**Computer Vision (2024 Fall) Project Report**

(DC227524) ETHAN, ZHANG JIAYANG

(DC227268) DIEGO, HUANG WU

Our code repository <https://github.com/Creepia/Vision_project.git>

# Introduction

In this project, we finished three types of tasks as below.

## Generating Image Pyramids

In this part, we would generate the Gaussian pyramids and Laplacian pyramids for gray images or color images then represent them in the required format.

## Hybrid Images

In this part, we would input two images and hybrid the first image’s low frequency and the second image’s high frequency, which are from Gaussian pyramids and Laplacian pyramids, into a hybrid pyramid. Then we would reconstruct the result image from the Nth layer of the hybrid pyramid.

## Blend Images

In this part, we would input two images and blend them into a new image with the method of blurring them in a pyramid. We have to different methods. The first way is calculating a left-right linear gradient weight for each layer and apply it for each layer blending. The left part of the result image is from the first images and right part is from the second image. We can use the window parameter to adjust the blending area. Another way is that we could use a mask and calculate the pyramid of the mask first then for each layer the mask is also blurred.

In the second part, we would show all image results and the flowchart for the program to do each task would be shown in the third part.

The source code in “.mlx” (MATLAB Interactive Script) format is attached. Or you can find the plain text code at the end of this report.

# Results

Then we show all the image results for each task from next page.

