PROJECT 2
LAPLACIAN PYRAMIDS AND IMAGE BLENDING

OVERVIEW

In this project, you will investigate two applications: *Laplacian pyramid blending* and *hybrid images*. You can use the former to make smooth transitions between arbitrary images, such as the fish sandwich depicted in Figure 1. The latter can be used to generate interesting optical illusions as seen in Figure 4.

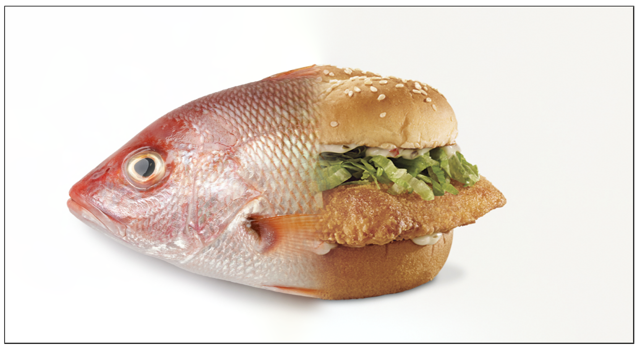

Figure 1: A fish sandwich

LAPLACIAN PYRAMID BLENDING

1. First, generate a Laplacian pyramid. A *Laplacian pyramid* (see section 3.5.3 of the Szeliski textbook) encodes an image as a succession of progressively smaller Laplacian images, built atop a base layer consisting of a blurred and reduced copy of the original image. Given an input image I, we can construct a Laplacian pyramid $P = (L_0, L_1, L_2, \ldots, L_N)$ of depth $N + 1$ according to the following rules:

- Define G_0 to be the input image itself, so $G_0 = I$.
- Given G_i, obtain G_{i+1} by convolving G_i with a small Gaussian, and then discarding the odd-numbered rows and columns of the image, to obtain a half-sized result.
- Given G_{i+1}, produce $G^{\uparrow}_{i+1}$ by enlarging G_{i+1} to be the same size as G_i while smoothing the result.
- For any $i < N$, define $L_i = G_i - G^{\uparrow}_{i+1}$.
- Finally, let $L_N = G_N$

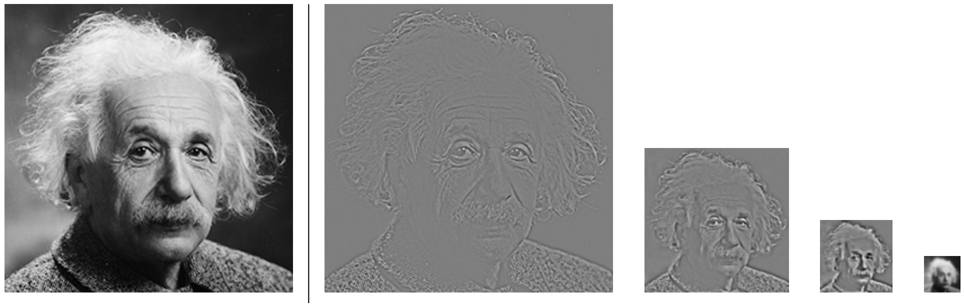

Figure 2: *Left of line:* input image. *Right of line:* Four-level Laplacian pyramid.

When constructing the Laplacian pyramid, please keep the following tips in mind:

- ▷ Creating G_{i+1} from G_i may be accomplished via the **`cv2.pyrDown`** function.
- ▷ Creating $G_{i+1}^{\uparrow}$ from G_{i+1} may be accomplished via the **`cv2.pyrUp`** function. You will almost certainly want to supply the **`dstsize`** parameter so that $G_{i+1}^{\uparrow}$ and G_i have the same shape.
- ▷ Be careful not to directly subtract two **`numpy.uint8`** images – the result will certainly overflow. Instead, convert images to **`numpy.float32`** datatype before subtracting.
- ▷ When displaying floating-point images via **`cv2.imshow`**, OpenCV expects the intensities to vary between 0.0 and 1.0. Therefore, a reasonable method to display a floating-point Laplacian image (e.g. for debugging purposes) might be something like:

```
cv2.imshow(window, 0.5 + 0.5*(L / numpy.abs(L).max()))
```

