full_script

September 14, 2019

1 Aluna: Andressa Gabrielly Macedo Marçal

1.1 RA: 262878

1.1.1 MO443/MC920

1.1.2 imports

[1]:
```
import cv2
import numpy as np
import PIL.Image
```

1.1.3 seta pixel; imagem e shape

[2]:
```
def set_pixel(im, x, y, new):
    im[x, y] = new
```

2 Floyd e Steinberg

[4]:
```
def floyd(im):  # Metodo Floyd-Steinberg

    w7 = 7/16.0
    w3 = 3/16.0
    w5 = 5/16.0
    w1 = 1/16.0

    for y in range(0, height-1):
        for x in range(1, width-1):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
```

```
            set_pixel(im, x+1, y, im[x+1, y]+quant_err * w7)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + quant_err * w3)
            set_pixel(im, x, y+1, im[x, y+1] + quant_err * w5)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + quant_err * w1)

    return im
```

3 Stevenson e Arce

[5]:
```
def stevenson(im):  # Metodo de Stevenson e Arce

    w32 = 32/200.0
    w12 = 12/200.0
    w26 = 26/200.0
    w30 = 30/200.0
    w16 = 16/200.0
    w5 = 5/200.0

    #pega o shape da imagem
    width, height = im.shape

    for y in range(0, height-2):
        for x in range(0, width-2):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
            set_pixel(im, x+1, y, im[x+1, y] + w32 * quant_err)
            set_pixel(im, x+2, y, im[x+2, y] + w12 * quant_err)
            set_pixel(im, x-2, y+1, im[x-2, y+1] + w26 * quant_err)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + w30 * quant_err)
            set_pixel(im, x, y+1, im[x, y+1] + w16 * quant_err)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + w12 * quant_err)
            set_pixel(im, x+2, y+1, im[x+2, y+1] + w26 * quant_err)
            set_pixel(im, x-2, y+2, im[x-2, y+2] + w12 * quant_err)
            set_pixel(im, x-1, y+2, im[x-1, y+2] + w5 * quant_err)
            set_pixel(im, x, y+2, im[x, y+2] + w12 * quant_err)
            set_pixel(im, x+1, y+2, im[x+1, y+2] + w12 * quant_err)
            set_pixel(im, x+2, y+2, im[x+2, y+2] + w5 * quant_err)
    return im
```

4 Burkes

[6]:
```
def burkes(im):  # Metodo de Burkes

    w8 = 8/32.0
    w4 = 4/32.0
    w2 = 2/32.0

    width, height = im.shape

    for y in range(0, height-2):
        for x in range(0, width-2):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
            set_pixel(im, x+1, y, im[x+1, y] + w8 * quant_err)
            set_pixel(im, x+2, y, im[x+2, y] + w4 * quant_err)
            set_pixel(im, x-2, y+1, im[x-2, y+1] + w2 * quant_err)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + w4 * quant_err)
            set_pixel(im, x, y+1, im[x, y+1] + w8 * quant_err)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + w4 * quant_err)
            set_pixel(im, x+2, y+1, im[x+2, y+1] + w2 * quant_err)
```

5 Sierra

[7]:
```
def sierra(im):   # Metodo de Sierra

    w5 = 5/32.0
    w4 = 4/32.0
    w3 = 3/32.0
    w2 = 2/32.0
    width, height = im.shape

    for y in range(0, height-2):
        for x in range(0, width-2):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
```

```
            set_pixel(im, x+1, y, im[x+1, y] + w5 * quant_err)
            set_pixel(im, x+2, y, im[x+2, y] + w3 * quant_err)
            set_pixel(im, x-2, y+1, im[x-2, y+1] + w2 * quant_err)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + w4 * quant_err)
            set_pixel(im, x, y+1, im[x, y+1] + w5 * quant_err)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + w4 * quant_err)
            set_pixel(im, x+2, y+1, im[x+2, y+1] + w2 * quant_err)
            set_pixel(im, x-2, y+2, im[x-2, y+2] + w2 * quant_err)
            set_pixel(im, x-1, y+2, im[x-1, y+2] + w3 * quant_err)
            set_pixel(im, x, y+2, im[x, y+2] + w2 * quant_err)
    return im
```

6 Stucki

[8]:
```
def stucki(im):   # Metodo de Stucki

    w8 = 8/42.0
    w7 = 7/42.0
    w5 = 5/42.0
    w4 = 4/42.0
    w2 = 2/42.0
    w1 = 1/42.0
    width, height = im.shape

    for y in range(0, height-2):
        for x in range(0, width-2):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
            set_pixel(im, x+1, y, im[x+1, y] + w7 * quant_err)
            set_pixel(im, x+2, y, im[x+2, y] + w5 * quant_err)
            set_pixel(im, x-2, y+1, im[x-2, y+1] + w2 * quant_err)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + w4 * quant_err)
            set_pixel(im, x, y+1, im[x, y+1] + w8 * quant_err)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + w4 * quant_err)
            set_pixel(im, x+2, y+1, im[x+2, y+1] + w2 * quant_err)
            set_pixel(im, x-2, y+2, im[x-2, y+2] + w1 * quant_err)
            set_pixel(im, x-1, y+2, im[x-1, y+2] + w2 * quant_err)
            set_pixel(im, x, y+2, im[x, y+2] + w4 * quant_err)
            set_pixel(im, x+1, y+2, im[x+1, y+2] + w2 * quant_err)
            set_pixel(im, x+2, y+2, im[x+2, y+2] + w1 * quant_err)
    return im
```

7 Jarvis, Judice e Ninke

[9]:
```
def jarvis(im):   # Metodo de Jarvis, Judice e Ninke

    w1 = 1/48.0
    w3 = 3/48.0
    w5 = 5/48.0
    w7 = 7/48.0

    width, height = im.shape

    for y in range(0, height-2):
        for x in range(0, width-2):
            old_pixel = im[x, y]
            if old_pixel < 128:
                new_pixel = 0
            else:
                new_pixel = 255
            set_pixel(im, x, y, new_pixel)
            quant_err = old_pixel-new_pixel
            set_pixel(im, x+1, y, im[x+1, y] + w7 * quant_err)
            set_pixel(im, x+2, y, im[x+2, y] + w5 * quant_err)
            set_pixel(im, x-2, y+1, im[x-2, y+1] + w3 * quant_err)
            set_pixel(im, x-1, y+1, im[x-1, y+1] + w5 * quant_err)
            set_pixel(im, x, y+1, im[x, y+1] + w7 * quant_err)
            set_pixel(im, x+1, y+1, im[x+1, y+1] + w5 * quant_err)
            set_pixel(im, x+2, y+1, im[x+2, y+1] + w3 * quant_err)
            set_pixel(im, x-2, y+2, im[x-2, y+2] + w1 * quant_err)
            set_pixel(im, x-1, y+2, im[x-1, y+2] + w3 * quant_err)
            set_pixel(im, x, y+2, im[x, y+2] + w5 * quant_err)
            set_pixel(im, x+1, y+2, im[x+1, y+2] + w3 * quant_err)
            set_pixel(im, x+2, y+2, im[x+2, y+2] + w1 * quant_err)
    return im
```

