Tyler Ambroziak
Ryan Fox
CS 638-1 (Dyer)
Spring 2010

Virtual Barber

Abstract

What would you look like without a beard? Or how about with a different type of beard? Think of the beards as a layer on top of the face rather than part of the face itself. Using techniques developed by Nguyen, Lalonde, Efros, and De la Torre, the beard layer can be automatically removed by estimating a beardless face from a database of beardless faces. To refine the image synthesis, you can then using the differences between the original and synthesized images to define a beard layer mask. To examine less drastic measures, we extended the paper's techniques to include "style filters" which leave part of the beard intact by masking only parts of the beard layer. Some applications of this technique include helping match pictures of wanted persons who may have changed their appearance or helping men choose a facial hair style that fits their personality.

Introduction

For some men, their beard is a defining feature of who they are. Some guys go for years without shaving. Because it is such a big part of who they are, some men may have a difficult time “letting go”, even when they decide that it is time to shave. If there were some way to preview what you would look like without a beard or with a different style of facial hair before actually shaving, the decision to shave or not to shave could be made much easier for all those men who are on the fence about shaving. The problem for us to address, then, is how can you take a single image of a bearded face and realistically remove the beard while preserving the rest of the face?

Motivation

Our project is based off of the paper "Image-based Shaving" (Nguyen, Lalonde, Efros, and de la Torre, 2008). The authors used 3 different approaches in this paper. After constructing a non-beard subspace (NBS) using 738 beardless faces, they first made a rough estimate of the beardless face by solving a linear least squares problem. This estimate basically darkened light areas and lightened dark ones, but didn't effectively remove the beard. To improve on this, the authors then used an iteratively reweighted least squares technique, which iteratively decreases the weight of the beard pixels, thus decreasing their influence on the final product. This fairly simply approach did a good job at removing the beard, but it also removed other features, such as glasses, scars, and moles. While removing these features may be desirable in some cases, the purpose of the program was to remove the beard while preserving the rest of the face. To overcome this problem, the authors developed a third technique, which utilized PCA and graph cuts to define a beard layer mask and use that mask to target their synthesis to the beard region. This final approach resulted in some very good shaves.

One major shortcoming of this paper was that it only performed a clean shave on an image. A person had no option of keeping some of their facial hair (i.e. changing the style). The addition of this feature was the major extension of our implementation.

Method

Our main steps of our method are as follows:

a) Construct non-beard subspace (NBS)
b) Input bearded image
c) Estimate beardless version using robust statistics in conjunction with NBS
d) Apply beard filter to retain wanted facial hair

Constructing the non-beard subspace

The non-beard subspace (NBS) is the key to the whole image-based shaving process. To construct our NBS, we used images from two different image databases: the IMM Face Database (Stegmann, Ersbøll, and Larsen, 2003) and the CMU Multi-PIE Database (Sim, Baker, and Bsat, 2003). These datasets were very large, mostly due to the fact that they both contained multiple images of the same individual under different conditions (i.e. lighting, pose, expression, etc.). In total, there were about 100 unique individuals in the two databases. We chose to construct our NBS using only clean-shaven males with neutral expressions. Introducing female faces or faces with different expressions might introduce extra unwanted variability. In total, our NBS consisted of 60 unique individuals.

Once the images for the NBS were selected, we had to register the images by aligning and cropping them. To align the faces, we first had to define feature points on each face. Although the IMM database and other face databases contain pre-defined feature points, we thought it might be difficult to standardize (between databases) which points were defined. That is, we wanted to make sure that the same points were defined in our entire set of images. Thus, we manually defined our own 28 feature points on the jaw line, lips, nose, eyes, and eyebrows of every face in the NBS using Matlab's *cpselect*. Since this was very time consuming, this created a bottleneck for the size of our NBS. Our NBS was miniscule (60 images) compared to the NBS in the original paper (738 images). In retrospect, it probably would have been better to use the pre-defined feature points (or at least a subset of them), but hindsight is always 20/20. Once the feature points were defined, each face image was warped to a base set of feature points and cropped to 95x93 pixels. This completed the registration of the images.

