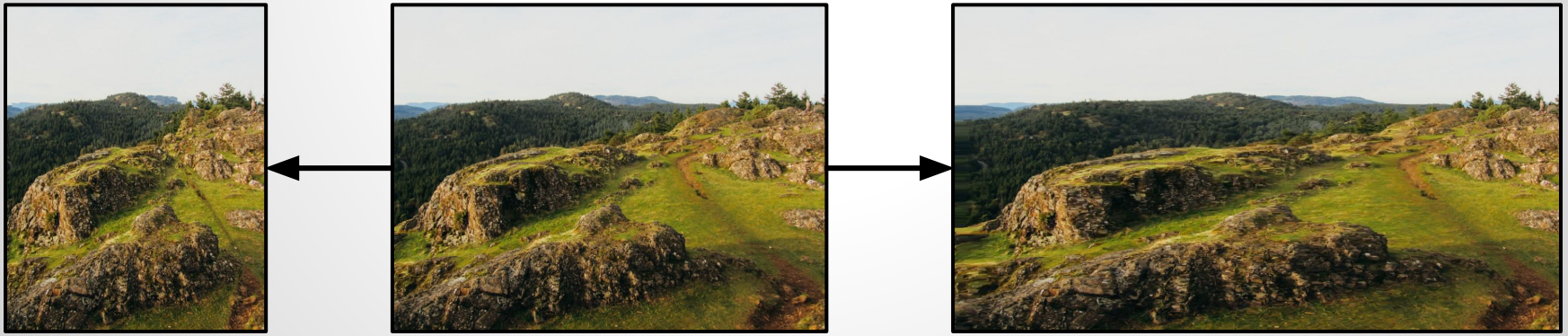
# Comp Photography
# Final Project

Full Name1: Daniel Kane
GTID1: 902970183
Full Name2: Farzon Lotfi
GTID2: 902462754

Spring 2019

# Ripping at Seams

Application that resizes images with minimal distortion using graph cuts.



# The Goal of Your Project

Original project scope: Growing/shrinking the width/height or an image using seam carving. Stitch two images together using seam carving.

What motivated you to do this project? We talking about the subject in class but never discussed the details beyond square differencing or had an assignment about it. So we decided to implement it ourselves.

# Scope Changes

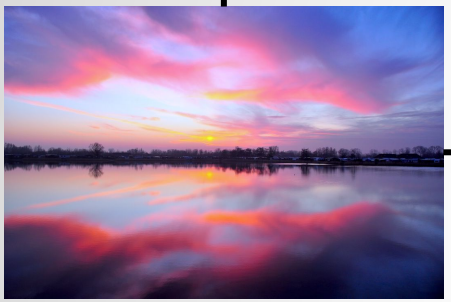
- Did you run into issues that required you to change project scope from your proposal?
  - Yes, we had speed issues that required us to change languages and timing issues that required us to drop stitching.
- Give a detailed explanation of what changed. Stitching two images together using seam carving proved to be too time consuming and difficult alongside resizing images so we decided to just implement the resizing features. We also changed from JavaScript/OpenProcessing to using C++ because seam carving was too slow on the browser.
  - Don’t just take our word for it feel free to try out our working js demo and watch your browser slow to a crawl when you swap test.jpeg for test.jpg computing only 10 seams for removal.

