Individual Homework: Seam Carving (Part 2)[[1]](#footnote-0)

# Objectives

* Further practice reading from files.
* Validation of input data including the use of stream states.
* Work with two dimensional arrays.
  + Traversing through.
  + Accessing elements.
  + Staying within array bounds.
* Work with dynamic arrays.
* Compute information based on information from different parts of an array.

# Overview

Sometimes you want to use an image, but it needs to be resized for your use. For example, what if we need to make this image narrower without changing the height?

Original surfer image.  


Traditionally this is done by cropping or re-scaling.

Cropping can remove important information such as the two people on the right of the image.

Cropped surfer image.  
Island to left and surfer are clearly visible. Only one surfer from the cluster on the right is visible.


Rescaling ends up making the image look squashed.

Rescaled surfer image.  


Something that would be ideal is a way to resize the image that does not distort the important details of the image. Seam carving is a method of resizing images in a content-aware manner that works well for some types of images. Unimportant pixels are removed while pixels that convey important details are preserved. Interestingly, the seam carving approach was first published in 2007[[2]](#footnote-1).

[Watch the original SIGGRAPH video describing the technique](https://www.youtube.com/watch?v=6NcIJXTlugc).

Seam carved surfer image.  


# Requirements & Roadmap

* Continue with the code from last week.
  + Note, if you don’t need all of the data validation cases from last week running to do this week’s labwork.
* Download test ppm files
  + [Download zip of test ppm files](https://drive.google.com/file/d/1COmGYMOIDOpRSm4pirLUuNQORBQI0vTJ/view)
  + [See test ppm files on Google Drive](https://drive.google.com/drive/u/1/folders/1XHMQ3Cbmf9NeHNdZoy51MnHq82XZHDPw)
    - View these files to see what they look like.
    - As you test your program, view the modified ppm files to see how well your solution is working.
    - Bigger files will take longer to run, especially later in part 2.
* You man only include
  + <iostream>
  + <fstream>
  + <string>
  + <sstream>
  + <cmath>
  + “functions.h”
* Read over the starting code.
  + The struct Pixel is defined in functions.h
  + The main program already reads in a file name, the width and height, and the target width and height after carving. You can use that to test your program.
  + You will need to use an image viewer to view your PPM file. See “[Viewing ppm files](#_3fn2fi580pf)” section below for information on viewing your PPM files.
  + We are using **dynamic arrays** for this homework.
* You cannot have a memory leak.
* Implement **first**  
  int loadVerticalSeam(Pixel\*\* image, int start\_col,   
   int width, int height, int\* seam);
  + This function will traverse through an image starting at the first row of the given column (start\_col). See “[Loading a vertical seam](#_r5zkhbutl0b)” below for how the traversal works. See “[Seam Representation](#_bo2ii6cxwg0g)” below for how seams are represented.
  + The function returns the total energy of the seam
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the column to start the seam.
  + The third parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The fourth parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + The fifth parameter is an array to be loaded with column values for each row.
  + Pseudocode
    - Set seam for the first row to the starting column
    - Initialize total energy to the energy for pixel (start\_col, 0)
    - For each subsequent row
      * Calculate the energy of each possible next column
      * Set the seam for current row to the column with the minimal energy
      * Add the minimal energy to the total energy
    - Return total energy
* Implement after loadVerticalSeam  
  int\* findMinVerticalSeam(Pixel\*\* image, int width, int height);
  + This function will traverse through each column, loading its seam in order to find the seam with lowest energy. See “[Finding a Minimal Vertical Seam](#_133htmxvt0wh)” below for how this works.
  + The function returns a pointer to a seam that is the seam with minimal energy.
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The third parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + Pseudocode
    - Set minimal energy to the result of loading the seam for the first column
    - Note minimal seam was loaded by function that returned minimal energy
    - For each column
      * Get energy for the column and load candidate seam
      * If the energy is less than minimal energy
        + Set minimal energy to energy
        + Update minimal seam to be the candidate seam
  + Note: you will have to create seams to pass to loadVerticalSeam
    - You cannot have a memory leak.
* Implement after findMinVerticalSeam  
  void removeVerticalSeam(Pixel\*\* image, int width, int height,   
   int\* verticalSeam);
  + This function removes the pixels from the image corresponding to the vertical seam. See “[Removing a Vertical seam](#_qqcd4w5lj3n1)” below for how the removal works. See “[Seam Representation](#_bo2ii6cxwg0g)” below for how seams are represented.
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The third parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + The fourth parameter is an array indicating which pixels should be removed.
  + Pseudocode
    - For each row
      * Remove pixel (seam[row], row) - note that seam[row] will be the column.
* Uncomment code that calls functions to remove seams.
  + Pseudocode
    - While width is greater than target width or height is greater than target height
      * If width is greater than target width
        + Find minimal vertical seam
        + Remove minimal vertical seam
        + Update width (i.e. decrement by one)
      * If height is greater than target height
        + Find minimal horizontal
        + Remove minimal horizontal seam
        + Update height (i.e. decrement by one)
  + Note: it is important to remove the vertical seam prior to the horizontal seam since this can affect the subsequent calculation of energies for pixels if you do the extra credit.
* Ensure the entire program does not have a memory leak.