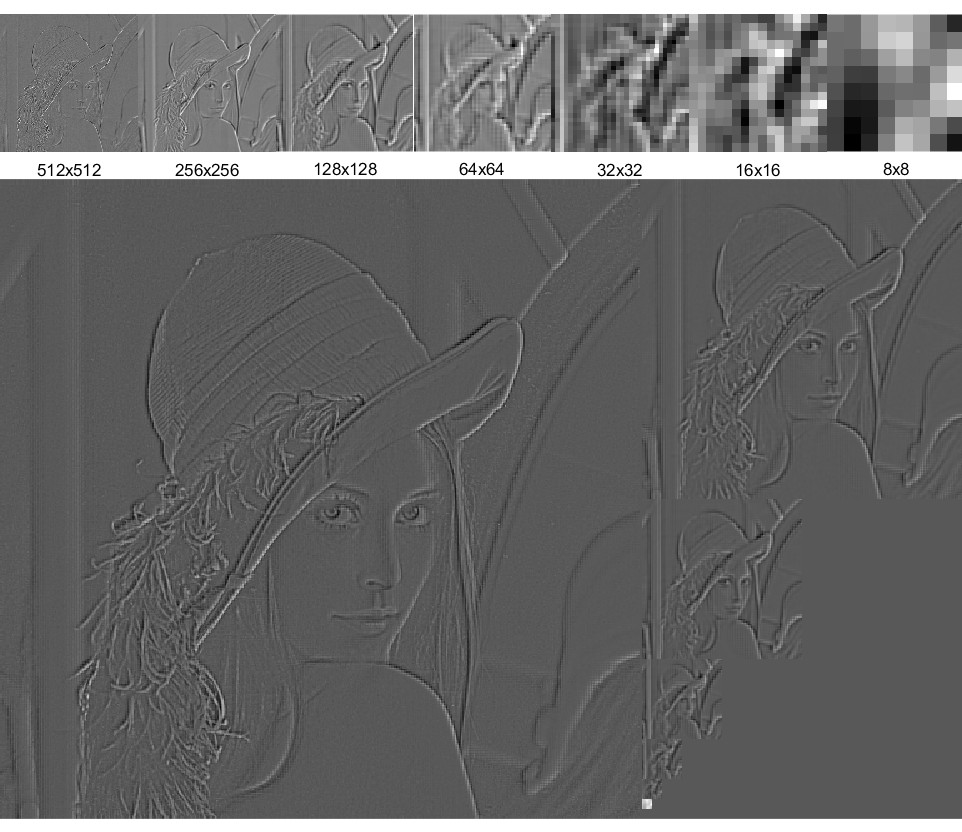
## Generating Image Pyramids

### Classical Image – Lenna (Gray)

#### Gaussian Pyramid



### Laplacian Pyramid



### Character Icon – Emu (Color)

#### Gaussian Pyramid

A cartoon of a child holding a fish

Description automatically generated

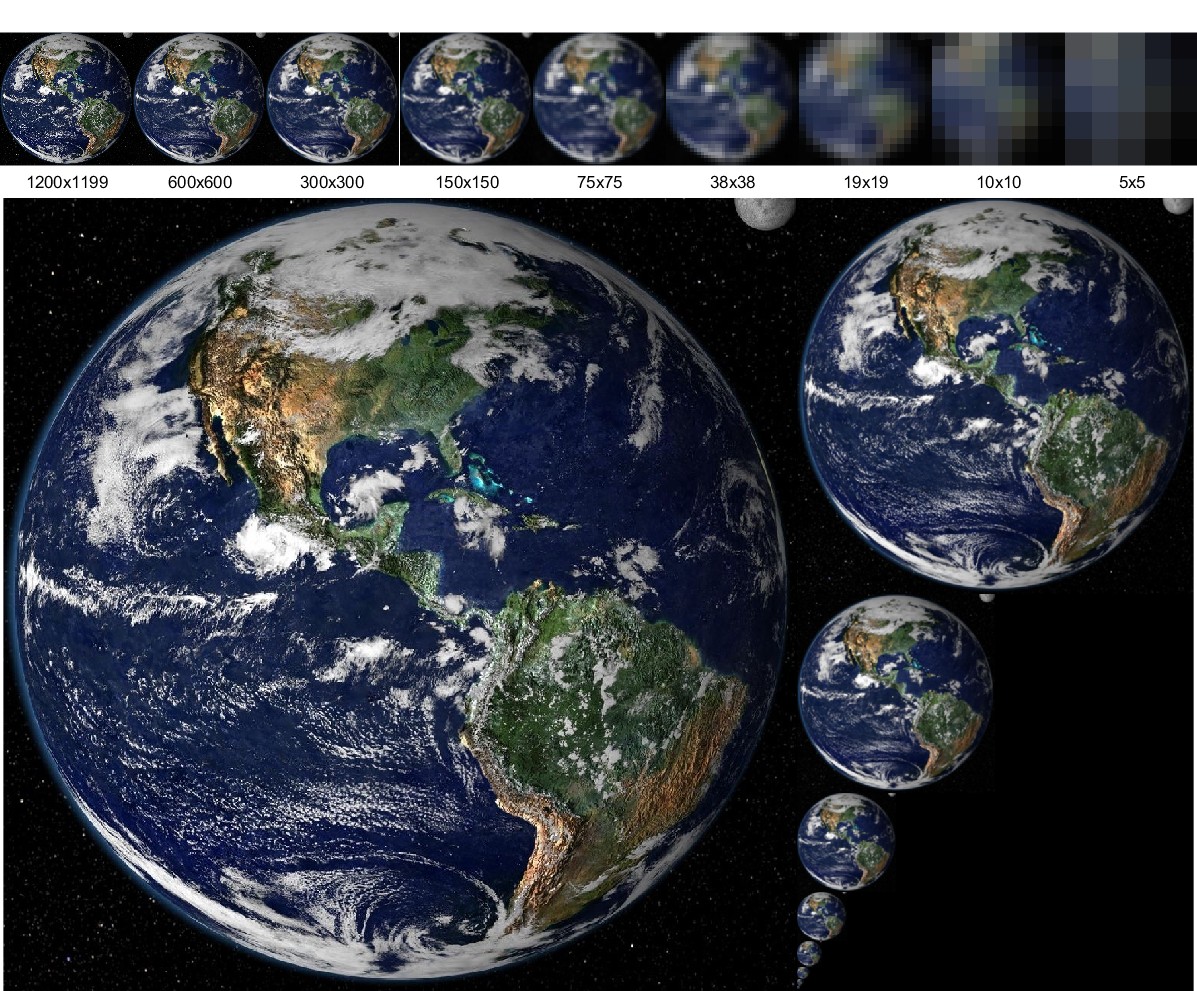
#### Laplacian Pyramid

A screenshot of a cartoon

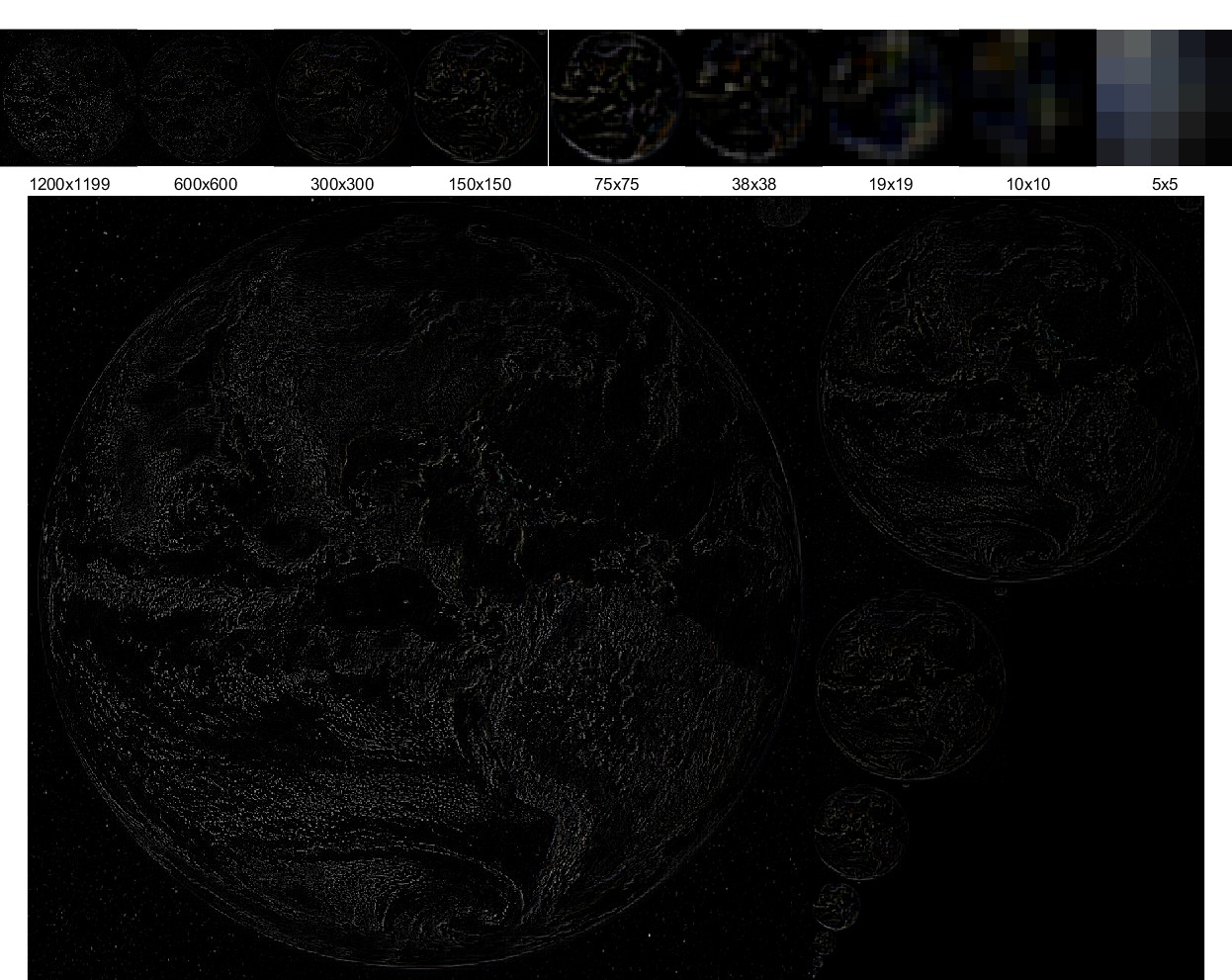
Description automatically generated

### Satellite Image – Earth (Color)

#### Gaussian Pyramid



#### Laplacian Pyramid



## Hybrid Images

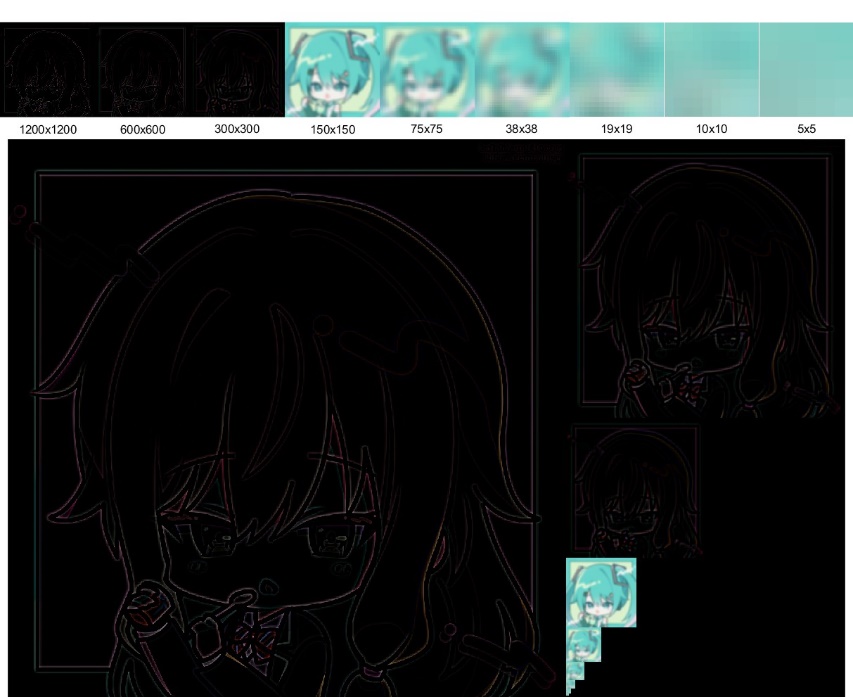