# Showcase

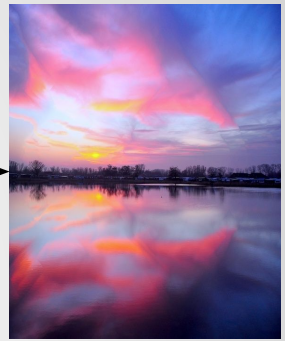
Input

Output

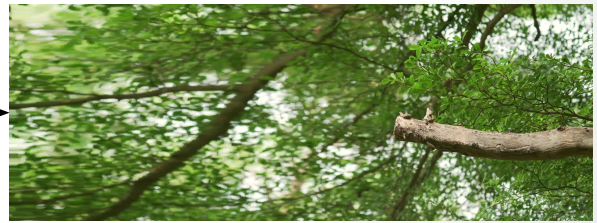
Shrink the width



Shrink the height



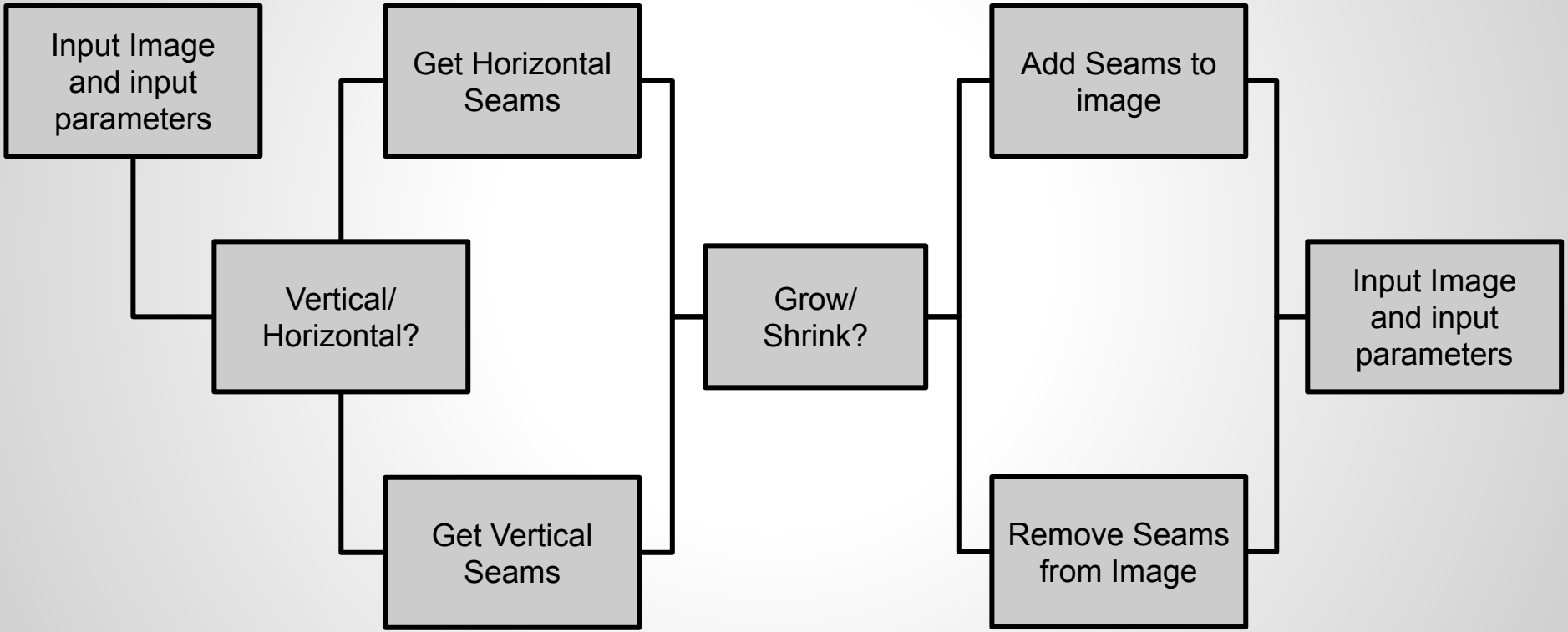
Grow the width



Grow the height

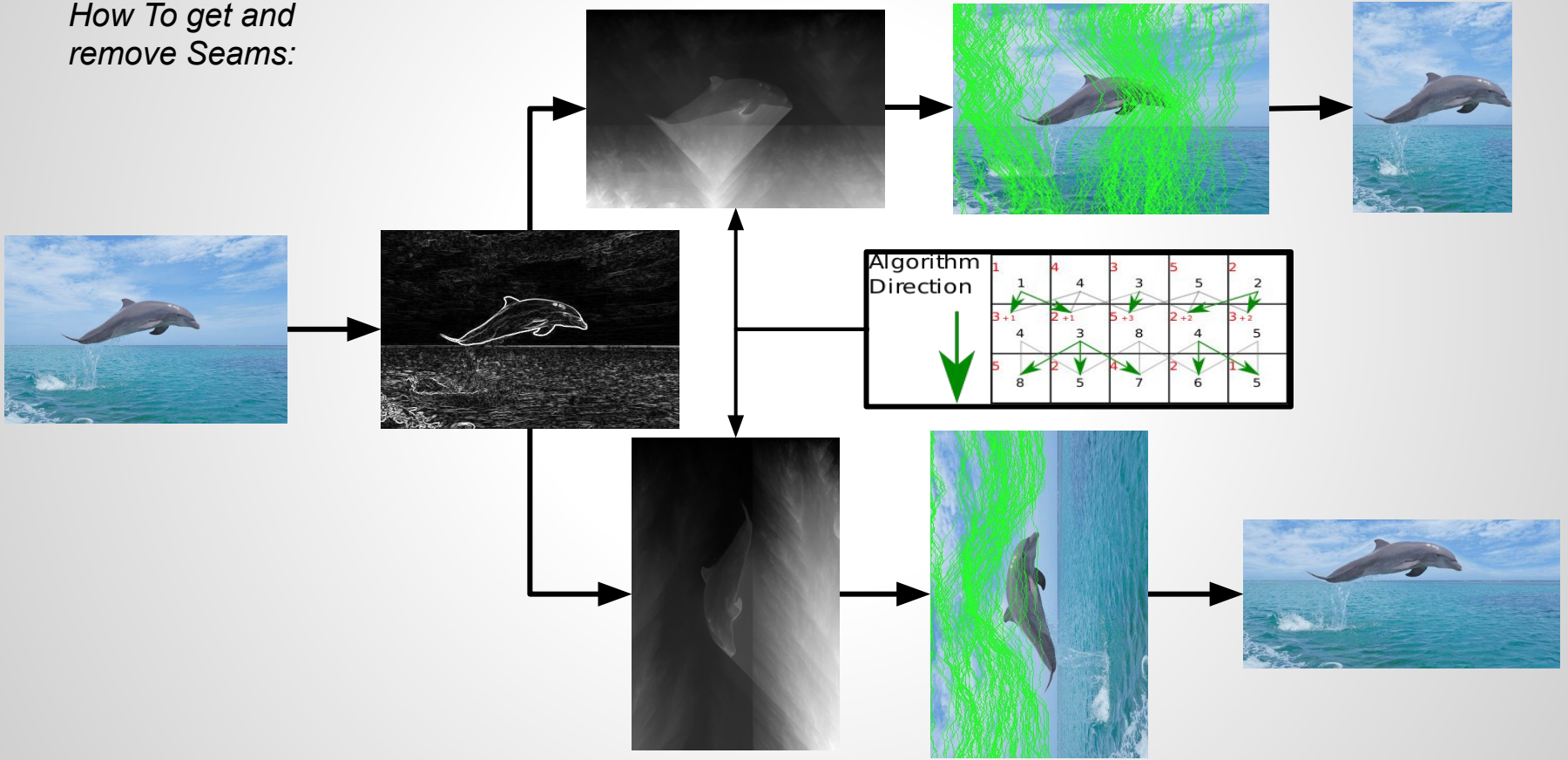
Notice how the prominent feature (the big branch) remains largely unchanged after resize.

# Project Pipeline



# Project Pipeline

*How To get and remove Seams:*



# Demonstration:  Result Sets

Shrunk 200 Pixels Horizontal ⟵ Original Image ⟶ Grown 200 Pixels Horizontal

YouTube Video Demo:  https://youtu.be/Curd1u6-itE

# Demonstration:  Result Sets

Grown 200 Pixels Horizontal

Original Image

Shrunk 200 Pixels Vertically

# Project Development

- Our progress is code complete. We started by first writing the gradient function. Originally we tried to do a convolution of partial derivatives for X and Y like mentioned in the paper (see *GetEnergyImg* in source code for more details on this approach).
- But this turned out to not give good results when we were trying to compute our Intensity Matrix (where the score/edge weight of or minimum weight path resided). So we swapped this out for a similar technique that used the Sobel edge detector. (see *computeGradientMagnitude* for more details in functional desc. slide).
- Next we wanted to compute the Intensity matrix. Here we wrote a DP algorithm to computed a path for the minimum intensity from the current path of the 3 next pixels in our graph (see *computePathIntensityMat* in functional desc. slide).
- At this point we get the least important path returned as an int vector of rows mapped to minimum columns, this is the same for both the growth and shrinking cases we covered. (see getLeastImportantPath for More details on this  in function desc. slide).

$$e_1(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

# Project Development

- The remaining two functions we had to write are complements of one another *removeLeastImportantPath and addLeastImportantPath*. Here we take the  seam we just got and in the removal case memcpy a row up to the min seam and then after the min seam col  for every row.
- Difficulties encountered initially after doing the removal  was were were able to scale the columns but not rows. We quickly found that the minimum code to get rows working was to transpose the image and run the original algorithm then transpose the final image.
- Next in the *addLeastImportantPath case To perform this we need two image clones of the original, one that would shrink per seam and another that would grow per seam. The image that would shrink per seam is where got the least import path from and then we would add this seam to the right of the growing image.*
- *At this point the rest of the project was just UI level changes. First and foremost we wanted a CLI  mode were we could precisely pick the seams we wanted to remove/add and the images we wanted to work with. Next we wanted a slider so we could dynamically resize our image. While the app was running.*

# Computation: Code Functional Description

- computeGradientMagnitude:
  - Computes this function $\sqrt{dx^2 + dy^2}$ where dx is the image gradient in the x direction and dy is the image gradient in the y direction. We use openCV's Sobel filter to compute these gradients on the image.
- computePathIntensityMat:
  - Uses the energy matrix computed by computeGradientMagnitude to find the paths with the least accumulative energy for each row/column in the image. It does this by choosing the smallest energy pixel of the three neighboring pixels in direction of path for each pixel in the image.
- getLeastImportantPath:
  - This function finds the finds the smallest value pixel of the last row/col of the matrix computed by computePathIntensityMat. It then works its way backwards picking the least valued neighboring pixel and adding it to the seam. Once it hits the other side of the image the resulting seam will be the path with the least energy in the image.
- removeLeastImportantPath / addLeastImportantPath:
  - This function removes/adds the least important seam by doing mem-copies for the length of the seam/path. If we are removing a path we copy everything except the pixel represented by the seam. IF we are adding a path we copy everything to the left of the seam pixel the seam pixel again and then everything to the right of the pixel (in the new image row or column).

# Computation: Code Functional Description

- OpenCV:
  - We used OpenCV for C++ in every part of the project. OpenCV's Mat object is useful for storing images and has several useful functions that can be used with it. Any large scale matrix computation was done using openCV.
  - The simple Slider GUI was done using openCV's highgui.

# Any additional details?

- Read the dependencies file for how to build with more details:
- For all make sure you have c++17 standard compiler.
- For Mac
  - brew install opencv
- For Linux
  - sudo apt-get install libopencv-dev pkg-config
- For Windows
  - install the linux subsystem for windows and Xmig for X11 windows
  - run the linux steps
  - Or build\find windows binaries from somewhere
- After installing dependencies you can now call make

# Teamwork

- Describe your original division of labor, and how that worked out.
  - Daniel Kane
    - Growing Images
  - Farzon Lotfi
    - Shrinking Images

# Resources

- Dolphin photo:
  - https://www.natgeokids.com/wp-content/uploads/2014/06/dolphins-facts-3s.jpg
- Landscape Photos
  - https://www.shutterstock.com/image-photo/beautiful-scenery-lake-sky-575688598
  - https://image-cdn.hypb.st/https%3A%2F%2Fhypebeast.com%2Fimage%2F2019%2F04%2Fearth-day-fashion-industry-tips-tw.jpg?w=960&cbr=1&q=90&fit=max
  - https://image.ibb.co/ey311m/pexels_photo_459225_75.jpg
  - https://www.victoriatrails.com/images/photos/lone-tree-hill-3.jpg
  - https://ortega3d.files.wordpress.com/2010/03/ortega_arvore_port011.jpg
- Paper resource
  - https://perso.crans.org/frenoy/matlab2012/seamcarving.pdf

# Appendix: Your Code

**Code Language: C++**

**List of code files:**

- **Main.cpp**
- **SeamCarving.cpp**
- **SeamCarving.hpp**
- **SeamCarvingHorizontal.cpp**
- **SeamCarvingVertical.cpp**

# Credits or Thanks

- Shai Avidan and Ariel Shamir for writing  paper on Seam carving that was actually comprehensible.