Wavelets

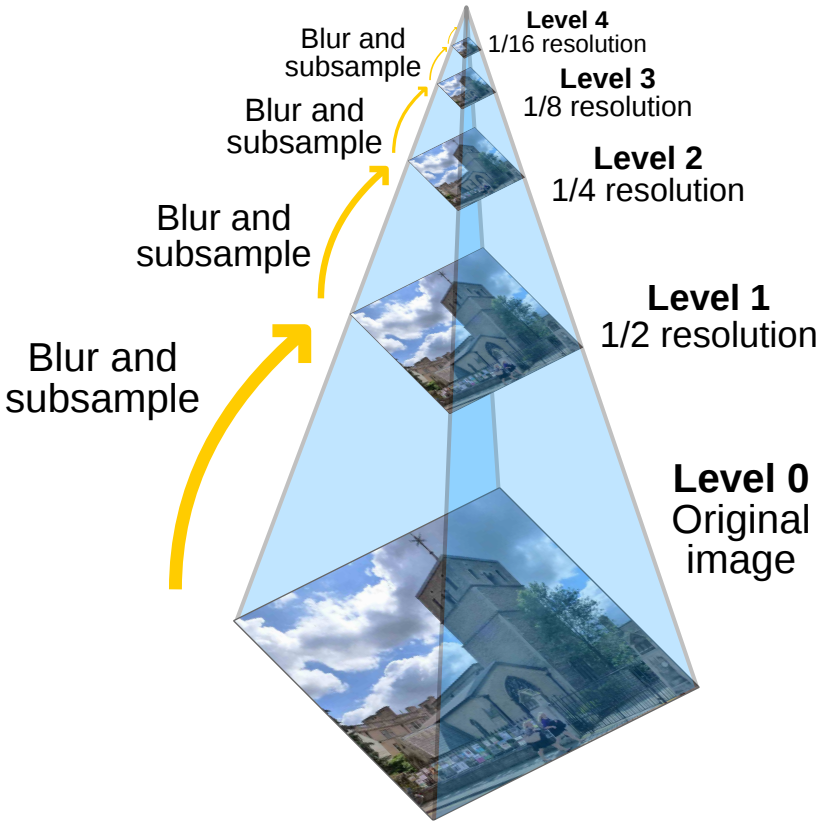
Introducción

- La transformada de Fourier es muy utilizada en procesamiento de imagenes. Hemos visto varios ejemplos.
- A partir de los años 80 se empezaron a utilizar otras transformadas que facilitan la compresión, transmisión y análisis de las imágenes.
- Las ondículas (wavelets) son señales de pequeña duración (en contraste con las señales generadoras de Fourier que son de duración infinita) de frecuencia variable.
- Que sean de duración finita permite que tengan información del momento en el que ocurren (como las notas en una partitura) en contraste a la transformada de Fourier que nos da información si ciertas señales ocurren o no.

Representación piramidal

- Es una forma de representar imágenes en distintas resoluciones.
- A medida que nos movemos en forma ascendente en la pirámide la imagen baja su tamaño y su resolución. La base tiene una resolución de NxN o de $2^J x 2^J$ donde $J = log_2 N$. En general el nivel j ($0 \leq j \leq J$) tiene una resolución de $2^j x 2^j$. El nivel 0 tiene una resolución de $1x1$ la cual no nos aporta mucha información. Se trabajará con imágenes que van desde el nivel J hasta el nivel J-P, con $P \leq J$. Por lo tanto vamos a tener P+1 niveles, donde $j = J - P, \ldots, J-2, J-1, J$.
  La cantidad total de pixeles será:

$$N^2(1 + \frac{1}{(4)^1} + \frac{1}{(4)^2} + \frac{1}{(4)^4} + \ldots + + \frac{1}{(4)^P}) \leq \frac{4}{3}N^2$$



Podemos ir subiendo y bajando la resolución de la imagen. Se llama residuo a la diferencia entre una de las imágenes de alta resolución y la imagen de baja resolución interpolada.