### Character Icons – Miku & Nene

#### Original Input Images

 A cartoon of a child with blue hair

Description automatically generated

#### Hybrid Pyramid



#### Result in different scaling

A cartoon of a child

Description automatically generated

Parameters: N=4, Gaussian kernel size=11, sigma=5.

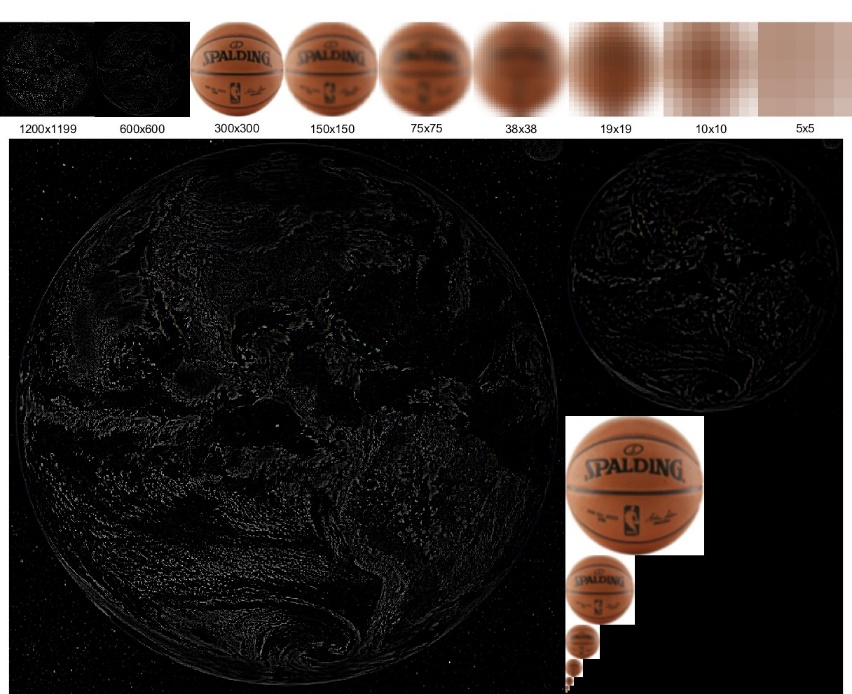
(We used a horizontal flip to match the facing direction)

### Basketball & Earth

#### Original Input Images

#### Hybrid Pyramid



#### Result in different scaling



Parameters: N=3, Gaussian kernel size=13, sigma=5.

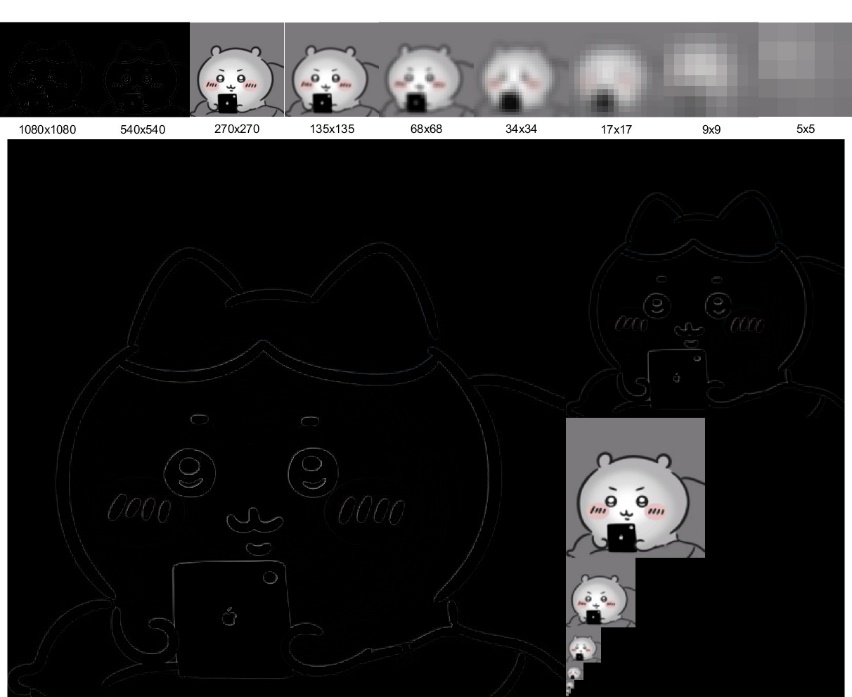
(We used a cropping and resizing to match the resolution and position of the objects)

