Programming Assignment #2

Announcement:	24 January 2017
Submission Deadline:	07 February 2017

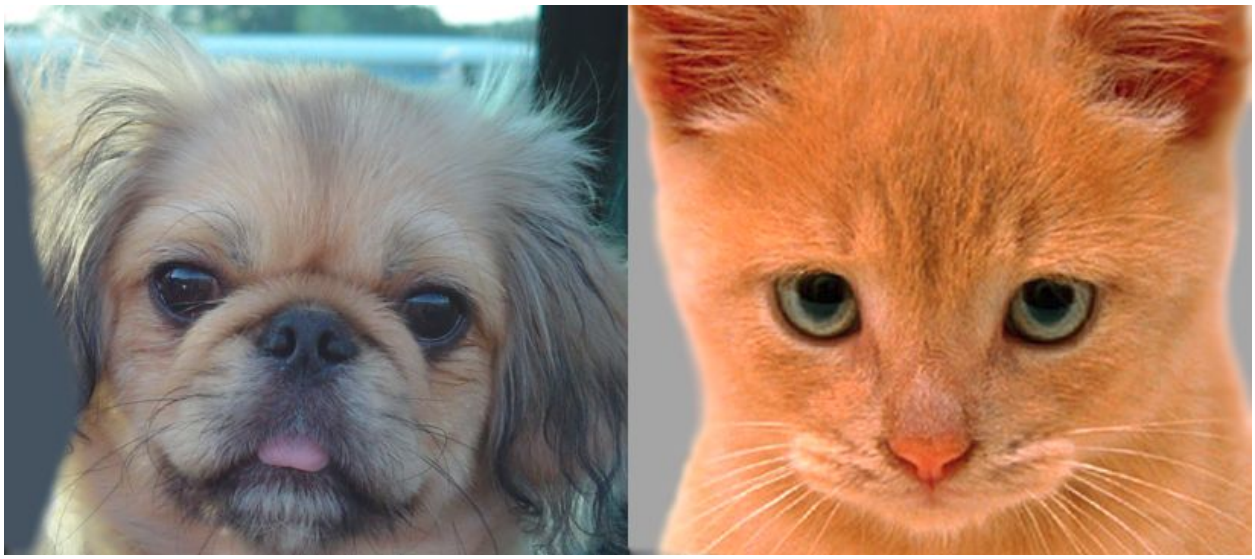
Description

The goal of this assignment is to implement a simplified version of “hybrid images” as introduced in their SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns. A *hybrid image* is a static image that changes in interpretation as a function of the viewing distance. This is based on the observation that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances.

Part 1: Using OpenCV’s cv::GaussianBlur(...) for image filtering

Implement an application in OpenCV which takes two input images and constructs a hybrid image. The hybrid image should be constructed by combining the low frequency of the first image with the high frequency of the second image. For a low-pass filter, Oliva et al. suggest using a standard 2D Gaussian filter. For a high-pass filter, they suggest using the impulse filter minus the Gaussian filter (which can be computed by subtracting the Gaussian-filtered image from the original). The kernel sizes and variances of each filter should be chosen with some experimentation. An example is shown below.

Input images



Low and high frequency images (high frequency image offset by +128)

Hybrid image

Part 2: Implement your own image filtering function

Image filtering (or convolution) is a fundamental image processing tool. Implement your own filtering function and use it to construct hybrid images. Compare the results with OpenCV's filtering.

Details:

- The function should be called filtering(...) and have the following signature:
  void filtering(cv::Mat const &input_image, cv::Mat const &kernel, cv::Mat &output_image);
  The filtering function should support the convolution with arbitrary shaped filters, as long as both dimensions are odd (e.g. 7x9 filters but not 4x5 filters). You should pad the input image with zeros prior to the application of the kernel and return a filtered image *output_image* which has the same resolution as the *input_image*.
- Implement a helper function called GaussianKernel(...) with the following signature:
  void GaussianKernel(int sizex, int sizey, double sigma, cv::Mat &kernel);
  This function takes as input the size of the kernel and the variance sigma. It returns *kernel* which contains values computed based on the Gaussian function.

Part 3: Alternative high-pass filter - [Graduate program requirement only]

The paper suggests that for a high pass filter you should use the impulse filter minus the Gaussian filter (which can be computed by subtracting the Gaussian-filtered image from the original). Implement the following alternatives for the high pass filter:

- Sobel operator
- Difference of Gaussians
- Laplacian of Gaussians

Using both words and images compare the results of each of the filters with the suggested. The report should be included in the zip file.

Submission (electronic submission through EAS only)

Please create a zip file containing your C/C++ code and a readme text file (.txt).

In the readme file document the features and functionality you have implemented, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc.

Additional Information

- Five pairs of aligned images are provided for testing and experimentation here

Credits

Assignment developed by James Hays based on a similar project by Derek Hoiem.