Individual Homework: Seam Carving

Objectives

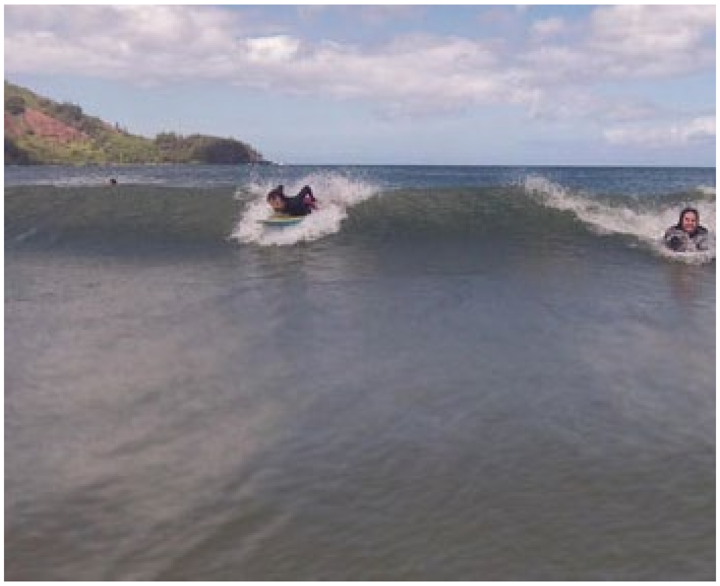
- Work with two-dimensional arrays.
    - Traversing
    - Accessing elements.
    - Staying within array bounds
- Work with dynamic arrays
- Compute information based on information from different parts of an array.
- Implement an algorithm

Overview

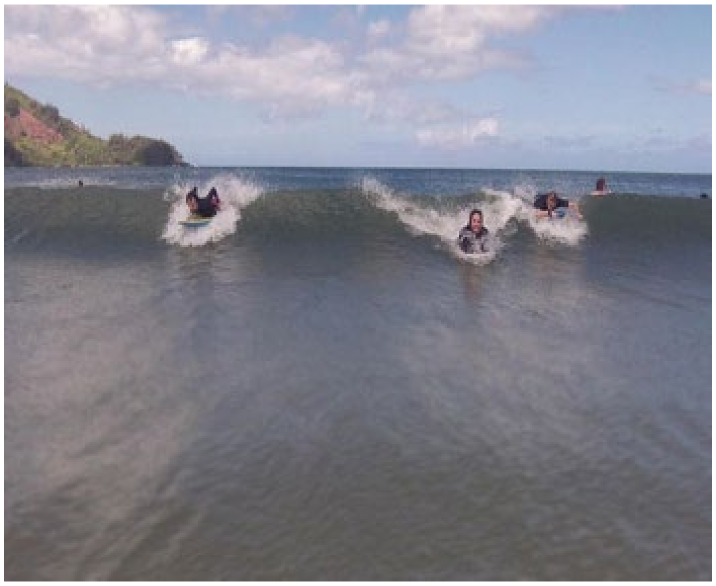
Sometimes you want to use an image, but it needs to be resized for your use. For example, what if we need to make this image narrower without changing the height?

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.



Rescaling ends up making the image look squashed.



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.

Watch the original SIGGRAPH video describing the technique.

Roadmap

This is a guide on how to approach this homework. Specific requirements for functions are in the Requirements section.

1. Get the start code and test files from Mimir.
   - Files
     - `seamcarving.cpp`
       - Update as needed
     - `functions.h`
       - Update as needed (you probably won't need to)
     - `functions.cpp`
       - Update as needed.
   - Test Files
     - Some PPM files.
       - Note that ppm files are names with widthXheight for your convenience as you select target widths for debugging and testing.
     - Feel free to make your own. I use the GIMP to convert images to PPM files.
       - Note: you will need to remove comments since our code will not handle comments.
2. This homework manipulates image files.
   - Review Images and RGB Color Model below

- Review Image File Format (PPM)
    - You will need to use an image viewer to view PPM files that your program generates. See “Viewing ppm files” section below for information on viewing your PPM files.