## 

## Extra Credit

* Implement  
  int loadHorizontalSeam(Pixel\*\* image, int start\_row,   
   int width, int height, int\* seam);
  + This function will traverse through an image starting at the first column at the given start\_row. See “[Loading a Horizontal Seam](#_sa49ys37mty4)” below for how the traversal works. See “[Seam Representation](#_bo2ii6cxwg0g)” below for how seams are represented.
  + The function returns the total energy of the seam
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the column to start the seam.
  + The third parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The fourth parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + The fifth parameter is an array to be loaded with column values for each row.
  + Pseudocode
    - Analogous to loadVerticalSeam
* Implement after loadHorizontalSeam  
  int\* findMinHorizontalSeam(Pixel\*\* image, int width, int height);
  + This function will traverse through each column, loading its seam in order to find the seam with lowest energy. See “[Finding a Minimal Horizontal Seam](#_tnk8xh4fwnuh)” below for how this works.
  + The function returns a pointer to a seam that is the seam with minimal energy.
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The third parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + Pseudocode
    - Analogous to findMinVerticalSeam
  + Note: you will have to create seams to pass to loadHorizontalSeam
    - You cannot have a memory leak.
* Implement after findMinHorizontalSeam  
  void removeHorizontalSeam(Pixel\*\* image, int width, int height,   
   int\* verticalSeam);
  + This function removes the pixels from the image corresponding the horizontal seam. See “[Removing a Horizontal seam](#_pgnjevyichk)” below for how the removal works. See “[Seam Representation](#_bo2ii6cxwg0g)” below for how seams are represented.
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  + The third parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  + The fourth parameter is an array indicating which pixels should be removed.
  + Pseudocode
    - Analogous to removeVerticalSeam

# Supporting Information

## Images and RGB Color Model

Images are a two dimensional matrix of pixels where each pixel is a color composed of a red, a green, and a blue value. For example RGB(80, 0, 0) is Aggie maroon.[[3]](#footnote-2)

In image processing, pixel (x,y) refers to the pixel in column x and row y where pixel (0, 0) is the upper left corner of the image and pixel (width-1, height-1) is in the lower right corner.

***Warning:*** This is column-major ordering which is transposed from the row-major ordering that is used for cartesian coordinates where the first index is the row and the second index is the column and (0, 0) is in the lower left corner. In image files the width is essentially the number of columns and the height is the number of rows. So the impact is that you will index with [col][row] instead of with [row][col].

Coordinates for a 3X4 (3 columns (i.e. width) by 4 rows (i.e. height)) image are shown in the following table.

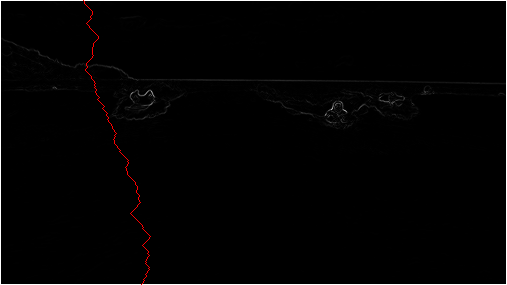
|  |  |  |
| --- | --- | --- |
| (0, 0) | (1, 0) | (2, 0) |
| (0, 1) | (1, 1) | (2, 1) |
| (0, 2) | (1, 2) | (2, 2) |
| (0, 3) | (1, 3) | (2, 3) |

We will use a Pixel struct (defined in functions.h) that holds a value for red, green and blue. The image will be a 2 dimensional array of Pixels.

## 

## Seam Carving

Seam carving involves three major steps.