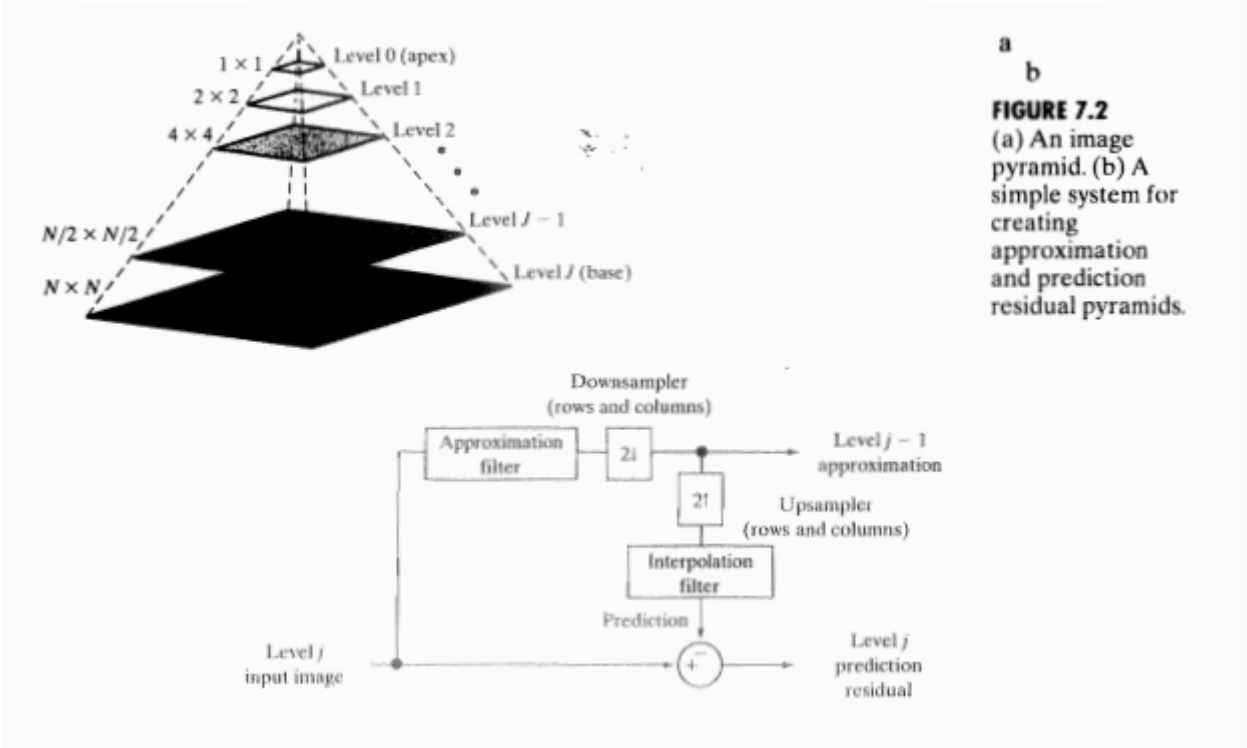


FIGURE 7.2
(a) An image pyramid. (b) A simple system for creating approximation and prediction residual pyramids.

Vamos a definir dos pirámides: la pirámide de aproximación y la pirámide de residuos. A partir de la imágen de nivel J-P y la pirámide de residuos, puede obtener la pirámide de aproximación.

Tipos de aproximaciones posibles:

- Media
- Filtro Gaussiano
- Subsampling

También se pueden definir varios tipos de interpolación:

- Nearest Neighbor
- Bilineal
- Bicúbica

OPENCV utiliza filtro Gaussiano para bajar de resolución las imágenes. El mismo es un filtro con los siguientes coeficientes:

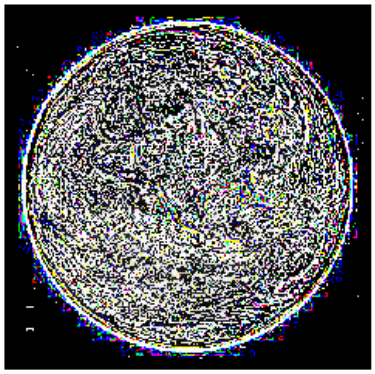
$$H = \frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

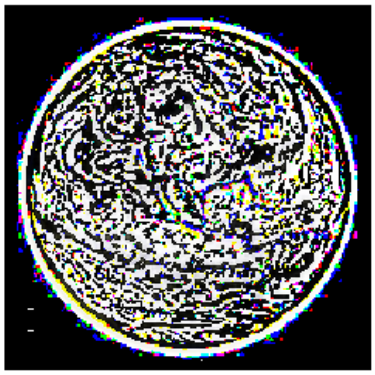
A este tipo de pirámide se la llama pirámide Gaussiana. A la pirámide cuyo nivel j se forma con la diferencia del nivel j+1 de la pirámide Gaussiana expandido a j y el nivel j de la misma pirámide Gaussiana se la llama pirámide Laplaciana.

