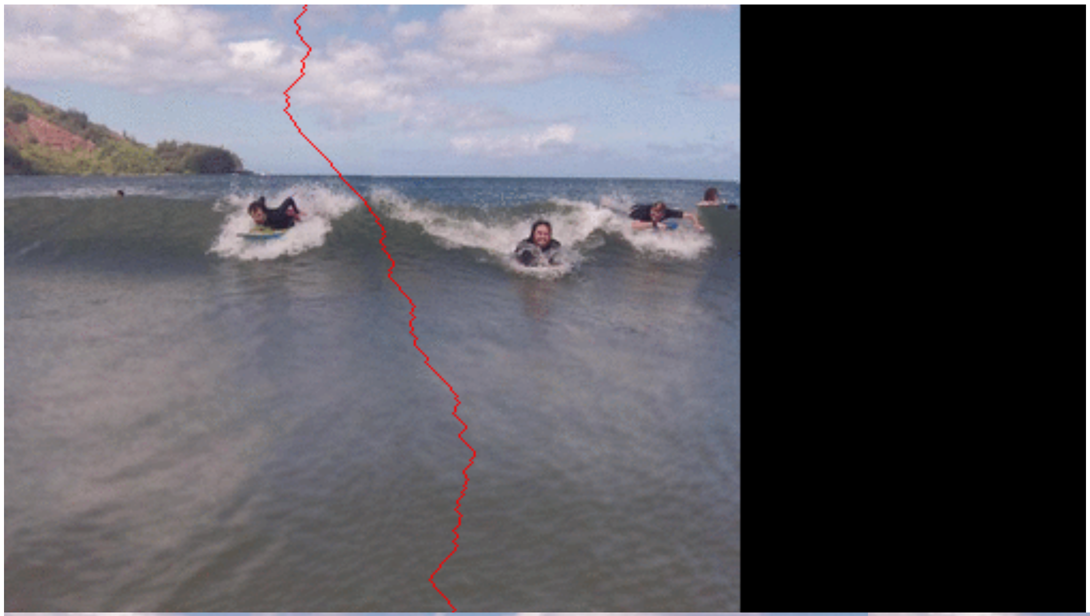
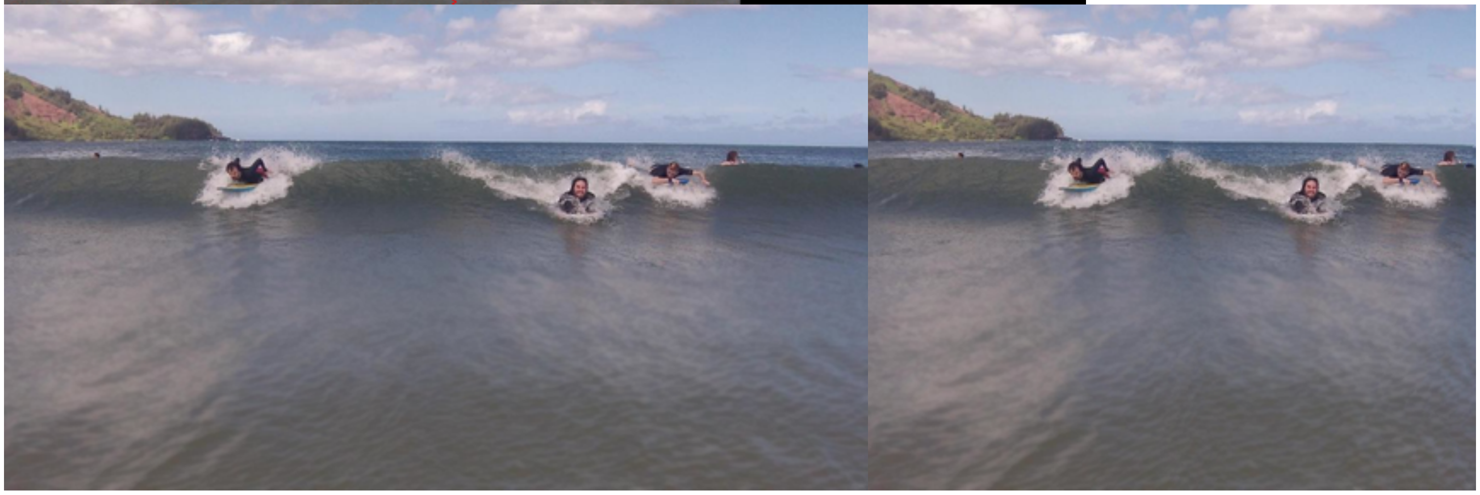
Project 2: Seam Carving