3. Review the Code.
    - Compile it and run it before making any changes.
        - Make sure you input the targetWidth and targetHeight the same as the width and height of the image, otherwise it will crash since some functions are not fully implemented yet.
            - So **all** other errors are introduced by you!
        - Note the filename and `targetWidth` and `targetHeight` are command line parameters.
            - If I compile the program as seam, here is an example run
                - `./seam blocks38X36.ppm 35 36`
            - To run on visual studio code, edit launch.json and add your test values for filename, targetWidth, and targetHeight as args and update as needed..
                - `"args": ["sunset200X125.ppm", "175", "100"],`
    - Read the header file.
        - You are required to implement the indicated functions in `functions.h`.
            - Do not make changes to provided functions!
        - Source files have been marked with `// TODO` where you need to write code.
    - Look over the provided code and functions.
        - In an ideal world we would have you write all the functions. To let you focus and shorten the time to implement, we’ve provided many functions that you should already know how to do or should after the next week or so.
            - Even though you may not have written these functions, they could be the basis for future exam problems.
        - The struct Pixel is defined in `functions.h`
4. Implement Functions
    - Write your code so that it is easy to understand!
        - See prompt from prior homework for more details.
    - Tackle functions in a meaningful order to reduce your coding and debugging time.
        - Look at dependencies.
            - `energy` function is needed for `getVerticalSeam`
            - `getVerticalSeam` relies on a correctly implemented `energy` function.
            - `removeVerticalSeam` relies on a seam created by `getVerticalSeam`
    - Recommended compiler line
        - `g++ -std=c++17 -Wall -Wextra -pedantic-errors -Weffc++ -fsanitize=undefined,address seamcarving.cpp functions.cpp`
    - Recompile and rerun after completing each function.

        - Note: You can test and debug before you add all functionality to a function. Get one aspect to work and then add functionality in measured increments until you complete the full functionality.
        - Check for errors.
        - If no errors, move on
        - Else, start debugging
    - Once you have some functionality, submit to Mimir.
        - If the basic tests for that function pass, move on
        - Else, start debugging
    - Continue by picking new tests and writing just enough code to pass them, adding more functionality each time.

Requirements

Allowed includes

- `<iostream>`
- `<fstream>`
- `<string>`
- `<sstream>`
- `<cmath>`
- `"functions.h"`

`energy` function

This function will return the dual-gradient energy of the pixel. How to calculate this is in the “Computing the energy of a pixel” section below.

- Signature
    - `int energy(const Pixel*const* image, int column, int row, int width, int height);`
- Return and Result
    - returns integer energy
- Parameters
    - First parameter is a pointer to a pointer to a Pixel.
        - With this structure, we can access our array with the [][] notation.
        - We are using dynamic arrays, so we won’t know the array dimensions at compile time.
            - Therefore, we cannot use [][] in the function signature since we would have to specify the second array dimension at compile time.
            - The const prevents the array from being modified within the function.
    - Second parameter `column` is the column which is the first value in the coordinate (column, row).
    - Third parameter `row` is the row which is the second value in the coordinate (column, row).

- Hints
  - Avoid accessing arrays outside array bounds!
  - Debugging hint:
    - Write a function that prints out a table of the energy for each pixel coordinate.
      - Helps see if your energies are correct.
  - If you write a set of if/else statements encompassing 9 conditions, you introduce more places to make errors.
    - You might make things easier on yourself if you figure out how to simplify things to avoid duplicating work and reducing the places for errors to occur.

getVerticalSeam function

This function will traverse through an image starting at the first row of the given column (`start_col`). See "Loading a vertical seam" below for how the traversal works. See "Seam Representation" below for how seams are represented.

- Signature
  - `int getVerticalSeam(const Pixel*const* image, int start_col, int width, int height, int* seam);`
- Return and Result
  - The function returns the total energy of the seam.
  - seam loaded with the column values for each row (i.e. index of the array)
- Parameters
  - First parameter is a 2d array of Pixels (structs) that hold a color value
  - Second parameter is the column to start the seam.
  - Third parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  - Fourth parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  - Fifth parameter is an array to be loaded with column values for each row.

removeVerticalSeam function

This function removes the pixels from the image corresponding to the vertical seam. See "Removing a Vertical seam" below for how the removal works. See "Seam Representation" below for how seams are represented.

