In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

print("Setup Complete")
```

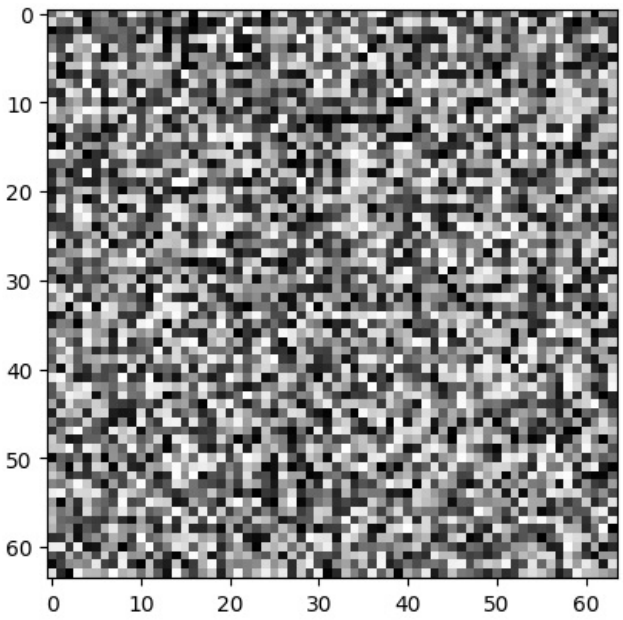
```
Setup Complete
```

Вступ

Для початку розглянемо малюнок згенерований задопомогою рандомних чисел:

In [ ]:
```
generation_matrix = np.random.randint(255, size=(64, 64))
plt.imshow(generation_matrix, cmap="gray")
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a251c81550>
```



В результаті маємо рисунок, що нагадує шум телевізора, коли є поганий зв'язок.

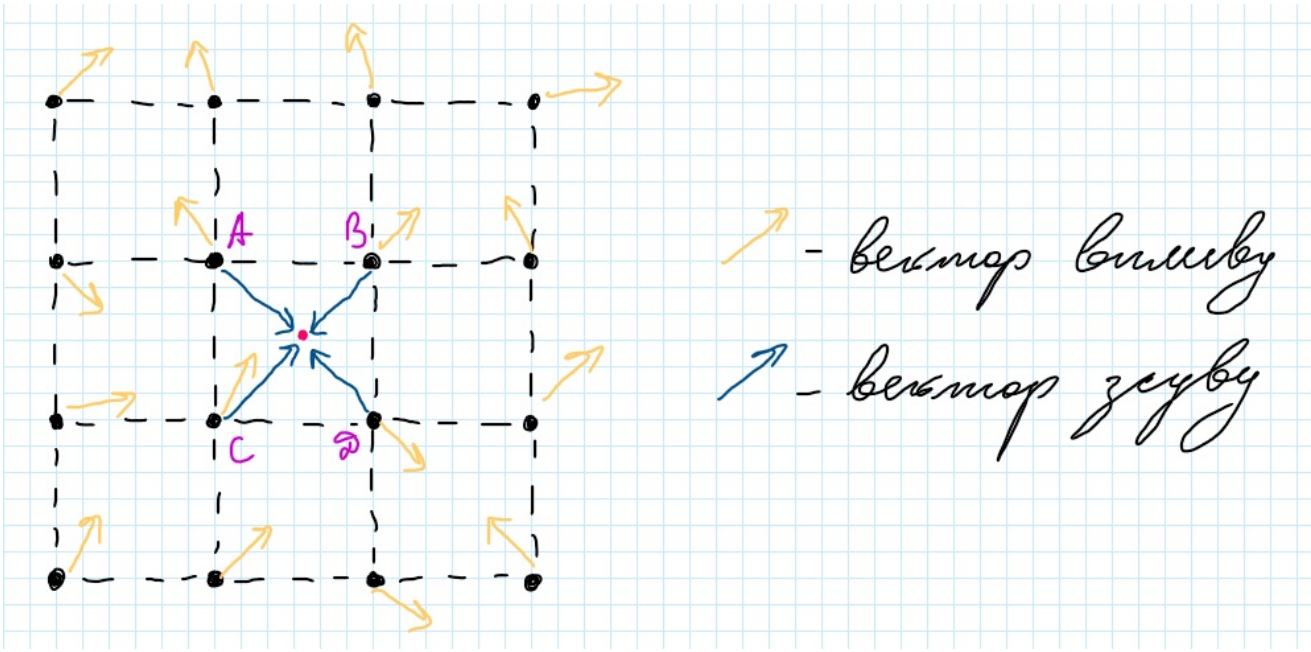
Для роботи цей шум не є придатний, бо значення є "абсолютно" рандомними. Для імітації "природи" наприклад текстури дерева, необхідні "плавні" радномні значення, адже в природі повністю випадковим є тільки космічний шум, все інше повинне є залежним. Отже потрібно імітувати цю залежність. Для цього будемо використовувати шум Перліна.

Шум Перліна

Шум Перліна має 3 основні етапи:

- сітка
- скалярний добуток
- інтерполяція

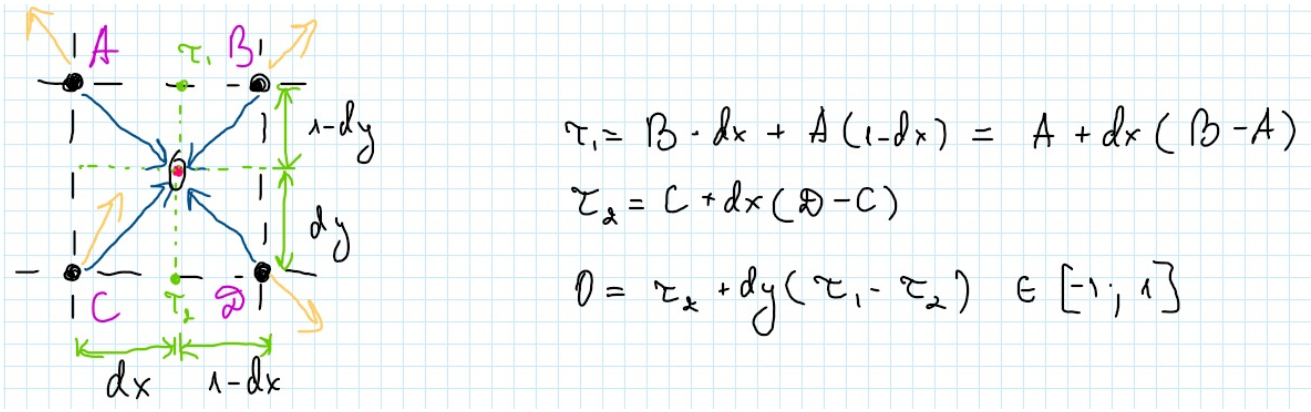
Почнемо з першомго - сітка: Отже уявимо деяку сітку, вектори впливу є вибираються рандомним чином, а ось вектори зсуву обчислюються



Другий етап "скалярний добуток", обираємо наприклад верхню ліву точку (A), обчислюємо скалярний добуток векторів, що виходять з цієї точки, так робимо для кожної точки.

Значення скалярного добутку будемо записувати в точку відносно якої обчислюємо

Третій етап - інтерполяція: Використовуємо лінійну інтерполяцію віднсоно точок A, B, C, D і знаходимо значення червоної точки. Це значення буде лежати в межах [-1; 1]

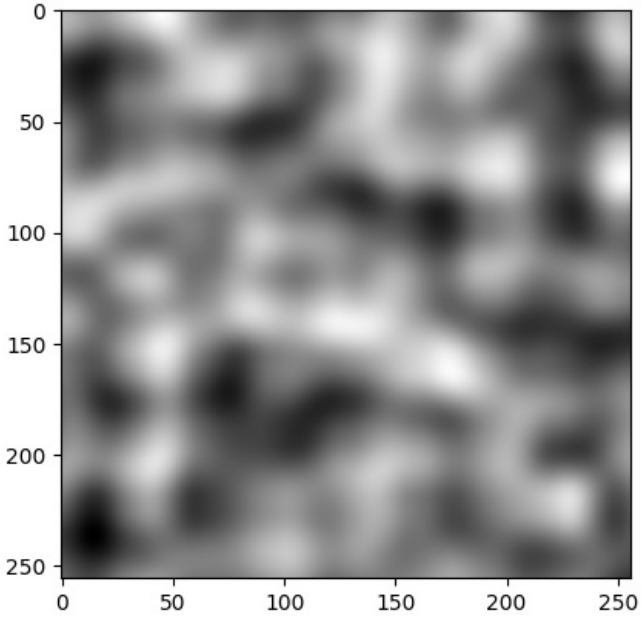


Знизу написана імплементація шуму Перліна

- **x** : x'ва частина від `np.meshgrid`
- **y** : y'ва частина від `np.meshgrid`
- **z** : координата z
- **seed** : використовується для `np.random.seed`

In [ ]:
```
def check_index(ptable, index):
    if (index > ptable.size - 1).any():
        index = index % (ptable.size - 1)
    return index

