Bizarre Image Fourier Transform Explained with Python

Understand how image Fourier Transform works and implement it with the shortest Python code

Andrew Zhu · Follow
Published in Python in Plain English
6 min read · Feb 21, 2022

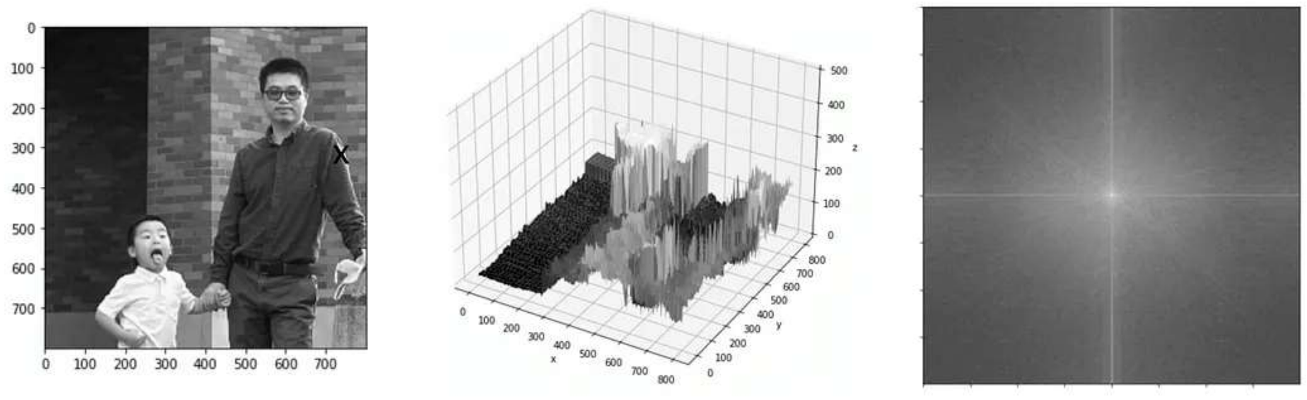


Image 2D Fourier Transform is so useful and powerful, that you will use it sooner or later if you work with Machine Learning or vision automation. Call API or libraries to use it is a piece of cake, fully understanding 2D Discrete Fourier Transform isn't that easy.

In this article, I am trying to explain how Image Fourier Transform works in an intuitive way with short and plain Python code.

Python code is runnable and tested in Jupyter notebook with Python3.8(Ubuntu and Windows 10).

From 1-D Fourier transform to 2-D Image transform

Fourier Transform can break up a complex wave signal into a number of sine waves of various frequencies and amplitudes like the below GIF shows.

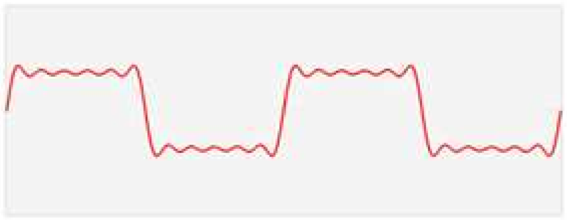


Image from wiki

If you want to know more about 1D Fourier Transform, my previous article may help:

Clean Up Data Noise with Fourier Transform in Python

Use Fourier Transform to clean up time series data in the shortest Python code

towardsdatascience.com

This 1D Fourier Transform works for signals like sound or microwaves, or even some stock prices. The same mechanism can be applied to 2D signals too.

When you think of an image as a 2D signal, Fourier transforms the image into many 2D-**sine waves** (like ripples on the water surface).

