Image Content-Aware Resize

[Introduction 2](#_Toc373439458)

[Main idea 2](#_Toc373439459)

[Solution 1: Remove Pixels with Min Energy 3](#_Toc373439460)

[Solution 2: Remove Seams with Min Backward Energy 3](#_Toc373439461)

[Solution 3: Remove Seams with Min Forward Energy 5](#_Toc373439462)

[To remove K columns (or K rows) only 7](#_Toc373439463)

[To remove K1 columns and K2 rows simultaneously 7](#_Toc373439464)

[Project Requirements 8](#_Toc373439465)

[Given 8](#_Toc373439466)

[Input 8](#_Toc373439467)

[Deliverables 9](#_Toc373439468)

[Implementation 9](#_Toc373439469)

[Documentation 9](#_Toc373439470)

[Grades 9](#_Toc373439471)

[BONUS TASKs 10](#_Toc373439472)

# Introduction

“Have a look on [this video](http://www.youtube.com/watch?v=qadw0BRKeMk) before proceeding to read this document.”

Image resizing is a standard tool in many image processing applications. It works by uniformly resizing the image to a target size. However, this standard resizing doesn’t consider the image content, leading to stretch/elongate the objects inside the image as shown in figure 1.

Original image (186×274) Standard resizing (186×174)

Figure 1: Elongation/Stretching effect of standard image resizing

On the other hand, Content-Aware Resizing can be used to effectively resizing the image while considering its content to avoid the stretching/elongation effects on the image objects, see figure 2 and compare the result with figure 1

** **

Original image (186×274) Content-Aware resizing (186×174)

Figure 2: effect of content-aware image resizing, the Tower is not elongated

# Main idea

Content-aware resizing mainly uses an **energy** function to define the importance of pixels. The pixels with low energy are less important and can be eliminated when resizing the image. Figure 3 show the energy function of the Tower image, the pixels with large energy are white while those of small energy are black. Observe that the Tower pixels are the most important pixels with the largest energy, so it should not be removed by the Content-aware resizing.

****

Figure 3: Energy function of the Tower image, white colors are high energies while black are low energies.

# Solution 1: Remove Pixels with Min Energy

Successively removing the pixels with minimum energy from each row leads to corrupted image, as shown in figure 4.

****

Figure 4: Effect of successively removing pixels with minimum energy from each row

# Solution 2: Remove Seams with Min Backward Energy

So, instead of removing separate pixels as in solution 1, the content-aware resizing search for a **connected path** of **minimum total energy** pixels crossing the image from top to bottom, or from left to right, this path is called a **“Seam”.** Figure 5 show the minimum seam in the Tower image.

By successively removing seams we can reduce the size of an image in one or both directions. Seam selection ensures that while preserving the image structure, we remove more of the low energy pixels and fewer of the high energy ones.

****

Figure 5: First smallest vertical seam in the Tower image

However, removing the seam with minimum total energy may lead to **create some artifacts** in the resulting image as shown in Figure 6. These artifacts are created because we choose to remove the seam with the current least amount of energy from the image, **ignoring new energy** that is inserted into the resulting image. The inserted energy is due to new edges created by previously non adjacent pixels that become neighbors once the seam is removed.





Figure 6: Removing the seams with min total energy lead to artifacts as shown in red rectangle because they insert more energy to the image than remove

# Solution 3: Remove Seams with Min Forward Energy

To solve this issue, the algorithm looks forward at the resulting image instead of backward at the image before removing the seam. At each step, we search for the **seam** whose removal **inserts the minimal amount of energy** into the image. These are seams that are not necessarily minimal in their energy, but will **leave fewer artifacts** in the resulting image, after removal. See figure 7.



Figure 7: Removing the seams that will inserts the minimal amount of energy lead to reduce artifacts significantly as shown in red rectangle. Compare with figure 6

To ensure the connectivity of the pixels within a single path "Seam", each location should be connected to one of the three preceding locations as shown in figure 8.

i, j

i, j-1

i+1, j-1

i-1, j-1

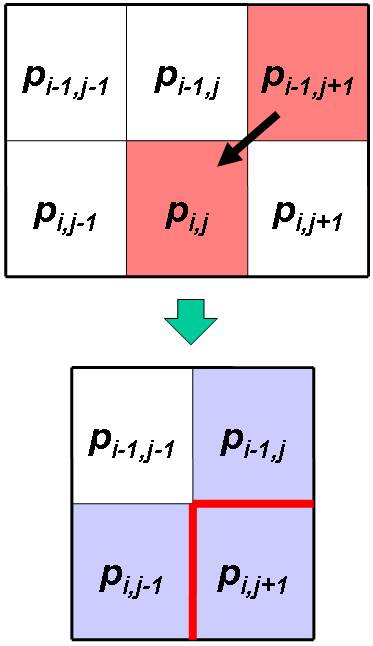
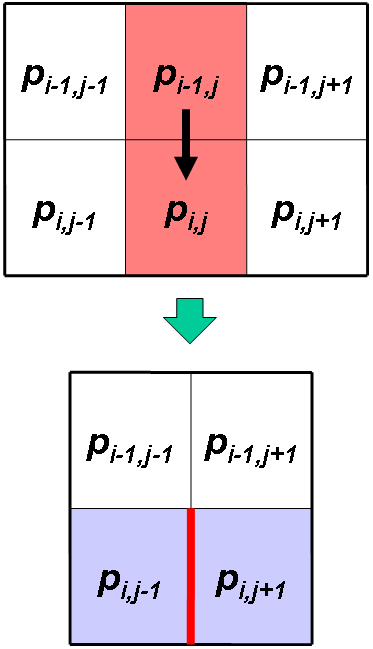
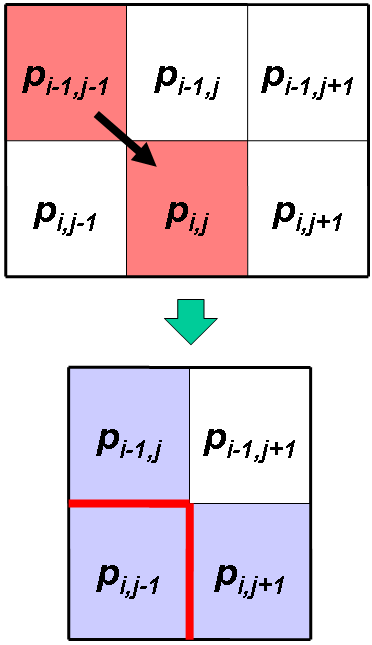
…

…

…

Figure 8: To ensure connectivity of single vertical seam, each pixel should be connected to one of the three above locations… and so on

To measure the amount of energy that will be inserted into the image after removing a specific seam, we need to **calculate the energy** between the pixels that become new neighbors if the seam is removed. Depending on the direction of the seam, three such cases are possible (see Figure 9). The seam that will **insert minimum total energy** should be **removed.**



(c)

(b)

(a)

Figure 9: Calculating the three possible vertical seam step costs for pixel p[i,j] using forward energy. After removing the seam, new neighbors (in blue) and new pixel edges (in red) are created. In each case the cost is defined by calculating the energy between new adjacent pixels. Note that the new edges created in row i – 1 were accounted for in the cost of the previous row pixel.

In case (a):

Cost = energy between pixels **p[i, j-1]** & **p[i, j+1]** PLUS energy between pixels **p[i, j-1]** & **p[i-1, j]**

In case (b):

Cost = energy between pixels **p[i, j-1]** & **p[i, j+1]**

In case (c):

Cost = energy between pixels **p[i, j-1]** & **p[i, j+1]** PLUS energy between pixels **p[i, j+1]** & **p[i-1, j]**

## To remove K columns (or K rows) only

For each path "Seam" to be removed

1. Find the vertical (or horizontal) path "seam" that will **insert minimum total energy** if it's removed.
2. Remove its pixels from the original image, reducing the image width (or height) by one

Repeat until all required Seams are removed

Note that we **repeat** the searching for the best seam after each step because when we remove a seam, the pixels' neighbors are changed and thus the next minimum seam needs to be recalculated on the new image.

## To remove K1 columns and K2 rows simultaneously

This begs the question of what is the correct order of seam removing. Remove vertical seams first? Horizontal seams first? Or alternate between the two?

We define the search for the optimal order as an **optimization problem** that **minimizes the total inserted energy** after removing columns and rows. Thus we need to search for the order of removing the required rows and columns such that the total inserted energy of them is **minimized.**

We should find an **efficient solution** that achieves a **balance** between the **minimization needs** and the **execution time**.

Figure 10 shows some explanation about the possibilities of removing:

Remove 1 row

Remove 1 row

Remove 1 column

Remove 1 row

Remove 1 column

Remove 1 row

Remove 1 column

And so on… until removing K1 columns and K2 rows. We need to find the order of removing that minimizes the total inserted energy.

Note that we can start with removing one column instead of one row.

Figure 10: different possibilities of removing columns and rows… we need to minimize the total inserted energy

# Project Requirements

1. **Efficient** method to remove K columns (or K rows) separately
2. **Efficient** method to remove K1 rows and K2 columns by determining the best order of removing that minimize the total inserted energy

## Given

C# Code contains **ImageOperations** class with the following functionalities:

1. Open image & load it in a 2D array stored in a global variable of type MyColor[[1]](#footnote-1) [,]called ImageMatrix

MyColor [,] OpenImage(string ImagePath)

1. Get width and height of the image matrix

int GetHeight(MyColor[,] ImageMatrix)

int GetWidth(MyColor[,] ImageMatrix)

1. Calculate the energy between two pixels

int CalculatePixelsEnergy(MyColor Pixel1,MyColor Pixel2)

1. Display an image on a given PictureBox control

void DisplayImage(MyColor[,] ImageMatrix,PictureBox PicBox)

1. Apply the standard resizing technique to resize the given image and return the resized one, you can use this function for comparing the results standard resizing and content-aware resizing

MyColor[,] NormalResize(MyColor[,] ImageMatrix, int NewWidth, int NewHeight)

## Input

1. Colored image to be resized
2. Number of columns and rows to be removed from the image

# Deliverables

## Implementation

1. Implement the content-aware resizing algorithm to minimize the input image by the given number of columns and rows by applying:
2. Efficient method to remove K columns (or K rows) separately
3. Efficient method to remove K1 rows and K2 columns by determining the best order of removing

## Documentation

1. Design of your solution in pseudocode (main functions **only**)
2. Analysis of your code

# Grades

|  |  |
| --- | --- |
| **Deliverable** | **Grade** |
| Remove columns (or rows) separately | 30% |
| Remove rows and columns by determining the best order of removing | 30% |
| Design of your solution in pseudo-code | 20% |
| Analysis of your pseudo-code | 20% |

# BONUS TASKs

1. Fast way to update the energy matrix of the affected locations ONLY after removing each column/row instead of recalculating the whole matrix
2. One of the applications for the content-aware resizing is to remove some objects from the original image. The object(s) to be removed should be highlighted by the user using the mouse. Then you should remove the selected object(s) using your implementation of the content-aware resizing. Figure 11 shows an example.

Figure 11: Simple object removal: the user marks a region for removal (green), and possibly a region to protect (red), on the original image (see inset in left image). On the right image, consecutive vertical seam were removed until no ‘green’ pixels were left.

1. Another bonus is to maximize the image by inserting seams instead of removing them. Figure 12 shows an example.



Figure 12: maximize the image by inserting seams

1. MyColor is a structure defined in the code to hold the Red, Green, Blue values of each pixel [↑](#footnote-ref-1)