Paint Like Me

An implementation of Painterly Rendering with Curved Brush Strokes of Multiple Sizes (Hertzmann)

In [2]:
```
import os
import numpy as np
import cv2
import cairo

from matplotlib.colors import LogNorm
from scipy import signal, ndimage
import random
```

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [4]:
```
datadir = "images/"
```

Creating Gaussian Filter

This is from my Hybrid Images MP, here i'm using it to create different layers in my painting (for detailing)

In [283]:
```
# 2D GAUSSIAN FILTER
def createGaussianFilter(image, sigma):
    ksize = int(np.ceil(sigma)+1)
    fil = cv2.getGaussianKernel(ksize, sigma) # 1D kernel
    fil = fil*np.transpose(fil) # 2D kernel by outer product
    return cv2.filter2D(image, -1, fil)
```

Painting

In [165]:
```
def paint_layer(canvas, refference_img, radius, grid_size, T, min_stroke_length, max_stroke_leng
  #  a new set of strokes, initially empty (S)
  S = []

  # create a pointwise difference image (D)
  D = np.sqrt(np.sum((canvas - refference_img)**2, axis=-1))

  canvas_height, canvas_width, _ = canvas.shape
  grid = int(grid_size * radius)  # stepsize grid

  # gradient is computed from the Sobelfiltered luminance
  luminance = 0.30*refference_img[:,:,0] + 0.59*refference_img[:,:,1] + 0.11*refference_img[:,:

  gradient_x = ndimage.sobel(luminance,1)
  gradient_y = ndimage.sobel(luminance,0)

  for x in range(0, canvas_width, grid):
    for y in range(0, canvas_height, grid):

      M = D[y:y+grid, x:x+grid]
      area_err = M.sum()/(grid**2)

      if area_err > T :
        # Find the largest error point
        indices = np.argmax(M)
        y1, x1 = np.unravel_index(indices, M.shape)

        # make brush stroke
        s = make_spline_stroke(radius, x+x1, y+y1, refference_img, canvas, min_stroke_length, ma
        S.append(s)

  # paint all strokes in S on the canvas (randomorder)
  random.shuffle(S)
```

```
34
35    cairo_canvas = cairo.ImageSurface(cairo.FORMAT_RGB24, canvas_width, canvas_height)
36    canvas_context = cairo.Context(cairo_canvas)
37    canvas_context.set_line_cap(cairo.LINE_CAP_ROUND)
38
39    # set the line width
40    line_width = canvas_context.device_to_user_distance(2 * radius, 2 * radius)
41    canvas_context.set_line_width(np.min(line_width))
42
43    for s in S:
44      paint_strokes(s, refference_img, canvas_context)
45
46    return cairo_canvas
```

In [167]:

```
1  def paint_strokes(s, refference_img, canvas_context):
2    stroke_color = (refference_img[s[0][0],s[0][1]])
3    sy, sx = s[0]
4
5    R,G,B = tuple(stroke_color)
6    canvas_context.set_source_rgb(R, G, B)
7
8    # start stroke path
9    canvas_context.move_to(sx, sy)
10
11   # draw lines
12   for sy,sx in s:
13     canvas_context.line_to(sx, sy)
14     canvas_context.move_to(sx, sy)
15
16   # end
17   canvas_context.close_path()
18   canvas_context.stroke()
```

In [331]:

```
1  def make_spline_stroke(radius, x0, y0, refference_img, canvas, min_stroke_length, max_stroke_le
2    # K = a new stroke with radius R and color strokeColor
3    K = [(y0, x0)]
4    x,y = x0, y0
5    lastDx,lastDy = (0,0)
6
7    # pointilism has a stroke len of 0, so add 1
8    max_stroke_length += 1
9
10   for i in range(1, max_stroke_length):
11     canvas_color_diff = np.linalg.norm(refference_img[y,x] - canvas[y,x])
12     stroke_color_diff = np.linalg.norm(refference_img[y,x] - refference_img[y0, x0])
13
14     if (i > min_stroke_length and (canvas_color_diff < stroke_color_diff)):
15       return K
16
17     # detect vanishing gradient
18     gx = gradient_x[y,x]
19     gy = gradient_y[y,x]
20
21     if (gx**2 + gy**2 == 0):
22       return K
23
24     # compute a normal direction
25     dx,dy = (-gy, gx)
26
27     # if necessary, reverse direction
28     if (lastDx * dx + lastDy * dy < 0):
29       dx,dy = -dx, -dy
30
31     # filter the stroke direction
32     curvature_filter = 1
33     dx, dy = curvature_filter*(dx,dy) + (1-curvature_filter)*(lastDx,lastDy)
34     dx, dy = (dx, dy) / np.sqrt(dx**2 + dy**2)
35
36     x, y =  (x + radius*dx, y + radius*dy)
37
38     # round up x, y
39     x = int(round(x))
40     y = int(round(y))
41
42     if x >= refference_img.shape[1] or y >= refference_img.shape[0]:
43       return K
```

```
    lastDx,lastDy = (dx,dy)

    # add the point (x,y) to K
    K.append((y,x))

  return K
```