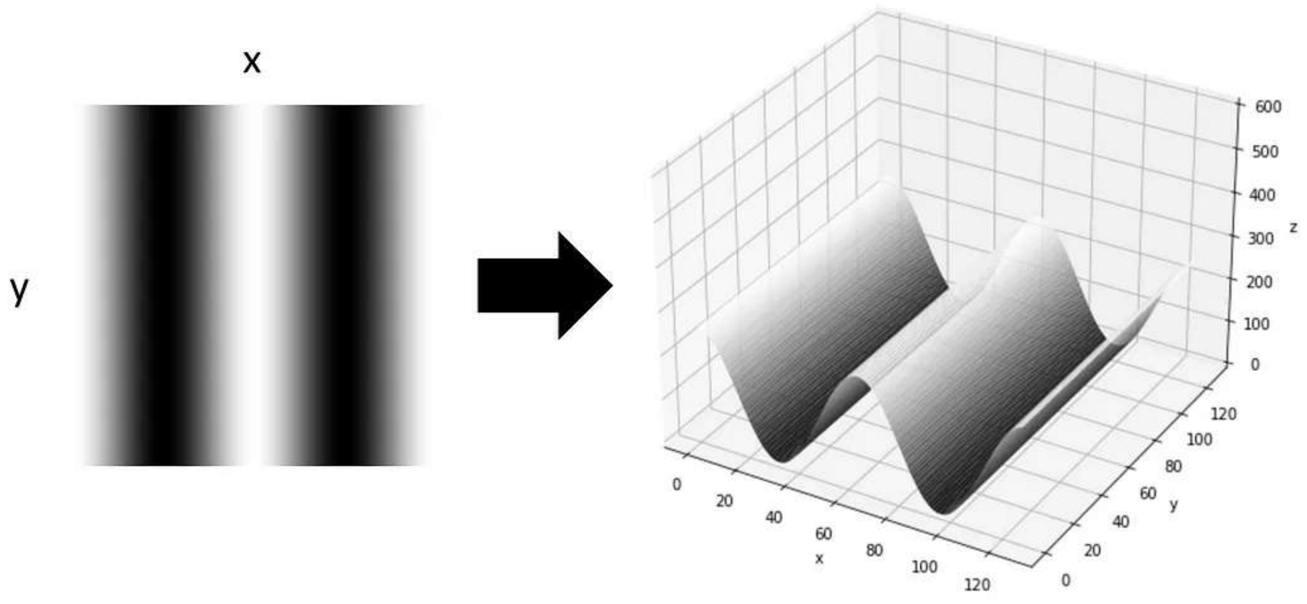
In the 1D Fourier Transform, A composite signal can be broken down into a number of sine waves:

$$y = a\sin(bx + c)$$

While in the 2D image case, an image can be broken down into a number of 2D sine waves:

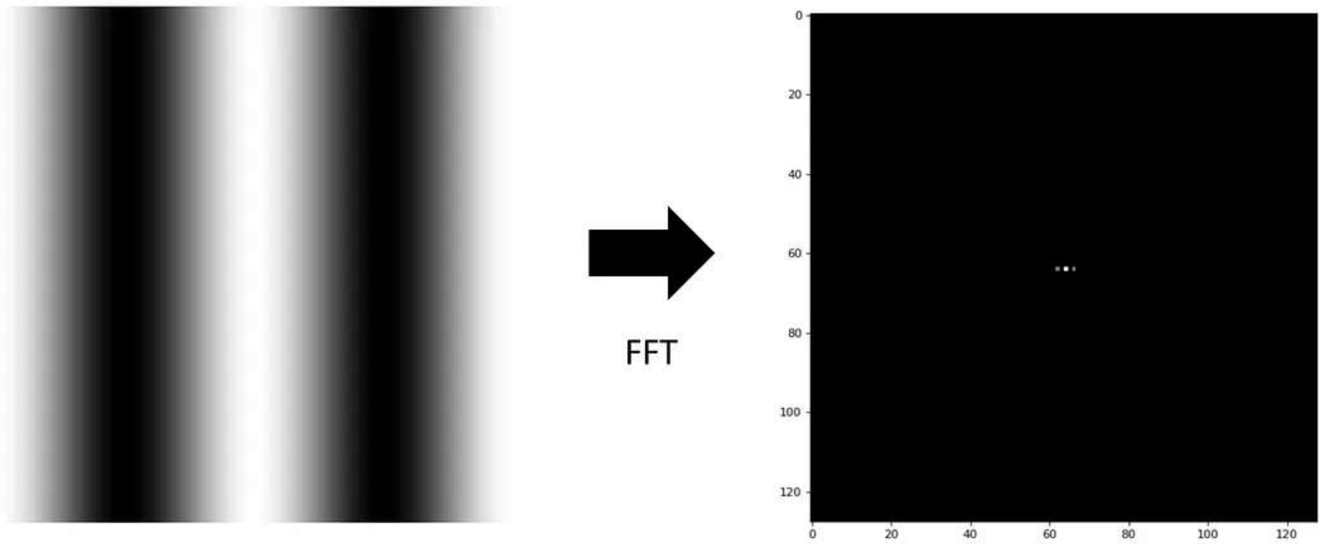
$$z = a\sin(hx + ky + p)$$

For example, consider the following image as a 2D sine wave:



view image as a sine plane

Apply the Fourier Transform to this image, we will have 3 dots in the frequency spectrum.

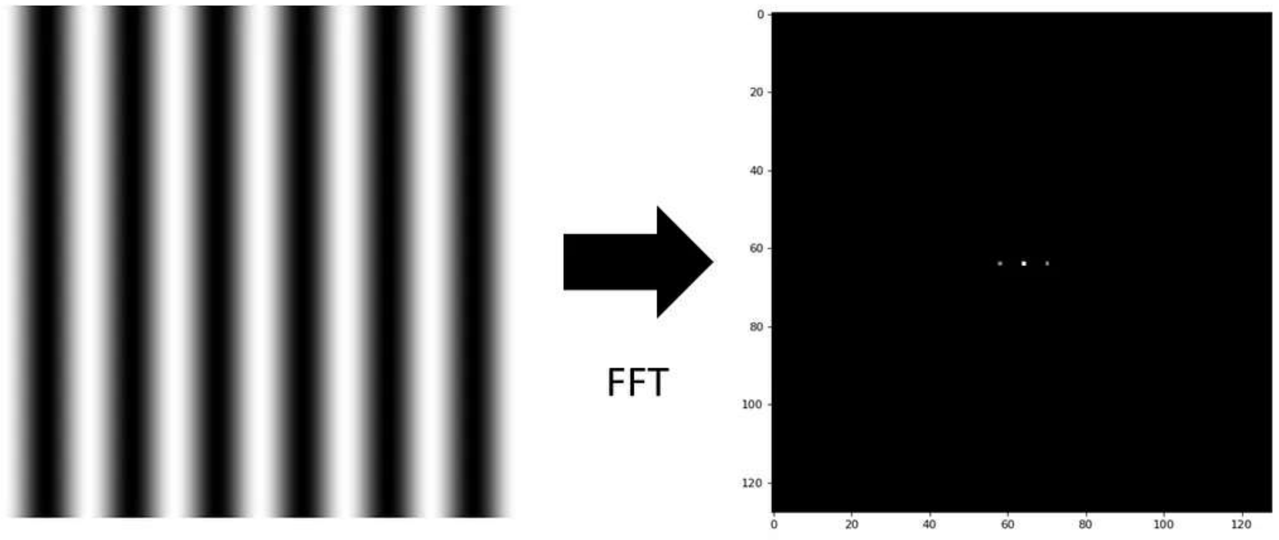


Left: original image: Right: three dots in the Frequence domain

The above sample image is formed by a single 2D sine wave. `a = 255` (grayscale max brightness). `h =1` , `k=0` , `p =0.` `h` and `k` are actually the frequency of x and y.

In the right-side frequency spectrum. The most right dot's position is in (h=1,k=0). The middle one is the center, the most left dot in position (-1,0) is the mirrored dot of the right one. The dot brightness represents the amplitude of the original image.

Let's test another image with a higher wave frequency.



The dot in the frequency domain spread wider because the h (represents the frequency of the sine plane) is higher this time.

Load image with Python

You can load images into Python with many solutions, I found scikit-image is the most useful. Because scikit-image's `io.imread` support not only read data from local disk, and also reach to a remote image with a URL.

```
from skimage import io
from skimage.color import rgb2gray

img = io.imread('/path/to/image.jpg')
#img = io.imread('http://url/to/image.jpg')

gray_img = rgb2gray(img)

io.imshow(gray_img)
```

Before heading to a TF image to its frequency spectrum, we need to be careful with the grayscale image format. What? isn't a greyscale image used `0 ~ 255` to represent a pixel? yes, yes, that is what the textbook told us.

In Python code, grayscale images can be represented in many forms. Take scikit-image for example. Grayscale images can also be represented with float numbers from `0 ~ 1`. Here is the complete list.

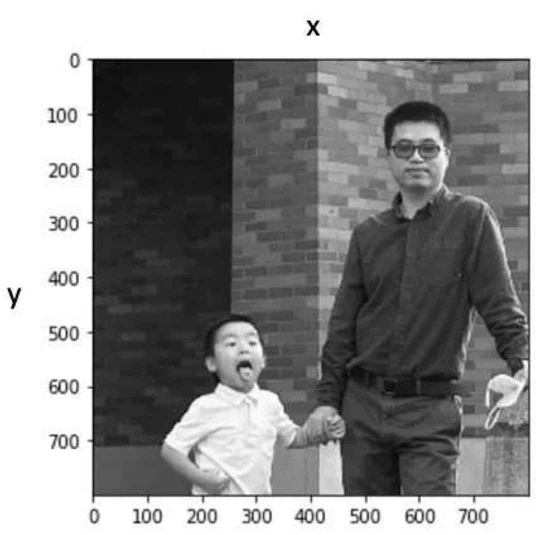
To ensure all grayscale is represented in a unified format, we can transform all formats to a single one, say, the 0 ~ 255 format(one byte per pixel). scikit-image also provides functions to do this.

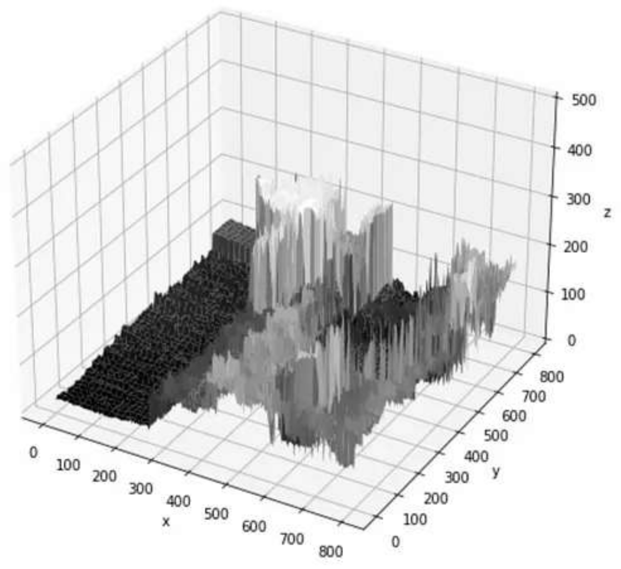
```
from skimage.util import img_as_ubyte
gray_img = img_as_ubyte(rgb2gray(img))
```

This `img_as_ubyte` function will ensure all grayscale images pixel is represented with a number in scope `0 ~ 255`.

Image Fourier Transform with Python

Let me use my Medium avatar as the sample image.



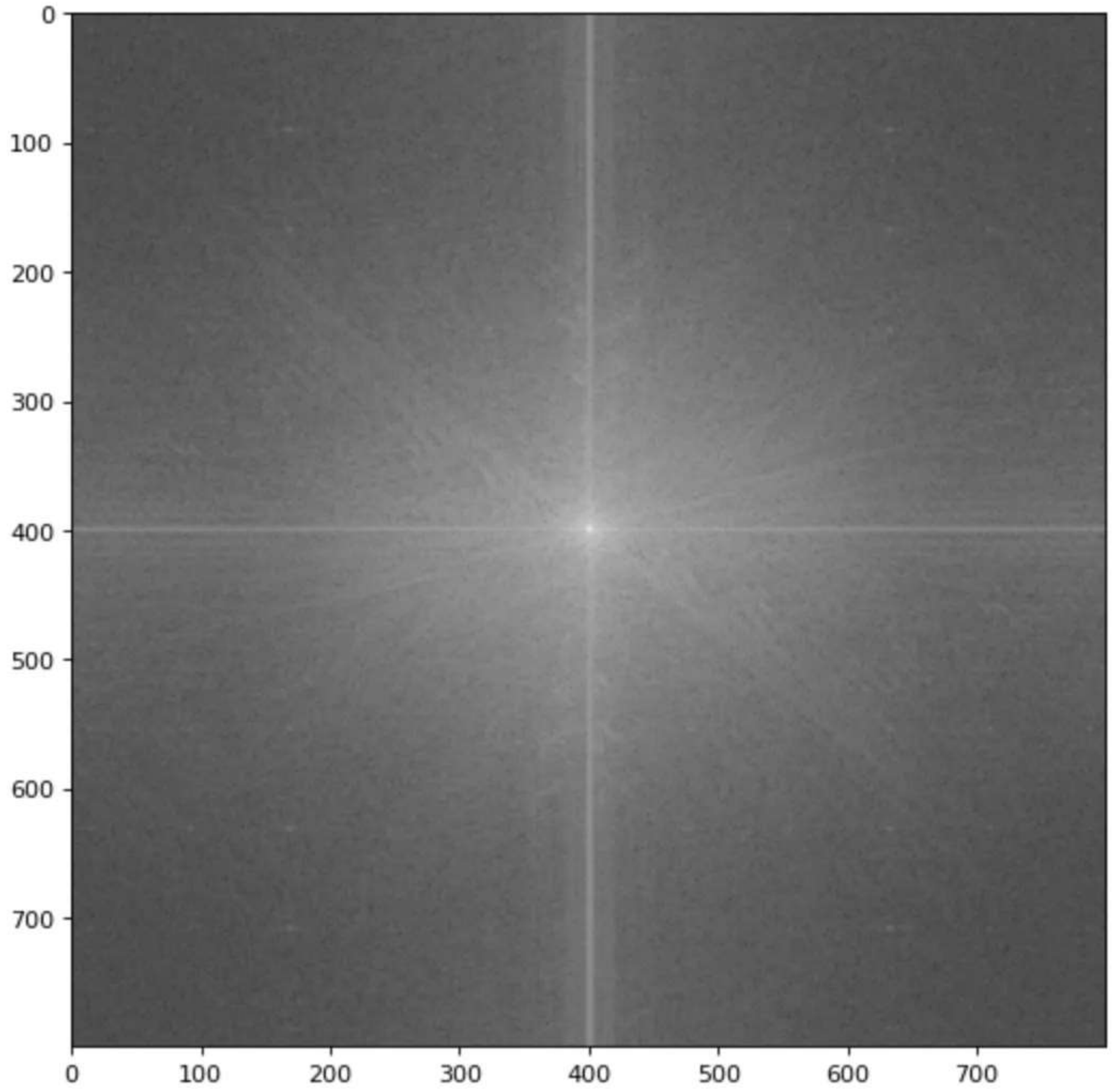


Charles Zhu and Andrew Zhu in U.W. Seattle left: grayscale image; right: 3D pixel distribution

Use Numpy's Fast Fourier Transform function `fft2` :

```
import numpy as np
f = np.fft.fft2(img)
f_s = np.fft.fftshift(f)
plt.figure(num=None, figsize=(10, 8), dpi=80)
plt.imshow(np.log(abs(f_s)), cmap='gray');
```

5 lines of Python code generate the "bizarre" Frequency-domain chart.



