Boris Burkov (bb393)
Najla Elmachtoub (ne48)
Fahad Karim (fk66)

FACES

Introduction

When taking photographs, it is difficult to capture a perfect photo. In portraits, for example, one might take a photo of someone with her eyes closed or mouth open. This is the motivation for our project, "Faces." With this tool, multiple photos of a person can be taken in sequence, and one can select out desired regions from each to create a composite image. Another application is merging two different faces together. Especially for this latter application, seams between the two faces need to be blended so that the resulting image is continuous in colors and gradients. We want our tool to greatly outperform a simple "cut and paste" job. On the Nokia N900, we developed a user interface in which the user can take two consecutive images and then use the touch screen to highlight parts from each; the selected areas will be merged together to form a new, corrected image that is saved to the phone.

Related Work

Our work is strongly based on the papers listed below.

Interactive Digital Photomontage: Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, Michael Cohen.

Fast Approximate Energy Minimization via Graph Cuts**:** Yuri Boykov, Member, IEEE, Olga Veksler, Member, IEEE, and Ramin Zabih, Member, IEEE

Poisson Image Editing: Patrick Perez, Michel Gangnet, Andrew Blake.

In addition, we used Vladimir Kolmogorov's code for building graph datastructures and running graph cuts on them (graph.cpp/h, maxflow.cpp/h, block.cpp/h). Everything else in our project we implemented ourselves. The only new OpenCV functions we used are related to creating and querying histograms. A discussion on this is provided below.

Technical Description

Segmentation

Given two images, a set of constraints, and a set of cost functions, we seek to find the best labeling for each pixel in the composite image. This is achieved by constructing an appropriate graph and running graph cuts software developed by Vladimir Kolmogorov. Both input images need to be of the same size, and each image needs a corresponding "mask" of the same size as

well. This mask represents a constraint—pixels covered by the mask *must* be selected by the graph cuts algorithm (unless the two masks overlap). If the input images and masks have N pixels, the graph we construct will have N + 2 nodes – one for each pixel and two additional source and sink nodes. There is a bidirectional edge between each pair of adjacent pixels, and each pixel also links to both the source and the sink. As explained in the Photomontage paper, the capacities on these edges depend on what type of objective you are following. For our project, we decided to use a combination of different objectives and to let the user adjust parameters to control the importance of each one:

Skin cost: We construct a histogram of all the colors in the "highlighted region" using OpenCV's histogram datastructure. For each other pixel in an image, we can then query the histogram to figure out how well that pixel fits the colors in the highlighted region. Increasing the "skin cost" parameter will give a larger weight to this parameter. If, for example, you highlight part of someone's hair, this parameter will help to select other pixels of the same hair color.

Intensity cost: This parameter controls how important it is that neighboring pixels in the composite image have similar intensities. Larger values represent greater importance.

Gradient cost: This parameter controls how important it is that neighboring pixels in the composite image have similar gradients. Again, larger values represent greater importance.

f and g are the two images.

p and q are neighboring pixel coordinates in the images.

$I_f(p)$ is the intensity of pixel p in image f.

$G_f(p)$ is the horizontal gradient of pixel p in image f.

$$\text{intensity\_cost} = \text{max\_intensity\_cost} \cdot \frac{\left(\|I_f(p) - I_g(p)\| + \|I_f(q) - I_g(q)\| \right)}{2}$$

$$\text{gradient\_cost} = \text{max\_intensity\_cost} \cdot \frac{\left(\|G_f(p) - G_g(p)\| + \|G_f(q) - G_g(q)\| \right)}{2}$$

Distance cost: This parameter roughly controls how far away from the user-highlighted region the labeling for that image should 'spread.' Of course, if only small regions in both images are highlighted, then this distance cost will not do as much since all the pixels do need to be labeled. For example, if one highlights the entire upper-half of a face in image1 and only the bottom lip in image2, setting a low distance cost will make the labeling from image2 more reserved. It is likely that only the bottom lip and maybe a small area around it will be used from image2. Instead of actually calculating the distance from each pixel to the closest part of the highlighted region, we approximate this distance cost by running a mean filter across the image's mask. The distance cost effectively varies the kernel's size since increasing the kernel size means allowing more spread. The resulting values in the altered mask are proportional to the distance cost assigned to each pixel.

The sum of the distance cost and the skin cost more or less defines the capacities of the edges from each pixel node to the source and sink nodes. These are costs that are independent of neighboring pixels. Here, the source and sink nodes represent image1 and image2. The sum of the gradient and intensity costs for two neighboring pixels sets the capacity for the edge between them. Running graph cuts on this algorithm returns a labeling of each pixel to either source or sink. This, effectively, labels each pixel as either image1 or image2.

Gradient domain fusion

Once we have decided which regions we will be using from each image, we must still decide what composite image to ultimately return. A basic strategy is to just take pixels from the respective images as specified by the labeling. This works reasonably well for some cases, particularly when working with two images of the same person, where skin tones are very similar. However, in other cases, we may run into cases where there are noticeable seams between the different regions. In order to resolve this issue, we use a technique outlined in "Interactive Digital Photo Montage" and specified in "Poisson Image Editing": gradient domain fusion.

Creating an image that is smooth in appearance can be done by solving for an image that is similar to the naive composite, but has exactly the gradient that would arise from "copying and pasting". So if we view the resulting image as a function F on R2, and let f be the first image and g the second, we need it to satisfy two equations.

$$\Delta F = \Delta g$$
$$F|_{\partial\Omega} = f|_{\partial\Omega}$$

Inside the smaller region segmented by graph cuts, we need the gradient of F to be as close as possible to the gradient of g, and we need the value of F to equal the value of f on the boundary. This ensures that we have a seamless color transition on the boundary, while still forming the appropriate composite of the two original images. An F that satisfies this must satisfy Poisson's equation. That is, the Laplacian of F must be equal to the Laplacian of g inside the region, with the same boundary condition.

To actually solve for F, we must solve a discretized version of the problem for each channel and then re-merge the channels to get the final solution. We can give the discrete version of Poisson's equation at each pixel, and add extra terms to the equations near the boundary, which are pixels that are not in the region, but have a pixel in their 4-neighborhood that is. We can represent the entirety of the problem as a system of linear equations, with one equation per pixel in the region. Then, using cvSolve we can find an exact solution to this system, which gives a good result in that channel. Merging them together, we get the final, smoothed result.

Unfortunately, this approach has a noticeable drawback. Allocating this enormous matrix and using cvSolve on such a large system are extremely taxing on both memory and time. This

problem is exacerbated on the phone by restrictions on memory and processing power. Although the best solution is to use an approximation technique to solve the differential equation, we settled on a simple solution that worked reasonably well. If we found that the smaller of the two regions was still too large to be tractable, we downsampled the entire discretized problem with a bicubic downsampling until it became the appropriate size. We then solved this workable problem, and upsampled back up to the original size. As long as there are no more than one or two downsamples, the results are still quite nice.

To run

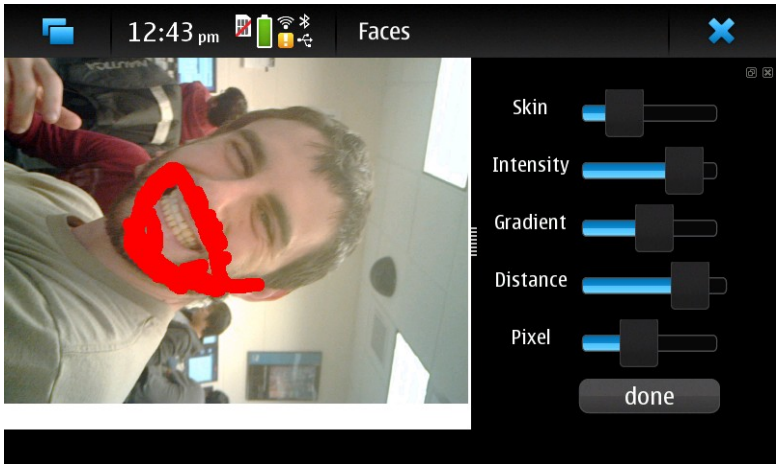
Desktop version

To run the desktop version, you need two images and two masks – all of exactly the same size. The masks should be colored white wherever you want to constrain/highlight the corresponding image, and black everywhere else. You run the program using the following command: ./Face <img1> <img2> <mask1> <mask2> [<skin_cost> <intensity_cost> <gradient_cost> <distance_diameter>]. The two images and masks are requires, but all the parameters are optional as there are some default parameters. Note that adjusting these parameters can significantly change the result, and some of the results shown on the next page did not use the default parameters (but usually something close to them).

Phone version

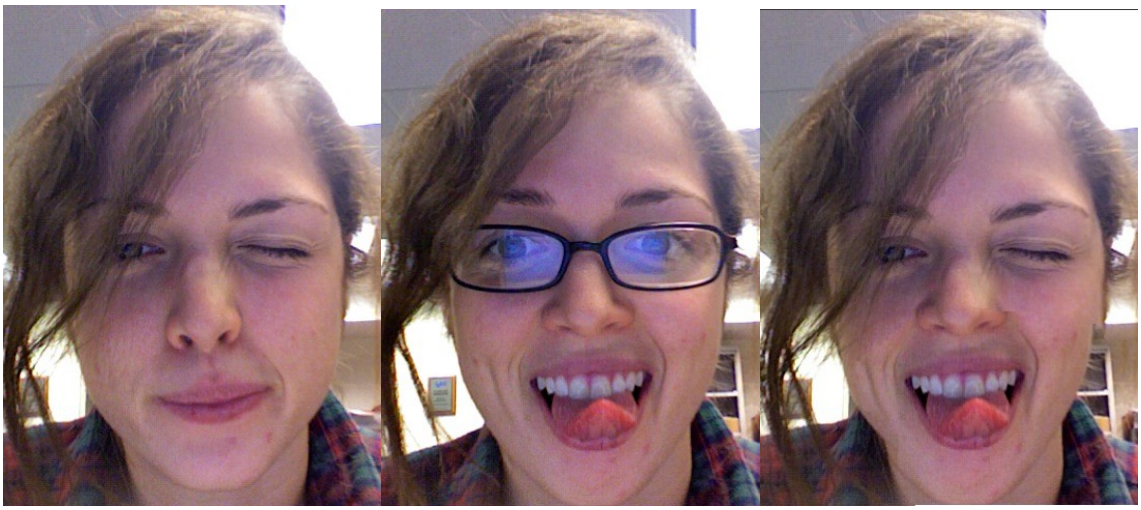
OpenCv was used to store the image and save them to the phone with the channels in the right order.

- Run ./FinalProject to start the user interface. Take two portraits with the same orientation. For best results, the faces should be on the same location on the screen and the same size. These images will be saved to the phone.
- Afterwards, the first image will be displayed with several sliders. The first four vary the cost weights for skin, intensity, the gradient, and distance. The last slider, 'pixels,' dictates how much space around the points you select will be included in the mask. The initial values are the default (suggested) values. Tap the screen with your finger or the stylus to highlight which parts of the image you want to keep. Repeat for the second photo. See image below.



- The result will then be computed (this may take several minutes on the phone). The result will be displayed and saved to the phone.

Experimental Results

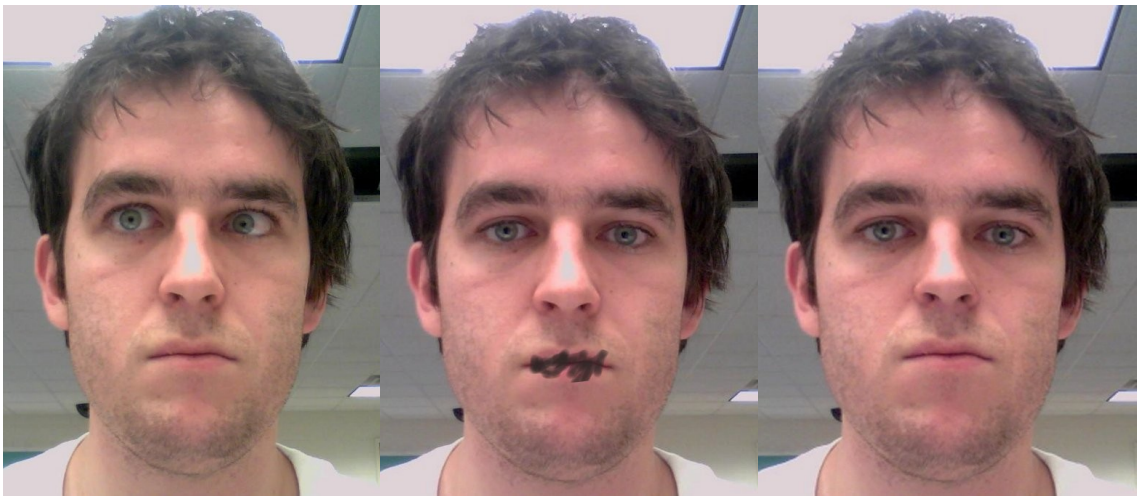


(1) (2) (3)

All three photos are of Najla. Taking the eyes from picture (1) and the mouth from picture (2) gives picture (3).

(1) (2) (3)

All three photos are of Diego. Taking the mouth from picture (1) and the eyes from picture (2) gives picture (3).



(1) (2) (3)

All three are photos of Boris. Taking the mouth from picture (1) and the eyes from picture (2) gives picture (3).

(1) (2) (3)

Photo (1) is of Najla and photo (2) is of Brian. Taking the top half of Najla's face and the bottom part of Brian's face yields picture (3). Notice that the skin tone in the final image is not the same as either Najla's or Brian's.

(1) (2) (3)

Photo (1) is of Fahad. We are using the previous photo of Najla again. Taking Najla's eyes and Fahad's face yields picture (3). Image (2) is the result achieved without gradient domain fusion. The seams are quite visiable – it looks like Najla's eyes were cut and pasted onto Fahad's face. In picture (3), however, you can notice that GDF has smoothed out the seams and made the skin tone look continuous. Producing picture (2) takes only seconds, whereas producing (3) can take several minutes – GDF is definitely the bottleneck of the program.

Discussion

We achieved our goal of seamlessly combining parts of faces. The results were often nice, but took too long. Sometimes results were skewed due to unaligned and/or differently scaled faces; for example, only part of an eye or both eyebrows would be in the final image. The phone version is not as accurate and takes a bit longer due to memory. A lot of improvements could be made to both the algorithms used and the user interface. These are discussed below.

Future Work

The phone interface can be improved. Its functionality is simple, and if we had more time we could have added zooming to make highlighting more precise, if the user wanted it that way. Users generally have slightly shaky hands and are not great judges of how lined up their images are. To assist the user with this, we could draw an outline for where the face should be when he or she is taking pictures. An even better way to handle this in the future would be to make it so the faces are automatically lined up by the program; pictures can be taken at different angles and distances that will skew results if they are not aligned.

For a more complete solution of the smoothing problem, it would be best to use a fast approximation algorithm, or using a different package from OpenCV for the solution. The second approach would be especially fruitful on the desktop, where libraries that offer sparse matrix (our equations generate matrices with no more than five non-zero entries per row, so they are quite sparse) computation exist. Even if using sparse matrices resolves the memory issue, to increase the speed an approximation is necessary. It appears as though a multigrid

approach could be made to work, but we were unable to resolve the difficulties in setting up the correct boundary conditions. This is definitely the best approach to improving the speed of the program.

Another area of future work is to allow the user to merge regions from more than just two images. The graph cuts algorithm described can only handle two images – and in fact the only way to scale this problem to multiple images is to introduce an approximate solution. There are two options for this, both of which are described in the Graph Cuts paper listed earlier: alpha-beta swapping and alpha-expansion. Both of these algorithms loop through the set of images and attempt to either swap the labeling of two pixels or expand a particular labeling by one pixel in each step. This is iterated until no change is observed.

Responsibilities

All of us participated in integration, testing, coding, and debugging. To split up the work, we had Boris focus on fully understanding and implementing Gradient Domain Fusion, while Fahad and Najla worked together to get the Graph Segmentation implemented and all the User Interface and Phone-integration sorted out. Fahad focused more on Graph Cuts while Najla focused more on the GUI. Najla did a lot of research on the phone limitations that helped determine the memory and image-format limitations we had for the phone version.