def perlin(x, y, z = 0, seed = 0):
    # print(f"x:\n{x},\n\n y:\n{y}")
    # print("==============================")
    np.random.seed(seed)
    ptable = np.arange(256, dtype=int)

    np.random.shuffle(ptable)

    ptable = np.stack([ptable, ptable, ptable]).flatten()
    try:
        xi, yi, zi = int(x), int(y), int(z)
    except:
        xi, yi, zi = x.astype(int), y.astype(int), int(z)
    # print(f"xi:\n{xi},\n\n yi:\n{yi}")
    # print("++++++++++++++++++++++++++++++")
```

```
    xg, yg, zg = x - xi, y - yi, z - zi
    # print(f"xg:\n{xg},\n\n yg:\n{yg}")
    # print("-----------------------------")
    xf, yf, zf = fade(xg), fade(yg), fade(zg)
    # print(f"xf:\n{xf},\n\n yf:\n{yf}")

    n000 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi)] + yi)
                    xg, yg, zg)
    n010 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi)] + yi
                    xg, yg - 1, zg)
    n110 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi + 1)] +
                    xg - 1, yg - 1, zg)
    n100 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi + 1)] +
                    xg - 1, yg, zg)

    n001 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi)] + yi)
                    xg, yg, zg - 1)
    n011 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi)] + yi
                    xg, yg - 1, zg - 1)
    n111 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi + 1)] +
                    xg - 1, yg - 1, zg - 1)
    n101 = gradient(ptable[check_index(ptable, ptable[check_index(ptable, ptable[check_index(ptable, xi + 1)] +
                    xg - 1, yg, zg - 1)


    n00 = lerp(n000, n100, xf)
    n10 = lerp(n010, n110, xf)
    n01 = lerp(n001, n101, xf)
    n11 = lerp(n011, n111, xf)

    n0 = lerp(n00, n10, yf)
    n1 = lerp(n01, n11, yf)
    return lerp(n0, n1, zf)

def lerp(a, b, x):
    return a + x * (b - a)

def fade(f):
    return ((6*f - 15)*f + 10)*np.power(f, 3)

def gradient(c, x, y, z):
    vectors = np.array([[1, 1, 0], [1, -1, 0], [1, 0, 1], [1, 0, -1],
                        [0, 1, 1], [0, 1, -1], [-1, 1, 0], [-1, -1, 0],
                        [-1, 0, 1], [-1, 0, -1], [0, -1, 1] ,[0, -1, -1],
                        ])
    # print(f"vectors:\n{vectors}")
    gradient_co = vectors[c % 12]
    # print(f"gradient_co:\n{gradient_co}")
    return gradient_co[:, :, 0]*x + gradient_co[:, :, 1]*y + gradient_co[:, :, 2]*z

lin_array = np.linspace(0, 8, 256)
x, y = np.meshgrid(lin_array, lin_array)

