in the Youtube video.

Figure 8: ‘show’ function in action. Removing 1st (top), 2nd (middle), and 3rd (bottom) seams

VI. Result

As expected, the seam carving algorithm works marginally better than previous methods. Below is an example of horizontal shrinking with seam carving

Figure 9: Picture to be made narrower

(Source: Breaking Bad)

Figure 10: Horizontal shrinking. Scaling result (left) and seam carving (right) result

Horizontal expanding also looks better

Figure 11: Horizontal expanding. Scaling result (top) and seam carving (bottom) result

We can also shrink or expand pictures vertically, in the implementation, this was done by temporarily rotating the image 90°, applying the algorithm, and then rotating back to restore the original image orientation. In the following example, a different picture is used to demonstrate vertical size change of the picture

Figure 12: Picture to be made shorter

(Source: The Mandalorian)

Figure 13: Vertical shrinking. Scaling result (left) and seam carving (right) result

Notice that the subject of the picture (the mandalorian) is not shrunk with seam carving, providing greater clarity of the image. However, there is a deformation of the sun behind the subject. To exclude an object with less details to be chosen for a seam, we can use an object mask.

The idea of an object mask is to set the individual energy value of the object to a specific value, to 'trick' the dynamic programming. If we want to preserve the object, we set the energy of the object pixels to be a really high value, and set it to a really low value if we want to remove it. I didn't implement this feature in my implementation.

Figure 14: Picture with objects to be removed [6]

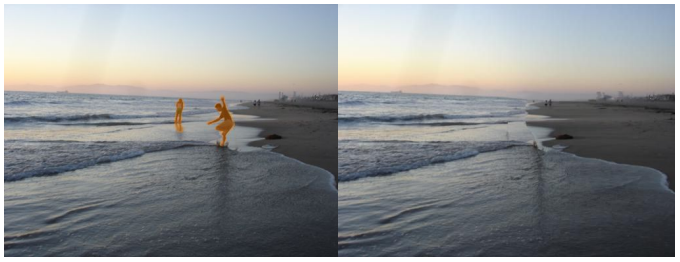

Figure 15: Object mask (left) and the final image (right) [6]

There are still many improvements that can be made to my implementation of this algorithm, the main one is using an object mask for object targeting or removal, making the mask can be done by adding a 'canvas' of the picture and a brush on the GUI highlight the object on. The highlighted pixels value can then be set appropriately depending on the need.

In this paper, we have seen how content-aware image resizing resizes an image by preserving content focus, how the seam carving algorithm works in conjunction with dynamic programming, and several tweaks that can be enforced to further tweak the performance of these methods.

Video Link at Youtube

As part of this paper, a Youtube video was made to better explain and demonstrate the implementation of seam carving. The video can be found at https://youtu.be/cMkUT4mbjvE. The project repository will be publicly available on my github page at https://github.com/littlemight not long after the publication of this paper.

Acknowledgment

I would like to express my gratitude to Mrs. Nur Ulfa Maulidevi, Mrs. Masayu Leylia Khodra, and Mr. Rinaldi Munir as our lecturer in Algorithm Strategy Course for the knowledge that they shared upon us. I also would like to thank my family and friends for helping and supporting me in the creation of this paper. I also would like to thank everyone whose open source projects and/or tutorials have helped me implement this simple demonstration of content-aware image resizing.

References

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

[2] Rubinstein, M., Shamir, A., & Avidan, S. (2008). Improved seam carving for video retargeting. ACM SIGGRAPH 2008 Papers on - SIGGRAPH 08. dl.acm.org/doi/10.1145/1399504.1360615

[3] Anbarjafari, G. (n.d.). 3. Resizing image. Retrieved May 1, 2020, from https://sisu.ut.ee/imageprocessing/book/3

[4] Ashish. (2018, September 26). Understanding Edge Detection (Sobel Operator). Retrieved May 1, 2020, from https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900

[5] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika ITB.

[6] Chan, L. (n.d.). Project 2: Seam Carving. Retrieved May 1, 2020, from http://www.cs.cmu.edu/afs/andrew/scs/cs/15-463/f07/proj2/www/lisachan/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Mei 2020

Michel Fang, 13518137