Final Project
Bandpass Filtering & Image Blending

Jeff Blair

December 5, 2019

1 Introduction

1.1 The Problem

The aim of this project is to devise an algorithm that can seamlessly integrate an object from one photo into another. One such application of such an algorithm would be for fantasy movies, placing characters into fictional artist-drawn landscapes with a convincing degree of visual fidelity.

Inputs:

(a) Source

(b) Mask

(c) Target

1.2 A Naive Approach

The easiest approach to solving this problem is to simply use a binary mask to place pixels from the source into the target. As we see, this approach produces very sharp, obvious edges where the viewer can easily tell that the image has been modified.

Figure 2: No blending

1.3 A Better Idea

One idea to solve this issue is to create a border with some pixel width around the object, and linearly interpolate the pixel values in order to smooth the image. However for large distances in pixel values, this solution runs into the same problem, and the larger we make the border (better smoothing) the worse the level of detail in our photo becomes.

In this project, we will use a smarter way of smoothing this edge, by using band-pass filters to construct Gaussian and Laplacian pyramids for the image.

2 Constructing the pyramids:

2.1 Gaussian pyramid (Low-Pass):

The first part of this approach is to construct a Gaussian pyramid from a selected Region of our image, indicated by the mask. This is accomplished by using the original image as a base, and creating each new level of the pyramid by applying a 5x5 Gaussian filter to the level below and down-sampling by a factor of two. Each level above the base is more blurred than the last, and we can consider the pixel values still left over in the upper parts of the pyramid the most "important" pixel values of the source.

Code Snippet:

```
def buildGuassianPyramid(img, size):
    """
    Construct size # of blurred and downsampled images of img
    """
    G = img
    pyramid = [G]
    for i in range(size):
        G = convolveAndDownsample(G)
        pyramid.append(G)

    return pyramid
```

2.2 Laplacian Pyramid (High-Pass):

The second part of our approach is to build Laplacian pyramids for both the source and target images. The Laplacians are obtained as a difference of Gaussians for each level of the pyramid. Like the Gaussian pyramid, each level of the Laplacian pyramid can be thought of the most important edges of each image, as the filter will blur out any weak edges as the pyramid gets higher.

$$L_i = G_i - K * G_i \tag{1}$$

Code Snippet:

```
def buildLaplacianPyramid(guassianPyramid):
    """
    Construct edge detecting Laplacian pyramid using a guassian Pyramid
    L_i = G_i - (convolve(K, G_i))
    """
    pyramid = []
    for i in range(len(guassianPyramid)-1):
        Gi = upsample(guassianPyramid[i+1])
        G = guassianPyramid[i]
        r, c = G.shape[:2]
        L = G - Gi[:r, :c]
        pyramid.append(L)

    pyramid.append(guassianPyramid[-1])
    return pyramid
```

2.3 Combining and Collapsing:

Now that we have our Laplacian pyramids L_S, L_T from the source and target, and our Gaussian Pyramid G from the mask, we can form a combined pyramid B from L_S and L_T using levels of G as weights.

$$B_i = G_i L_{Si} + (1 - G_i) L_{Ti} \tag{2}$$

Now that we have a combined image, all that's left to do is collapse our pyramid to get back the blended image. This can be done by up-sampling the top level by 2, and adding it to the level below it. We repeat this process until we reach the bottom level of the combined pyramid, producing the blended image.

Code Snippet:

```
def reconstructImageFromPyramid(pyramid):
    """
    Sum over the images in the pyramid, upsampling at each level.
    Starts from the bottom. Upsamples and adds to the second last, and repeats
    """
    for i in range(len(pyramid)-1, 0, -1):
        r, c = pyramid[i - 1].shape[:2]
        pyramid[i - 1] += upsample(pyramid[i])[:r, :c]

    return pyramid[0]
```

3 Experimental Results:

3.1 Image Comparison

(a) No blending (b) Blending

As we can see, the blended image produces a far more convincing output. In addition, the amount of levels required to produce convincing results is quite small. As a result, outputs for color images can be produced within seconds, even for very high resolution images.

3.2 Visual Artifacts

Even though the algorithm does a considerably better job at producing a convincing blended image, there are several drawbacks to using this approach.

3.2.1 Transparency Artifacts

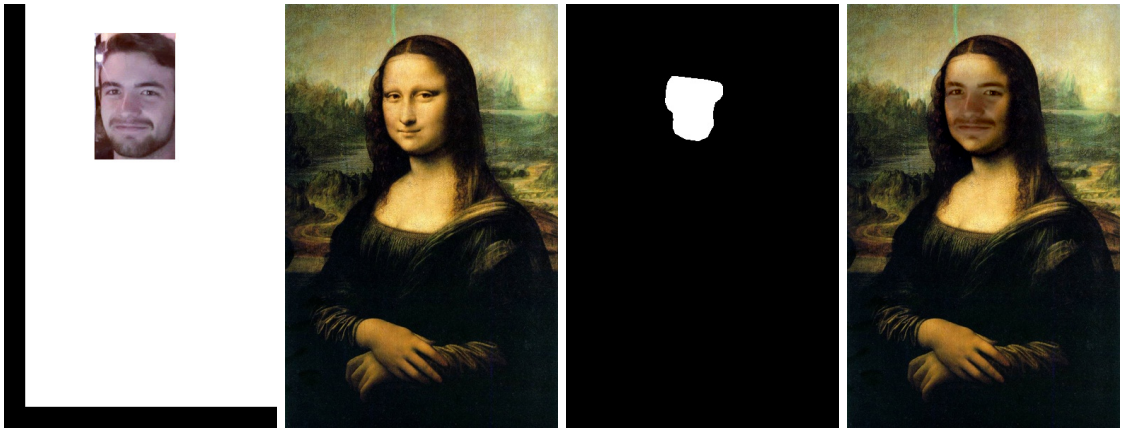
The following images suffer from one of the main reasons this algorithm can fail to produce a convincing output. In situations where lighting is significantly different between the source and target, the result can have unwanted transparency as a result of the blending process.

(a) Good result (b) Bad result

As we can see in the right photo, the blending process has made the plane slightly transparent in an attempt to resolve the lighting differences between the source and the target. Even in the good case on the left, this transparency issue is somewhat evident.

3.2.2 Edge artifacts

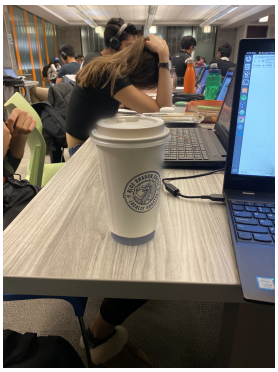
Edge artifacts are especially noticeable on faces, or any other situation where the color values of the images have stark differences that need to be reconciled as a whole, and not just on the border of the region.



(a) Source image (b) Target (c) Mask (d) Output

Figure 5: Trying to paint my face on the Mona Lisa

The algorithm can also fail to correctly adjust the differences between the source and the target photo, where edges are important to the successful production of a convincing output.



(a) Bad result

4 Improving the Algorithm

4.1 Poisson Image Editing

In the paper "P. Pérez, M. Gangnet, and A. Blake. Poisson image editing", an algorithm is given that produces an even better way to smooth differences around the border of the region. The paper shows a better approach to the problem is the solution to two constraints on our source and target image, and boundary Ω:

Constraint 1: Laplacians must be equal inside the region

$$\nabla^2 T(x,y) = \nabla^2 S(x,y), (x,y) \in \Omega \tag{3}$$

Constraint 2: Pixel values must be equal outside the region

$$T(x,y) = S(x,y), (x,y) \in \partial\Omega \tag{4}$$

The paper shows that satisfying these two constraints nicely equates to solving a system of linear equations. This largely solves the edge artifacts that are present as a result of the band-pass algorithm.

5 Works Cited

1) A lecture on both band pass filtering and poisson image editing by Rich Radke:
https://www.youtube.com/watch?v=UcTJDamstdk&t=2235s

2) Lecture Slides from CMU:
http://graphics.cs.cmu.edu/courses/15-463/2005_fall/www/Lectures/Pyramids.pdf

3) Perez et al: Poisson image editing
https://www.cs.virginia.edu/~connelly/class/2014/comp_photo/proj2/poisson.pdf