Charles & Andrew avatar in the frequency spectrum

The above frequency spectrum is been shifted by `np.fft.fftshift(f)` , the center point is the zero frequency area, frequency increase by spreading out of the axes.

The horizontal lines are sourced from the edges of brick pillars(vertical), the vertical line in the frequency spectrum is sourced from the horizontal lines(bricks) from the grayscale image.

The data in `f` is a 2d complex number array and contains all the information from the original grayscale image, so that we can use the inverse function to transform back the image from frequency spectrum to spatial spectrum(like 1D time domain):

```
img_i = np.fft.irfft2(f,img.shape)
plt.imshow(img_i,cmap='gray')
```

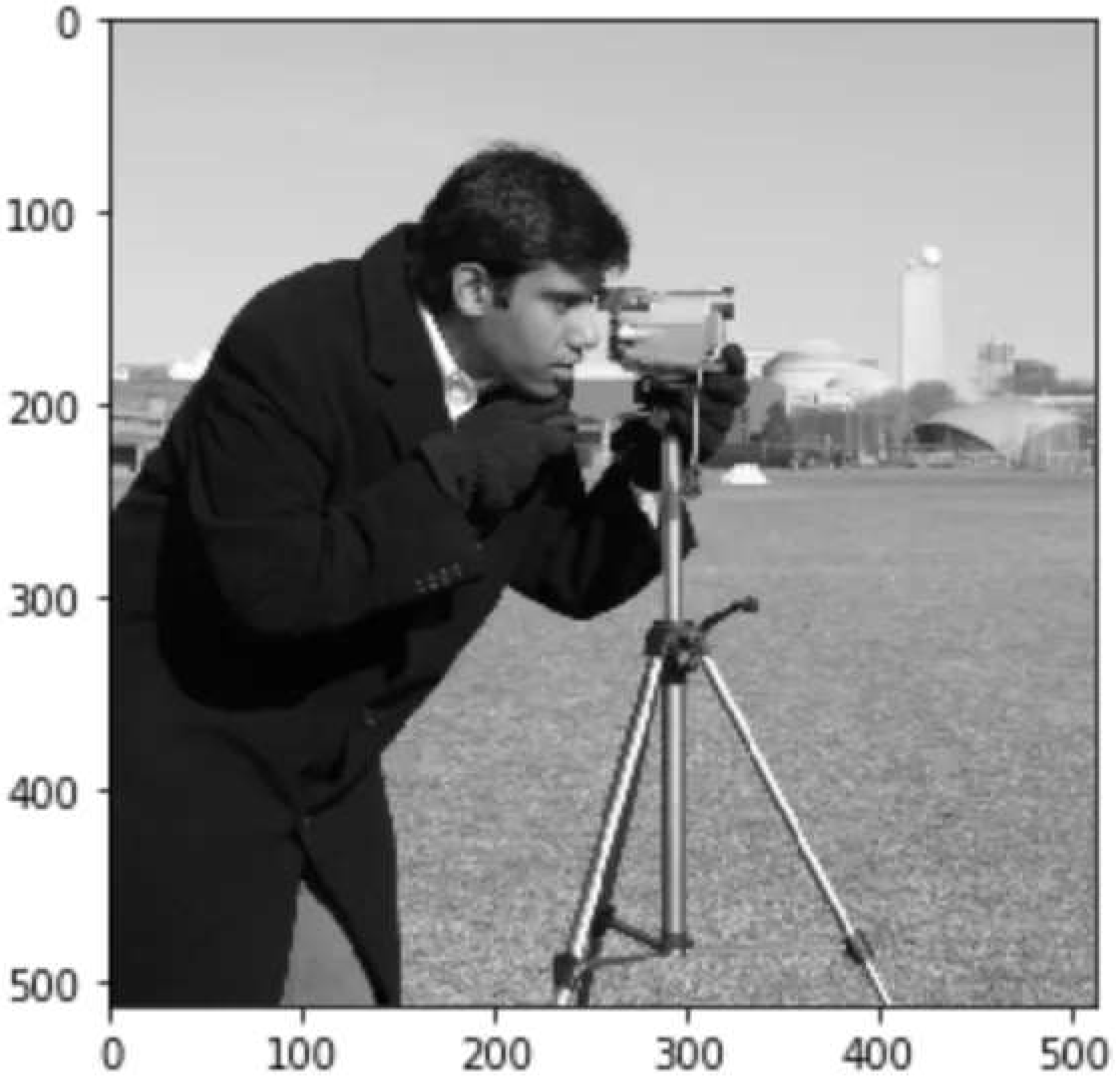
No information is lost in the process of transforming.