### Characters – Chiikawa & Hachiwari

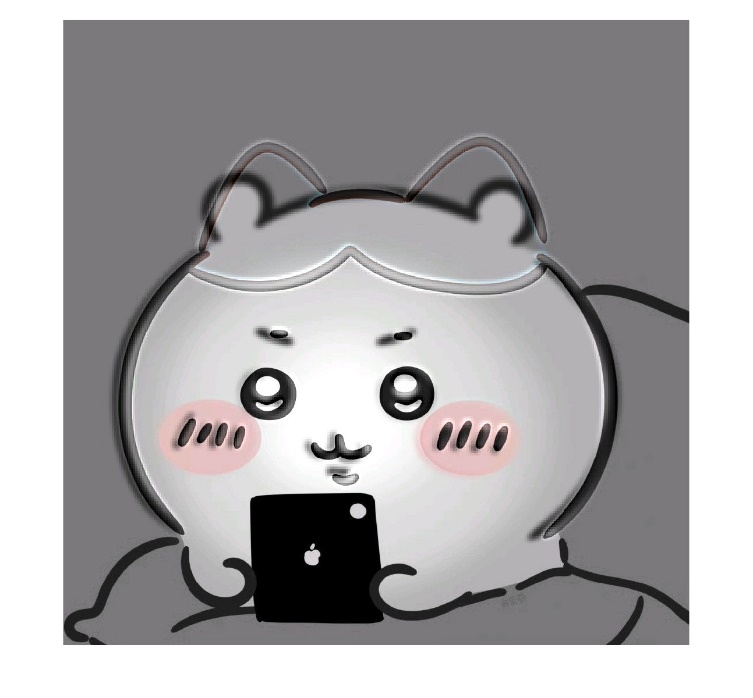
#### Original Input Images

#### Hybrid Pyramid



#### Result in different scaling



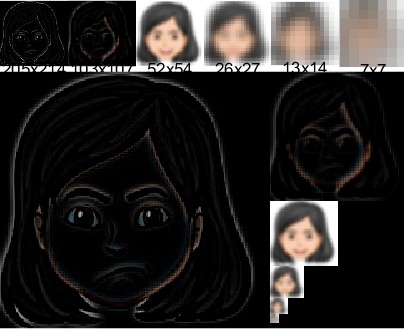
Parameters: N=3, Gaussian kernel size=7, sigma=3.

### Emotion Icons – Smile & Angry

#### Original Input Images

#### Hybrid Pyramid



#### Result in different scaling



Parameters: N=3, Gaussian kernel size=9, sigma=3.

## Blending

### Characters – Usagi & Chii (Left-Right)

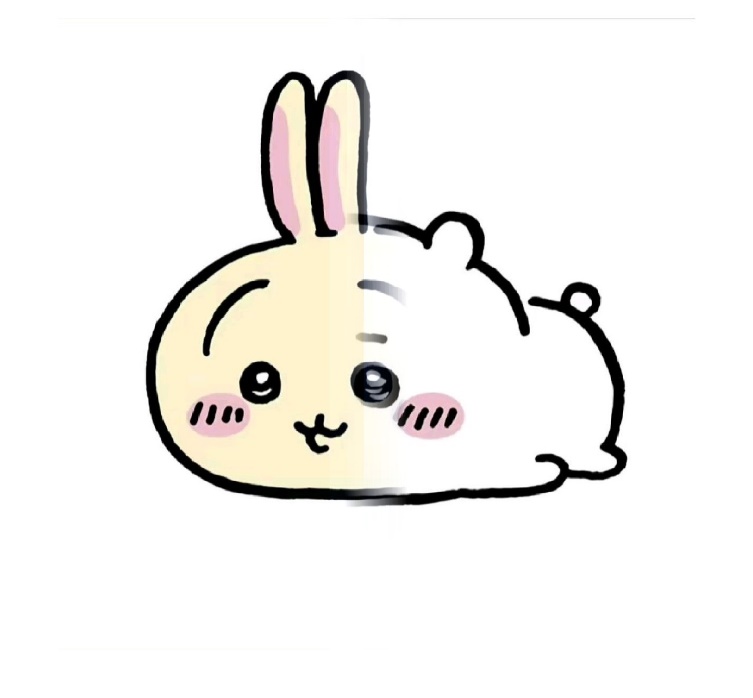
#### Original Input Images

#### Blend Pyramid



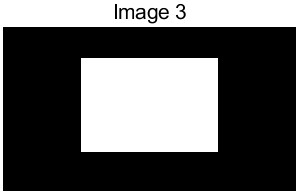
#### Result



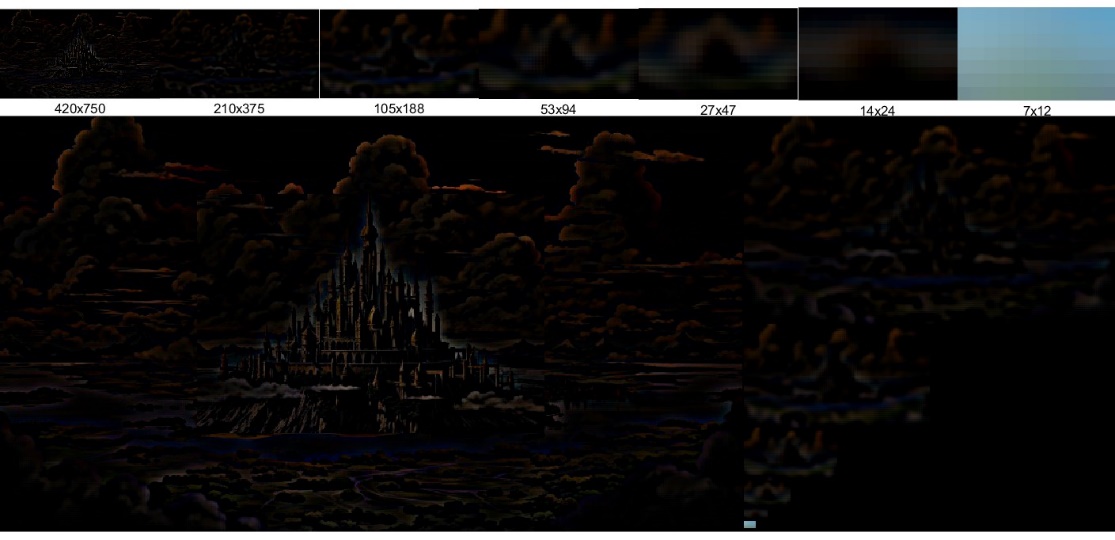
Parameters: Gaussian kernel size=25, sigma=5, window=[0.45 0.55].

### Castle Scenes – Scene1 & Scene2 (Regional)

#### Original Input Images

#### Blend Pyramid



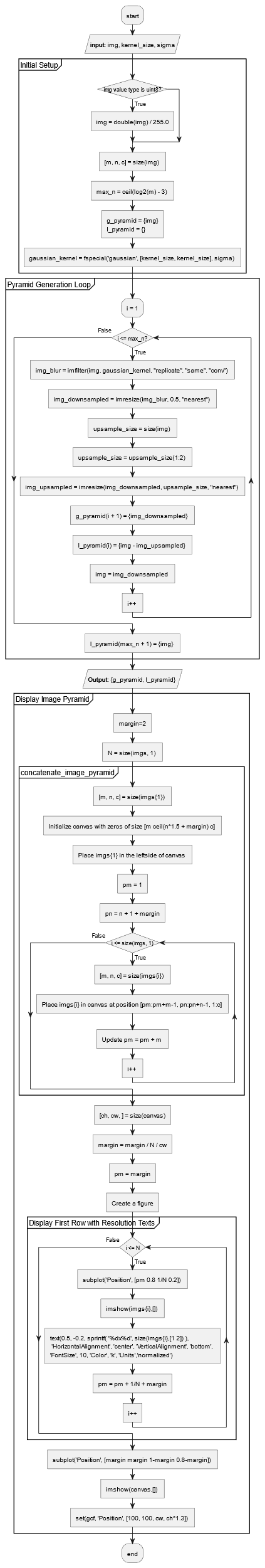
#### Result



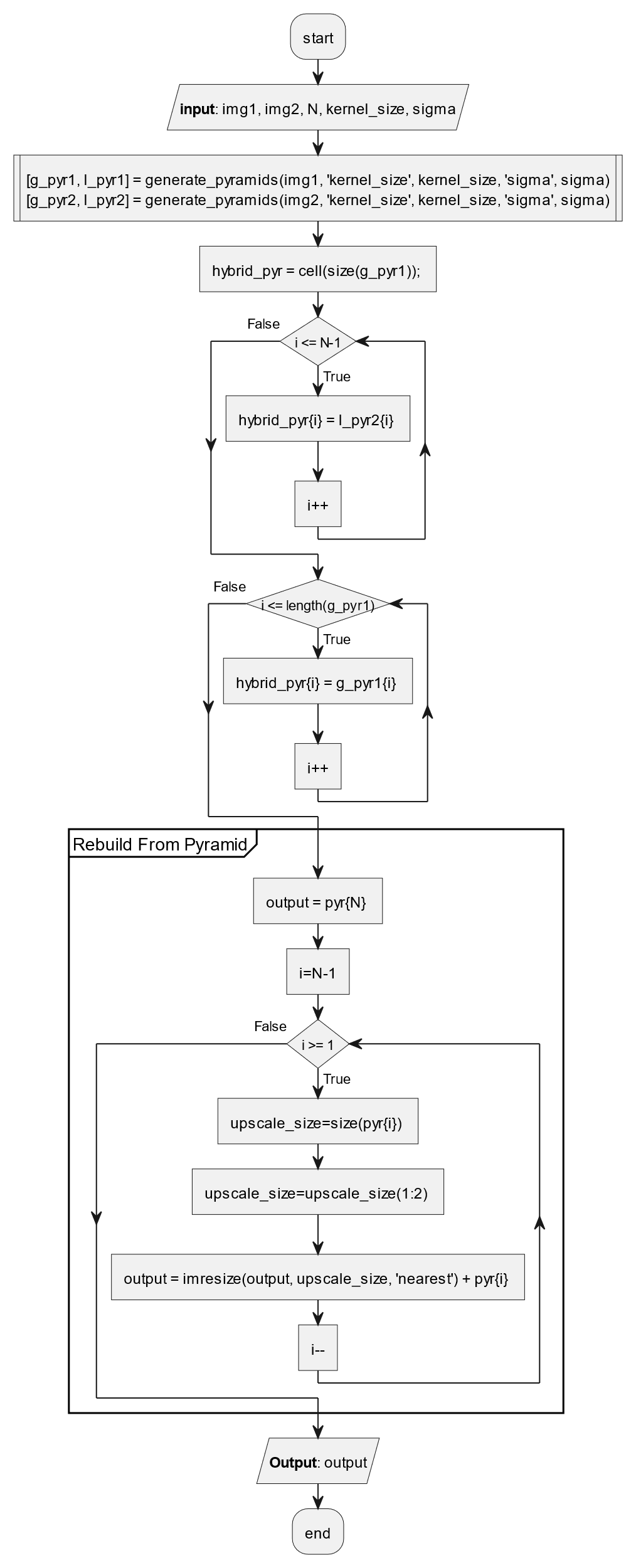
Parameters: Gaussian kernel size=25, sigma=5.