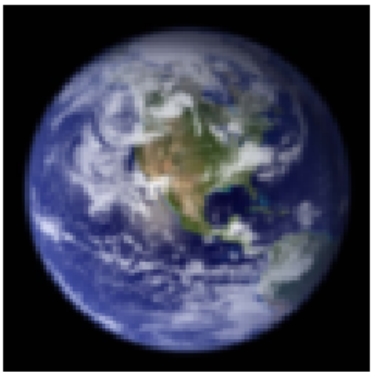
Ejemplo de pirámide de imágenes

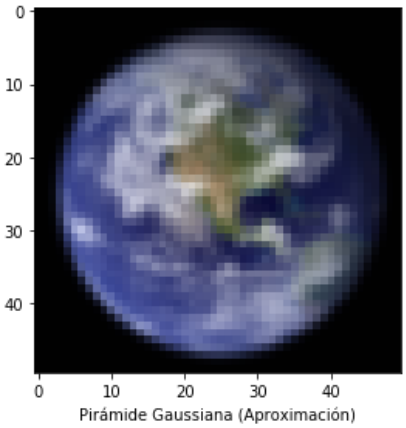
In [2]:

```
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
niveles=4
img = cv2.imread('mundo.jpg')
lower_reso = cv2.pyrDown(img)
higher_reso2 = cv2.pyrUp(lower_reso)
residuo=img-higher_reso2
f, axarr = plt.subplots(niveles, 2, figsize=(15, 5*niveles))
for i in range(niveles-1):
    axarr[i,0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    img_temp=cv2.pyrDown(img)
    residuo=img-cv2.pyrUp(img_temp)
    axarr[i,1].imshow(cv2.cvtColor(residuo, cv2.COLOR_BGR2RGB))
    img=img_temp
    axarr[i,0].axis("off")
    axarr[i,1].axis("off")
axarr[niveles-1,0].imshow(cv2.cvtColor(img_temp, cv2.COLOR_BGR2RGB))
axarr[niveles-1,0].set_xlabel("Pirámide Gaussiana (Aproximación)")
axarr[niveles-1,1].imshow(cv2.cvtColor(img_temp, cv2.COLOR_BGR2RGB))
axarr[niveles-1,1].set_xlabel("Pirámide Laplaciana (Residuo)")
axarr[i,0].axis("off")
axarr[i,1].axis("off")
plt.show()
```









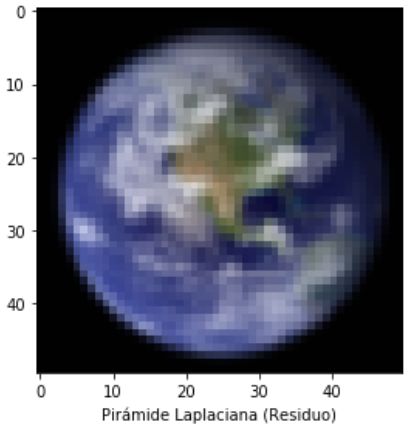


Image Blending utilizando Image Pyramids

La opción trivial para pegar dos imágenes es tomar una fracción de una de las matrices de una de las imágenes, y utilizarla para sobreescribir otra imagen. Como se ve a continuación, las discontinuidades entre ambas imágenes hacen que el efecto no quede bien logrado:

In [11]:

```
import sys
import os
import numpy as np
import cv2
import scipy
from scipy.stats import norm
from scipy.signal import convolve2d
import math
import numpy as np

rows=1024
cols=1024
blend_pixels=20
blend_array_a=np.ones([rows,cols]) #Primero lleno la imagen de unos para luego s
obreescribir con ceros
blend_array_a[:,int(0.5*cols+blend_pixels):]=np.zeros([rows,int(cols*0.5-blend_p
ixels)])
blend_array_a[:,int(0.5*cols-
blend_pixels):int(0.5*cols+blend_pixels)]=np.dot(np.ones([rows, 1]),
[np.linspace(1,0,blend_pixels*2)])
cv2.imwrite('./mask{}_{}.jpg'.format(cols,blend_pixels), blend_array_a*255)
```

Out[11]:

True

In [12]:

```
'''split rgb image to its channels'''
def split_rgb(image):
    red = None
    green = None
    blue = None
    (blue, green, red) = cv2.split(image)
    return red, green, blue

'''generate a 5x5 kernel'''
def generating_kernel(a):
    w_1d = np.array([0.25 - a/2.0, 0.25, a, 0.25, 0.25 - a/2.0])
    return np.outer(w_1d, w_1d)

'''reduce image by 1/2'''
def ireduce(image):
    out = None
    kernel = generating_kernel(0.4)
    outimage = scipy.signal.convolve2d(image,kernel,'same')
    out = outimage[::2,::2]
    return out

'''expand image by factor of 2'''
def iexpand(image):
    out = None
    kernel = generating_kernel(0.4)
    outimage = np.zeros((image.shape[0]*2, image.shape[1]*2), dtype=np.float64)
    outimage[::2,::2]=image[:,:]
    out = 4*scipy.signal.convolve2d(outimage,kernel,'same')
    return out

'''create a gaussain pyramid of a given image'''
```