plt.imshow(perlin(x, y, 1.4, seed=1), cmap="gray")
plt.show()
```

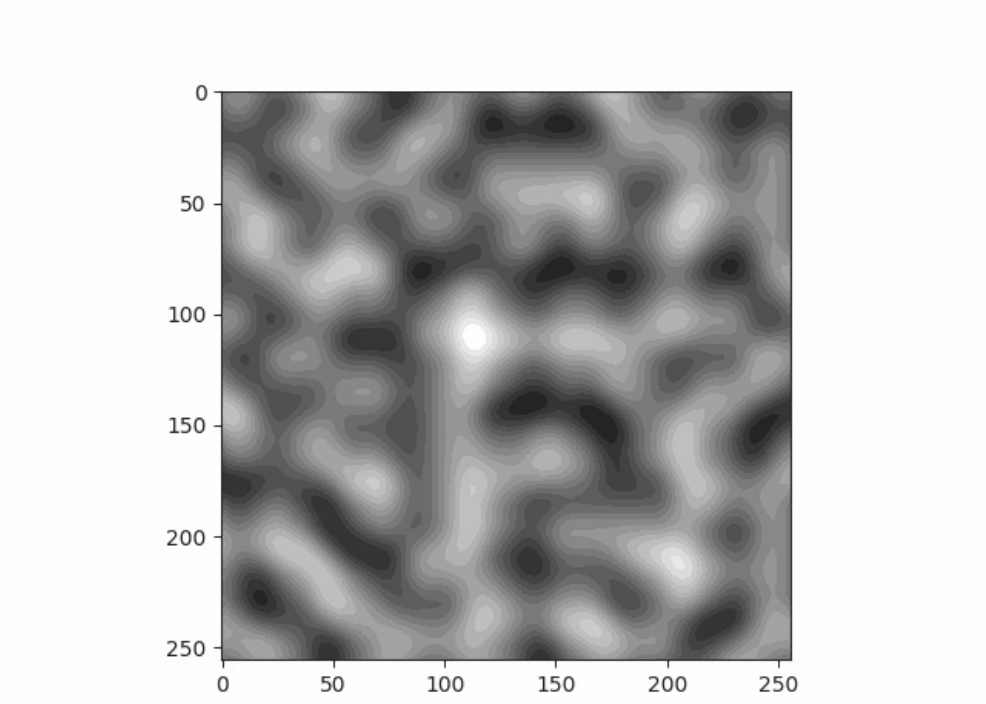


Зверху приклад 2D шуму Перліна

код що знизу використовується для збереження графіків у вигляді `.gif`. Для того, щоб не перезаписувати з кожним рестартом ядра, було вирішено закоментувати такі острівці коду

In [ ]:
```
import matplotlib.animation as animation


# fig, ax = plt.subplots()

# z = np.linspace(0, 8, 256)

# ims = []
# for z_i in z:
#     im = ax.imshow(perlin(x, y, z_i, seed=10), cmap="gray", animated=True)
#     if z_i == 0:
#         ax.imshow(perlin(x, y, z_i, seed=10), cmap="gray")  # show an initial one first
#     ims.append([im])

# ani = animation.ArtistAnimation(fig, ims, interval=20, blit=True,
#                                 repeat_delay=10)

# ani.save("gif/movie.gif")

# plt.show()
```

А ось приклад 3D шуму Перліна, який будемо використовувати далі



Векторне поле

Розіб'ємо полотно з розміром 600 на 400 пікселів на сітку з кроком 20 пікселів, на перетині сітки розташуємо початок векторів, а за допопмогою шуму Перліна будемо зчитувати занчення шуму в точці де знаходиться початок вектора і застосовувати це значення, щоб повернути цей вектор, а саме $\frac{\text{значення шуму перліна}+1}{2}$ з $\frac{\text{н}}{2}$ а ч е н н
таким чином з меж $[-1; 1]$ ми переходио в межі від 0 до 1 помноживши це значення на 2π. Отримаємо значення від 0 до 2π і це значення будемо використовувати щоб крутити вектор(надалі замість 2π будемо використовувати інші значення, щоб отримати інші результати ☻)

In [ ]:
```
window_shape = (600, 400)

