TEMPLATE CONVOLUTION & HYBRID IMAGES IN OPENIMAJ

Introduction

Hybrid images, as presented by Oliva, Torralba & Schyns[2006][1], display an image that appears to change depending on viewing distance. This is achieved by "by superimposing two images at two different spatial scales"[1], and the resulting image makes use of the "multi-scale perceptual mechanisms of human vision"[1] in order to alter the perceived content at various viewing distances. The low and high spatial scales are obtained by applying a low-pass or a high-pass filter to the input images.

Template Convolution

One way of producing a low-pass filtered version of an image is to convolve the image with a template consisting of Gaussian values. The convolution algorithm implemented in this project was adapted from *Feature Extraction & Image Processing*[2, p83], and works by traversing both axes of the image and at each pixel, "generating a running summation of the pixel values within the template's window multiplied by the respective template weighting coefficient."[2]. The value of this summation is then assigned to the pixel at the centre point of the template in a new output image. The algorithm is shown in full below.

```
//kernel can be arbitrary size but both dimensions must be odd
private float[][] kernel;

public MyConvolution(float cutoff, float[][] kernel) {
    this.kernel = kernel;
}

@Override
/* convolve image with kernel and store result back in image
 *
 * Adapted from Feature Extraction & Image Processing - Nixon, Aguado (Code Snippet 3.5)
 */
public void processImage(FImage image)
{
    int imgRows = image.height;
    int imgCols = image.width;

    int tempRows = kernel.length;
    int tempCols = kernel[0].length;

    int tempROWS_half = (int) Math.floor(tempRows/2);
    int tempCOLS_half = (int) Math.floor(tempCols/2);

    //set a temporary image to black
    FImage temporary  = new FImage(imgCols, imgRows);
    temporary.fill(0f);


    for(int x=tempROWS_half+1; x<imgCols-tempROWS_half; x++){ //address all columns except border
        for(int y=tempCOLS_half+1; y<imgRows-tempCOLS_half; y++){ //address all rows except border
            float sum = 0;
            for(int iWin=1; iWin<tempRows; iWin++){ //address template rows
                for(int jWin=1; jWin<tempCols; jWin++){ //address template columns
                    sum = sum + image.getPixel(x+iWin-tempROWS_half-2, y+jWin-tempCOLS_half-2) * kernel[jWin][iWin];
                }
            }
            temporary.setPixel(x, y, sum);
        }
    }

    //Put all pixels within range 0.0-1.0
    temporary = temporary.normalise();

    //modify passed image
    image.internalAssign(temporary);
}
```

Hybrid Creation

The convolution algorithm detailed above is applied to each of the two selected images with a Gaussian kernel to obtain their low-pass versions. The sigma value of the Gaussian, or its variance, is by default 4 for image 1, and 8 for image 2, but is user adjustable. This value is defined "as the frequency for which the amplitude gain of the filter is 1/2"[1], and each pair of cut-off values represents a set of parameters by which the appearance of the hybrid image can be altered.

After obtaining a low-pass version of image, the *createHybrid* function produces corresponding high-pass images by subtracting the low-pass images from the originals, using the OpenIMAJ API method, *MBFImage.subtract()*. The 2 potential hybrid images are created by adding the low-pass version of one image with the high-pass of the other, and vice versa. The core code for *createHybrid* is shown below.

```
//set size to function of sigma:
size = (int) (8.0f * cutoff1 + 1.0f); // (this implies the window is +/- 4 sigmas from the centre of the Gaussian)
if(size % 2 == 0) size++; //ensure size is odd

FImage gauss2 = Gaussian2D.createKernelImage(size, cutoff2);
MyConvolution conv2 = new MyConvolution(cutoff2, gauss2.pixels);

//LP - convolve image with guassian filter
LPimage1 = image1.process(conv1);
LPimage2 = image2.process(conv2);

//HP - subtract a low pass version of an image from itself
HPimage1 = image1.subtract(LPimage1);
HPimage2 = image2.subtract(LPimage2);

//create both versions of hybrid
hybrid1 = HPimage1.add(LPimage2);
hybrid2 = HPimage2.add(LPimage1);
```

Results

In order to test the algorithm at its various stages, and with varying parameters, a simple interface was designed using swing and awt components. After selecting 2 matching images and checking their previews, a user can choose to display either hybrid image combination(fig1), the high and low-pass versions of each image (fig2), or a scaled representation of the default hybrid (fig3), the last of which is intended to simulate the effect of increased viewing distance.