1. **Energy Calculation** (You did this last week in Part 1)  
   The first step is to calculate the energy of a pixel, which is a measure of its importance—the higher the energy, the less likely that the pixel will be included as part of a seam (as we'll see in the next step). In this assignment, you will use the dual-gradient energy function, which is described in the “[Computing the energy of a pixel](#_hsvef6jluhbk)” section below.  
   Here is a visualization of the dual gradient of the surfing image above.  
   The energy is high (white) for pixels in the image where there is a rapid color gradient (such as the boundary between the sea and sky and the boundary between the surfing Josh Hug on the left and the ocean behind him). The seam-carving technique avoids removing such high-energy pixels.
2. **Seam Identification** (You’ll do this week in Part 2)  
   The next step is to find a vertical seam of minimum total energy.   
   Seams cannot wrap around the image (e.g., a vertical seam cannot cross from the leftmost column of the image to the rightmost column). See “[Identifying Seams](#_b7xa5p1eezh6)” below for details on identifying the appropriate seam to remove.  
   Finding a horizontal seam is analogous.
3. **Seam Removal** (You’ll do this week in Part 2)  
   The final step is remove from the image all of the pixels along the vertical or horizontal seam.

### Computing the energy of a pixel

Recall the notation covered above in “Images and RGB Color Model”

You will use the *dual-gradient energy function*: The energy of pixel (*x*, *y*) is Δ*x*2(*x*, *y*) + Δ*y*2(*x*, *y*), where the square of the *x-*gradient Δ*x*2(*x*, *y*) = *Rx*(*x*, *y*)2 + *Gx*(*x*, *y*)2 + *Bx*(*x*, *y*)2, and where the central differences *Rx*(*x*, *y*), *Gx*(*x*, *y*), and *Bx*(*x*, *y*) are the absolute value in differences of red, green, and blue components between pixel (*x* + 1, *y*) and pixel (*x* − 1, *y*). The square of the *y*-gradient Δ*y*2(*x*, *y*) is defined in an analogous manner. To handle pixels on the borders of the image, calculate energy by defining the leftmost and rightmost columns as adjacent and the topmost and bottommost rows as adjacent. For example, to compute the energy of a pixel (0, *y*) in the leftmost column, we use its right neighbor (1, *y*) and its left neighbor (*width* − 1, *y*).

Consider the 3-by-4 image with RGB values (each component is an integer between 0 and 255) as shown in the table below.

|  |  |  |
| --- | --- | --- |
| (255, 101, 51) | (255, 101, 153) | (255, 101, 255) |
| (255,153,51) | (255,153,153) | (255,153,255) |
| (255,203,51) | (255,204,153) | (255,205,255) |
| (255,255,51) | (255,255,153) | (255,255,255) |

* *Non-border pixel example.* The energy of pixel (1, 2) is calculated from pixels (0, 2) and (2, 2) for the *x*-gradient  
  *Rx*(1, 2) = 255 − 255 = 0,  
  *Gx*(1, 2) = 205 − 203 = 2,  
  *Bx*(1, 2) = 255 − 51 = 204,  
    
  yielding Δ*x*2(1, 2) = 02 +  22 + 2042 = 41620;  
    
  and pixels (1, 1) and (1, 3) for the *y*-gradient  
  *Ry*(1, 2) = 255 − 255 = 0,  
  *Gy*(1, 2) = 255 − 153 = 102,  
  *By*(1, 2) = 153 − 153 = 0,  
    
  yielding Δ*y*2(1, 2) = 02 + 1022 + 02 = 10404.  
    
  Thus, the energy of pixel (1, 2) is 41620 + 10404 = 52024. Similarly, the energy of pixel (1, 1) is 2042 + 1032 = 52225.
* *Border pixel example.* The energy of the border pixel (1, 0) is calculated by using pixels (0, 0) and (2, 0) for the *x*-gradient  
  *Rx*(1, 0) = 255 − 255 = 0,  
  *Gx*(1, 0) = 101 − 101 = 0,  
  *Bx*(1, 0) = 255 − 51 = 204,  
    
  yielding Δ*x*2(1, 0) = 02 + 02 + 2042 = 41616;  
    
  and pixels (1, 3) and (1, 1) for the *y*-gradient  
  *Ry*(1, 0) = 255 − 255 = 0,  
  *Gy*(1, 0) = 255 − 153 = 102,  
  *By*(1, 0) = 153 − 153 = 0,  
    
  yielding Δ*y*2(1, 2) = 02 + 1022 + 02 = 10404.  
    