Pointillism

The art of putting many, many dots together to create shapes

In [400]:
```
im1_file = datadir + 'flower.jpg'
im1 = np.float32(cv2.imread(im1_file, cv2.IMREAD_GRAYSCALE) / 255.0)

# for full color RGB
im1_color = np.float32(cv2.imread(im1_file) / 255.0)
im1_rgb = cv2.cvtColor(im1_color, cv2.COLOR_BGR2RGB)

im1_height, im1_width = im1.shape
plt.imshow(im1_rgb)
```

Out[400]: <matplotlib.image.AxesImage at 0x7fbabb6aafd0>

In [401]:
```
source_img = im1_rgb
im_height = im1_height
im_width  = im1_width

# ==== Brush Properties ====
brush_sizes = [4, 2]
blur_filter = 20
grid_size = 0.5
approx_threshold = 0.05
min_stroke_length = 0
max_stroke_length = 0
```

In [402]:

```
# paint the canvas
canvas = np.zeros((im_height, im_width,3))
blur_img = np.zeros((len(brush_sizes), im_height, im_width, 3))

for i, radius in enumerate(brush_sizes):
  # apply Gaussian blur
  blur = createGaussianFilter(source_img, blur_filter*radius)
  reference_image = source_img * blur
  blur_img[i] = reference_image


# display blur layers
plt.figure()
plt.imshow(blur_img[0])

plt.figure()
plt.imshow(blur_img[1])
```

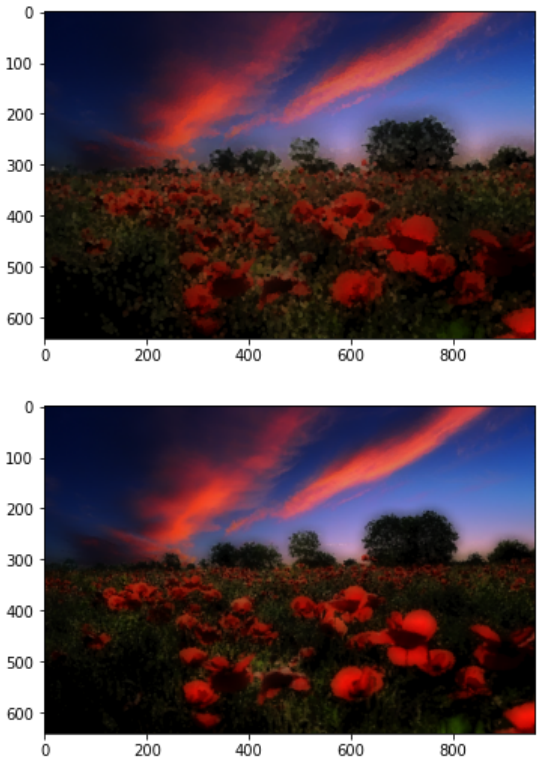
Out[402]: <matplotlib.image.AxesImage at 0x7fbaec4b6a60>

In [403]:

```
# paint a layer
layer_img = np.zeros((len(brush_sizes), im_height, im_width, 3), dtype=np.uint8)

for i, radius in enumerate(brush_sizes):
  reference_image = blur_img[i]

  cairo_canvas = paint_layer(canvas, reference_image, radius, grid_size, approx_threshold, min_s
  cairo_canvas.write_to_png("Pointillism.png")

  layer_img[i] = np.ndarray(shape=(im_height, im_width, 4),
                    dtype=np.uint8,
                    buffer=cairo_canvas.get_data())[:, :, 0:3]
```

