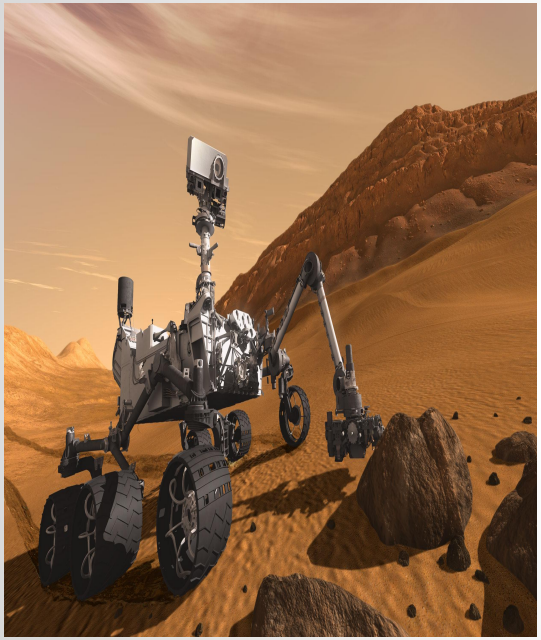
Comp Photography Assignment #6

Ngoc (Amy) Tran
Fall 2015

Input Images + Mask



“Black” image

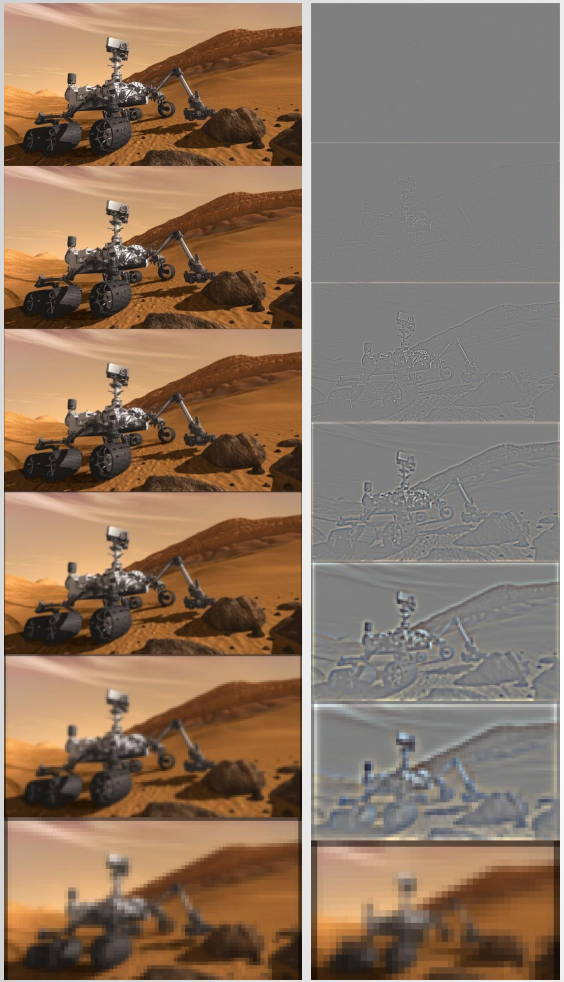
“Mask” Image

“White” image

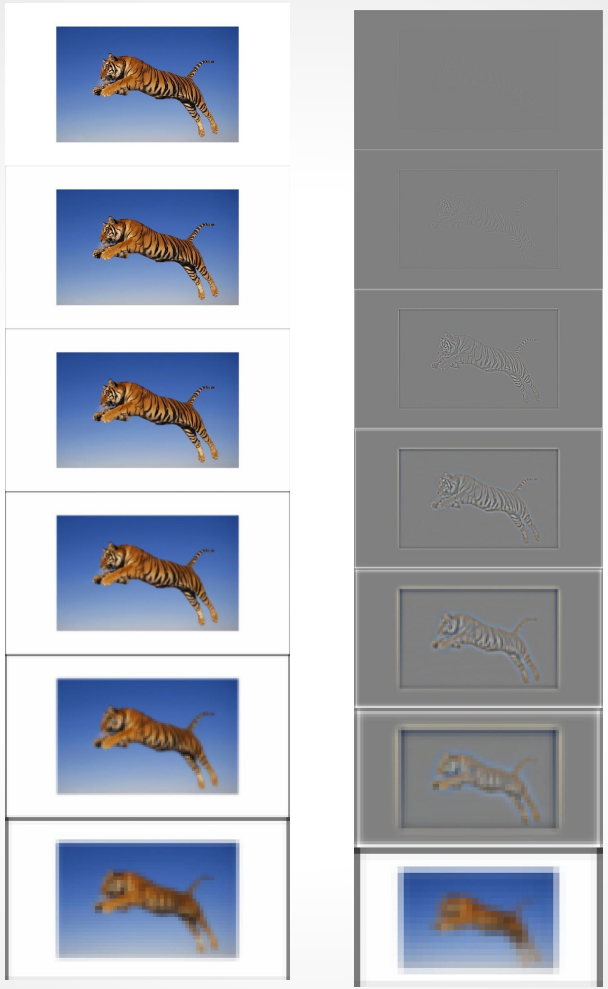
Created Mask

- I create mask using GIMP 2.8 and input “white” image
- First, I resize the image “white” image to be the same size as “black” image
- Then create duplicate layer of the white image
- And create mask layer by “Add Layer Mask”, use “White (full opacity)” option
- Then I use Paintbrush tool to paint over the tiger and use tiger as my mask
- Note: I use GIMP tutorial to create Layer Mask

Outputs



Black- Gaussian & Laplacian Pyramids



White- Gaussian & Laplacian Pyramids

Results

Part 0: Reduce & Expand function

1) **Reduce:** The reduce function takes an image, convolves it and then subsamples it down to a quarter of the size.
- I first create a symmetric kernel using provided function generatingKernel(a) with a=0.4
- Use 2-dimensional arrays “scipy.signal.convolve2d(image, kernel, 'same')” to convolve the kernel image
- Then take result output image sliced in steps of two for both rows and columns of the input image which reduced the image size in half (That is 25% of the original size since the image is 2D)

2) **Expand:** The expand function does opposite of the reduce function. It create an image that is twice the size of input image.
- Create a symmetric kernel using provided function generatingKernel(a) with a=0.4
- Create new array by using numpy.zeros() function and multiply both rows and columns by 2 to double image size
- Then slicing operation, every 2 pixels in the new array gets set with a pixel from the input image, resulting in a black image with evenly dispersed pixel dots of the original image
- Use 2-dimensional arrays “scipy.signal.convolve2d(image, kernel, 'same')” to convolve the kernel image

Part 0: continue

- Finally, multiply output image by a factor of 4 in order to scale it back up.

➢The reason multiply the result by 4, because input image spread out over 4 positions in the output image. So, the output image multiply by 4 to restore the original image's pixels information. When reduced an image, it's quarter (¼) of the original image's pixel. So, multiply by 4 just go back the original image (¼ * 4 = 1, 1 exemplifying as the whole, expanded, non-blurry, original image)

Part 1: Gaussian and Laplacian Pyramids

3) **gaussPyramid**: This function take an image and builds a pyramid out of it

- Take the levels input passed in, i create a for loop through range 0 -> levels
- During the loop, i made repeated calls reduce function, appending the output to a list
- Increasing positions in the list contained smaller images.

4) **lapIPyramid:** This function take a GaussPyramid and turns it into a Laplacian pyramid

- This function is similar to gaussPyramid(), Every elements of the list now corresponds to a layer of the Laplacian pyramid, containing the different two layers of the Gaussian pyramid.
- For every layer in the Gaussian pyramid, I expanded the next layer using the expand function
- Using 2 if conditionals, i remove subarray if the expanded image shape is larger than expected dimension to ensure size is appropriate
- After check, I use the algorithm that provided in this assignment “output[k] = gauss_pyr[k] - expand(gauss_pyr[k + 1])”
- Then, I append the result output of the different to the pyramid to the list

Part 2: Blend functions

5) **blend:** This function perform an alpha-blend of the two Laplacian pyramids according to the mask pyramid

- Using the Laplacian pyramid of the black and white input images
- Create another image pyramid list that blends two pyramids with gradient transparency level, where the value 0 takes 100% of the black image and 0% is the white image
- Use for loop that iterates through each entry of the input pyramid, and apply the algorithm “output[i, j] = current_mask[i, j] * white_image[i, j] + (1 - current_mask[i, j]) * black_image[i, j]” that blends black, white Laplacian images base on the gradient mask values.
- The result is appended to the returning list and continues to iterate until all the entries have been blended.

```
def blend(laplPyrWhite, laplPyrBlack, gaussPyrMask):
blended_pyr = []
for laplWhite, laplBlack, gaussMask in zip(laplPyrWhite, laplPyrBlack, gaussPyrMask):
   blended_pyr.append(gaussMask * laplWhite + (1 - gaussMask) * laplBlack)
return blended_pyr
```

Part 2: Collapse functions

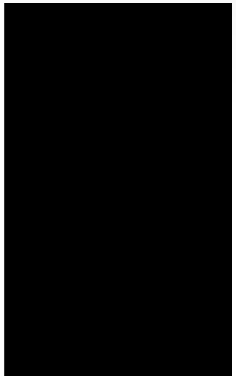
6) **collapse:** This function is given a Laplacian pyramid and is expected to ‘flatten’ it to an image.

- Creating an array output as the same size of the last position in pyramid to ensure it has good size of the output.
- Approach this function as follows, start at the smallest layers of the pyramid. So, the start loop by reserved pyramid.
- And add it to the second to smallest layer, and continue the process loop until it at the largest image.
- Sum it and returned

```
def collapse(pyramid):
output = pyramid[-1]
for image in reversed(pyramid[:-1]):
   #Expand current sum, making sure to have the layer sizes agree
   output = image + expand(output)[:image.shape[0], :image.shape[1]]
 return output
```

Addition image Blending using Pyramids

“White” Image



“Black” Image

The final output