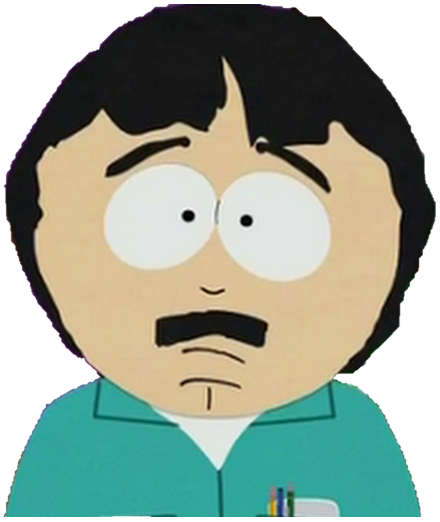
```
def gauss_pyramid(image, levels):
    output = []
    output.append(image)
    tmp = image
    for i in range(0,levels):
        tmp = ireduce(tmp)
        output.append(tmp)
    return output

'''build a laplacian pyramid'''
def lapl_pyramid(gauss_pyr):
    output = []
    k = len(gauss_pyr)
    for i in range(0,k-1):
        gu = gauss_pyr[i]
        egu = iexpand(gauss_pyr[i+1])
        if egu.shape[0] > gu.shape[0]:
            egu = np.delete(egu,(-1),axis=0)
        if egu.shape[1] > gu.shape[1]:
            egu = np.delete(egu,(-1),axis=1)
        output.append(gu - egu)
    output.append(gauss_pyr.pop())
    return output
'''Blend the two laplacian pyramids by weighting them according to the mask.'''
def blend(lapl_pyr_white, lapl_pyr_black, gauss_pyr_mask):
    blended_pyr = []
    k= len(gauss_pyr_mask)
    for i in range(0,k):
        p1= gauss_pyr_mask[i]*lapl_pyr_white[i]
        p2=(1 - gauss_pyr_mask[i])*lapl_pyr_black[i]
        blended_pyr.append(p1 + p2)
    return blended_pyr
'''Reconstruct the image based on its laplacian pyramid.'''
def collapse(lapl_pyr):
    output = None
    output = np.zeros((lapl_pyr[0].shape[0],lapl_pyr[0].shape[1]), dtype=np.floa
t64)
    for i in range(len(lapl_pyr)-1,0,-1):
        lap = iexpand(lapl_pyr[i])
        lapb = lapl_pyr[i-1]
        if lap.shape[0] > lapb.shape[0]:
            lap = np.delete(lap,(-1),axis=0)
        if lap.shape[1] > lapb.shape[1]:
            lap = np.delete(lap,(-1),axis=1)
        tmp = lap + lapb
        lapl_pyr.pop()
        lapl_pyr.pop()
        lapl_pyr.append(tmp)
        output = tmp
    return output

def main():

    image1 = cv2.imread('./Lorde_ya_ya.png')
    image2 = cv2.imread('./randy.png')
    mask = cv2.imread('./mask1024_20.jpg')
#    image1 = cv2.imread('./orange.jpg')
#    image2 = cv2.imread('./apple.jpg')
#    mask = cv2.imread('./mask512_20.jpg')
    r1= None
    g1= None
```

```
    b1= None
    r2= None
    g2= None
    b2= None
    rm= None
    gm = None
    bm = None

    (r1,g1,b1) = split_rgb(image1)
    (r2,g2,b2) = split_rgb(image2)
    (rm,gm,bm) = split_rgb(mask)

    r1 = r1.astype(float)
    g1 = g1.astype(float)
    b1 = b1.astype(float)

    r2 = r2.astype(float)
    g2 = g2.astype(float)
    b2 = b2.astype(float)

    rm = rm.astype(float)/255
    gm = gm.astype(float)/255
    bm = bm.astype(float)/255

    # Automatically figure out the size
    min_size = min(r1.shape)
    depth = int(math.floor(math.log(min_size, 2))) - 4 # at least 16x16 at the h
ighest level.

    gauss_pyr_maskr = gauss_pyramid(rm, depth)
    gauss_pyr_maskg = gauss_pyramid(gm, depth)
    gauss_pyr_maskb = gauss_pyramid(bm, depth)

    gauss_pyr_image1r = gauss_pyramid(r1, depth)
    gauss_pyr_image1g = gauss_pyramid(g1, depth)
    gauss_pyr_image1b = gauss_pyramid(b1, depth)

    gauss_pyr_image2r = gauss_pyramid(r2, depth)
    gauss_pyr_image2g = gauss_pyramid(g2, depth)
    gauss_pyr_image2b = gauss_pyramid(b2, depth)

    lapl_pyr_image1r  = lapl_pyramid(gauss_pyr_image1r)
    lapl_pyr_image1g  = lapl_pyramid(gauss_pyr_image1g)
    lapl_pyr_image1b  = lapl_pyramid(gauss_pyr_image1b)

    lapl_pyr_image2r = lapl_pyramid(gauss_pyr_image2r)
    lapl_pyr_image2g = lapl_pyramid(gauss_pyr_image2g)
    lapl_pyr_image2b = lapl_pyramid(gauss_pyr_image2b)

    outpyrr = blend(lapl_pyr_image2r, lapl_pyr_image1r, gauss_pyr_maskr)
    outpyrg = blend(lapl_pyr_image2g, lapl_pyr_image1g, gauss_pyr_maskg)
    outpyrb = blend(lapl_pyr_image2b, lapl_pyr_image1b, gauss_pyr_maskb)

    outimgr = collapse(blend(lapl_pyr_image2r, lapl_pyr_image1r,
gauss_pyr_maskr))
    outimgg = collapse(blend(lapl_pyr_image2g, lapl_pyr_image1g,
gauss_pyr_maskg))
    outimgb = collapse(blend(lapl_pyr_image2b, lapl_pyr_image1b,
gauss_pyr_maskb))
    # blending sometimes results in slightly out of bound numbers.
    outimgr[outimgr < 0] = 0
```

```
    outimgr[outimgr > 255] = 255
    outimgr = outimgr.astype(np.uint8)

    outimgg[outimgg < 0] = 0
    outimgg[outimgg > 255] = 255
    outimgg = outimgg.astype(np.uint8)

    outimgb[outimgb < 0] = 0
    outimgb[outimgb > 255] = 255
    outimgb = outimgb.astype(np.uint8)

    result = np.zeros(image1.shape,dtype=image1.dtype)
    tmp = []
    tmp.append(outimgb)
    tmp.append(outimgg)
    tmp.append(outimgr)
    result = cv2.merge(tmp,result)
    cv2.imwrite('./blended_SP.jpg', result)

if  __name__ =='__main__':
    main()
```

Blending hecho sin y con Image Pyramids:

Mezclamos:



con

Y obtenemos (sin Image Pyramids):

Utilizando Image Pyramids: