**Report On Seam Carving Implementation**

**Deekshith Sandesari**

**CS 435/535 Algorithms**

**Dr. Z. -H. Duan**

**April 6, 2013**

**Abstract**

This is a report on the analysis of Seam Carving Implementation. It also presents asymptotic analysis on the running time of the algorithms used in this program.

**Introduction**

**Seam Carving:**  
Digital images are often viewed in many different display devices with a variety of resolutions. Variation of resolution makes viewing images difficult because they usually are resized to accommodate limited space. Simple attempts at resizing include scaling and cropping. Scaling reduces perceivable detail and cropping cannot be done automatically. Also, cropping alters the image composition and is not always desirable. This project is an implementation of a different image resizing approach called seam carving. This algorithm for resizing method is developed by Shai Avidan and Ariel Shamir.

Seam carving allows a change in size of the image by modifying the least noticeable pixels in an image. A typical application for seam carving is to reduce the size of an image along one dimension. This can be done by finding one-pixel wide paths from the top to the bottom of the image and removing those paths. If the pixels in those paths are similar to surrounding pixels, then their removal may be unnoticed. Other seam carving applications include increasing the size of an image, changing the size of an image in two dimensions and even object removal.

**Analysis of Implementation**

The implementation uses the following functions:

1.readImage ()

2.processImg ()

I. calculatePixelEnergy ()

II. calculateCumilativeMinEnergy ()

III. seamCarving ()

3.saveImage ()

**1.readImage ():**

This function reads the input pgm format image file and stores in a 2-d array. The time taken is O (m x n) where m-rows and n-columns of the array.

**2.processImg ():**

This function implements seam carving that changes the size of an image by finding one pixel wide paths of lowest energy pixels crossing the image from top to bottom or from left to right and removing the path (seam).

The function takes vertical and horizontal seams to be removed as the input and for each iteration, it calculates seams by finding cumulative energy of the pixels using energy function.

**I. calculatePixelEnergy ()**

The visibility of a pixel can be defined using an energy function.

Energy is the absolute difference of values of pixel elements on the right side, left side, below and above. For each pixel energy value is calculated as shown below:

*Energy* [i][j] = Math.*abs*(*img*[i][j]-*img*[i-1][j])

+ Math.*abs*(*img*[i][j]-*img*[i+1][j]) + Math.*abs*(*img*[i][j]-*img*[i][j-1])

+ Math.*abs*(*img*[i][j]-*img*[i][j+1]);

For the boundary cases, the difference = 0 if one of the pixel is outside of the given image.

The time taken to calculate energy for each of pixel for 2-D array is O (m x n), where m-rows and n-columns of the array.

**II. calculateCumilativeMinEnergy ()**

Cumulative energy is sum of energy values and min cumulative energy values of above value, above left and above right value. The cumulative minimum energy for all possible connected vertical seams for each entry (i, j) can be calculated as shown:

*cumMinEnergy*[i][j] = *energy*[i][j]+

Math.*min*(*cumMinEnergy*[i - 1][j - 1], Math.*min*(*cumMinEnergy*[i - 1][j],

*cumMinEnergy*[i - 1][j + 1]));

For the boundary cases, the cumulative minimum energy is ignored for pixel that is outside of the given image.

The time taken to calculate cumulative energy for each of pixel in the 2-D array is

O (m x n), where m-rows and n-columns of the array.

**III. seamCarving ()**

To find the vertical seam, for each row i the function finds the minimum cumulative energy element from the ones on the left (columnindex-1), current value (columnindex) and from one on the right (columnundex+1). The minimum cumulative energy element can be calculated as shown:

minEnergy = Math.*min*(Math.*min*(*cumMinEnergy*[i][columnindex-1],

*cumMinEnergy*[i][columnindex]),

*cumMinEnergy*[i][columnindex+1]);

After finding the minimum energy element for a row, the function then removes the element from image. In this way a vertical seam can be removed from the image by calculating it for each row.

The calculation of minEnergy value for each row takes O (m) time where m is number of rows. But after finding the minEnergy value, the min value must be removed and this removal may take atmost O (n) time where n is no of columns i.e. if suppose the minEnergy values are all at the first row of the matrix then to remove all the first elements in each row of the array, the iterator needs to traverse the width of the array.