# Flow Chart

## Generating Image Pyramids

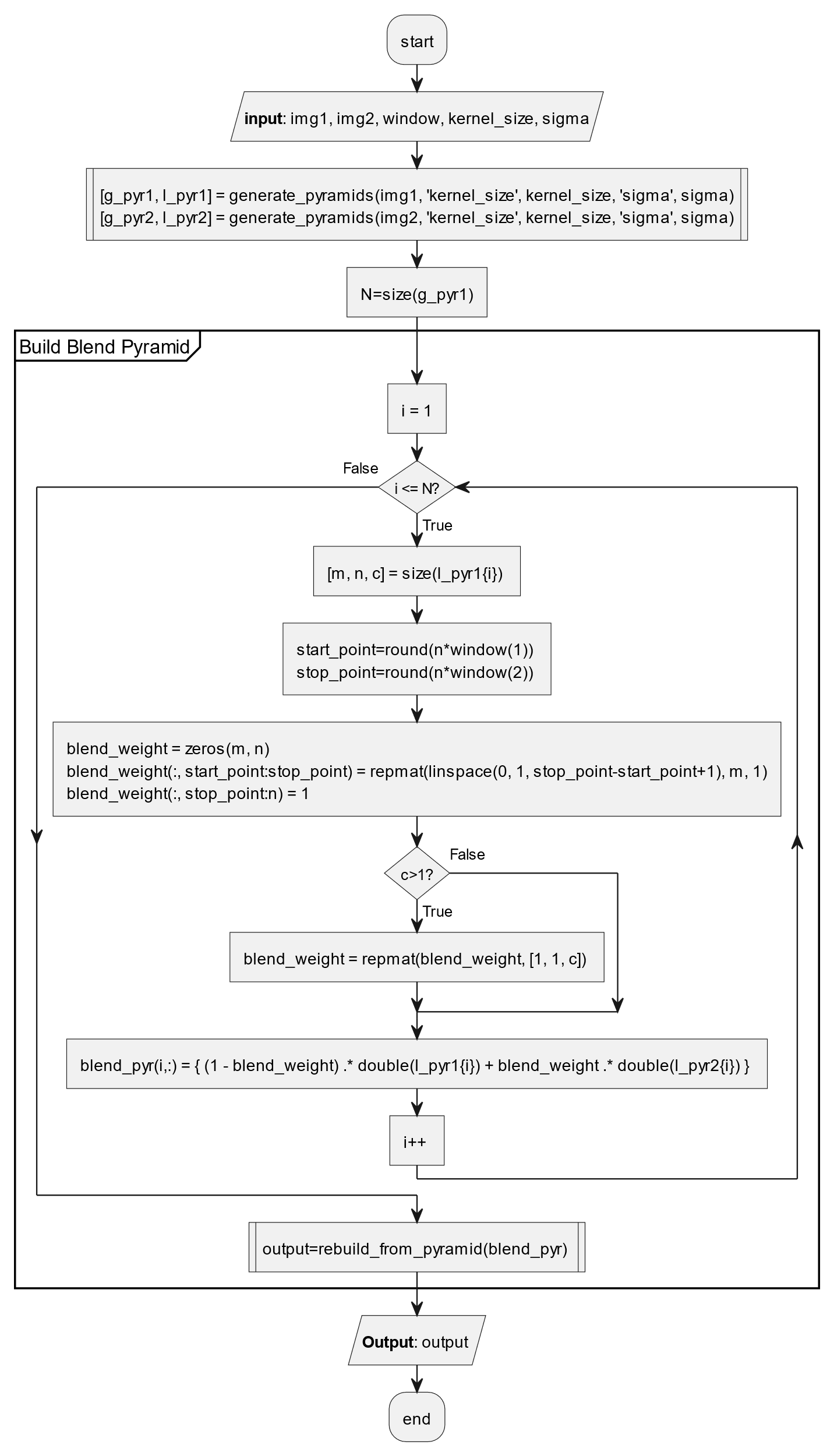


## Hybrid Images

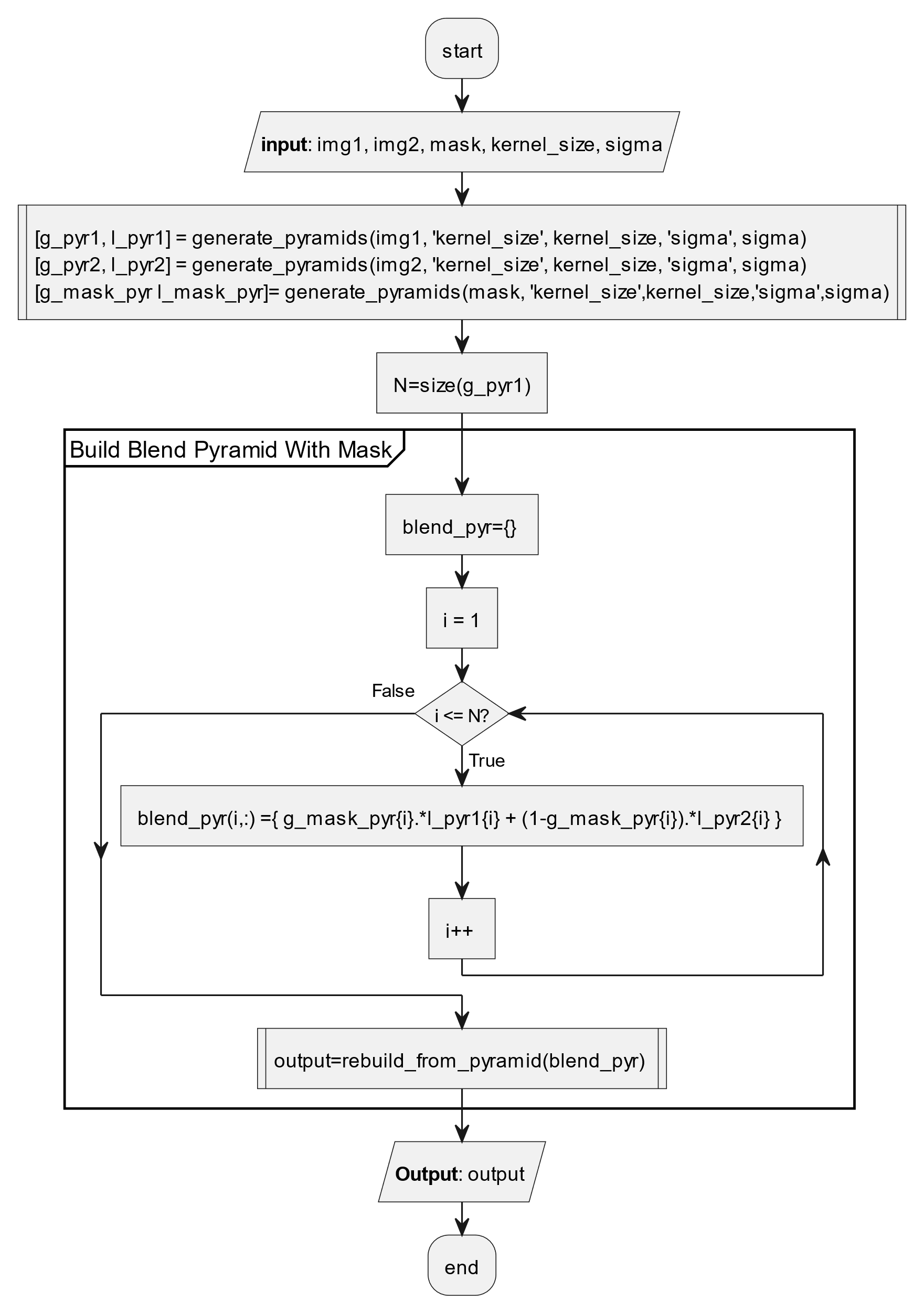


## Blend Images

### Left-Right Blending



### Mask Blending



# Code in Plain Text

% Read test images.