In [404]:
```
# display painted layers
plt.figure()
plt.imshow(cv2.cvtColor(layer_img[0], cv2.COLOR_BGR2RGB) / 255)

plt.figure()
plt.imshow(cv2.cvtColor(layer_img[1], cv2.COLOR_BGR2RGB) / 255)
```

Out[404]: <matplotlib.image.AxesImage at 0x7fbab4a2abe0>

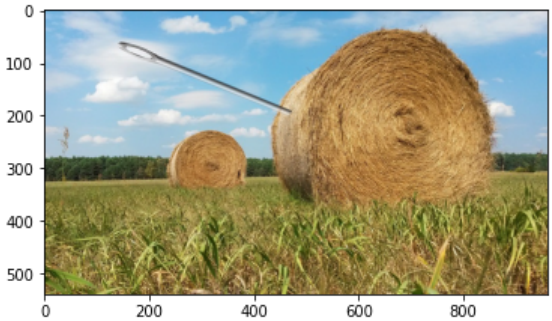


Impressionism

An art form from the 19th century and is well known for very defined, small brush strokes

In [322]:
```
im2_file = datadir + 'haystack.jpg'
im2 = np.float32(cv2.imread(im2_file, cv2.IMREAD_GRAYSCALE) / 255.0)

# for full color RGB
im2_color = np.float32(cv2.imread(im2_file) / 255.0)
im2_rgb = cv2.cvtColor(im2_color, cv2.COLOR_BGR2RGB)

im2_height, im2_width = im2.shape
plt.imshow(im2_rgb)
```

Out[322]: <matplotlib.image.AxesImage at 0x7fbab9f68460>



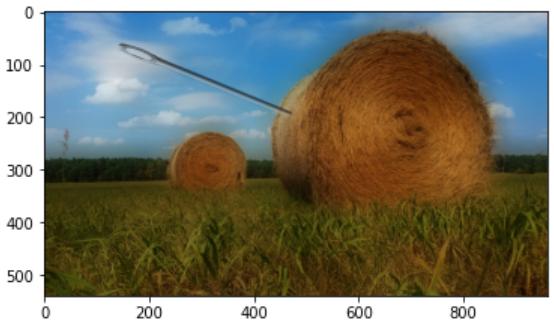
In [353]:
```
source_img = im2_rgb
im_height = im2_height
im_width  = im2_width
```

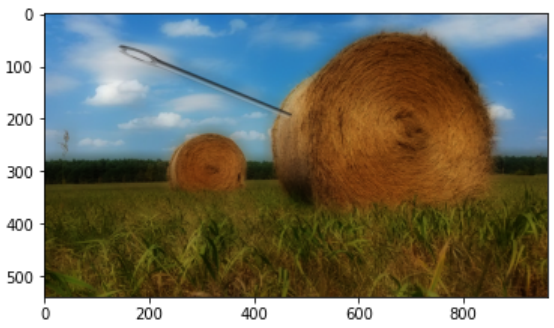
```
4
5  # ==== Brush Properties ====
6  brush_sizes = [8, 4, 2]
7  blur_filter = 10
8  grid_size = 1
9  approx_threshold = 0.05
10 min_stroke_length = 2
11 max_stroke_length = 16
```

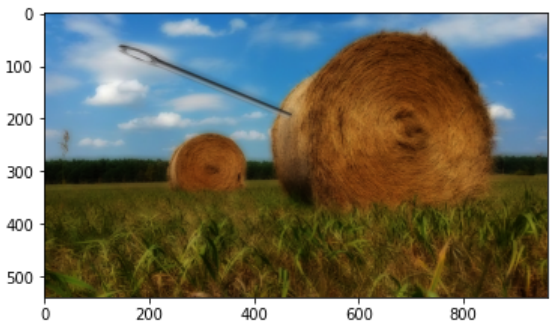
In [354]:

```
1  # paint the canvas
2  canvas = np.ones((im_height, im_width,3)) * np.inf
3  blur_img = np.zeros((len(brush_sizes), im_height, im_width, 3))
4
5  for i, radius in enumerate(brush_sizes):
6    # apply Gaussian blur
7    blur = createGaussianFilter(source_img, blur_filter*radius)
8    reference_image = source_img * blur
9    blur_img[i] = reference_image
10
11
12 # display blur layers
13 plt.figure()
14 plt.imshow(blur_img[0])
15
16 plt.figure()
17 plt.imshow(blur_img[1])
18
19 plt.figure()
20 plt.imshow(blur_img[2])
```