7.1 Transformação de [RGB] e [GRAY] do Algoritmo [Floyd e Steinberg]

[10]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza
```

```
blue = img[:, :, 0]  # pega o canal azul
blue = floyd(blue)

green = img[:, :, 1] #pega canal verde
green = floyd(green)

red = img[:, :, 2] #pega canal vermelho
red = floyd(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = floyd(gray) #aplica filtro com pontilhado em tons de cinza
```

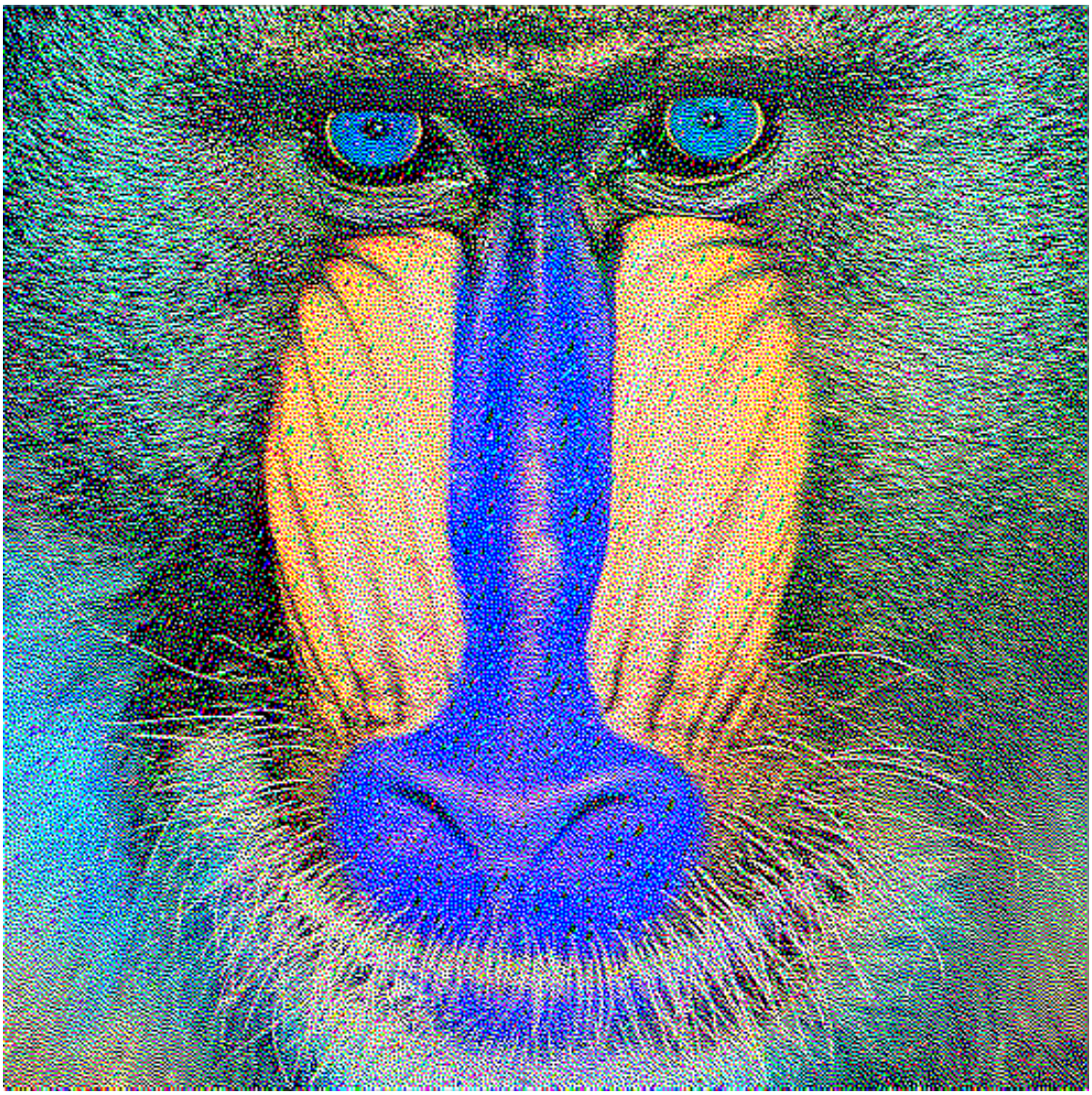
```
(512, 512, 3)
```

7.1.1 [EXIBIÇÃO]

FLOYD - GRAY

[11]:
```
PIL.Image.fromarray(image)
```

[11]:

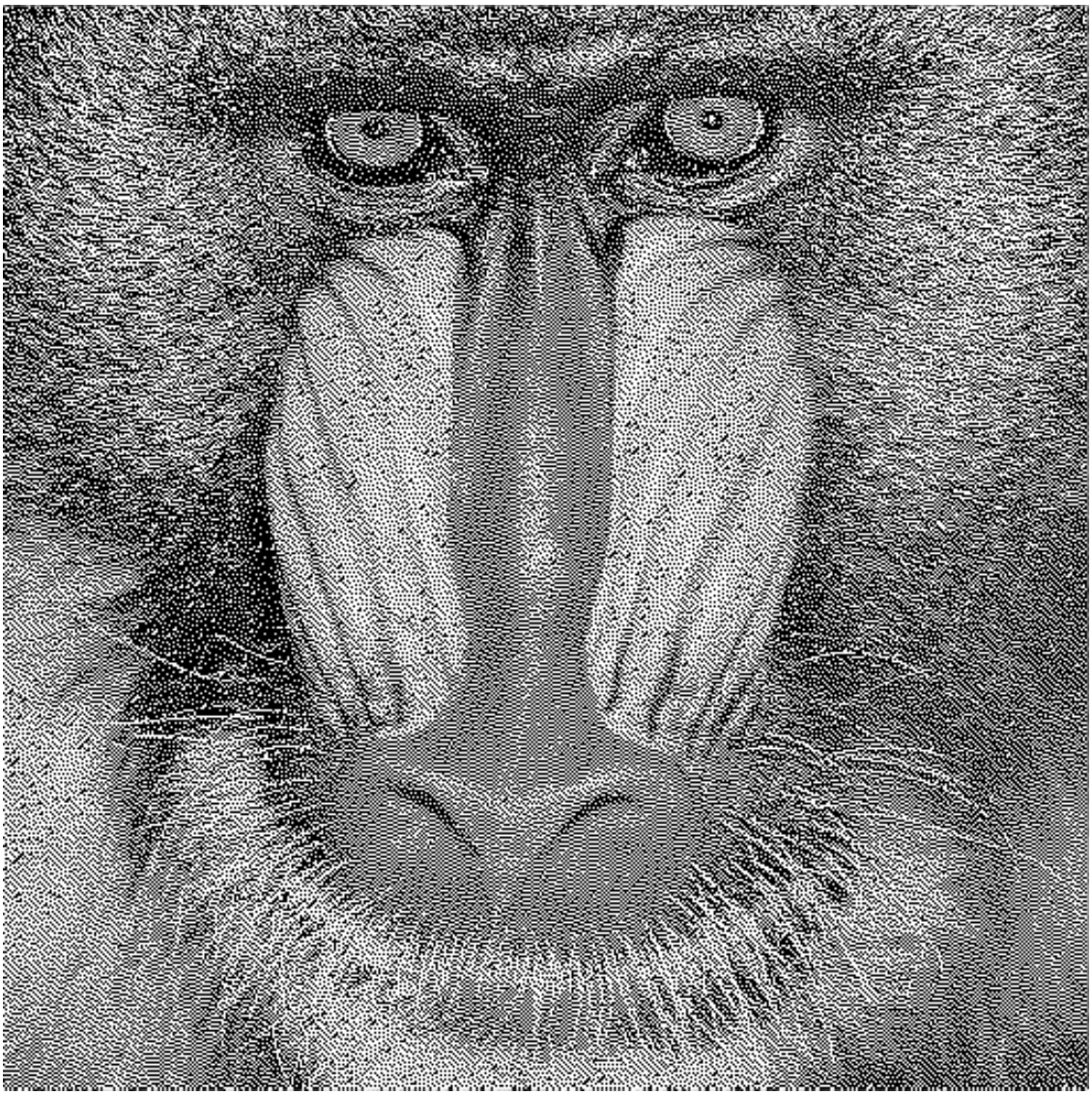


7.1.2 [EXIBIÇÃO]

FLOYD - RGB

[12]:
```
PIL.Image.fromarray(gray2)
```

[12]:



8 OBS:

8.0.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato.

8.1 Transformação de [RGB] e [GRAY] do Algoritmo [Stevenson e Arce]

[13]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
```

```
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza

blue = img[:, :, 0]  # pega o canal azul
blue = stevenson(blue)

green = img[:, :, 1] #pega canal verde
green = stevenson(green)

red = img[:, :, 2] #pega canal vermelho
red = stevenson(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = stevenson(gray) #aplica filtro com pontilhado em tons de cinza
```

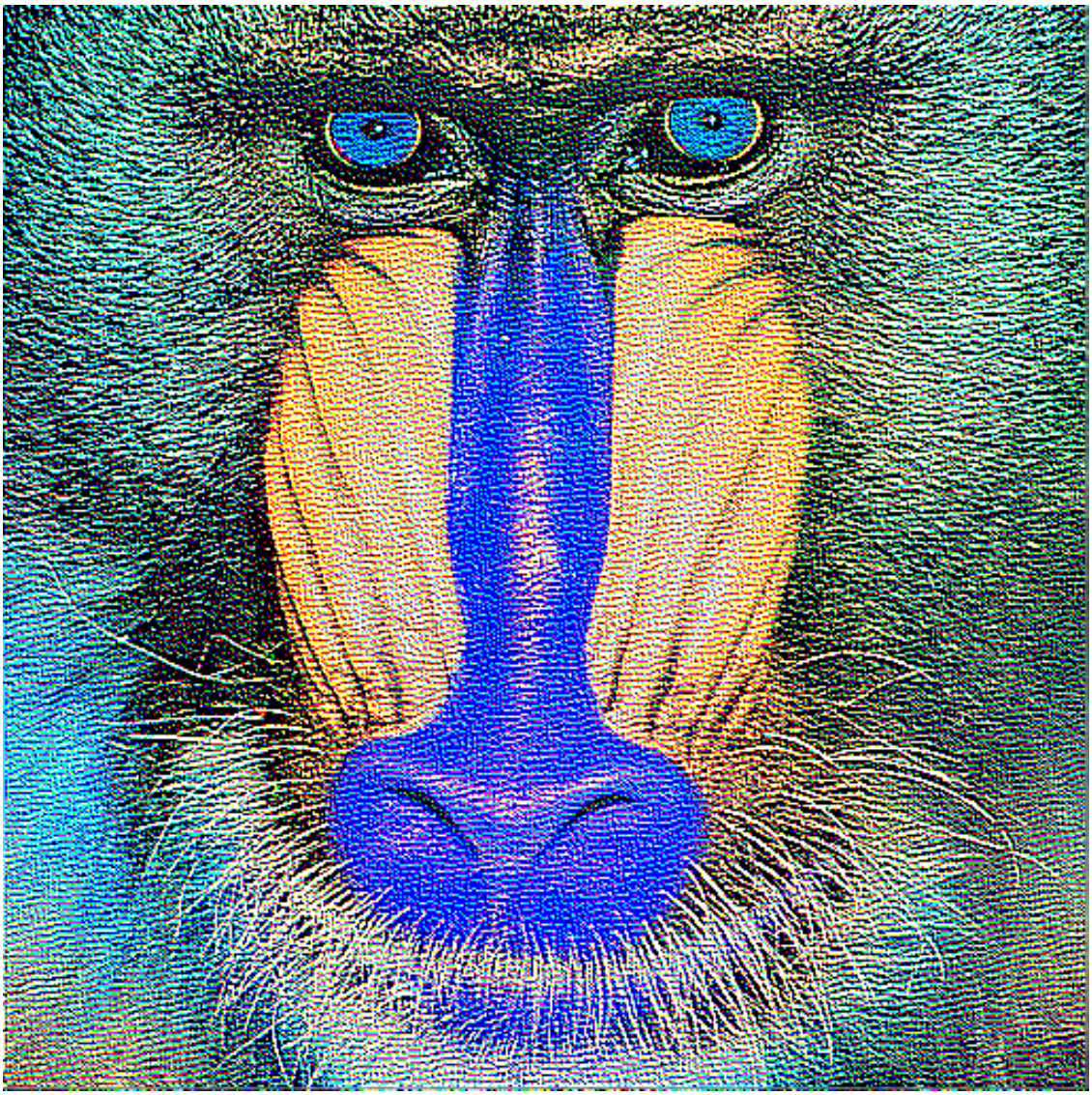
```
(512, 512, 3)
```

8.1.1 [EXIBIÇÃO]

STEVENSON - RBG

[14]:
```
PIL.Image.fromarray(image)
```

[14]:

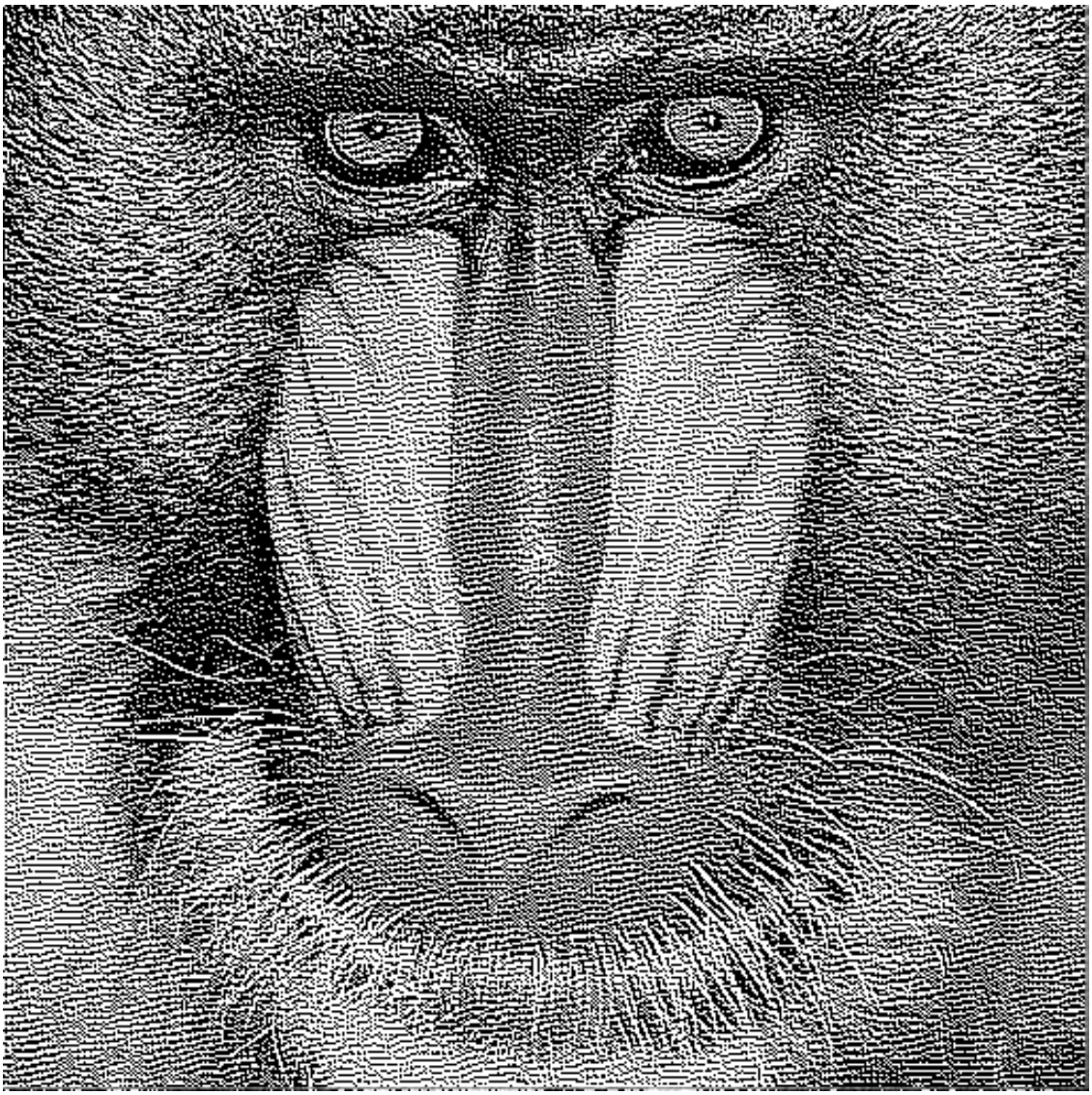


8.1.2 [EXIBIÇÃO]

STEVENSON - GRAY

[15]:
```
PIL.Image.fromarray(gray2)
```

[15]:



9 OBS:

9.0.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato.

9.1 Transformação de [RGB] e [GRAY] do [Algoritmo Burkes]

[16]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
```

```
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza

blue = img[:, :, 0]  # pega o canal azul
blue = burkes(blue)

green = img[:, :, 1] #pega canal verde
green = burkes(green)

red = img[:, :, 2] #pega canal vermelho
red = burkes(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = burkes(gray) #aplica filtro com pontilhado em tons de cinza
```

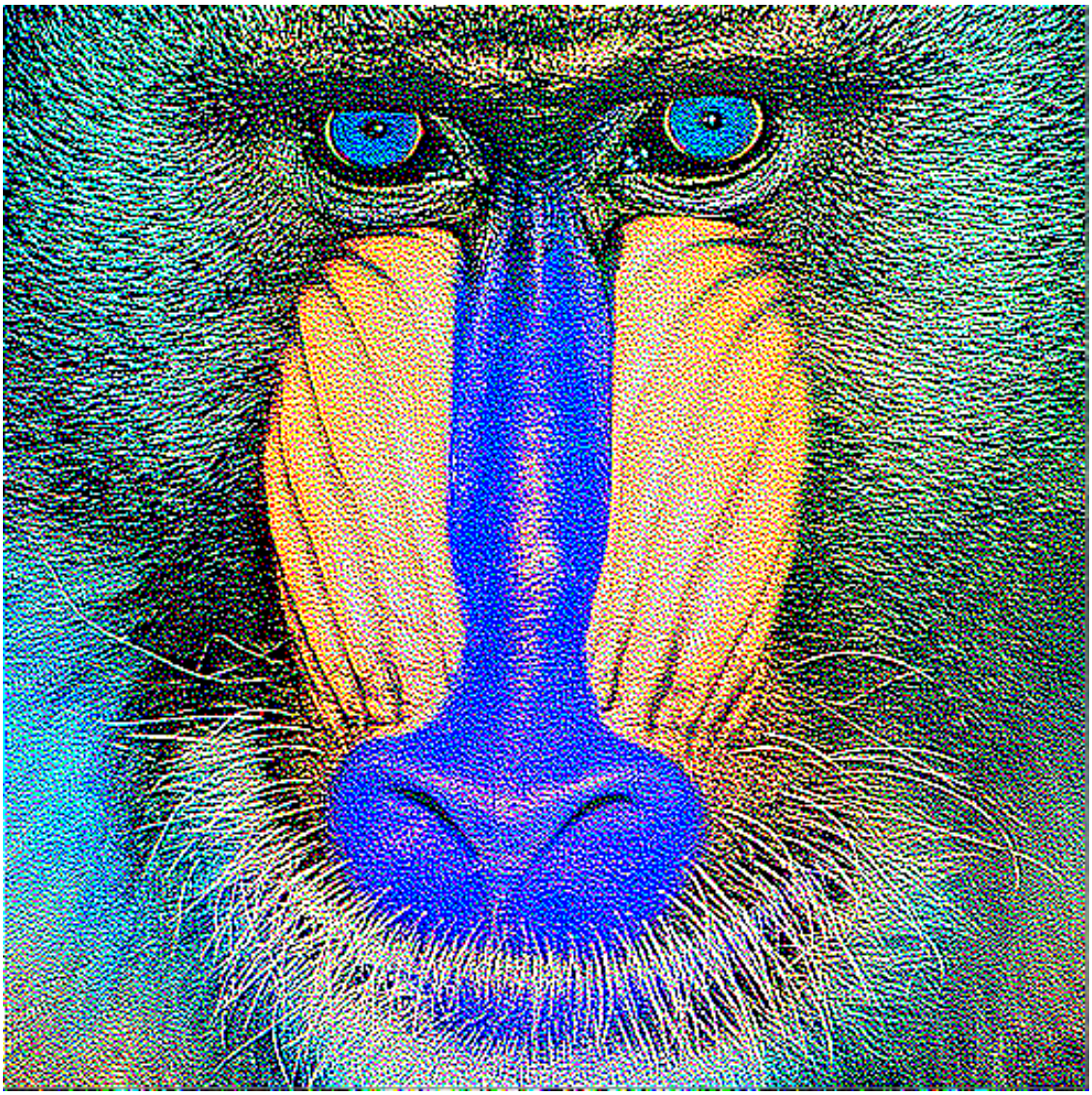
```
(512, 512, 3)
```

9.1.1 [EXIBIÇÃO]

BURKES - RGB

[56]:
```
PIL.Image.fromarray(image)
```

[56]:

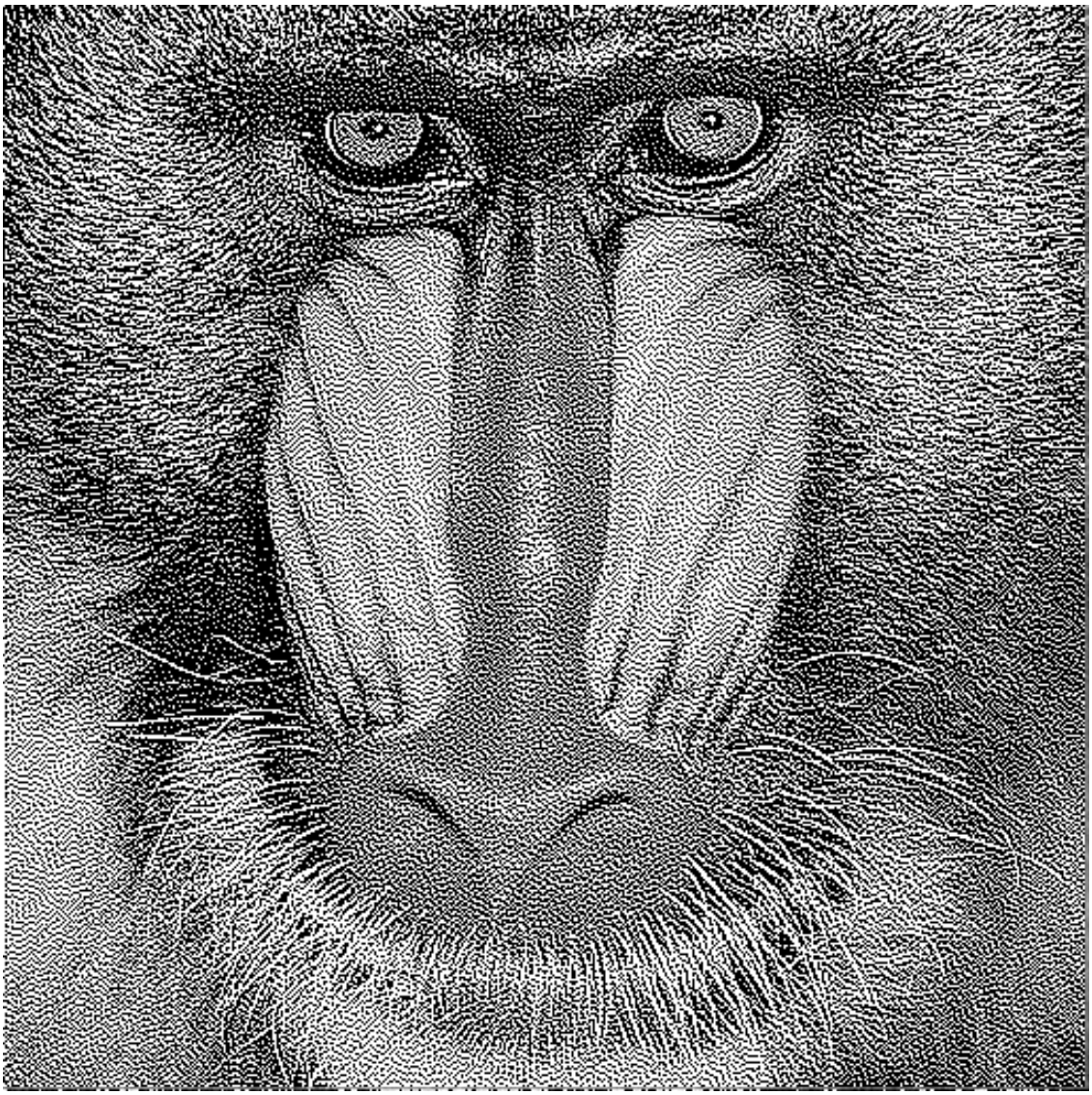


9.1.2 [EXIBIÇÃO]

BURKES - GRAY

[61]:
```
PIL.Image.fromarray(gray2)
```

[61]:



---

10 OBS:

10.0.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato.

10.1 Transformação de [RGB] e [GRAY] do Algoritmo [Sierra]

[20]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza

blue = img[:, :, 0]  # pega o canal azul
blue = sierra(blue)

green = img[:, :, 1] #pega canal verde
green = sierra(green)

red = img[:, :, 2] #pega canal vermelho
red = sierra(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = sierra(gray) #aplica filtro com pontilhado em tons de cinza
```

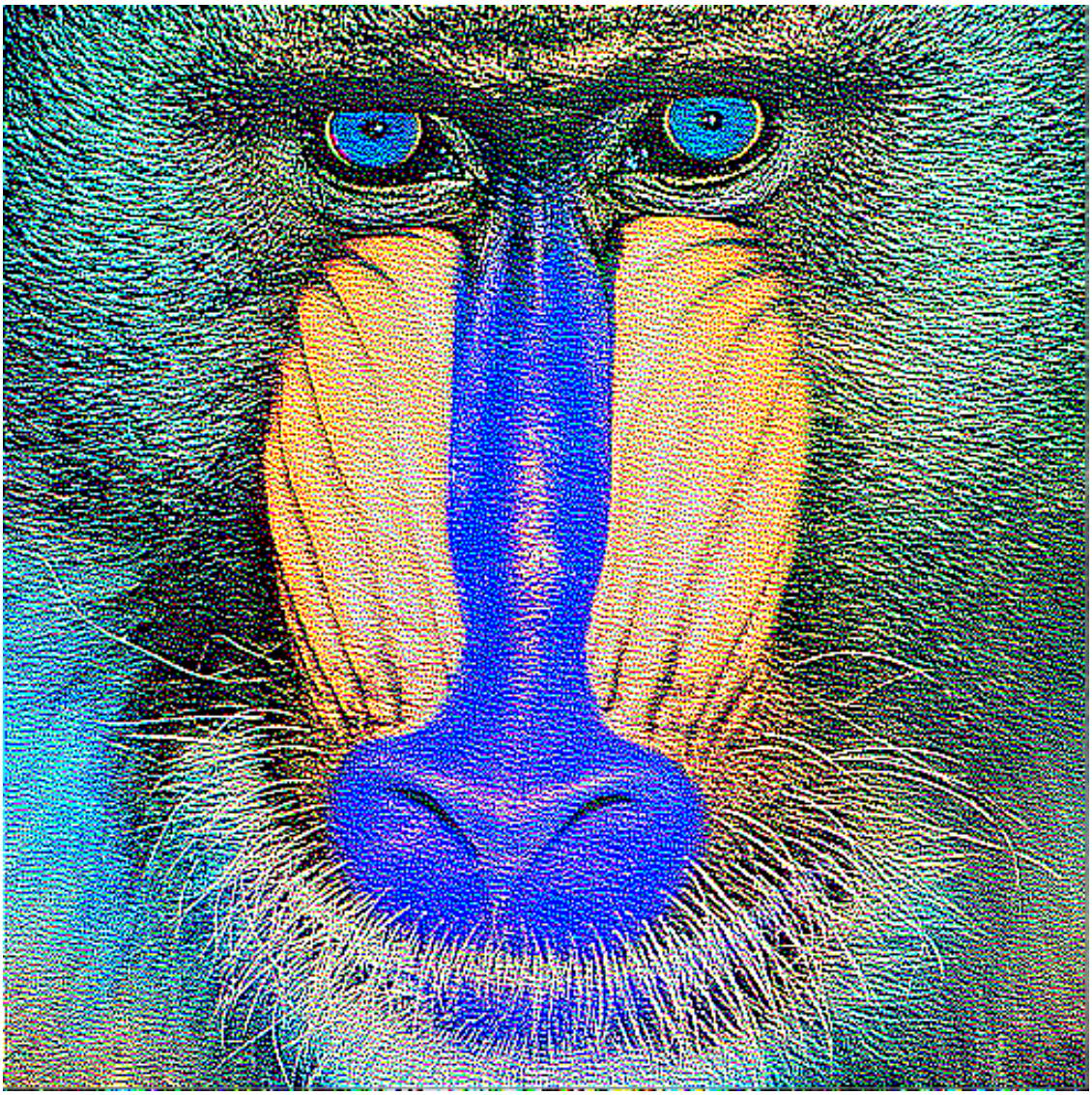
```
(512, 512, 3)
```

10.1.1 [EXIBIÇÃO]