After the faces were cropped and in register, each face image was vectorized using *reshape* in Matlab. Thus, each image was fully contained by this 26505 x 1 vector. After vectorizing each image in the NBS, we concatenated them together into a 26505 x 60 matrix to form the actual NBS matrix.

Input bearded image

Now, the interesting part begins. We start by inputting a bearded face. The process here is very similar to the preprocessing of the NBS. The user starts by specifying 28 control points using *cpselect*. To ensure that the points are correctly placed, we displayed an image from the NBS with control points overlaid. We also overlaid a set of control points on the input face, so the user could just move them into the correct position. After all of the control points were defined, the image was warped, cropped, and vectorized, just like we did with the NBS.

We realized that if a user wanted to use the same input image multiple times, defining the control points over and over again would be quite redundant. So we decided to associate the control points with the file name and store that information, so if the same input image is used more than once, the user only has to define the points the first time.

Estimate beardless face

We used two different techniques to generate the beardless faces. The first was simply a naïve projection of the bearded face into the NBS. We solved an overdetermined least squares problem using Matlab. For an input image x consisting of d pixels, let x* be the corresponding image with beard removed. We solve $x^* = V\hat{c}$ with

$$\hat{c} = \underset{c}{argmin} \|x - VC\|_2^2$$

where $\|u\|_2^2$ denotes the squared L_2 norm of u. This projects the image input, pixel by pixel, into the NBS, minimizing the error for each pixel. This method is fast and simple, but doesn't produce very good results, especially with a small NBS.

The results from the naïve (Figure 1) method show pixilation as well as a general darkening of the face. This is due to the fact that the beard is generally a significant part of the face, so it lightens the beard portions and darkens the rest of the face. In addition, some of the original facial features get modified. We believe this is due to the relatively small NBS employed.

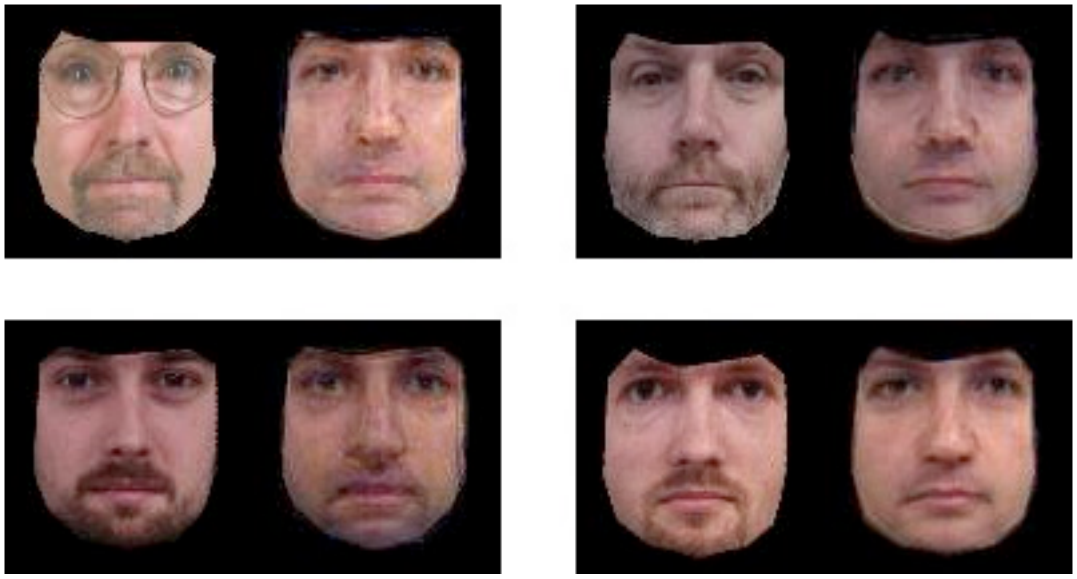
Figure 1: Original image and naïve beard removal for 4 faces