Out[354]: <matplotlib.image.AxesImage at 0x7fbab938aac0>







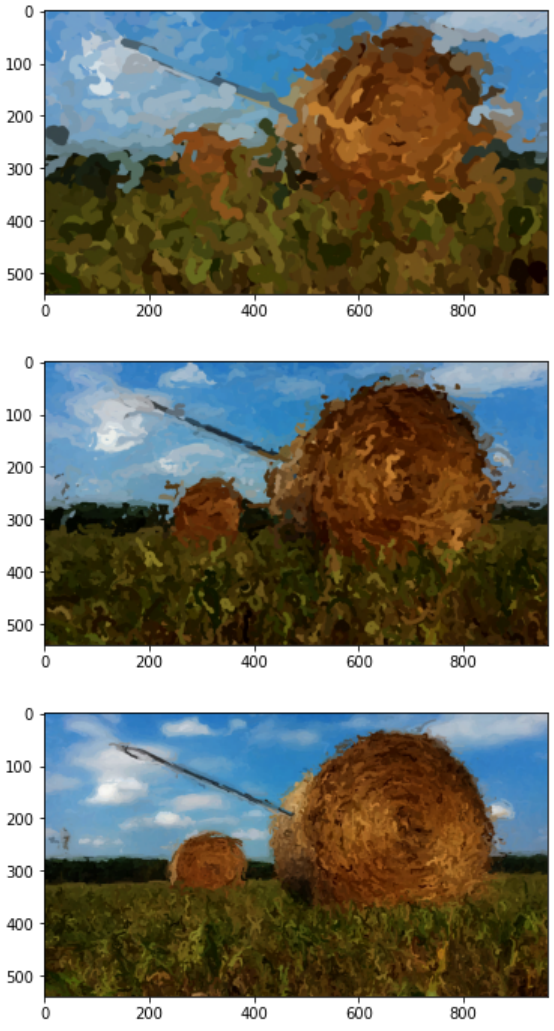
In [345]:

```
1  # paint a layer
2  layer_img = np.zeros((len(brush_sizes), im_height, im_width, 3), dtype=np.uint8)
```

```
for i, radius in enumerate(brush_sizes):
  reference_image = blur_img[i]

  cairo_canvas = paint_layer(canvas, reference_image, radius, grid_size, approx_threshold, min_s
  cairo_canvas.write_to_png("Impressionism.png")

  layer_img[i] = np.ndarray(shape=(im_height, im_width, 4),
                            dtype=np.uint8,
                            buffer=cairo_canvas.get_data())[:, :, 0:3]
```

In [359]:
```
# display painted layers
plt.figure()
plt.imshow(cv2.cvtColor(layer_img[0], cv2.COLOR_BGR2RGB) / 255)

plt.figure()
plt.imshow(cv2.cvtColor(layer_img[1], cv2.COLOR_BGR2RGB) / 255)

plt.figure()
plt.imshow(cv2.cvtColor(layer_img[2], cv2.COLOR_BGR2RGB) / 255)
```

Out[359]: <matplotlib.image.AxesImage at 0x7fbaba948c70>



Expressionism

more modern art form known for ‘expressive’ brush strokes

In [295]:
```
im3_file = datadir + 'Amsterdam.jpg'
im3 = np.float32(cv2.imread(im3_file, cv2.IMREAD_GRAYSCALE) / 255.0)

# for full color RGB
im3_color = np.float32(cv2.imread(im3_file) / 255.0)
```

```
im3_rgb = cv2.cvtColor(im2_color, cv2.COLOR_BGR2RGB)

im3_height, im3_width = im3.shape
plt.imshow(im3_rgb)
```

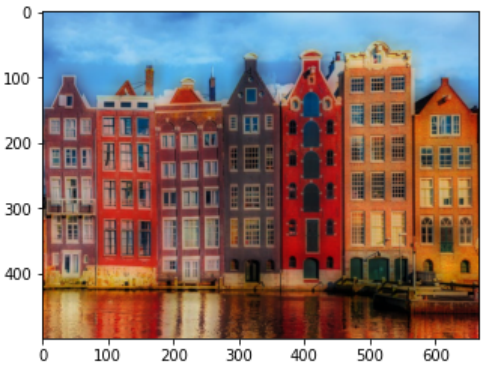
Out[295]: <matplotlib.figure.AxesImage at 0x7fbab6e5ae20>