Write a function **`pyr_build(img)`** which takes an 8-bit per channel RGB or grayscale image as input, and which outputs a list **`lp`** of Laplacian images (stored in **`numpy.float32`** format) corresponding to the image's Laplacian pyramid. Your code should produce results similar to Figure 2, but with more levels. Aim to have the coarsest level be about 8-16 pixels in its minimum dimension. For instance, the coarsest level of a six-level pyramid for a 640×480 input image would have a size of 20×15 pixels.

2. Reconstruct an image from a Laplacian pyramid Surprisingly, no information is lost when converting from an original image to the corresponding Laplacian pyramid. You can reconstruct the original image from the Laplacian pyramid $(L_0, L_1, \ldots, L_N)$ by following these steps:

- Let $R_N = L_N$
- Given R_i, let $R_{i-1} = R_i^{\uparrow} + L_{i-1}$, where $R_i^{\uparrow}$ is an enlarged and smoothed version of R_i with the same dimensions as L_{i-1}.

- When you reach R_0, the result should be virtually indistinguishable from the original image.

Keep in mind these practical implementation matters:

▷ You can use the **cv2.pyrUp** function to create $R_i^{\uparrow}$ from R_i. Be careful to specify destination size as you did when building the pyramid.

▷ You will want to assemble R as a **numpy.float32** array, but convert to **numpy.uint8** immediately prior to displaying. If you have any problems with overflow, you can use the **numpy.clip** function to restrict R to the range $[0, 255]$ before type-converting.

Write a function **pyr_reconstruct(lp)** that reconstructs the original image from a Laplacian pyramid, and test it to verify that it works before proceeding to the next section.

3. Combine two Laplacian pyramids. Given two original images **imgA** and **imgB**, it is possible to combine their corresponding Laplacian pyramids **lpA** and **lpB** in such a way that low-frequency features (such as solid color areas) are blended over a large distance, while high-frequency features (such as fine lines, ripples or edges) are blended over much shorter distences.

One good way to combine images is with a continuously-varying mask, sometimes called an *alpha mask*. Let **alpha** be a 2D floating-point array with pixel intensities in the range $[0.0, 1.0]$, and assume that it has the same height and width as two images **A** and **B**. Then you can use this **alpha_blend** function to combine the images:

```
def alpha_blend(A, B, alpha):

    A = A.astype(alpha.dtype)
    B = B.astype(alpha.dtype)

    # if A and B are RGB images, we must pad
    # out alpha to be the right shape
    if len(A.shape) == 3:
        alpha = numpy.expand_dims(alpha, 2)

    return A + alpha*(B-A)
```

At each level of the Laplacian pyramid, resize the original alpha mask (which had the same dimensions as **imgA** and **imgB**) to the size of each of the Laplacian images using **cv2.resize** with **interpolation** mode **cv2.INTER_AREA**. Then, use the resized alpha mask to blend the pair of Laplacian images at that level.[1] The result is shown in Figure 3.

Your task is to photograph (or obtain) a pair of images to blend together, and write a program to generate an alpha mask and perform Laplacian pyramid blending between the two images. One reasonable way of generating an alpha mask with OpenCV is to draw a geometric shape (i.e. using

[1]Note we do *not* ever compute the Laplacian pyramid of an alpha mask!

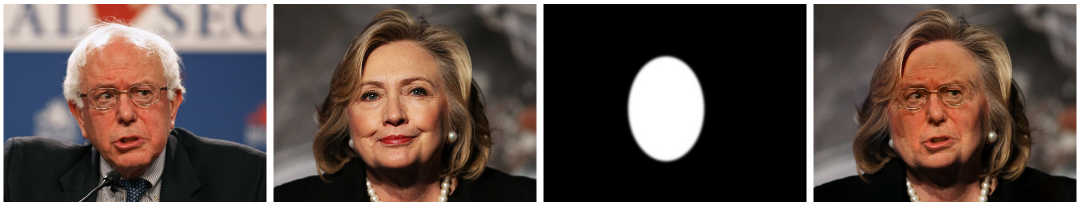

Figure 3: This disturbing composition uses a mask to combine Laplacian pyramid layers.

`cv2.ellipse`) and the smooth the resulting image using `cv2.GaussianBlur`. Your program should use the `pyr_build` and `pyr_reconstruct` functions you have already written.

In addition to submitting your program and the Laplacian-pyramid-blended output, you should also submit the "traditional" result of directly alpha-blending the two input images, without the pyramid. Hopefully, the pyramid result looks much more convincing than the traditional result.

HYBRID IMAGE

Moving on from Laplacian pyramids, a related concept is the *hybrid image* illusion, as shown in Figure 4. A hybrid image I can be created from two images A and B according to the formula

$$I = k_A \cdot \text{lopass}(A, \sigma_A) + k_B \cdot \text{hipass}(B, \sigma_B)$$

Here, the $\text{lopass}(X, \sigma)$ function implements a low-pass filter on the image X by blurring it with a Gaussian kernel with standard deviation σ, and hipass implements a high-pass filter according to

$$\text{hipass}(X, \sigma) = X - \text{lopass}(X, \sigma)$$

Typically $\sigma_A > \sigma_B$, and setting both k_A and k_B to 1 is a reasonable place to start.

Your task here is to obtain a pair of images A and B, and write a program to merge them into a hybrid image. As with the Laplacian pyramid code, you will probably want to work in `numpy.float32` as an intermediate format, and be careful about overflow when converting from floating-point back to `numpy.uint8` format. Please submit source code, source images, and output hybrid image.

WRITEUP

Please create a 2-4 page PDF writeup addressing the following questions:

- Who did what for this project?
- How did you obtain and align your images for each of the two tasks? Did you use any third-party software (e.g. Paintbrush, Photoshop), or write a program to help prepare the images or mask?

- What depth did you choose to build your Laplacian pyramid to, and why?
- Why does Laplacian pyramid blending blend low-frequency content over a larger distance than high-frequency content? See if you can illustrate this with some carefully chosen input image examples.
- How did you arrive at good values for the constants σ_A, σ_B, k_A, and k_B for the hybrid image generation? Describe the process.
- If you display your hybrid image at full size on your computer screen, how close do you need to be in order to primarily see image B? How far away do you need to get before you only see features from image A? Are these distances fairly consistent between you, your lab partner, and any unsuspecting friends you show your image to?
- What does the Laplacian pyramid of your hybrid image look like?

Please include your source images/masks, and program outputs as figures, along with whatever other images you think will be useful/informative.

EVALUATION CRITERIA
Your project will be evaluated by the following criteria (percentages are approximate):

a. source code, raw data, and output (20%) - Turn in the full source code for your project, including any additional files needed to run it. You should also submit the program outputs as image files. I expect your code to be neatly indented and reasonably commented. Please expect to spend plenty of time photographing/obtaining and aligning suitable input images for both main tasks above – it is critical to have reasonably well aligned inputs, especially for the second task!

b. Laplacian pyramid results (30%) - The results of your Laplacian pyramid blending should be free of sharp seams, and be clearly superior to traditional alpha blending.

c. hybrid image results (20%) - Your image should clearly transition from image A to image B as the viewer approaches from far away.

d. going further (10%) - Go above and beyond the tasks outlined above. Examples might include writing a helper program to align input images and/or generate masks; adapting either one of the above techniques to work on video (live or pre-recorded); visualizing intermediate computations/results of the above tasks; researching and discussing other uses of pyramid representations in computer vision, etc. Feel free to ask if you want to run any ideas by me!

e. written report (20%) - Your PDF report should address all of the questions mentioned above, and contain the relevant images as figures.

Submit your code, inputs, outputs, and PDF to Moodle by 11:55PM on Sunday, February 26.