img\_lenna=imread("Lenna.jpg");

img\_lenna=im2gray(img\_lenna);

img\_emu=imread("emu.jpg");

img\_nene=imread("nene.jpg");

img\_miku=imread("miku.jpg");

img\_scene1=imread('scene1.jpg');

img\_scene2=imread('scene2.jpg');

img\_sky\_green=imread('sky\_green.jpg');

img\_sky\_red=imread('sky\_red.jpg');

img\_usagi=imread('phone\_usagi.jpg');

img\_chii=imread('phone\_chii.jpg');

img\_8wari=imread('phone\_8wari.jpg');

img\_lying\_usagi=imread('lying\_usagi.jpg');

img\_lying\_chii=imread('lying\_chii.jpg');

img\_lying\_chii=imresize(img\_lying\_chii,size(img\_lying\_usagi,1:2),'nearest');

img\_woman\_expression1=imread('woman\_expression1.jpg');

img\_woman\_expression2=imread('woman\_expression2.jpg');

img\_basketball=imread('basketball.jpg');

img\_earth=imread('earth.jpg');

img\_earth=imcrop(img\_earth,[91 1 1038 1036]);

img\_earth=imresize(img\_earth,size(img\_basketball,1:2),'nearest');

% figure, imshow(img\_earth)

% Inner function implementation.

% I1 = impyramid(img\_lenna, 'reduce');

% I2 = impyramid(I1, 'reduce');

% I3 = impyramid(I2, 'reduce');

% figure, imshow(img\_lenna)

% figure, imshow(I1)

% figure, imshow(I2)

% figure, imshow(I3)

% Generate the image pyramid.

[g\_pyramid,l\_pyramid]=generate\_pyramids(img\_lenna);

display\_image\_pyramid(g\_pyramid);

display\_image\_pyramid(l\_pyramid);

[g\_pyramid,l\_pyramid]=generate\_pyramids(img\_emu);

display\_image\_pyramid(g\_pyramid);

display\_image\_pyramid(l\_pyramid);

[g\_pyramid,l\_pyramid]=generate\_pyramids(img\_earth);

display\_image\_pyramid(g\_pyramid);

display\_image\_pyramid(l\_pyramid);

% Hybrid images (lowfreq-highfreq).

hybrid\_miku\_nene=hybrid\_image(img\_miku,flip(img\_nene,2),4,'kernel\_size',11,'sigma',5);

figure,imshow(hybrid\_miku\_nene);

hybrid\_basketball\_earth=hybrid\_image(img\_basketball,img\_earth,3,'kernel\_size',13,'sigma',5);

figure,imshow(hybrid\_basketball\_earth);

hybrid\_chii\_8wari=hybrid\_image(img\_chii,img\_8wari,3,'kernel\_size',7,'sigma',3);

figure,imshow(hybrid\_chii\_8wari);

hybrid\_woman\_emotions=hybrid\_image(img\_woman\_expression2,img\_woman\_expression1,3,'kernel\_size',9,'sigma',3);

figure,imshow(hybrid\_woman\_emotions);

% Blend images (left-right).

blend\_lyings=pyramid\_blend\_lr(img\_lying\_usagi,img\_lying\_chii,'kernel\_size',25,'sigma',5,'window',[0.45 0.55]);

figure,imshow(blend\_lyings);

% Blend images (mask).

mask1=zeros(size(img\_scene1));

mask1(80:320,200:550,:)=1;

blendmask\_sc1\_sc2=pyramid\_blend(img\_scene1,img\_scene2,mask1,'kernel\_size',25,'sigma',5);

figure,imshow(blendmask\_sc1\_sc2);

% Implement the functions manually.

function [g\_pyramid,l\_pyramid]=generate\_pyramids(img,varargin)

ip=inputParser;

addParameter(ip, 'kernel\_size', 5);

addParameter(ip, 'sigma', 2);

parse(ip,varargin{:});

kernel\_size=ip.Results.kernel\_size;

sigma=ip.Results.sigma;

if class(img)=="uint8"

img=double(img)/255.0;

end

[m,n,c]=size(img);

max\_n=ceil(log2(m)-3);

g\_pyramid={img};

l\_pyramid={};

gaussian\_kernel = fspecial('gaussian', [kernel\_size kernel\_size], sigma);

for i=1:max\_n

img\_blur=imfilter(img,gaussian\_kernel,"replicate","same","conv");

% img\_blur=imfilter(img,gaussian\_kernel,"replicate");

img\_downsampled=imresize(img\_blur,0.5,"nearest");

upsample\_size=size(img);

upsample\_size=upsample\_size(1:2);

img\_upsampled=imresize(img\_downsampled,upsample\_size,"nearest");

g\_pyramid(i+1,:)={img\_downsampled};

l\_pyramid(i,:)={img-img\_upsampled};

img=img\_downsampled;

end

% disp(max\_n+1);

l\_pyramid(max\_n+1,:)={img};

end

function display\_image\_pyramid(imgs, varargin)

ip=inputParser;

addParameter(ip, 'margin', 2);

parse(ip,varargin{:});

margin=ip.Results.margin;

N = size(imgs, 1);

canvas = concatenate\_image\_pyramid(imgs, 'margin', margin);

% figure, imshow(canvas);

[ch,cw,~] = size(canvas);

margin = margin / N / cw;

pm = margin;

figure;

for i = 1:N

% disp([pm 0.6 1/N 0.45])

subplot('Position', [pm 0.8 1/N 0.2]);

imshow(imgs{i},[]);

text(0.5, -0.2, sprintf( '%dx%d', size(imgs{i},[1 2]) ), ...

'HorizontalAlignment', 'center', 'VerticalAlignment', 'bottom', ...