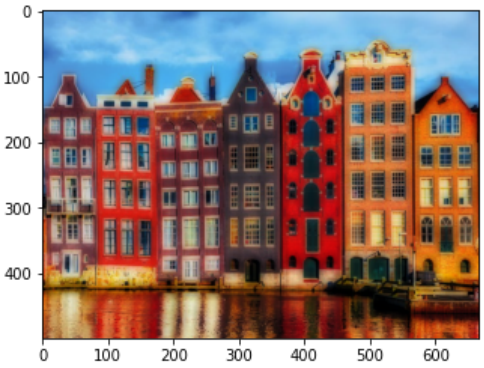
In [308]:

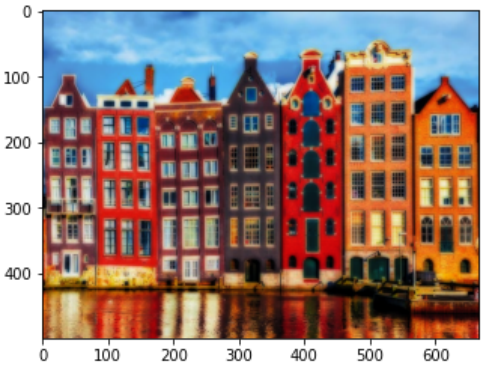
```
source_img = im3_rgb
im_height = im3_height
im_width  = im3_width

# ==== Brush Properties ====
brush_sizes = [8, 4, 2]
blur_filter = 5
grid_size = 1
approx_threshold = 0.05
min_stroke_length = 10
max_stroke_length = 16
```

In [309]:
```
# paint the canvas
canvas = np.ones((im_height, im_width,3)) * np.inf
blur_img = np.zeros((len(brush_sizes), im_height, im_width, 3))

for i, radius in enumerate(brush_sizes):
  # apply Gaussian blur
  blur = createGaussianFilter(source_img, blur_filter*radius)
  reference_image = source_img * blur
  blur_img[i] = reference_image


# display blur layers
plt.figure()
plt.imshow(blur_img[0])

plt.figure()
plt.imshow(blur_img[1])

plt.figure()
plt.imshow(blur_img[2])
```

Out[309]: <matplotlib.image.AxesImage at 0x7fbab56f01c0>





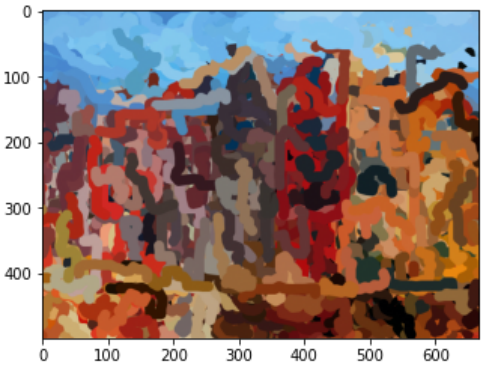


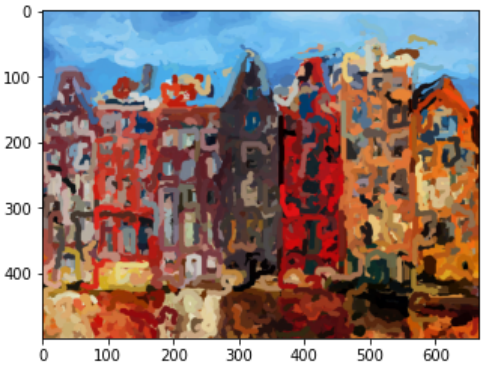
In [310]:
```
# paint a layer
layer_img = np.zeros((len(brush_sizes), im_height, im_width, 3), dtype=np.uint8)

for i, radius in enumerate(brush_sizes):
  reference_image = blur_img[i]
```

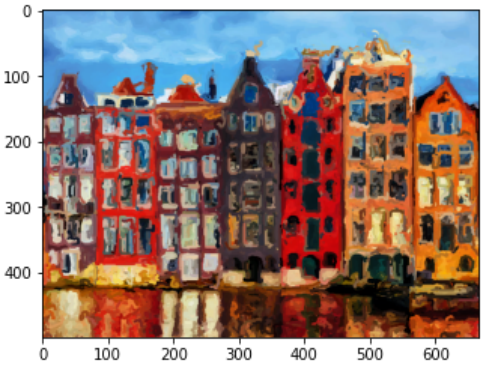
```
6
7   cairo_canvas = paint_layer(canvas, reference_image, radius, grid_size, approx_threshold, min_s
8   cairo_canvas.write_to_png("Expressionism.png")
9
10  layer_img[i] = np.ndarray(shape=(im_height, im_width, 4),
11                            dtype=np.uint8,
12                            buffer=cairo_canvas.get_data())[:, :, 0:3]
```

In [312]:
```
1 # display painted layers
2 plt.figure()
3 plt.imshow(cv2.cvtColor(layer_img[0], cv2.COLOR_BGR2RGB) / 255)
4
5 plt.figure()
6 plt.imshow(cv2.cvtColor(layer_img[1], cv2.COLOR_BGR2RGB) / 255)
7
8 plt.figure()
9 plt.imshow(cv2.cvtColor(layer_img[2], cv2.COLOR_BGR2RGB) / 255)
```

Out[312]: <matplotlib.image.AxesImage at 0x7fbac4d06670>







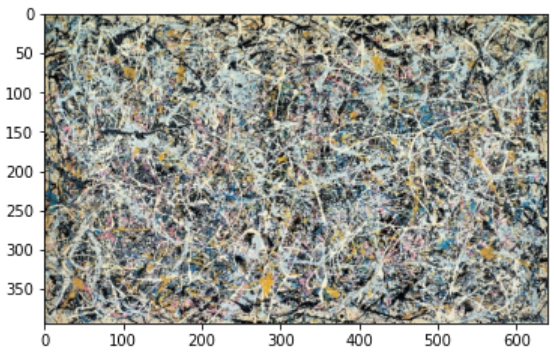
Finger Painting

Attempting to recreate something 4-year old Rima would have made

In [378]:
```
1 im4_file = datadir + 'pollock.jpg'
2 im4 = np.float32(cv2.imread(im4_file, cv2.IMREAD_GRAYSCALE) / 255.0)
```

```
# for full color RGB
im4_color = np.float32(cv2.imread(im4_file) / 255.0)
im4_rgb = cv2.cvtColor(im4_color, cv2.COLOR_BGR2RGB)

im4_height, im4_width = im4.shape
plt.imshow(im4_rgb)
```

Out[378]: <matplotlib.image.AxesImage at 0x7fbab996be20>



In [397]:
```
source_img = im4_rgb
im_height = im4_height
im_width  = im4_width

# ==== Brush Properties ====
brush_sizes = [10, 8]
blur_filter = 50
grid_size = 1
approx_threshold = 0.10
min_stroke_length = 4
max_stroke_length = 16
```

In [394]:
```
# paint the canvas
canvas = np.ones((im_height, im_width,3)) * np.inf
blur_img = np.zeros((len(brush_sizes), im_height, im_width, 3))

for i, radius in enumerate(brush_sizes):
  # apply Gaussian blur
  blur = createGaussianFilter(source_img, blur_filter*radius)
  reference_image = source_img * blur
  blur_img[i] = reference_image
```

In [398]:
```
# paint a layer
layer_img = np.zeros((len(brush_sizes), im_height, im_width, 3), dtype=np.uint8)

for i, radius in enumerate(brush_sizes):
  reference_image = blur_img[i]

  cairo_canvas = paint_layer(canvas, reference_image, radius, grid_size, approx_threshold, min_s
  cairo_canvas.write_to_png("Fingerpaint.png")

  layer_img[i] = np.ndarray(shape=(im_height, im_width, 4),
                    dtype=np.uint8,
                    buffer=cairo_canvas.get_data())[:, :, 0:3]
```

In [399]:

```
# display painted layers
plt.figure()
plt.imshow(cv2.cvtColor(layer_img[0], cv2.COLOR_BGR2RGB) / 255)

plt.figure()
plt.imshow(cv2.cvtColor(layer_img[1], cv2.COLOR_BGR2RGB) / 255)
```

Out[399]: <matplotlib.image.AxesImage at 0x7fbaf6b0be80>