  Thus, the energy of pixel (1, 2) is 41616 + 10404 = 52020.

Table of all pixel energies for the RGB sample shown above.

|  |  |  |
| --- | --- | --- |
| 20808 | 52020 | 20808 |
| 20808 | 52225 | 21220 |
| 20809 | 52024 | 20809 |
| 20808 | 52225 | 21220 |

### Identifying Seams

#### Seam Representation

Seams are one dimensional arrays of integers. For a vertical seam, the index is the row and the corresponding element represents the column. Analogously, for a horizontal seam, the index is the column and the corresponding element represents the row.

For example given this layout, the black backgrounds represent a vertical seam starting at (1, 0), column 1 and row 0.

|  |  |  |
| --- | --- | --- |
| (0, 0) | (1, 0) | (2, 0) |
| (0, 1) | (1, 1) | (2, 1) |
| (0, 2) | (1, 2) | (2, 2) |
| (0, 3) | (1, 3) | (2, 3) |

Since this is a vertical seam and the height is 4, the seam will have 4 elements in it. So, the one dimensional array for the seam above would look like this

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Value | Corresponding Pixel | Looking down |
| 0 | 1 | (1,0) | Start (⬇ in context of the table) |
| 1 | 2 | (2,1) | Left (⬊ in context of the table) |
| 2 | 1 | (1,2) | Right (⬋ in context of the table) |
| 3 | 0 | (0,3) | Right (⬋ in context of the table) |

#### 

#### Loading a Vertical Seam

A vertical seam is loaded by starting at the provided starting column and row 0 and utilizing a **greedy algorithm**. The total energy of the seam is the sum of the energies of each pixel represented in the seam. From the starting pixel, follow a path to the next pixel which is the pixel in the next row with minimal energy that is adjacent to the current pixel. For example, if we start at pixel (1, 0). Looking down, we walk toward the next pixel that is on the next row, the options are

* directly forward - pixel(1, 1) i.e. keep the same column (⬇ in context of the table)
* to the right - pixel(0, 1) i.e. decrease the column by 1 (⬋ in context of the table)
* to the left - pixel(2, 1) i.e. increase the column by 1 (⬊ in context of the table)

Sometimes, more than one pixel has the same minimum energy value. In those cases

* First prefer to go directly forward (i.e. keep the same column) (⬇ in context of the table)
* Next prefer to go left (i.e. increase the column by 1) (⬊ in context of the table)

Note: you will only go right (i.e. decrease the column by 1) when it is strictly less than other options.

Store the column index for the winner in the corresponding location in the seam.

*Note: seams cannot cross over the bounds of an image.*

For example, given the following image represented by a table with RGB values in each cell,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ( 78,209, 79) | ( 63,118,247) | ( 92,175, 95) | (243, 73,183) | (210,109,104) | (252,101,119) |
| (224,191,182) | (108, 89, 82) | ( 80,196,230) | (112,156,180) | (176,178,120) | (142,151,142) |
| (117,189,149) | (171,231,153) | (149,164,168) | (107,119, 71) | (120,105,138) | (163,174,196) |
| (163,222,132) | (187,117,183) | ( 92,145, 69) | (158,143, 79) | (220, 75,222) | (189, 73,214) |
| (211,120,173) | (188,218,244) | (214,103, 68) | (163,166,246) | ( 79,125,246) | (211,201, 98) |

The vertical seam starting at pixel (1,0) is shown by bolded numbers in the table below. The table has the energy for each pixel and the color corresponds to the table representing an image with RGB values above.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 57685 | **50893** | 91370 | 25418 | 33055 | 37246 |
| **15421** | 56334 | 22808 | 54796 | 11641 | 25496 |
| **12344** | 19236 | 52030 | 17708 | 44735 | 20663 |
| **17074** | 23678 | 30279 | 80663 | 37831 | 45595 |
| 32337 | **30796** | 4909 | 73334 | 40613 | 36556 |

Total energy of the seam is 50893 + 15421 +12344 +17074 + 30796 = 126528.