Appendix

Load a sample image from scikit-image

In case you want to give the above script a test, and don't download the image, you can use image from scikit-image.

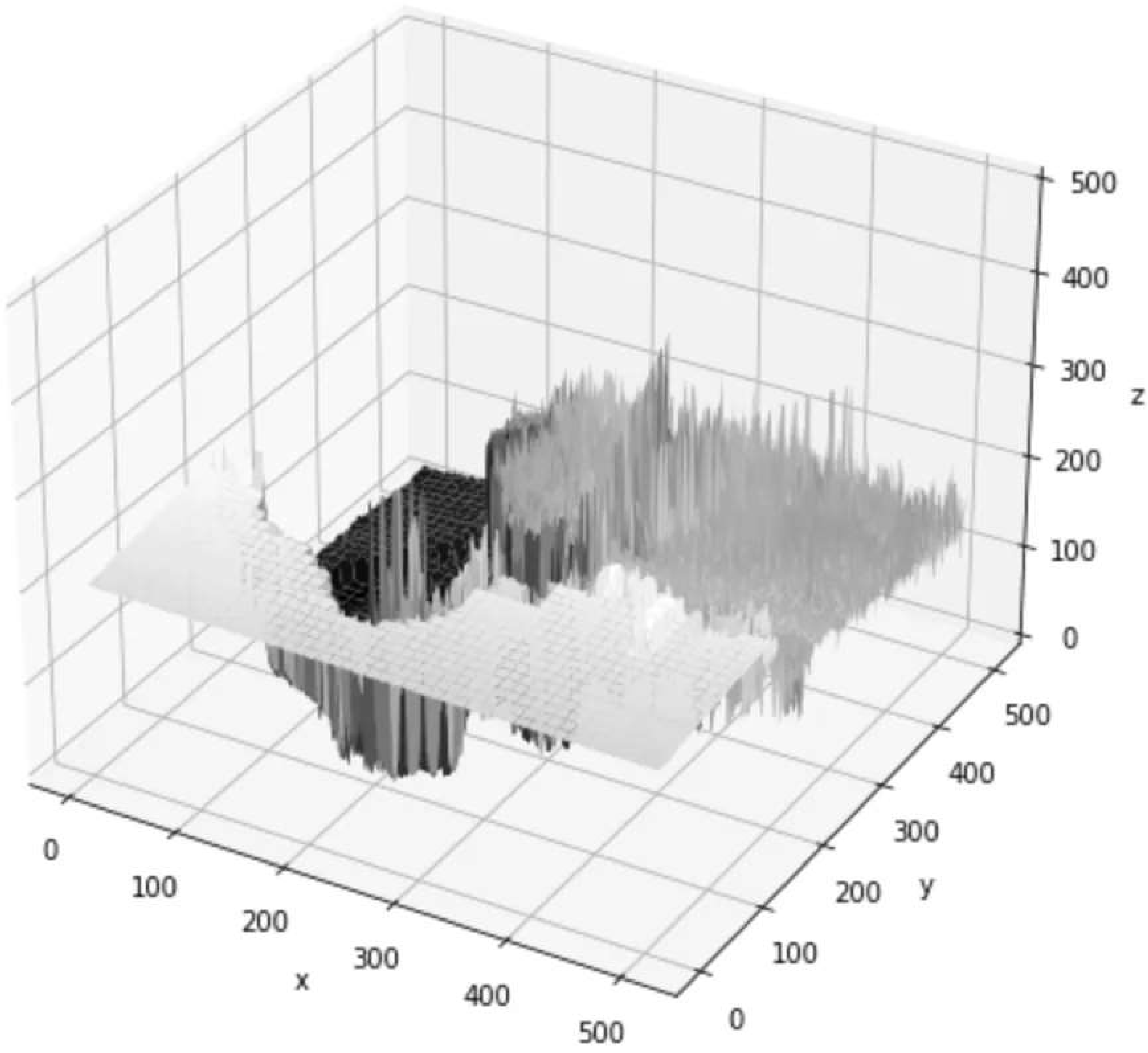
```
from skimage import io,data
from skimage.util import img_as_ubyte,
img = data.camera()
img = img_as_ubyte(img)
img = io.imshow(img)
```



Render an image on 3-D distribution scale

You can use this code to show the grayscale image on a 3D scale.

```
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d
yy,xx = np.mgrid[0:img.shape[0],0:img.shape[1]]
fig   = plt.figure(figsize=(10,8))
ax    = plt.axes(projection='3d')
ax.plot_surface(xx,yy,img,cmap='gray',edgecolor='none')
ax.set_zlim(0,500)
```



For more about matplotlib 3d chart rendering, I recommend this great article: Three-Dimensional Plotting in Matplotlib.

References

1. Grayscale:

Grayscale - Wikipedia

In digital photography, computer-generated imagery, and colorimetry, a grayscale image is one in which the value of…

en.wikipedia.org

2. Image data types and what they mean:

Image data types and what they mean - skimage v0.20.0.dev0 docs

In , images are simply numpy arrays, which support a variety of data types 1, i.e. "dtypes". To avoid distorting image...

scikit-image.org

3. Three-Dimensional Plotting in Matplotlib:

Three-Dimensional Plotting in Matplotlib

Now from this parametrization, we must determine the (x, y, z) positions of the embedded strip. Thinking about it, we...

jakevdp.github.io

4. Chapter 24: **Linear Image Processing** from the book **The Scientist and Engineer's Guide to Digital Signal Processing** by By **Steven W. Smith**, Ph.D.

Linear Image Processing

Wouldn't you rather have a bound book instead of 640 loose pages? Your laser printer will thank you! Order from...

www.dspguide.com

5. Fourier transforms of images:

Fourier transforms of images

The sounds we hear - whether music, speech, or background noise - are the result of vibrations of our ear drum...

plus.maths.org

If you have any questions, leave a comment and I will do my best to answer, If you spot an error or mistake, don't hesitate to mark them out. Thanks for reading.

*More content at **plainenglish.io**. Sign up for our **free weekly newsletter**. Get exclusive access to writing opportunities and advice in our **community Discord**.*

Machine Learning Python Image Processing Programming Data Science

Open in app ↗ Sign up Sign In

Search Medium

Follow

Written by Andrew Zhu

669 Followers · Writer for Python in Plain English

Data Scientist @ Microsoft | https://github.com/xhinker | https://twitter.com/xhinker | https://www.linkedin.com/in/andrew-zhu-23407223/

More from Andrew Zhu and Python in Plain English