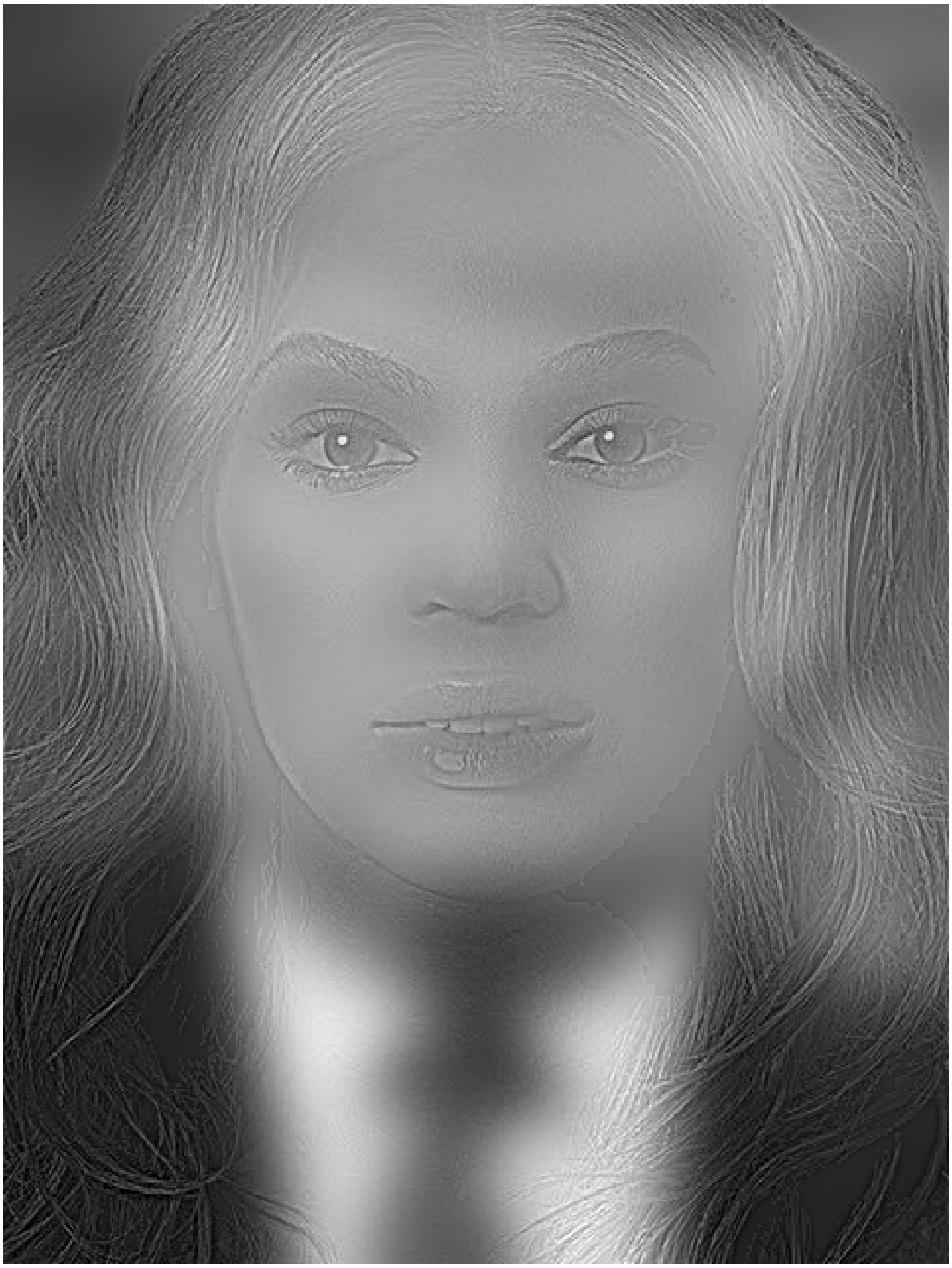

Figure 4: Appears to be Beyoncé from up close, but Trump from far away