Homework 6

Tasks:

1. Modify the code in Section 6.1 to perform more pyramid transformation iterations. For example, you can use a smaller template or increase the size of the image. Replace the **impyramid** function with down-sample function with a smaller factor (for example sub-sample to 0.8). Show the result and discuss.

Answer →

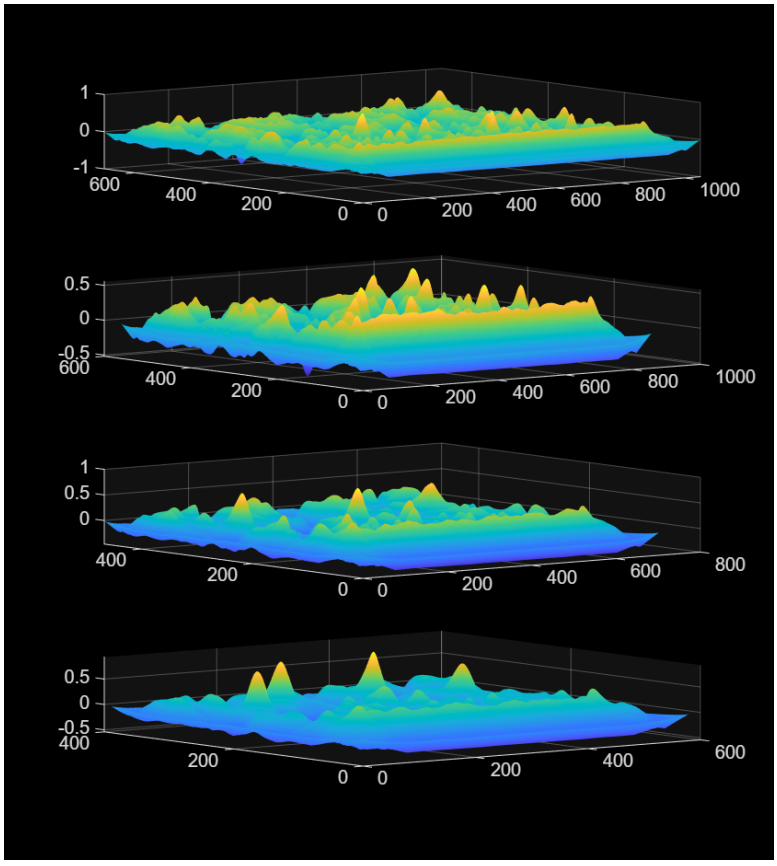
Here are the different combinations tested below:

Template Image	Main Image	Observations
Reduced by 0.5	Down sampled by 0.8 iteratively till it becomes 0.2 of original	More number of False Positives (3) observed than the impyramid one. Additionally, False Negatives (2) were observed as well.
Reduced by 0.25	Down sampled by 0.8 iteratively till it becomes 0.2 of original	Number of false positives increases as the template size is too small for the system to find a match with anything.
No Reduction	Down sampled by 0.8 iteratively till it becomes 0.2 of original	As the template size is large, the number of matches it finds on the original image reduces drastically and it only finds 4 image matches and 1 false positive at the edge.
Reduced by 0.5	Up scaled by 2 and then down sampled by 0.8 iteratively till it becomes 0.2 of original	More matches observed but the image rectangle sizes are not close to the original image.

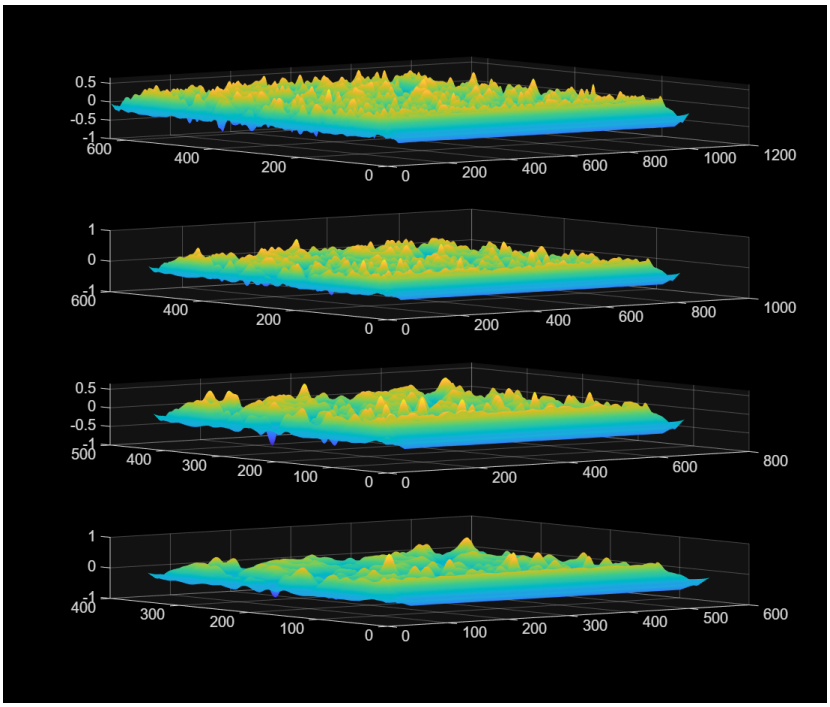
Overall Observations:

- As the template becomes smaller, the number of positive detections increases significantly.
- As the original image size is increased, the number of positive detections increases as well for the original template size.

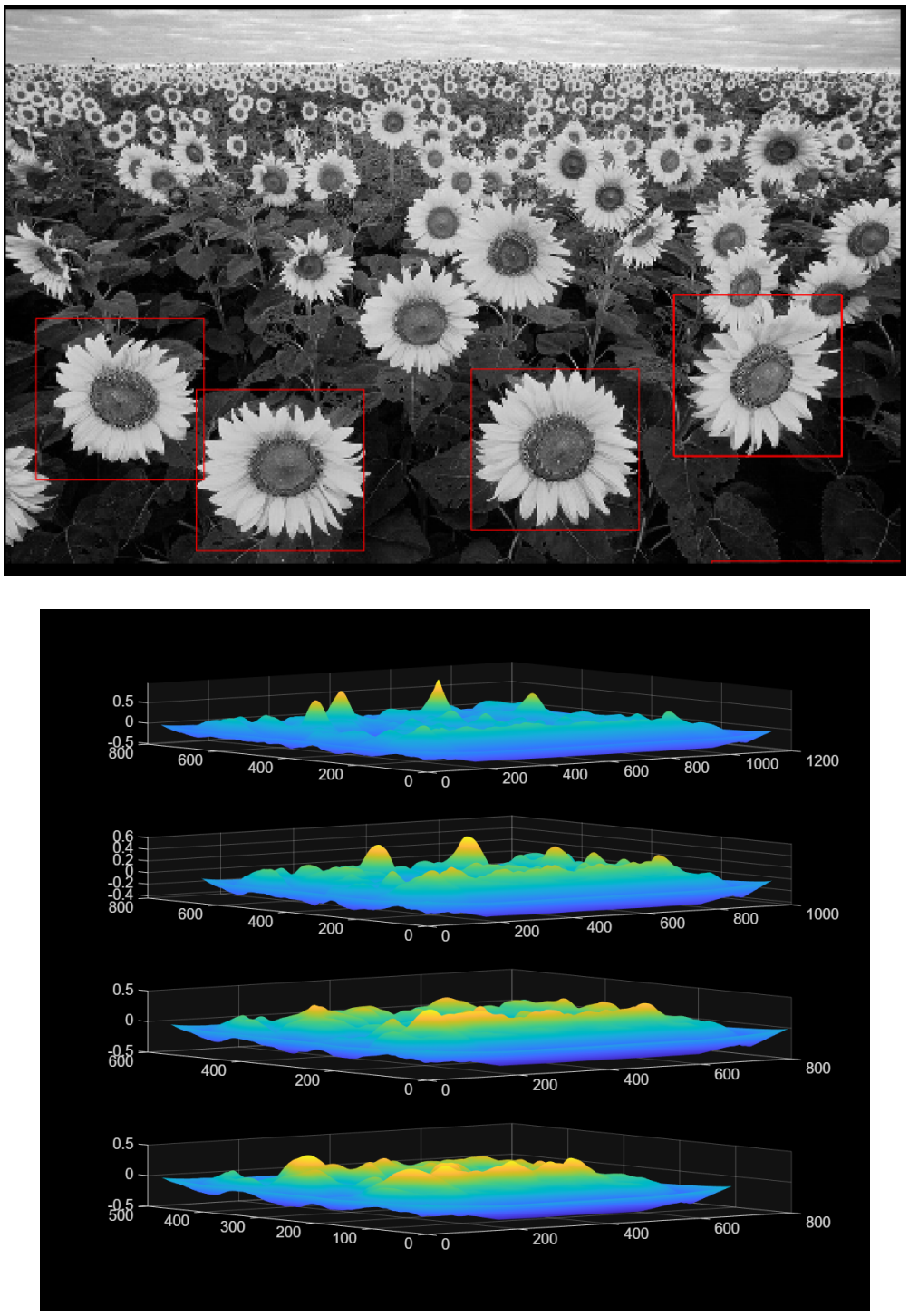
Case 1: Template reduced by 0.5 with bicubic interpolation. Main image down sampled to 0.8 of original size iteratively until the main image size is reduced to 0.2 of original image size.



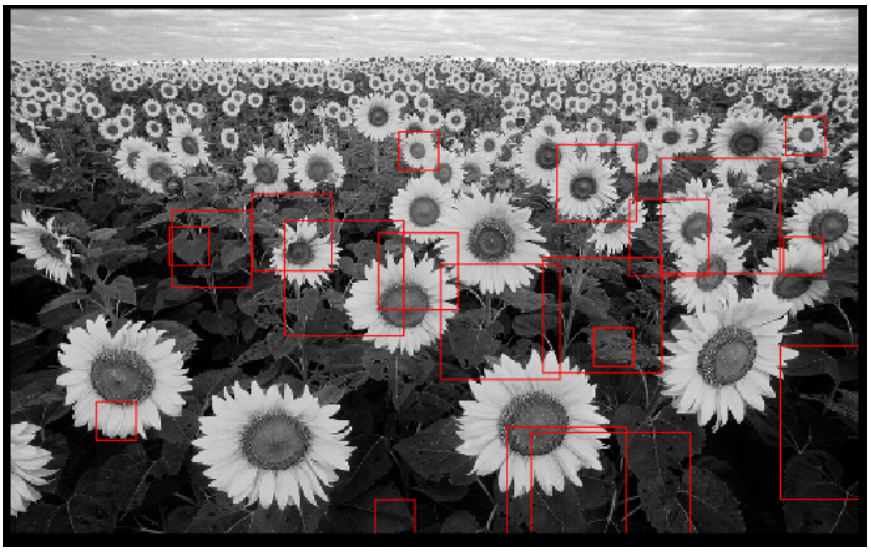
Case 2: Template reduced to 0.25 of original template size with bicubic interpolation and Main image down sampled to 0.8 of original size iteratively until the main image size is reduced to 0.2 of original image size.

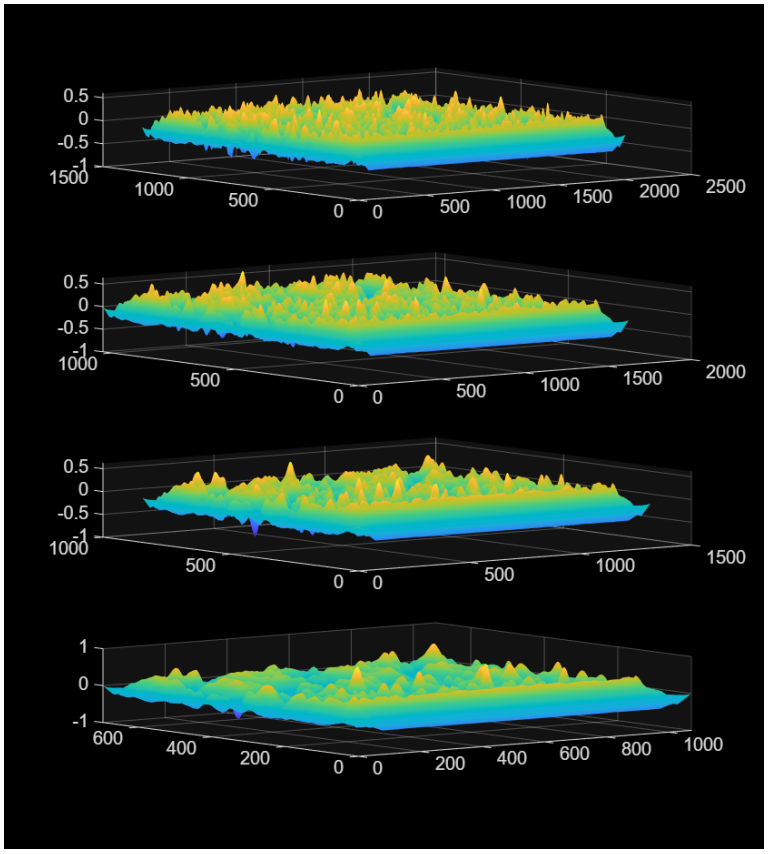


Case 3: Template image size unchanged (100%) and Main image down sampled to 0.8 of original size iteratively until the main image size is reduced to 0.2 of original image size.



