-SAI KRISHNA W.D.S
UNITY ID: swupadr

DIGITAL IMAGING SYSTEMS- PROJECT 3 REPORT

Part a) :

To generate Gaussian and Laplacian pyramid functions for a given image, and calibrating the number of levels for the construction of the pyramids:

So, first when an image is taken as the input we need to calculate the number of levels for the Gaussian Pyramid. This is done by measuring the number of rows and columns in the image, using the *image.shape* function.

Then, we find the area of the image, by multiplying the rows and columns of the image; and then take the log base 2 of it; to obtain the number of levels required for the gaussian pyramid.

Now, the gaussian pyramid is basically constructed using repetitive down-sampling and smoothing using a gaussian filter.

I have used a 3x3 gaussian kernel to implement the filtering process.

The following is the algorithm I used for implementing the Gaussian Pyramid:

1. Smooth the image with Gaussian filter
2. Sub sample the image by half
3. If reached desired size stop, else send the result to step 1

After the Gaussian pyramid for each layer has been constructed, I have each layer stored in a list called '*output*'.

Once the gaussian pyramid levels have been constructed, I computed the the Laplacian pyramid levels using the following algorithm:

```
1. Build a Gaussian pyramid
2. Until last element of Gaussian pyramid not reach do:
        ith Laplacian pyramid = ith Gaussian pyramid – expanded of (i+1)th Gaussian
        pyramid
```

So, for part a), I have designed 2 functions named gauss_pyramid and lapl_pyramid – which both take the parameters-(*input image, no. of layers*); and compute the image pyramids respectively.

The part a) code can be seen in the .py file attached named 'fullcode.py' in the zipped folder.

-SAI KRISHNA W.D.S
UNITY ID: swupadr

Part b) :

Generation of black and white mask with GUI using Rectangular ROI:

For generation of masks, I have used the rectangular ROI. The ROI is captured is captured with the help of mouse track events- which detect the place the mouse is clicked first and the place where the mouse is released after tracing the desired rectangle- from top left to it's bottom right on the foreground image.

The mask generation design and output images and .py file have been attached in the zipped folder.

Part c) :

Laplacian Blending of the images:

Now to implement the blending of images, we need three parameters – ie., the foreground image, the background image and the mask. I have taken all of the foreground , background and the mask sizes to be same for more computational efficiency.

I have chosen the background photo as Monalisa for all the cases; and I try to blend it with 3 different foreground images, those of Jackie Chan, Troll face and Donald Trump.

The masks are created for each foreground image in the code; and we first use the gauss_pyramid and lapl_pyramid functions to create the the image pyramids for the mask and the foreground and the background images.

After this, we blend the Laplacian pyramids of the 2 images by weighing them in accordance with the mask.

I have designed a function named '*blend*' that serves this purpose.

Once the different levels of the Laplacian pyramids of the foreground and the background images have been blended, we need to collapse the layers together to obtain the final blended output image.

I have designed a function named '*collapse*' that serves this purpose.

The output blended images and the entire code module has been put in the file named '*full code.py*' attached in the zipped folder.