Seam Carving

Seam-carving is a content-aware image resizing technique where the image is reduced in size by one pixel of width (or height) at a time. A vertical seam in an image is a path of pixels connected from the top to the bottom with one pixel in each row; a horizontal seam is a path of pixels connected from the left to the right with one pixel in each column. Below left is the original 505-by-287 pixel image; below right is the result after removing 150 vertical seams, resulting in a 30% narrower image. Unlike standard content-agnostic resizing techniques (such as cropping and scaling), seam carving preserves the most interest features (aspect ratio, set of objects present, etc.) of the image. Although the underlying algorithm (https://www.youtube.com/watch?v=6NcIJXTlugc) is simple and elegant, it was not discovered until 2007. Now, it is now a core feature in Adobe Photoshop and other computer graphics applications.



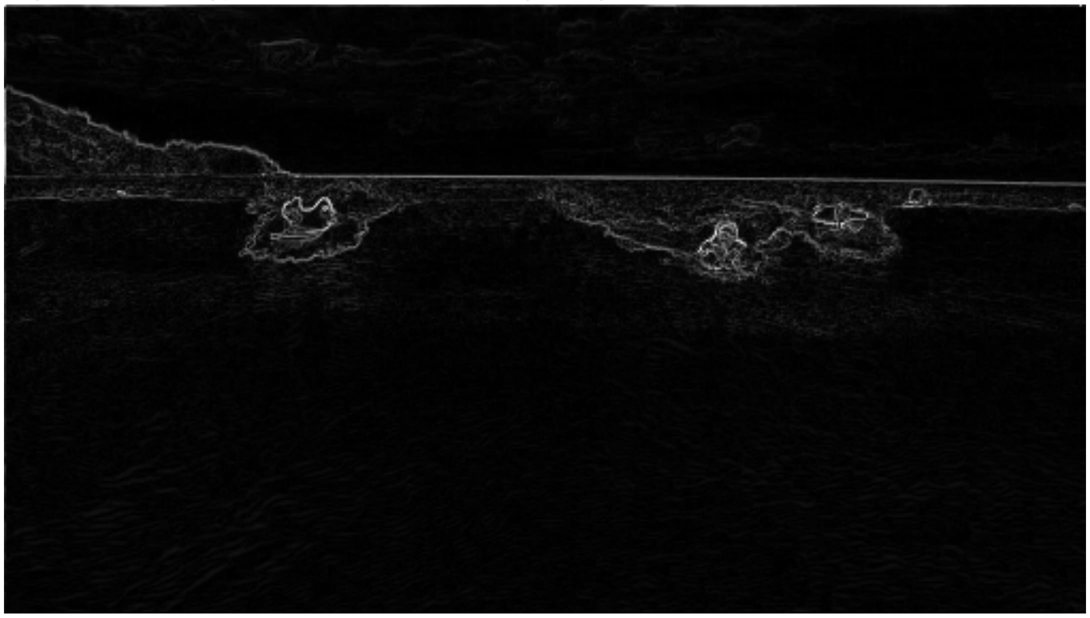


In this assignment, you will create a data type that resizes a H-by-W image using the seam-carving technique.You will first compute the dual-gradient energy function, and then find vertical “seams” – paths from the top to the bottom of the image – such that the sum of the dual-gradient energy values in the pixels along the path is as small as possible

Notation. In image processing, pixel (y, x) refers to the pixel in column x and row y, with pixel (0,0) at the upper-left corner and pixel (H-1, W-1) at the lower-right corner:

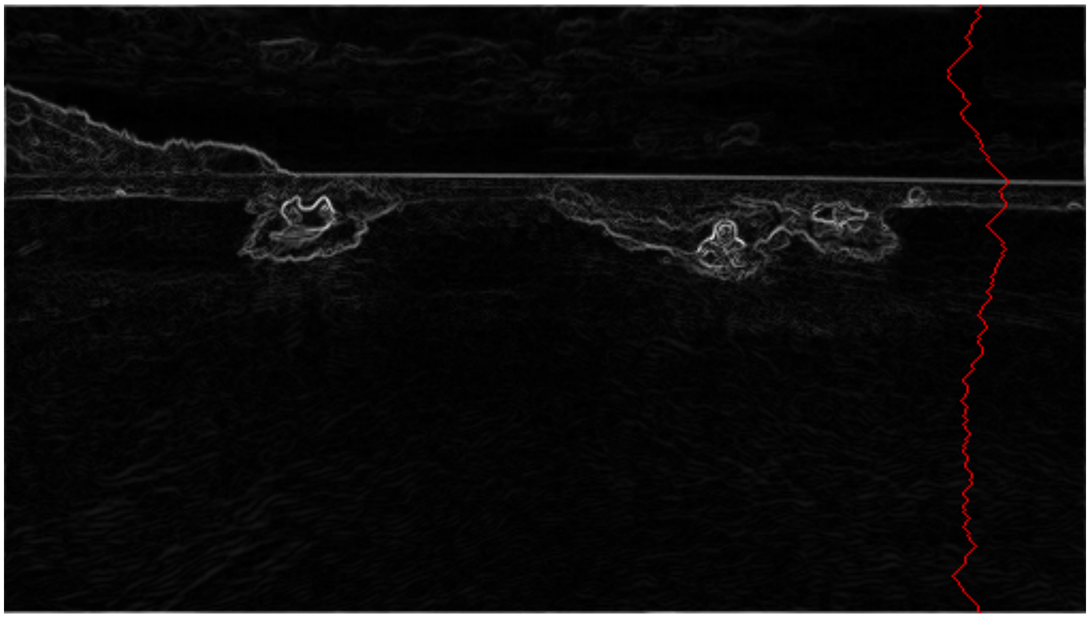
(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)
(3, 0)	(3, 1)	(3, 2)

Energy calculation. The first step is to calculate the energy of a pixel, which is a measure of its perceptual importance – the higher the energy, the less likely that the pixel will be included as part of a seam (as you will see in the next step). In this project, you will use the dual-gradient energy function, which is described below. Here is the dual-gradient energy function 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 (the original author of this assignment) on the left and the ocean behind him). The seam carving technique avoids removing such high-energy pixels.

Seam identification. The next step is to find a vertical seam of minimum total energy. The seam is a path through pixels from top to bottom such that the sum of the energies of the pixels is minimal. You will identify the minimum-energy seam using dynamic programming.



Seam removal. The final step is removing from the image all the pixels along the vertical seam.

Part 1: Dual-Gradient Energy Function

In this part, you will write the function

```
void calc_energy(struct rgb_img *im, struct rgb_img **grad);
```

The function will compute the dual-gradient energy function, and place it in the `struct rgb_img  *grad`.

The energy of pixel (y, x) is $\sqrt{\Delta_x^2(y,x) + \Delta_y^2(y,x)}$. Here,

$\Delta_x^2(y,x) = R_x(y,x)^2 + G_x(y,x)^2 + B_x(y,x)^2$

$\Delta_y^2(y,x) = R_y(y,x)^2 + G_y(y,x)^2 + B_y(y,x)^2$

R_x , R_y , ..., B_y are the differences in the red, green, and blue components of pixels surrounding the central pixel, along the x and y-axis. For example,

$R_x(y,x) = (y, x+1)_{red} - (y, x-1)_{red}$

In the mage below, for the pixel $(2, 1)$,

$R_x(2,1) = 255 - 255 = 0$

$G_x(2,1) = 205 - 203 = 2$

$B_x(2,1) = 255 - 51 = 204$

$R_y(2,1) = 255 - 255 = 0$

$G_y(2,1) = 255 - 153 = 102$

$B_y(2,1) = 153 - 153 = 0$

$\sqrt{\Delta_x^2(y,x) + \Delta_y^2(y,x)} = \sqrt{52024}$

For pixels at the edge of the image, you should "wrap around" the image. For example, in the image below for the the pixel $(0, 1)$

$R_x(0,1) = 255 - 255 = 0$

$G_x(0,1) = 101 - 101 = 0$

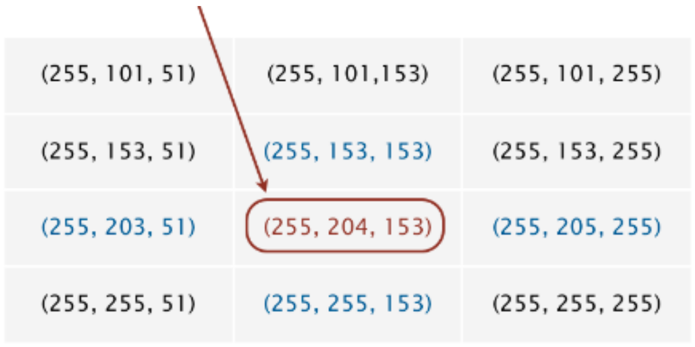
$B_x(0,1) = 255 - 51 = 204$

$R_y(0,1) = 255 - 255 = 0$

$G_y(0,1) = 153 - 255 = -102$

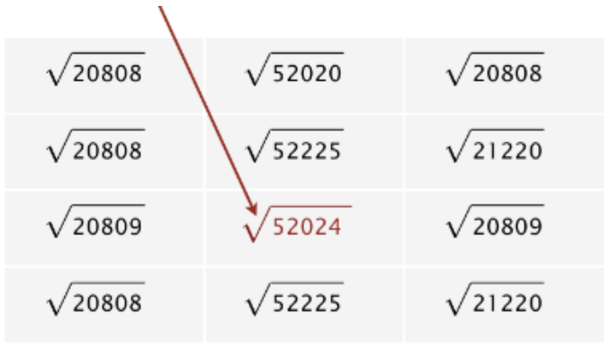
$B_y(0,1) = 153 - 135 = 0$

$\sqrt{\Delta_x^2(y,x) + \Delta_y^2(y,x)} = \sqrt{52020}$



(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)

a 3-by-4 image (RGB values)



$\sqrt{20808}$	$\sqrt{52020}$	$\sqrt{20808}$
$\sqrt{20808}$	$\sqrt{52225}$	$\sqrt{21220}$
$\sqrt{20809}$	$\sqrt{52024}$	$\sqrt{20809}$
$\sqrt{20808}$	$\sqrt{52225}$	$\sqrt{21220}$

dual-gradient energies

You are storing the dual-gradient energy in an image. To do that, divide the original energy by 10, and cast it to (`uint8_t`). For each pixel, set the r, g, and b channels to the same value (the energy divided by 10 and cast to `uint8_t`).

The resultant dual-gradient energy of the image `3x4.png` is:

```
        14      22      14
        14      22      14
        14      22      14
        14      22      14
```

(You can print out the dual-gradient energy using

```
  struct rgb_img *grad;

  calc_energy(im,  &grad);

  print_grad(grad);
```

)

Part 2: Cost Array

Define the function `dynamic_seam(struct rgb_img *grad, double **best_arr)` which allocates and computes the dunamic array `*best_arr` .

`(*best_arr)[i*width+j]` contains the minimum cost of a seam from the top of grad to the point (i, j).

For `6x5.png` , the dual-gradient image is

```
24      22      30      15      18      19
12      23      15      23      10      15
11      13      22      13      21      14
13      15      17      28      19      21
17      17      7       27      20      19
```

The best array is

```
24.000000       22.000000       30.000000       15.000000       18.000000       19.00
0000
34.000000       45.000000       30.000000       38.000000       25.000000       33.00
0000
45.000000       43.000000       52.000000       38.000000       46.000000       39.00
0000
56.000000       58.000000       55.000000       66.000000       57.000000       60.00
0000
73.000000       72.000000       62.000000       82.000000       77.000000       76.00
0000
```

Part 3: Recover the seam

Write a function

```
void recover_path(double *best, int height, int width, int **path);
```

This function allocates a path through the minimum seam as defined by the array best.

For the best array above, the path is `[3, 4, 3, 2, 2]`.

Part 4: Write a function that removes the seam

```
void remove_seam(struct rgb_img *src, struct rgb_img **dest, int *path);
```

The function creates the destination image, and writes to it the source image, with the seam removed.

Part 5: Not for submission

Run your program to repeatedly remove seams from an image, and visualize the result

```
    struct rgb_img *im;
    struct rgb_img *cur_im;
    struct rgb_img *grad;
    double *best;
    int *path;

    read_in_img(&im, "HJoceanSmall.bin");

    for(int i = 0; i < 5; i++){
        printf("i = %d\n", i);
        calc_energy(im,  &grad);
        dynamic_seam(grad, &best);
        recover_path(best, grad->height, grad->width, &path);
        remove_seam(im, &cur_im, path);

        char filename[200];
        sprintf(filename, "img%d.bin", i);
        write_img(cur_im, filename);


        destroy_image(im);
        destroy_image(grad);
        free(best);
        free(path);
        im = cur_im;
    }
    destroy_image(im);
```

What to submit

A file `seamcarving.c` that implements all the functions in `seamcarving.h` .

Credit: the assignment was originally designed by Josh Hug. Port to C by Michael Guerzhoy.