Case 4: Template image resized to 0.5 (as originally present in 6.1), Main image scaled up to 2 times original size and then the resized image is iteratively scaled down by 0.8 till resized image is 0.2 times of scaled up image.





Code (modified for Case 4):

```
%%
min_corel = 0.5;
% load image
I = im2gray(imread('L06 sunflower.png'));
I = imresize(I, 2, "bicubic");
```

```
% load template
T = im2gray(imread('L06 sunflower Template.png'));
% Make a smaller template
T = imgaussfilt(T,2);
T = imresize(T, 0.5,'bicubic');
fig1 = figure;
ax1 = axes(fig1);
imshow(I,'Parent', ax1);
fig2 = figure;
figure(fig2);
% While loop till reduced image size is 0.2 times original image size
I_tmp = I;
iter = 0;
while numel(I_tmp) / numel(I) >= 0.2
iter = iter + 1;
I_tmp = imgaussfilt(I_tmp,2);
% calculate and display correlation
c = normxcorr2(T,I_tmp);
subplot(4,1,iter);
surf(c);
shading flat;
% get local maxina pixels for correl >0.5
[yvals,xvals] = find(c > min_corel & islocalmax(c,1) == 1 & islocalmax(c,2) == 1);
% display result
%imshow(I_tmp)
for x=1:length(yvals)
fprintf('[%d] [%d] [%d]\n',yvals(x),xvals(x),c(yvals(x),xvals(x)));
yoffSet = iter*(yvals(x)-size(T,1));
xoffSet = iter*(xvals(x)-size(T,2));
rectangle(ax1,'Position',
[xoffSet,yoffSet,iter*size(T,2),iter*size(T,1)],'EdgeColor',[1 0 0]);
end
I_tmp = imresize(I_tmp, 0.8, "bicubic");
end
% Saving fig1 and fig2
saveas(fig1, 'original_image_with_detections.png');
saveas(fig2, 'correlation_surfaces.png');
```

2. Implement an image blending algorithm for two images according to section 3.5.5 in Szeliski textbook. For this, please pick any two images of your choice.

Answer → To create an image blending for two images, here is the workflow to be used:

1. Import two images
2. Create multi resolution pyramid images for both the images

3. Create laplacian pyramids for both the images
4. Create a mask of the foreground image
5. Create a gaussian pyramid of the mask
6. Create the blended image by blending the two images along with the mask using image matting and composting technique (Module 5.1)

Here are the two original images used:

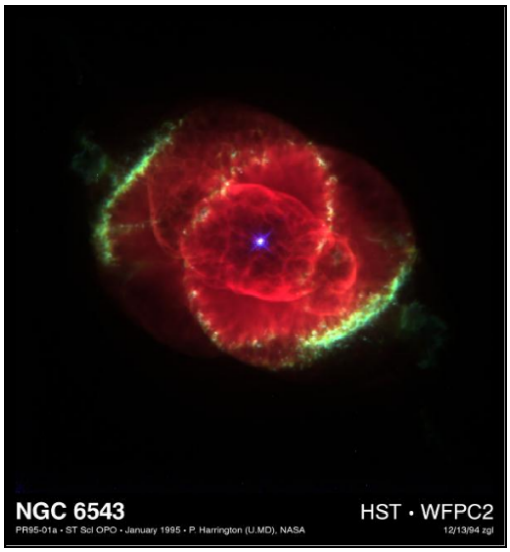
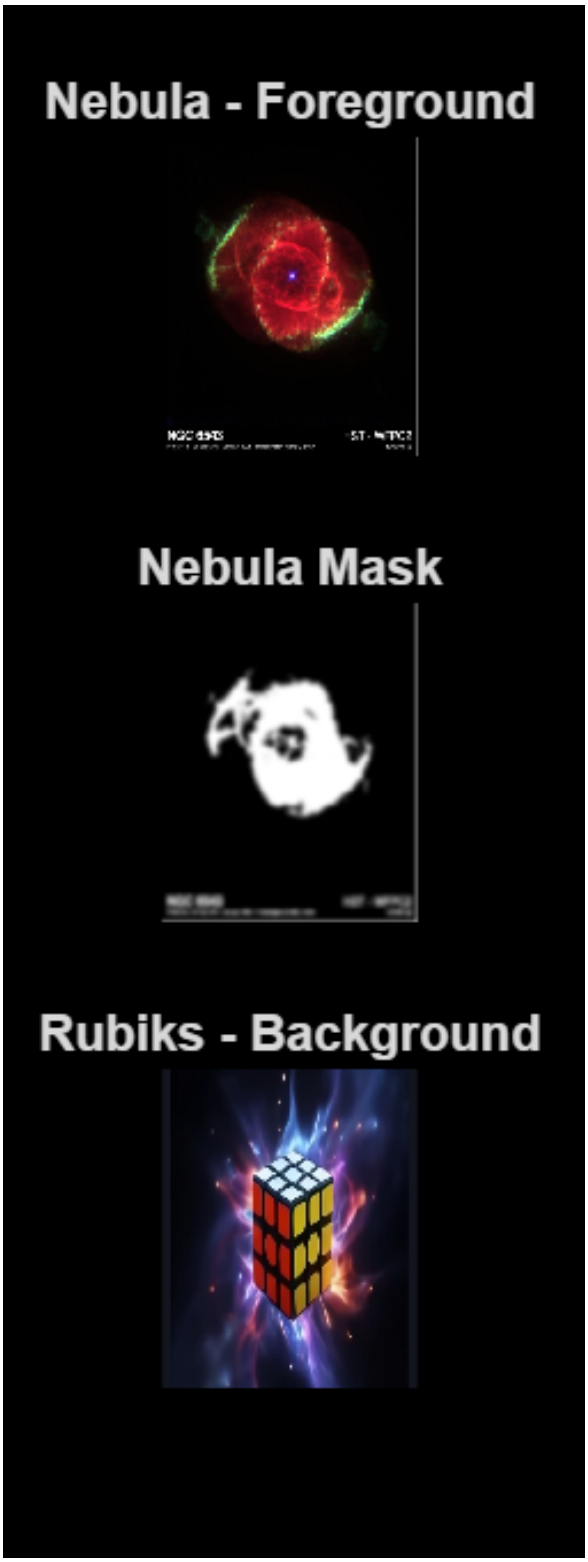
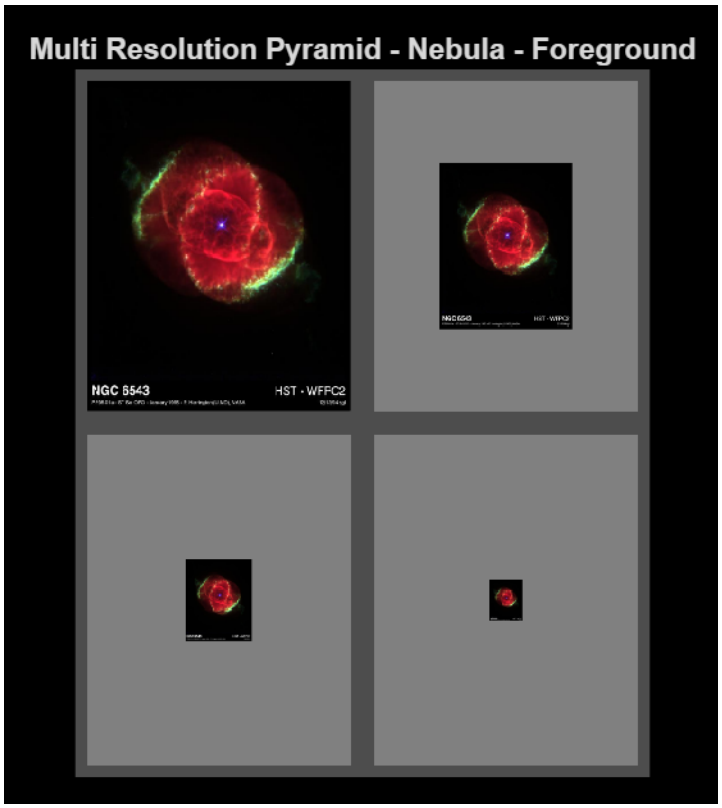
Image 1 - 

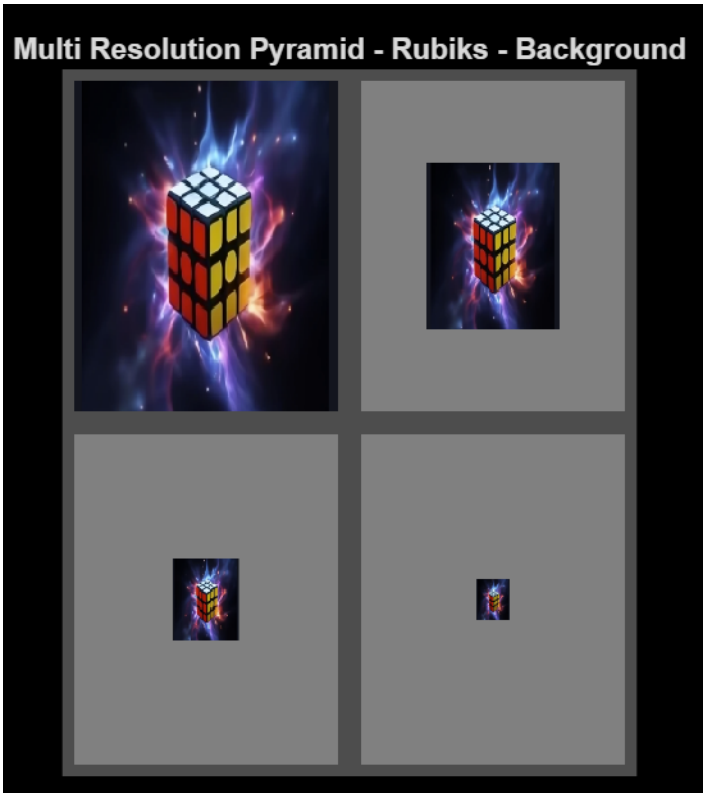
Image 2 - 

Combined Image with Mask:

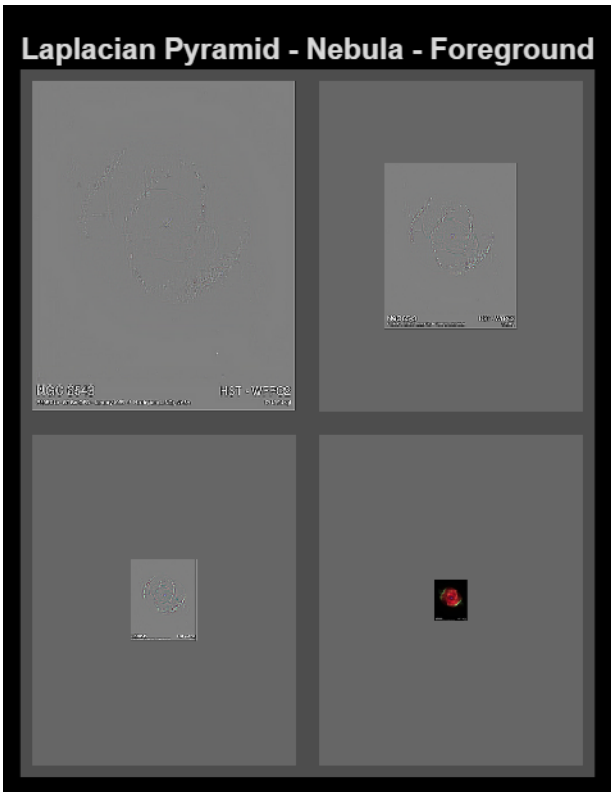


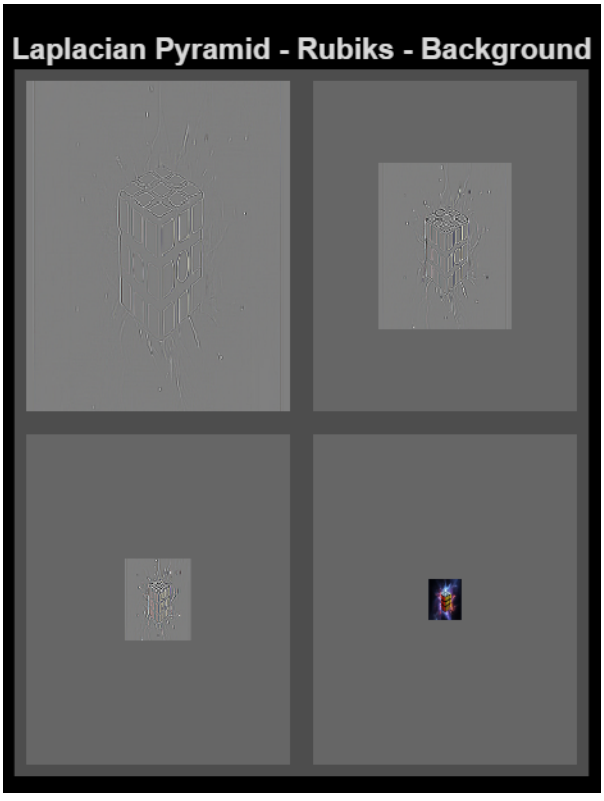
Multi resolution pyramid images for Image 1 and Image 2:





Laplacian pyramid for Images:





Blended Image -

Code:

```
%%
% Load images for Laplacian image blending
img1 = imread('ngc6543a.jpg');
%img2 = imread('street1.jpg');
img2 = imread('rubiks.png');
% Converting images to same size of 600 x 480 pixels
img1 = imresize(img1, [600 480],"bicubic","Antialiasing",true);
img2 = imresize(img2, [600 480],"bicubic","Antialiasing",true);
% Converting images to double for processing
img1 = im2double(img1);
img2 = im2double(img2);
% Creating a binary mask from image1 using thresholding
```

```
img1_gray = rgb2gray(img1);
mask = imbinarize(img1_gray);
mask = im2double(mask);
% Applying Gaussian filter on the mask for smooth blending
mask = imgaussfilt(mask, 5); % higher sigma for smoothness
% Displaying resized images
fig0 = figure;
figure(fig0);
subplot(3, 1, 1); imshow(img1); title("Nebula - Foreground");
subplot(3, 1, 2); imshow(mask); title("Nebula Mask");
subplot(3, 1, 3); imshow(img2); title("Rubiks - Background");
saveas(fig0, "Images and Mask.png");
% Creating multiresolution pyramid
mrp1 = multiresolutionpyramid(img1, 4);
mrp2 = multiresolutionpyramid(img2, 4);
% Function to create multiresolution pyramid
function mrp= multiresolutionpyramid(A, num_levels)
% adapted from https://blogs.mathworks.com/steve/2019/04/09/multiresolution-image-
pyramids-and-impyramid-part-2
M = size(A, 1);
N = size(A, 2);
if nargin < 2
lower_limit = 32;
num_levels = min(floor(log2([M N]) - log2(lower_limit))) + 1;
else
num_levels = min(num_levels, min(floor(log2([M N])))) + 2);
end
% Initialize the pyramid structure
mrp = cell(1, num_levels);
smallest_size = [M N] / 2^(num_levels - 1);
smallest_size = ceil(smallest_size);
padded_size = smallest_size * 2^(num_levels - 1);
Ap = padarray(A,padded_size - [M N],'replicate','post');
mrp{1} = Ap;
for k = 2:num_levels
mrp{k} = imresize(mrp{k-1},0.5,"bicubic");
end
mrp{1} = A;
end
% Visualize the multi resolution pyramid
fig1 = figure;
figure(fig1);
visualizePyramid(mrp1); title("Multi Resolution Pyramid - Nebula - Foreground");
saveas(fig1, "Multi Resolution Pyramid - Nebula - Foreground.png");
fig2 = figure;
figure(fig2);
visualizePyramid(mrp2); title("Multi Resolution Pyramid - Rubiks - Background");
saveas(fig2, "Multi Resolution Pyramid - Rubiks - Background.png");
```

```
% Function to visualize the multi resolution pyramid
function tiles_out = visualizePyramid(p)
% Steve Eddins
% MathWorks
M = size(p{1},1);
N = size(p{1},2);
for k = 1:numel(p)
Mk = size(p{k},1);
Nk = size(p{k},2);
Mpad1 = ceil((M - Mk)/2);
Mpad2 = M - Mk - Mpad1;
Npad1 = ceil((N - Nk)/2);
Npad2 = N - Nk - Npad1;
A = p{k};
A = padarray(A,[Mpad1 Npad1],0.5,'pre');
A = padarray(A,[Mpad2 Npad2],0.5,'post');
p{k} = A;
end
tiles = imtile(p,'GridSize',[NaN 2],'BorderSize',20,'BackgroundColor',[0.3 0.3 0.3]);
imshow(tiles)
if nargout > 0
tiles_out = tiles;
end
end
% Applying laplacian pyramid to each image
lapp1 = laplacianPyramid(mrp1);
lapp2 = laplacianPyramid(mrp2);
% Function to generate laplacian pyramid
function lapp = laplacianPyramid(mrp)
% Steve Eddins
% MathWorks
lapp = cell(size(mrp));
num_levels = numel(mrp);
lapp{num_levels} = mrp{num_levels};
for k = 1:(num_levels - 1)
A = mrp{k};
B = imresize(mrp{k+1},2,"bicubic");
[M,N,~] = size(A);
lapp{k} = A - B(1:M,1:N,:);
end
lapp{end} = mrp{end};
end
% Visualize laplacian pyramids
fig3 = figure;
figure(fig3);
showLaplacianPyramid(lapp1); title("Laplacian Pyramid - Nebula - Foreground");
saveas(fig3, "Laplacian Pyramid - Nebula - Foreground.png");
fig4 = figure;
```

```
figure(fig4);
showLaplacianPyramid(lapp2); title("Laplacian Pyramid - Rubiks - Background");
saveas(fig4, "Laplacian Pyramid - Rubiks - Background.png");
function showLaplacianPyramid(p)
% Steve Eddins
M = size(p{1},1);
N = size(p{1},2);
stretch_factor = 3;
for k = 1:numel(p)
Mk = size(p{k},1);
Nk = size(p{k},2);
Mpad1 = ceil((M - Mk)/2);
Mpad2 = M - Mk - Mpad1;
Npad1 = ceil((N - Nk)/2);
Npad2 = N - Nk - Npad1;
if (k < numel(p))
pad_value = -0.1/stretch_factor;
else
pad_value = 0.4;
end
A = p{k};
A = padarray(A,[Mpad1 Npad1],pad_value,'pre');
A = padarray(A,[Mpad2 Npad2],pad_value,'post');
p{k} = A;
end
for k = 1:(numel(p)-1)
p{k} = (stretch_factor*p{k} + 0.5);
end
imshow(imtile(p,'GridSize',[NaN 2],'BorderSize',20,'BackgroundColor',[0.3 0.3 0.3]))
end
% Creating a Gaussian pyramid of the mask
gp_mask = gaussianPyramid(mask, 4);
% Function to create gaussian pyramid
function gp = gaussianPyramid(A, num_levels)
% Determining how many iterations are needed
M = size(A, 1);
N = size(A, 2);
if nargin < 2
lower_limit = 32;
num_levels = min(floor(log2([M N]) - log2(lower_limit))) + 1;
else
num_levels = min(num_levels, min(floor(log2([M N])))) + 2);
end
% Initialize the pyramid structure
gp = cell(1, num_levels);
gp{1} = A;
for iter = 2:num_levels
% Perform gaussian pyramid reduction on the prior level
```

```
gp{iter} = impyramid(gp{iter-1}, 'reduce');
end
end
% Blending images (Image 1, Mask, Image2)
blended_img = reconstructFromLaplacianPyramid(lapp1, lapp2, gp_mask);
% Function to reconstruct image
function img = reconstructFromLaplacianPyramid(lapp1, lapp2, mask)
num_levels = numel(lapp2);
img = repmat(mask{num_levels}, [1, 1, 3]).* lapp1{num_levels} + (1-
repmat(mask{num_levels}, [1, 1, 3])).* lapp2{num_levels}; % Blending the top level of
pyramid
for k = num_levels-1:-1:1
img = imresize(img, 2, 'bicubic');
blend = repmat(mask{k}, [1, 1, 3]).* lapp1{k} + (1- repmat(mask{k}, [1, 1, 3])).*
lapp2{k}; % Blend with the mask
[M, N, ~] = size(blend);
img = img(1:M, 1:N, :) + blend;
end
end
% Display the blended image
figure;
imshow(blended_img);
title('Blended Image');
% Save the blended image to a file
imwrite(blended_img, 'blended_image.png');
```