So, the time complexity for seamCarving is O (m x n), where m –no of rows and n- no of columns of the 2D array. To remove V vertical seams, the running time is V \*O (m x n).

So, the total time complexity for removing V vertical seams and H horizontal seams is:  
V \*O (m x n) +H\*O (m x n) = (H+V)\*O (m x n).

**Horizontal seam:** The vertical seam can be removed by using the energy, cumulative energy and seam carving functions as shown above. But to remove the horizontal seams, transpose of the original image matrix is calculated and then the seams are removed. The transposed matrix is then transposed back to original image.

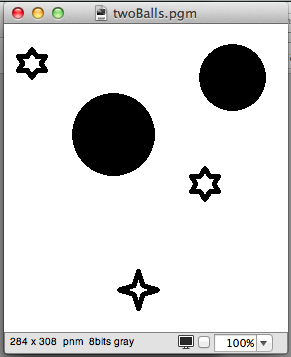
**3.saveImage ()**

This function writes the contents of processed input original\_image\_file to pgm file with name original\_image\_file\_name\_processed.pgm

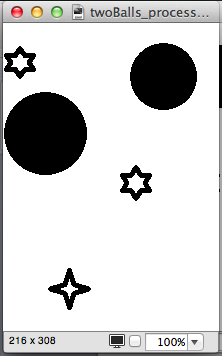
**Test Cases and Results**

**TestCase1- 2balls**

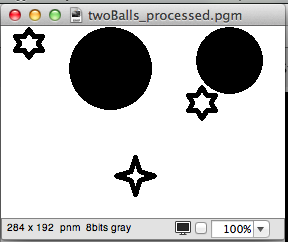
**Original Image:** **284 X 308.**The original image consists of 284 columns and 308 rows in the image.

****

**Vertical Seam (68 seams): 216 X 308.**After removing 68 vertical seams from the original image, the image now consists of 216 columns and 308 rows. The image after processing is shown below:

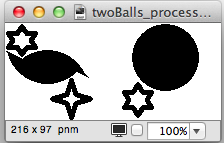
****

**Horizontal Seam (116 seams):** **284 X 192.**After removing 116 horizontal seams from the original image, the image now consists of 284 columns and 192 rows. The image after processing is shown below:

****

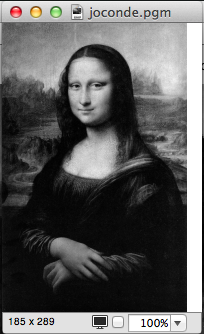
**Vertical and Horizontal Seam (68,211 seams):** **216 X 97.**

After removing 68 vertical and 211 horizontal seams from the original image, the image now consists of 216 columns and 97 rows. The image after processing is shown below:

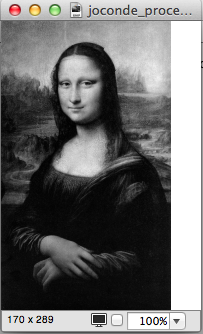
****

**TestCase6-MonaLisa**

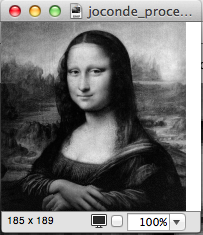
**Original Image:** **185 x 289**

****

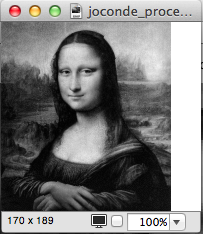
**Vertical Seam (15 seams): 170 X 289.**

****

**Horizontal Seam (100 seams): 185 X 189.**

****

**Vertical and Horizontal Seam (15,100 seams):** **170 X 189.**

****

**Test case 7:CAS (500 x 376)**

****

**Vertical Seam (100 seams): 400 X 376.**

****

**Horizantal Seam (100 seams): 500 X 226.**

****

**Vertical and Horizontal Seam (100,100 seams):** **400 X 226.**

****

**References**

[1] <http://en.wikipedia.org/wiki/Seam_carving>.  
  
[2] <http://www.faculty.idc.ac.il/arik/SCWeb/imret/imret.pdf>.  
  
[3] <http://www.cs.uakron.edu/~zduan/class/435/projects/project3/project3.htm>.