- Signature
  - `void removeVerticalSeam(Pixel** image, int& width, int height, int* verticalSeam);`
- Return and Result
  - width decremented by 1
- Parameters
  - First parameter is a 2d array of Pixels (structs) that hold a color value
    - Array is in column-major order
      - Always indexed with [column][row]

  - Second parameter is the width of the array (i.e. the number of columns) needed for traversing the array
  - Third parameter is the height of the array (i.e. the number of rows) needed for traversing the array.
  - Fourth parameter is an array indicating which pixels should be removed.

Memory Leaks

Ensure the entire program does not have any memory leaks.

Extra Credit

Handle removing horizontal seams as well!

- You can implement the `getHorizontalSeam` and `removeHorizontalSeam` functions for extra credit. They are analogous to the Vertical versions. See Finding a Minimal Horizontal Seam and Removing a Horizontal seam.

Challenge

Update the loadImage function so that it handles comments. You have the tools to do this.

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.

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.

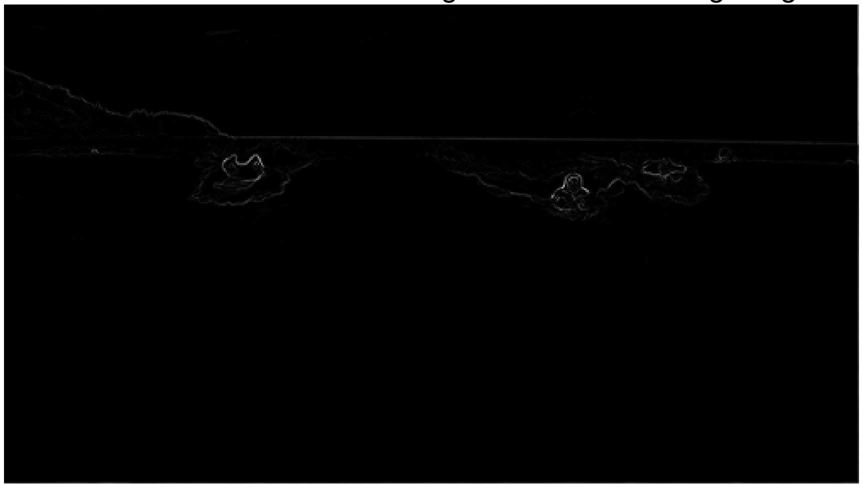
(0, 0)	(1, 0)	(2, 0)
(0, 1)	(1, 1)	(2, 1)
(0, 2)	(1, 2)	(2, 2)
(0, 3)	(1, 3)	(2, 3)

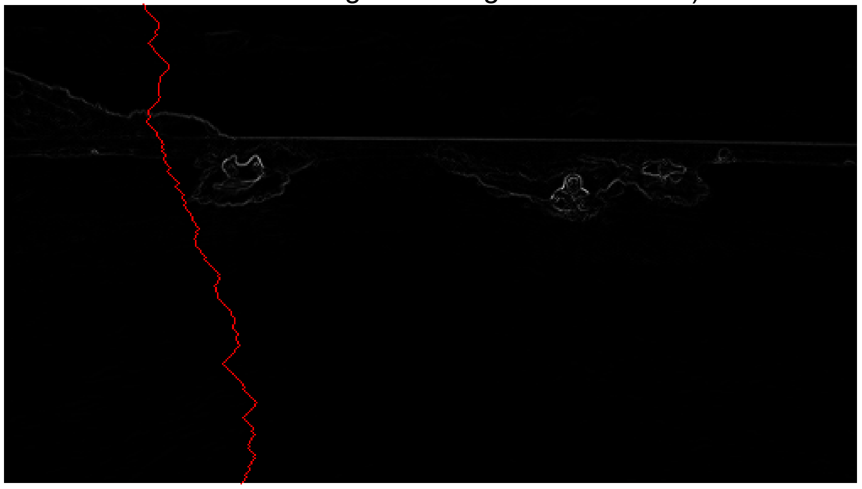
We will use a Pixel struct (defined in functions.h) that holds a value for red, green and blue. The image will be a 2 dimensional array of Pixels.

Seam Carving

Seam carving involves three major steps.

1. **Energy Calculation**
   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 "Computing the energy of a pixel" 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**
   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).
   
   Finding a horizontal seam is analogous.
3. **Seam Removal**
   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” below.

You will use the *dual-gradient energy function*: The energy of pixel (x, y) is $\Delta_x^2(x, y) + \Delta_y^2(x, y)$, where the square of the x-gradient $\Delta_x^2(x, y) = R_x(x, y)^2 + G_x(x, y)^2 + B_x(x, y)^2$, and where the central differences $R_x(x, y)$, $G_x(x, y)$, and $B_x(x, y)$ 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 $\Delta_y^2(x, y)$ 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. This is energy.ppm in your test files.

(0, 0): (255, 101, 51)	(1, 0): (255, 101, 153)	(2, 0): (255, 101, 255)
(0, 1): (255, 153, 51)	(1, 1): (255, 153, 153)	(2, 1): (255, 153, 255)
(0, 2): (255, 203, 51)	(1, 2): (255, 204, 153)	(2, 2): (255, 205, 255)
(0, 3): (255, 255, 51)	(1, 3): (255, 255, 153)	(2, 3): (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
  $R_x(1, 2) = 255 - 255 = 0$,
  $G_x(1, 2) = 205 - 203 = 2$,
  $B_x(1, 2) = 255 - 51 = 204$,

  yielding $\Delta_x^2(1, 2) = 0^2 + 2^2 + 204^2 = 41620$;

  and pixels (1, 1) and (1, 3) for the *y*-gradient
  $R_y(1, 2) = 255 - 255 = 0$,
  $G_y(1, 2) = 255 - 153 = 102$,
  $B_y(1, 2) = 153 - 153 = 0$,

  yielding $\Delta_y^2(1, 2) = 0^2 + 102^2 + 0^2 = 10404$.

  Thus, the energy of pixel (1, 2) is 41620 + 10404 = 52024. Similarly, the energy of pixel (1, 1) is $204^2 + 103^2 = 52225$.

- *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
  $R_x(1, 0) = 255 - 255 = 0$,
  $G_x(1, 0) = 101 - 101 = 0$,
  $B_x(1, 0) = 255 - 51 = 204$,

  yielding $\Delta_x^2(1, 0) = 0^2 + 0^2 + 204^2 = 41616$;

and pixels (1, 3) and (1, 1) for the *y*-gradient
$R_y(1, 0) = 255 - 255 = 0$,
$G_y(1, 0) = 255 - 153 = 102$,
$B_y(1, 0) = 153 - 153 = 0$,

yielding $\Delta_y^2(1, 0) = 0^2 + 102^2 + 0^2 = 10404$.

Thus, the energy of pixel (1,0) 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

Hint: Use the energy.ppm file. Create a loop that calculates the energy of each pixel and see if it matches the energies above!

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 where you are driving from the top of the image to the bottom of the image.

Index	Value	Corresponding Pixel	Following a Path
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). Following a path, we walk down toward the next pixel that is on the next row, the options are

- go directly forward - pixel(1, 1) i.e. keep the same column (⇣ in context of the table)
- go to the left - pixel(2, 1) i.e. increase the column by 1 (↘ in context of the table)
- go to the right - pixel(0, 1) i.e. decrease 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. This is seam.ppm in your test files.

(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.

For a vertical seam, the **index is the row**, and the **value is the column**. So, the vertical seam would look like:

Index/Row	Value/Col	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 image 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. $\boxed{\rightarrow}$
- Moving right increases the row. ↘
- Moving left decreases the row. ↗

For tiebreaks:

- First prefer to go directly forward (i.e. keep the same row) $\boxed{\rightarrow}$
- 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.

For a horizontal seam, the **index is the column** and the **value is the row**. So, the horizontal seam would look like:

Index/Col	Value/Row	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.

PPM Image File Format

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 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. You can view your PPM files on a website or use a program on your computer (See below). We will be using the plain PPM version, 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 identifying and ignoring comments. Comments are lines that start with the '#' character. I used the GIMP to create the PPM files provided with the starting code.

PPM File Specification

- Preamble
  - 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. Each three numbers represent the red, green, and blue values for a pixel. The pixels start at row 0, are provided column by column until all rows and columns have values. There are essentially no requirements for how the numbers are placed on lines. See examples below.

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
```

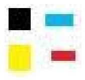
This version has all numbers on a single line.

```
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 found in your test files: blocks38X36.ppm 

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/)
  - You don't have to download any programs!
- The GIMP 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 (i.e. lines that starts with #).
- For Windows users, IrfanView is a free image viewer
  - Check the image-only box when installing.
  - Consent to allow IrfanView to associate to your image files.
  - After completing the installation, the image can be viewed by double-clicking on the PPM file.