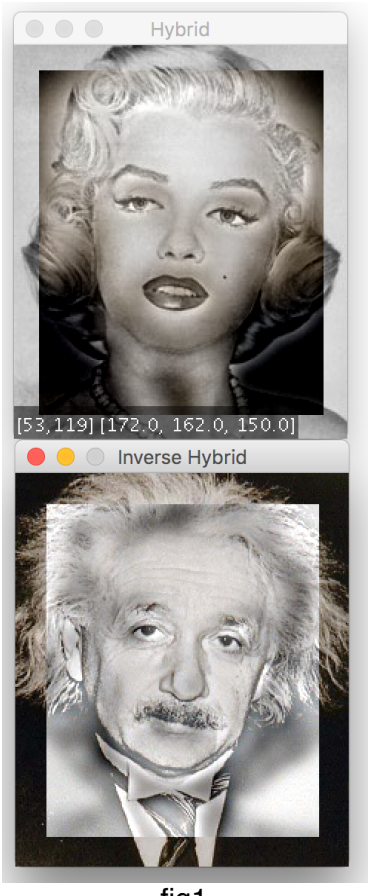

fig1

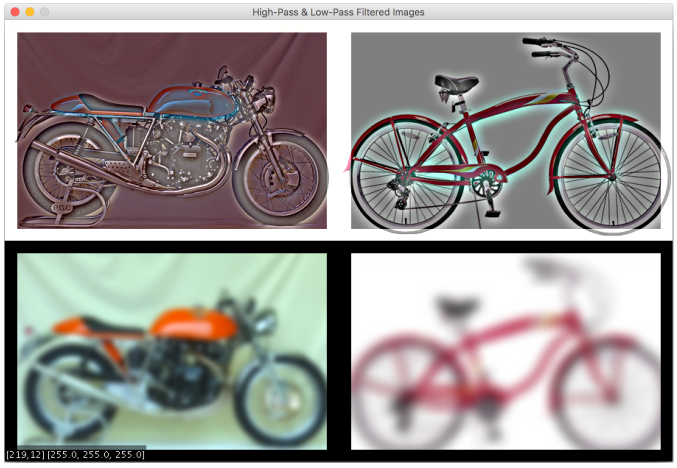

fig2

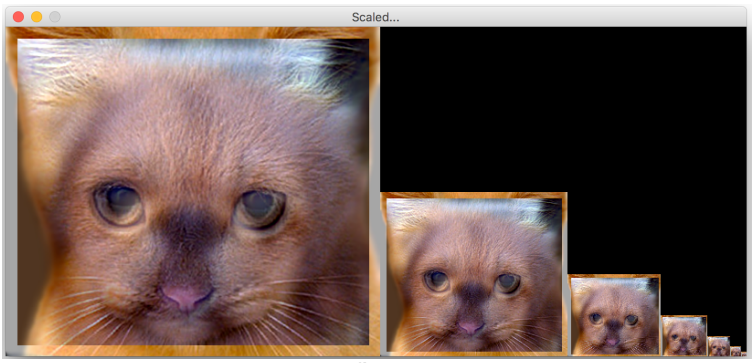

fig3

Discussion

The algorithm was able to produce these results with good speed, taking less than 10 seconds to process images with higher cutoff values, and around 1-2 seconds for the default parameters, which were judged to be optimal by human perception for most pairs of images. However the effectiveness of any hybrid image, and thus the success of this application, is largely dependent on the relationship of the two input images based on the rules of perceptual grouping[1]. In this sense, the application fulfils its requirements as it is able to combine any two images of the same size in a customisable manner. Further developments on this project would include altering the convolution algorithm to set border pixels to a constant value, perhaps, and implementing timer functionality, to facilitate rigorous and definitive testing of the algorithm's performance.

Bibliography

[1] - OLIVA, A., TORRALBA, A. and SCHYNS, P. (2006) - *Hybrid Images* - p527 - p532 SIGGRAPH ACM 2006, Available from: http://cvcl.mit.edu/publications/OlivaTorralb_Hybrid_Siggraph06.pdf

[2] - NIXON, M. and AGUADO, A. (2008) - *Feature Extraction & Image Processing* - 2nd Ed. Oxford:Elsevier