The second technique used for beard removal was an iterative process. We used a robust statistical method that weights the beard pixels less each time the process is iterated through. At each step we use the same method as before, but weight the pixels with a diagonal weight matrix **W**. So for the kth iteration, we solve the same equation as before but with $\hat{c}$ replaced by

$$c^{(k)} = \underset{c}{argmin} \| W(x - Vc) \|_2^2$$

The element w_{ii} in W is calculated by $w_{ii} = \frac{\sigma^2}{(e_i + \sigma^2)}$

where all other entries are 0. The deviation $\boldsymbol{\sigma}$ is updated each iteration as well and is defined by

$$\sigma = 1.4826 * \text{median}\left(\{|x_i - x_i^*| : i = [1, d]\}\right)$$

as well as the residual e obtained from the previous iteration:

$$e = x - Vc^{(k-1)}$$

The result x* is taken to be the limit of $c^{(k)}$ as k goes to infinity. The robust statistical method requires multiplying **W** X **W**' twice. Note that the size of x is 95 X 93 X 3, for d = 26505. W is a d X d matrix, for a total of 702 515 025 elements. Multiplying two matrices of this size would ordinarily require $(702\,515\,025)^2 \sim 10^{17}$ operations. However, we noted that the only nonzero entries in **W** are on the diagonal, so the result can be calculated by extracting that diagonal as a vector and squaring it. This reduced the calculation to $\sim 10^9$ operations. Before this optimization, one iteration took around 10 minutes. Afterward, one iteration took about 5 seconds. In practice, we found the first few iterations yielded significant improvements (fig. 2), with returns dropping off as the number of iterations was increased. The "sweet spot" balancing processing time and image quality seemed to be around 20 iterations.

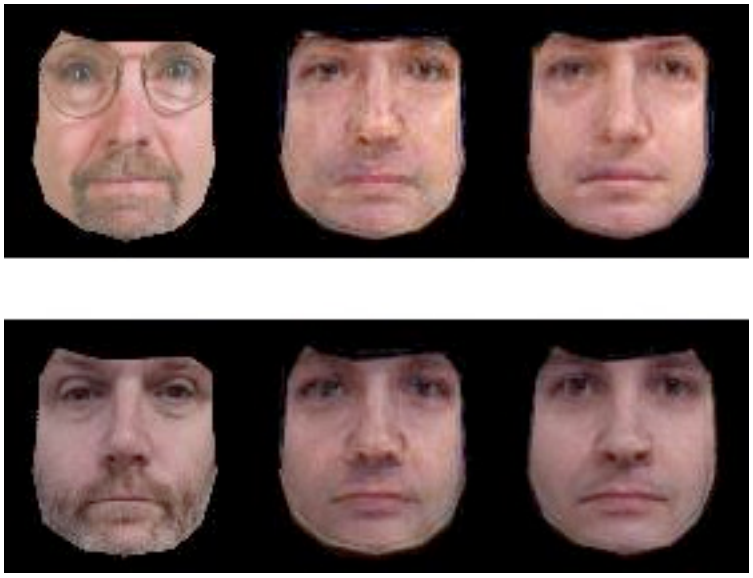
Figure 2: Original vs. Naïve method vs. Robust method

Apply beard filter

Up until this point, the goal of our paper has been essentially the same as the original paper by Nguyen et. al. Here, we add the option of leaving part of the facial hair intact, thus giving the source image a new “style”. This can be done simply by defining a mask for the regions that should be left intact, then combining the masked portion of the original image with the unmasked portion of the beardless image. We defined 6 different “style masks” for users to try on their input images, including 'clean', 'moustache', 'goatee', 'vandyck' (goatee + moustache), 'soulPatch', and 'fuManchu'. For as simple as this approach is, the results were surprisingly good:

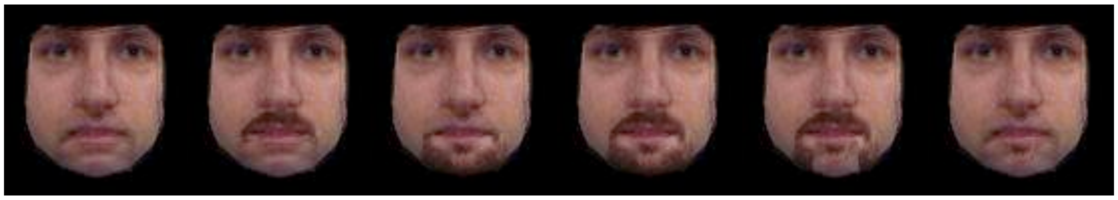
Figure 3: (L to R) 'clean', 'moustache', 'goatee', 'vandyck', 'fuManchu', and 'soulPatch'

As you can see, they are dependent on the facial hair coverage of the source. For example, the soul patch in Figure 3 is not as strong as the moustache, due to few beard pixels in the soul patch region. Overall, this technique works pretty well but could benefit from a few refinements such as blending, beard-pixel hallucination, etc.

Implementation Notes

To run our code, first download the zip file from our website. After you've unzipped the file and opened that directory in Matlab, you can run our code simply by calling

```
[before, after] = removeBeard(src_img, method, style);
```

where src_img is a string defining a relative path to the input image, method is either 'naïve' or 'iterative' depending on which method you'd like to use, and style is a string defining one of the styles, as defined in the previous section. The entire project was coded by Tyler and Ryan.

Results

If you would like to see more results than were included in our paper, please see our website (http://pages.cs.wisc.edu/~ambrozia/virtualBarber/). If this site cannot be found, please email Tyler Ambroziak at wrambro@gmail.com.

Improvements

Our implementation had several areas which we could have improved upon. The thing that would have helped our results out the most would be increasing the size of our non-beard subspace. This might have been easier to do if we had defined a way to utilize the pre-defined feature points in some of the image databases, but the authors of the original paper also had the advantage of more man power and more time to manually define the points. We also would have benefitted from implementing the third technique described in the original paper (PCA and graph cuts) to refine the region of synthesis. This would have preserved features that were lost due to the projection (i.e. glasses, scars, etc.) as well as prevented artifacts from the small size of our NBS (i.e. eyes/nose do not always seem to match source image).

Another simple addition would be include our function to warp the "de-bearded" face back onto the original image. Despite how trivial this was to implement, we chose not to include it for two reasons. First, since we have a high volume of results, reducing the output to the region of interest (i.e. the face) and eliminating extraneous and redundant information (i.e. the rest of the picture) is beneficial for comparing results side by side. Second, since the images were generally reduced in size when they were cropped to 95x93, warping the beardless face back to the original image would give a pixilated result. Alternatively, you could down-sample the rest of the image to match the size of the beardless face, but this did not appeal to us.

One final feature to implement would be beard transfer. That is, remove a beard from a bearded image or a composite of bearded images and superimpose it on a beardless face. The original paper touched on this a little bit, but did not explore it in great depth.

References

Minh Hoai Nguyen, Jean-François Lalonde, Alexei A. Efros and Fernando de la Torre. Image-based Shaving, *Computer Graphics Forum Journal (Eurographics 2008)*, 27(2), p. 627-635, 2008.

Terence Sim, Simon Baker, and Maan Bsat, "The CMU Pose, Illumination, and Expression Database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(12), p. 1615–1618, December 2003.

M. B. Stegmann, B. K. Ersbøll, and R. Larsen. FAME – a flexible appearance modelling environment. IEEE Trans. on Medical Imaging, 22(10):1319–1331, 2003.