SIERRA - RGB

[21]:
```
PIL.Image.fromarray(image)
```

[21]:

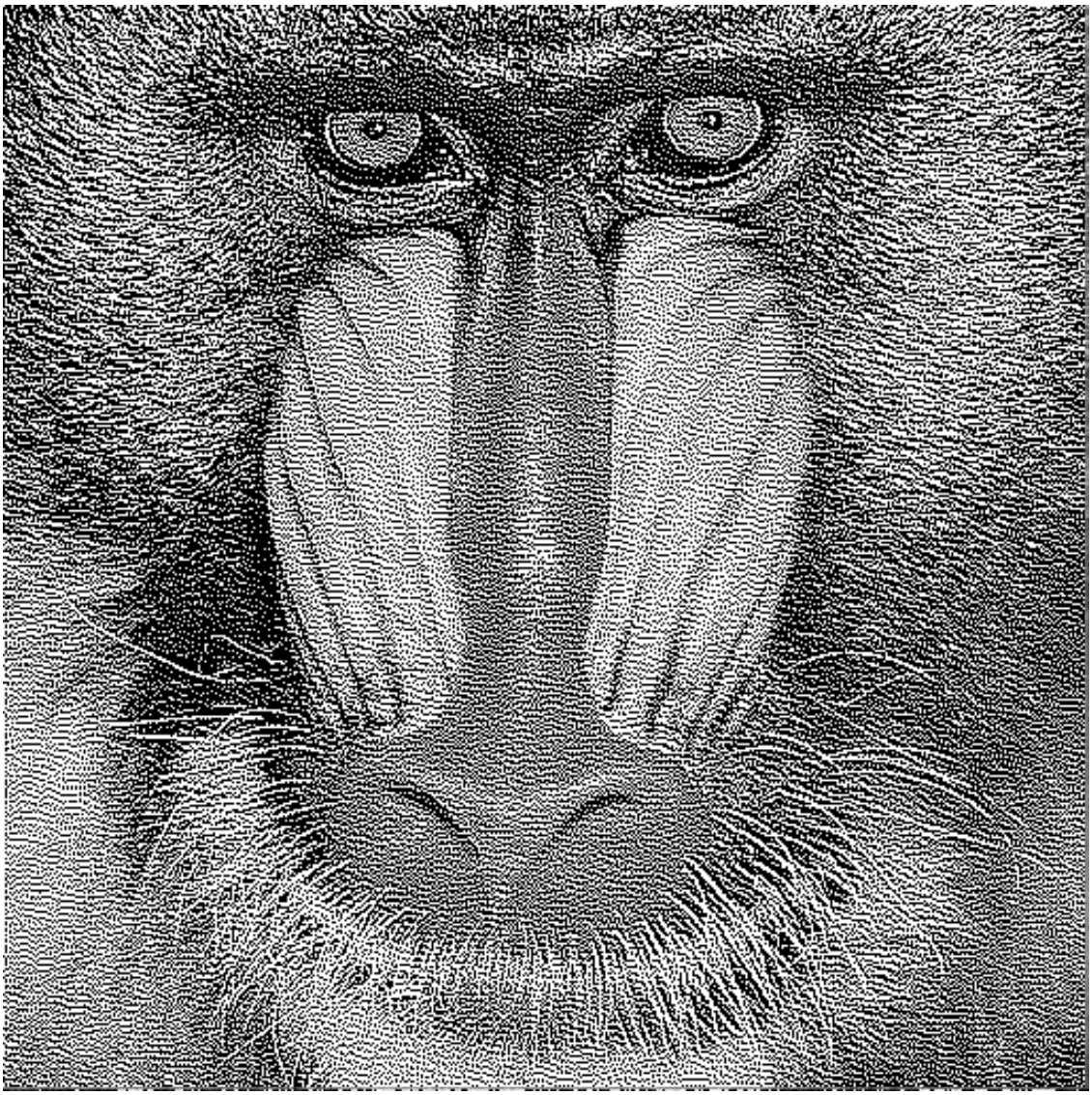


10.1.2 [EXIBIÇÃO]

SIERRA - GRAY

[22]:
```
PIL.Image.fromarray(gray2)
```

[22]:



---

11 OBS:

11.0.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato.

11.1 Transformação de [RGB] e [GRAY] do Algoritmo [Stucki]

[50]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza

blue = img[:, :, 0]  # pega o canal azul
blue = stucki(blue)

green = img[:, :, 1] #pega canal verde
green = stucki(green)

red = img[:, :, 2] #pega canal vermelho
red = stucki(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = stucki(gray) #aplica filtro com pontilhado em tons de cinza
```

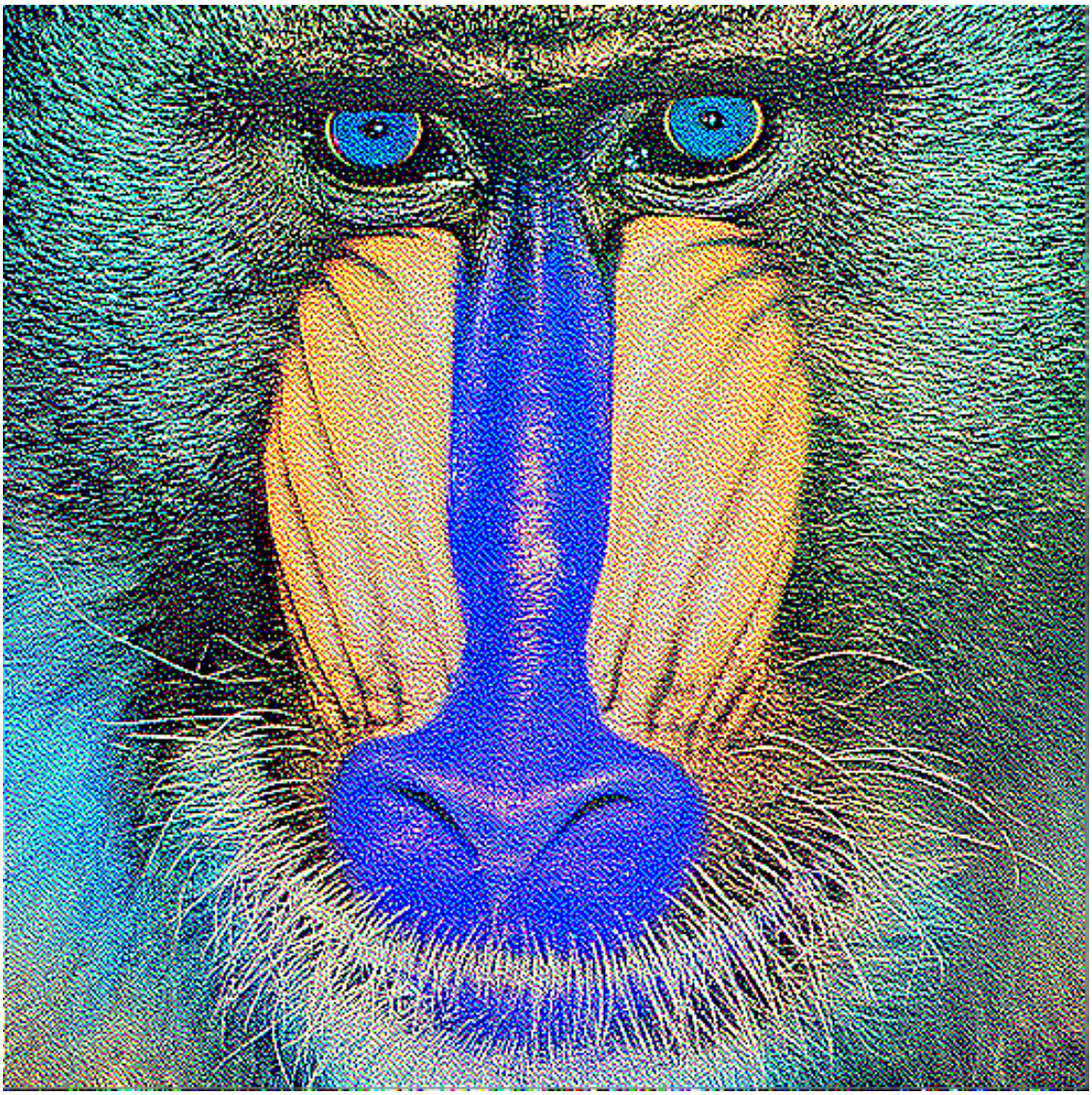
```
(512, 512, 3)
```

11.1.1 [EXIBIÇÃO]

STUCKI - RGB

[51]:
```
PIL.Image.fromarray(image)
```

[51]:

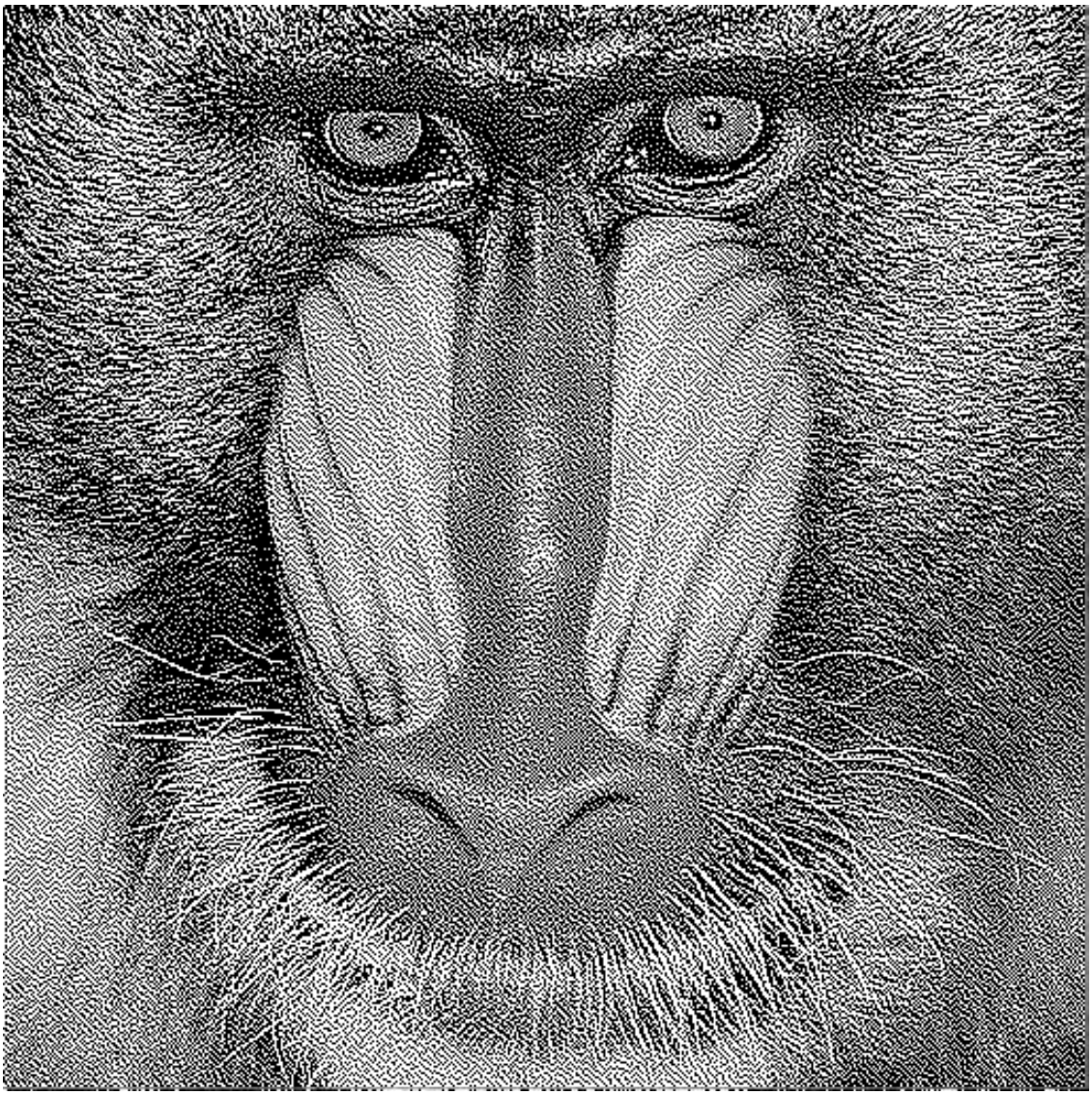


11.1.2 [EXIBIÇÃO]

STUCKI - GRAY

[52]:
```
PIL.Image.fromarray(gray2)
```

[52]:



---

12 OBS:

12.0.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato.

12.1 Transformação de [RGB] e [GRAY] do Algoritmo [Jarvis]

[62]:
```
# adicione o caminho para acessar a imagem
img = cv2.imread("/home/andressa/Documentos/testes/input/baboon.png")

img2 = img.copy()
width, height, z = img.shape
print(img.shape)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #transforma em tons de cinza

blue = img[:, :, 0]  # pega o canal azul
blue = jarvis(blue)

green = img[:, :, 1] #pega canal verde
green = jarvis(green)

red = img[:, :, 2] #pega canal vermelho
red = jarvis(red)

image = cv2.merge((blue, green, red))  #mesclando os 3 canais de cores (R,G,B)

gray2 = jarvis(gray) #aplica filtro com pontilhado em tons de cinza
```

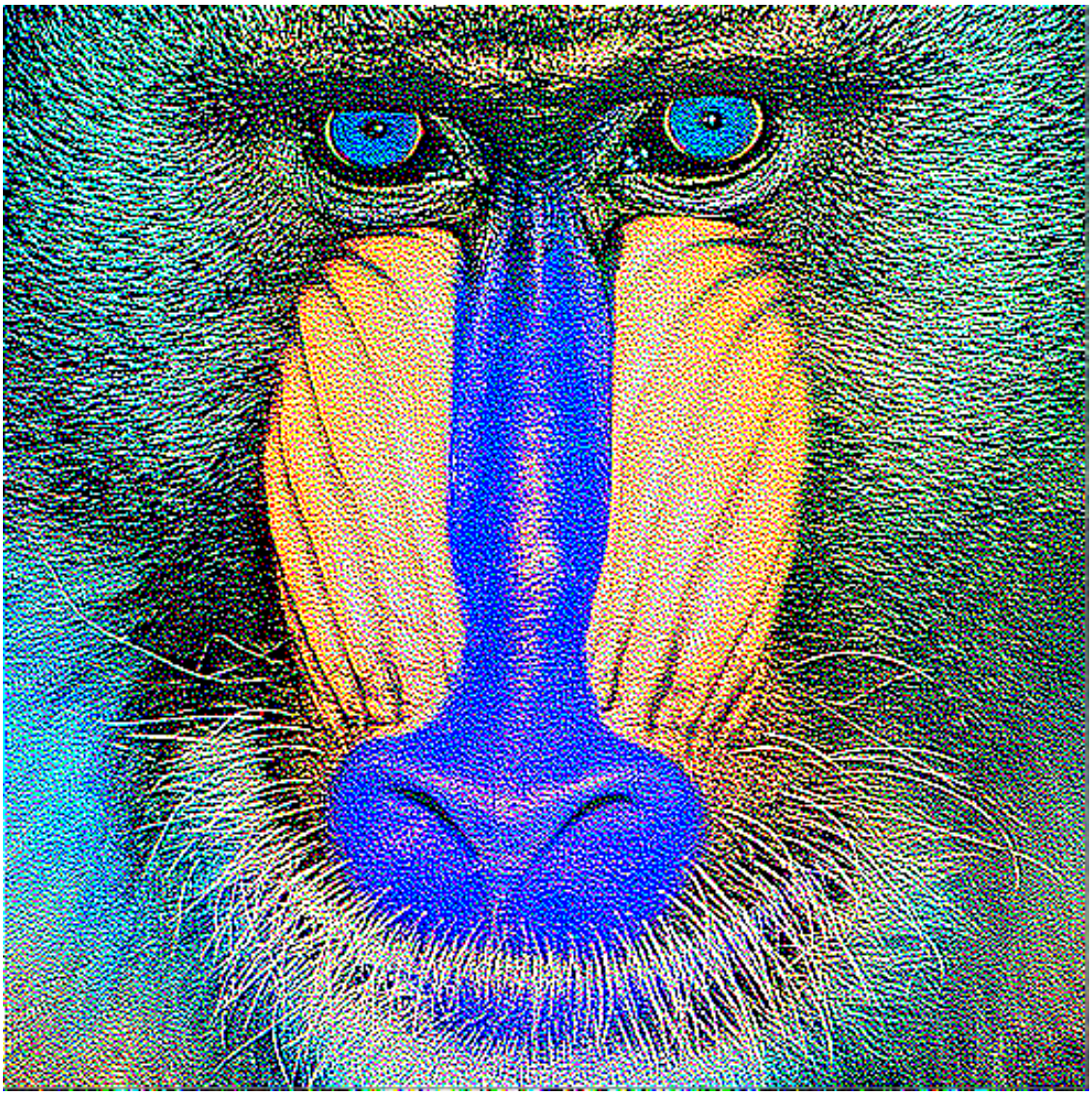
```
(512, 512, 3)
```

12.1.1 [EXIBIÇÃO]

JARVIS - RGB

[63]:
```
PIL.Image.fromarray(image)
```

[63]:

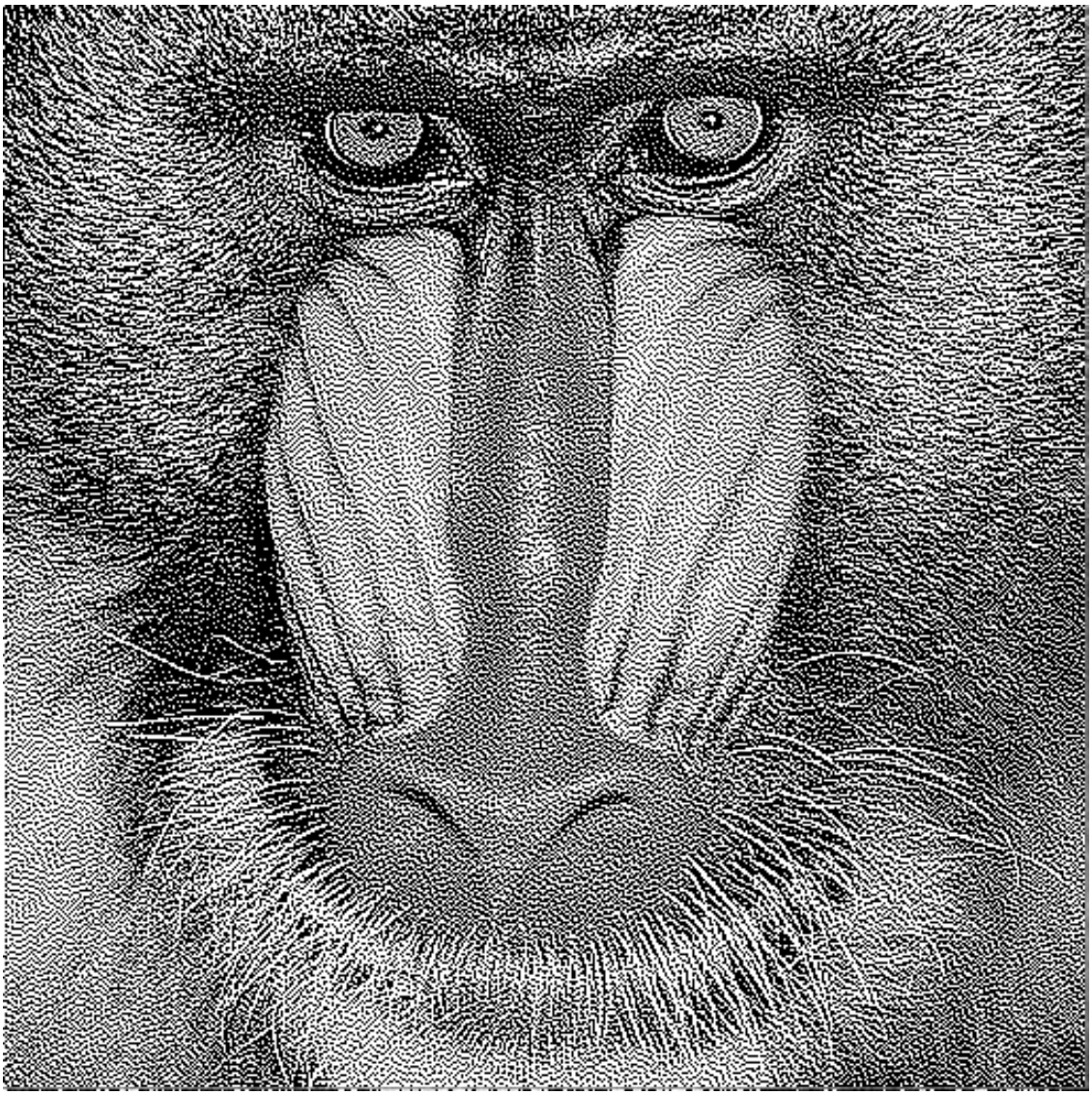


12.1.2 [EXIBIÇÃO]

JARVIS - GRAY

[64]:
```
PIL.Image.fromarray(gray2)
```

[64]:



---

13 OBS:

13.1 Notei que a exibição em cores não é a mesma executada pelo script via terminal, por gentileza, testar para averiguar o fato. Está com uma exibição mais saturada visualizando pelo notebook.

13.2 Ver resultados no diretório Output, também estará mais perceptível com as abordagens pedidas no trabalho.