'FontSize', 10, 'Color', 'k', 'Units','normalized');

pm = pm + 1/N + margin;

end

% disp(margin)

subplot('Position', [margin margin 1-margin 0.8-margin]);

imshow(canvas,[]);

set(gcf, 'Position', [100, 100, cw, ch\*1.3]);

end

function canvas=concatenate\_image\_pyramid(imgs,varargin)

ip=inputParser;

addParameter(ip, 'margin', 2);

parse(ip,varargin{:});

margin=ip.Results.margin;

[m,n,c]=size(imgs{1});

canvas=zeros([m ceil(n\*1.5+margin) c]);

canvas(1:m,1:n,1:c)=imgs{1};

pm=1;

pn=n+1+margin;

for i=2:size(imgs,1)

[m,n,c]=size(imgs{i});

canvas(pm:pm+m-1,pn:pn+n-1,1:c)=imgs{i};

pm=pm+m;

end

end

function output = hybrid\_image(img1, img2, N, varargin)

ip=inputParser;

addParameter(ip, 'kernel\_size', 5);

addParameter(ip, 'sigma', 2);

parse(ip,varargin{:});

kernel\_size=ip.Results.kernel\_size;

sigma=ip.Results.sigma;

% Generate the pyramids

[g\_pyr1, l\_pyr1] = generate\_pyramids(img1, 'kernel\_size',kernel\_size,'sigma',sigma);

[g\_pyr2, l\_pyr2] = generate\_pyramids(img2, 'kernel\_size',kernel\_size,'sigma',sigma);

hybrid\_pyr = cell(size(g\_pyr1));

% Build the hybrid pyramid

for i = 1:N-1

hybrid\_pyr{i} = l\_pyr2{i}; % High-frequency for img2

end

for i = N:length(g\_pyr1)

hybrid\_pyr{i} = g\_pyr1{i}; % Low-frequency for img1

end

% figure, imshow(concatenate\_image\_pyramid(hybrid\_pyr));

display\_image\_pyramid(hybrid\_pyr);

output = rebuild\_from\_pyramid(hybrid\_pyr,'N',N);

% show\_images({img1,img2,output});

end

function output = pyramid\_blend\_lr(img1, img2, varargin)

ip=inputParser;

addParameter(ip, 'kernel\_size', 5);

addParameter(ip, 'sigma', 2);

addParameter(ip, 'window', [0.3 0.7]);

parse(ip,varargin{:});

kernel\_size=ip.Results.kernel\_size;

sigma=ip.Results.sigma;

window=ip.Results.window;

% Generate the (laplacian) pyramids

[g\_pyr1, l\_pyr1] = generate\_pyramids(img1, 'kernel\_size',kernel\_size,'sigma',sigma);

[g\_pyr2, l\_pyr2] = generate\_pyramids(img2, 'kernel\_size',kernel\_size,'sigma',sigma);

N=size(g\_pyr1);

% Build the blend pyramid

blend\_pyr ={};

for i = 1:N

[m, n, c] = size(l\_pyr1{i});

start\_point=round(n\*window(1));

stop\_point=round(n\*window(2));

% Generate the linear gradient weight image

blend\_weight = zeros(m, n);

blend\_weight(:, start\_point:stop\_point) = repmat(linspace(0, 1, stop\_point-start\_point+1), m, 1);

blend\_weight(:, stop\_point:n) = 1;

if c > 1

blend\_weight = repmat(blend\_weight, [1, 1, c]);

end

% figure,imshow(blend\_weight);

% Apply the linear gradient weight image for this layer

blend\_pyr(i,:) = { (1 - blend\_weight) .\* double(l\_pyr1{i}) + blend\_weight .\* double(l\_pyr2{i}) };

end

% figure, imshow(concatenate\_image\_pyramid(blend\_pyr) ,[]);

display\_image\_pyramid(blend\_pyr);

output=rebuild\_from\_pyramid(blend\_pyr);

% show\_images({img1, img2, output});

end

function output=rebuild\_from\_pyramid(pyr,varargin)

ip=inputParser;

addParameter(ip, 'N', size(pyr));

parse(ip,varargin{:});

N=ip.Results.N;

% disp(N);

output = pyr{N};

for i = N-1:-1:1

upscale\_size=size(pyr{i});

upscale\_size=upscale\_size(1:2);

output = imresize(output, upscale\_size, 'nearest') + pyr{i};

end

% disp(output);

% figure, imshow(output);

end

function output = pyramid\_blend(img1, img2, mask, varargin)

ip=inputParser;

addParameter(ip, 'kernel\_size', 5);

addParameter(ip, 'sigma', 2);

parse(ip,varargin{:});

kernel\_size=ip.Results.kernel\_size;

sigma=ip.Results.sigma;

% Generate the pyramids

[g\_pyr1, l\_pyr1] = generate\_pyramids(img1, 'kernel\_size',kernel\_size,'sigma',sigma);

[g\_pyr2, l\_pyr2] = generate\_pyramids(img2, 'kernel\_size',kernel\_size,'sigma',sigma);

[g\_mask\_pyr l\_mask\_pyr]= generate\_pyramids(mask, 'kernel\_size',kernel\_size,'sigma',sigma);

N=size(g\_pyr1,1);

% disp(g\_mask\_pyr)

% display\_image\_pyramid(g\_pyr1);

% Build the blend pyramid

blend\_pyr = {};

for i = 1:N

% disp({g\_mask\_pyr{i}, l\_pyr1{i} } );

blend\_pyr(i,:) ={ g\_mask\_pyr{i}.\*l\_pyr1{i} + (1-g\_mask\_pyr{i}).\*l\_pyr2{i} };

end

display\_image\_pyramid(blend\_pyr);

% disp(blend\_pyr);

% disp(blend\_pyr);

output = rebuild\_from\_pyramid(blend\_pyr);

% show\_images({img1 img2 mask output});

end

function show\_images(imgs)

% Display input list of images

num\_imgs = length(imgs); % Determine the number of images

% Calculate suitable layout

rows = ceil(sqrt(num\_imgs)); % Determine the number of rows

cols = ceil(num\_imgs / rows); % Determine the number of columns

figure;

for i = 1:num\_imgs

subplot(rows, cols, i); % Create subplot

imshow(imgs{i},'InitialMagnification','fit'); % Display image

title(['Image ', num2str(i)]); % Add title

end

end