The vertical seam would look like:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Value | Corresponding Pixel | Energy |
| 0 | 1 | (1, 0) | 50893 |
| 1 | 0 | (0, 1) | 15421 |
| 2 | 0 | (0, 2) | 12344 |
| 3 | 0 | (0, 3) | 17074 |
| 4 | 1 | (1, 4) | 30796 |

#### Finding a Minimal Vertical Seam

You will want to find the vertical seam with minimal energy. You will have to determine the seam starting at the first row of each column and keep track of the seam and the corresponding energy that is lowest. The minimal vertical seam for the image shown in the table of RGB values is shown with bolded energies.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 57685 | 50893 | 91370 | **25418** | 33055 | 37246 |
| 15421 | 56334 | 22808 | 54796 | **11641** | 25496 |
| 12344 | 19236 | 52030 | **17708** | 44735 | 20663 |
| 17074 | 23678 | **30279** | 80663 | 37831 | 45595 |
| 32337 | 30796 | **4909** | 73334 | 40613 | 36556 |

Total energy of the seam is 25418 + 11641 + 17708 + 30279 + 4909 = 89955.

The minimal vertical seam would look like:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Value | Corresponding Pixel | Energy |
| 0 | 3 | (3, 0) | 25418 |
| 1 | 4 | (4, 1) | 11641 |
| 2 | 3 | (3, 2) | 17708 |
| 3 | 2 | (2, 3) | 30279 |
| 4 | 2 | (2, 4) | 4909 |

If you have more than one seam with minimal energy, then prefer the one that starts at the lowest column index.

#### Removing a Vertical seam

Once a minimal vertical seam has been identified, you will need to remove the seam. You will iterate through each row and remove the column indicated by the value in the seam. Note this will be by copying the higher columns over into the lower index locations. We will not resize the array to make it smaller. However, we will update the width to represent the new width of the image.

For example the image above represented by the table of RGB values would become after removing the minimal vertical seam above.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ( 78,209, 79) | ( 63,118,247) | ( 92,175, 95) | (210,109,104) | (252,101,119) | ~~(252,101,119)~~ |
| (224,191,182) | (108, 89, 82) | ( 80,196,230) | (112,156,180) | (142,151,142) | ~~(142,151,142)~~ |
| (117,189,149) | (171,231,153) | (149,164,168) | (120,105,138) | (163,174,196) | ~~(163,174,196)~~ |
| (163,222,132) | (187,117,183) | (158,143, 79) | (220, 75,222) | (189, 73,214) | ~~(189, 73,214)~~ |
| (211,120,173) | (188,218,244) | (163,166,246) | ( 79,125,246) | (211,201, 98) | ~~(211,201, 98)~~ |

Note the width would now be 5. The other values are still there, but will never be accessed since we will use an updated width of 5 to control loops that access the values in the array.

#### Loading a Horizontal Seam (Extra Credit)

This is analogous to loading a vertical seam but now the path starts in the left hand column (i.e. index 0). In the horizontal seam, the index now represents the column and the value is the row.

Moving options are now:

* Moving forward keeps the same row. ➡
* Moving right increases the row. ⬊
* Moving left decreases the row. ⬈

For tiebreaks:

* First prefer to go directly forward (i.e. keep the same row) ➡
* Next prefer to go left (i.e. decrease the row by 1) ⬈

Note: you will only go right (i.e. increase the row by 1) when it is strictly less than other options.

The horizontal seam starting at pixel (0, 4) is shown by bolded numbers in the table below. The table has the energy for each pixel and the color corresponds to the table representing an image with RGB values above.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 57685 | 50893 | 91370 | 25418 | 33055 | 37246 |
| 15421 | 56334 | **22808** | 54796 | **11641** | 25496 |
| 12344 | **19236** | 52030 | **17708** | 44735 | **20663** |
| **17074** | 23678 | 30279 | 80663 | 37831 | 45595 |
| 32337 | 30796 | 4909 | 73334 | 40613 | 36556 |

Total energy of the seam is 17074 + 19236 + 22808 + 17708 + 11641 + 20663 =.109130.

The horizontal seam would look like:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Value | Corresponding Pixel | Energy |
| 0 | 3 | (0, 3) | 17074 |
| 1 | 2 | (1, 2) | 19236 |
| 2 | 1 | (2, 1) | 22808 |
| 3 | 2 | (3, 2) | 17708 |
| 4 | 1 | (4, 1) | 11641 |
| 5 | 2 | (5, 2) | 20633 |

#### Finding a Minimal Horizontal Seam (Extra credit)

Finding a horizontal seam is analogous to finding the minimal energy vertical seam. You will start at the first column of each row.

If you have more than one seam with minimal energy, then prefer the one that starts at the lowest row index.

#### Removing a Horizontal seam (extra credit)

Removing a horizontal seam is analogous to removing a vertical seam.

## Image File Format (PPM)

You are probably already familiar with common image formats such as JPEG, PNG, and GIF. However, these formats all use some type of data compression to keep file sizes relatively small. However, we are not ready to tackle these formats in C++.

We are going to use an image format that only requires basic text file I/O.

[The PPM (portable pixel map) format](http://netpbm.sourceforge.net/doc/ppm.html) is a specification for representing images using the RGB color model. PPM is not used widely because it is very inefficient (for example, it does not apply any data compression to reduce the space required to represent an image.) However, PPM is very simple, and there are programs available for Windows, Mac, and Linux that can be used to view ppm images. Even more conveniently, you can use an online tool with your browser to [view your PPM files online](http://paulcuth.me.uk/netpbm-viewer/) or convert into a widely supported format such as JPG. We will be using the [plain PPM version](http://netpbm.sourceforge.net/doc/ppm.html#plainppm), which stores the data in ASCII (i.e. plain text) rather than in a binary format. Since it is plain text, we will be able to use text file I/O to read and write these image files.

Note that the pixels in a PPM file are given row by row, so is essentially row-major ordering which is transposed from the array image format which is column major.

If you do create your own plain / ASCII PPM files make sure you **remove the comments**, since we are not addressing how to identify and ignore comment lines. Comments are lines that start with the ‘#’ character. I used [the GIMP](https://www.gimp.org/) to create the PPM files provided with the starting code.

### PPM File Specification

* First line: string “P3”
* Second line: width (number of columns) and height (number of rows)
* Third line: max color value (for us, 255)
* Rest of the file: list of RGB values for the image, expressed as a raster of rows, from top to bottom. Each row contains the RGB values (i.e., three values) for each column.

### PPM Examples

***Note:*** We have added colors to emphasize that every three numbers represent a single pixel. This version has each row on a separate line.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0  255 255 255 255 0 255 0 0 0 0 255 0  255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239 239 239 |

This version is the same as above, but with spaces added to help you visualize the file.

|  |
| --- |
| P3 4 4 255  0 0 0 255 0 0 0 0 0 0 255 0  255 255 255 255 0 255 0 0 0 0 255 0  255 255 0 0 0 255 125 0 255 255 0 125  0 0 255 255 255 0 125 125 125 239 239 239 |

This version has all numbers on a single line.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0 255 255 255 255 0 255 0 0 0 0 255 0 255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239 239 239 |

Alternatively, it could be saved with one pixel per line (i.e. 3 numbers per line) or even one number per line (this is what the GIMP did when I used it to create PPM files).

The following also works, but makes no sense to a human reading it.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0  0 0 0 0 255  0 255 255 255 255  0 255 0 0 0  0 255 0 255 255  0 0 0 255 125  0 255 255 0 125  0 0 255 255 255  0 125 125 125 239  239 239 |

Sample PPM File: [blocks.ppm](https://drive.google.com/drive/u/1/folders/1XHMQ3Cbmf9NeHNdZoy51MnHq82XZHDPw) 

### Viewing PPM files

You’ll need to view your PPM files to see the results of your program. Unfortunately, PPM is not supported by many image viewers.

Some options for viewing your files include:

* [Drag files onto this website (http://paulcuth.me.uk/netpbm-viewer/)](http://paulcuth.me.uk/netpbm-viewer/)
  + You don’t have to download any programs!
* [The GIMP](https://www.gimp.org/) is an open source version of Photoshop.
  + ***Warning:*** This is a very large program.
  + If you use the GIMP to create any PPM files, you will need to remove the comment line that it adds (line that starts with #).

1. [Seam carving assignment source: http://nifty.stanford.edu/2015/hug-seam-carving/](http://nifty.stanford.edu/2015/hug-seam-carving/) [↑](#footnote-ref-0)
2. Shai Avidan and Ariel Shamir. 2007. Seam carving for content-aware image resizing. ACM Trans. Graph. 26, 3, Article 10 (July 2007). DOI: https://doi-org.srv-proxy2.library.tamu.edu/10.1145/1276377.1276390 [↑](#footnote-ref-1)
3. [TAMU Web Color Pallette (https://brandguide.tamu.edu/web/web-color-palette.html)](https://brandguide.tamu.edu/web/web-color-palette.html) [↑](#footnote-ref-2)