cell_step = 20
x_window = np.arange(0, window_shape[0] + cell_step, cell_step)
y_window = np.arange(0, window_shape[1] + cell_step, cell_step)

scale = 2


def rotate_matrix(angle: float):
    return np.array([[np.cos(angle), np.sin(angle)],
                     [-np.sin(angle), np.cos(angle)]])
```

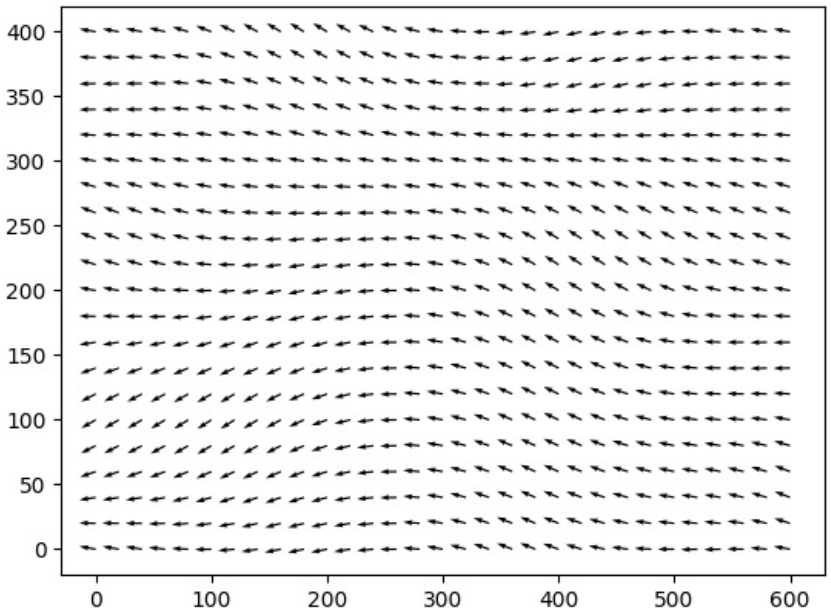
```
x_grid, y_grid = np.meshgrid(x_window, y_window)

# plt.scatter(x_grid, y_grid)

u = np.sin(x_grid)
v = np.cos(y_grid)

perlin_noise = (perlin(x_grid/window_shape[0] * scale, y_grid/window_shape[1] * scale)+1)/2
# print(perlin_noise)
```

Функція знизу використовується для того, щоб повернути вектор відносно мапи шуму

- **`noise_map`** : мапа шуму
- **`y, x`** : координати початку вектора
- **`length`** : довжина вектора
- **`scale_vector_angle`** : коефіцієнт множення кута повороту

In [ ]:
```
def vectors(noise_map, y, x, length, scale_vector_angle = 1):

    # old version \/
    # print(f"perlin noise shape:{noise_map.shape}")
    # print(f"{x.size}, {y.size}")
    # vectors_arr_x = np.zeros((x.size, y.size))
    # vectors_arr_y = np.zeros((x.size, y.size))

    # for x_i in range(x.size):
    #     for y_i in range(y.size):
    #         start_vector = np.array([0, length])
    #         # print(f"noise map value:{noise_map[x_i, y_i]}, x_i={x_i}, y_i={y_i}")
    #         rotated_vector = np.dot(rotate_matrix(-np.pi*noise_map[x_i, y_i]*(18/18)), start_vector)
    #         # print(rotated_vector)
    #         vectors_arr_x[x_i, y_i] = rotated_vector[0]
    #         vectors_arr_y[x_i, y_i] = rotated_vector[1]


    # [[np.cos(angle), np.sin(angle)],
    #  [-np.sin(angle), np.cos(angle)]]
    # optimazied version

    reshaped_noise = noise_map.reshape(x.size*y.size)
    angled_noise = -np.pi*reshaped_noise*scale_vector_angle+np.pi*(2-scale)/2

    by = np.sin(angled_noise)*length
    dy = np.cos(angled_noise)*length

    return by.reshape(y.size, x.size), dy.reshape(y.size, x.size)
```

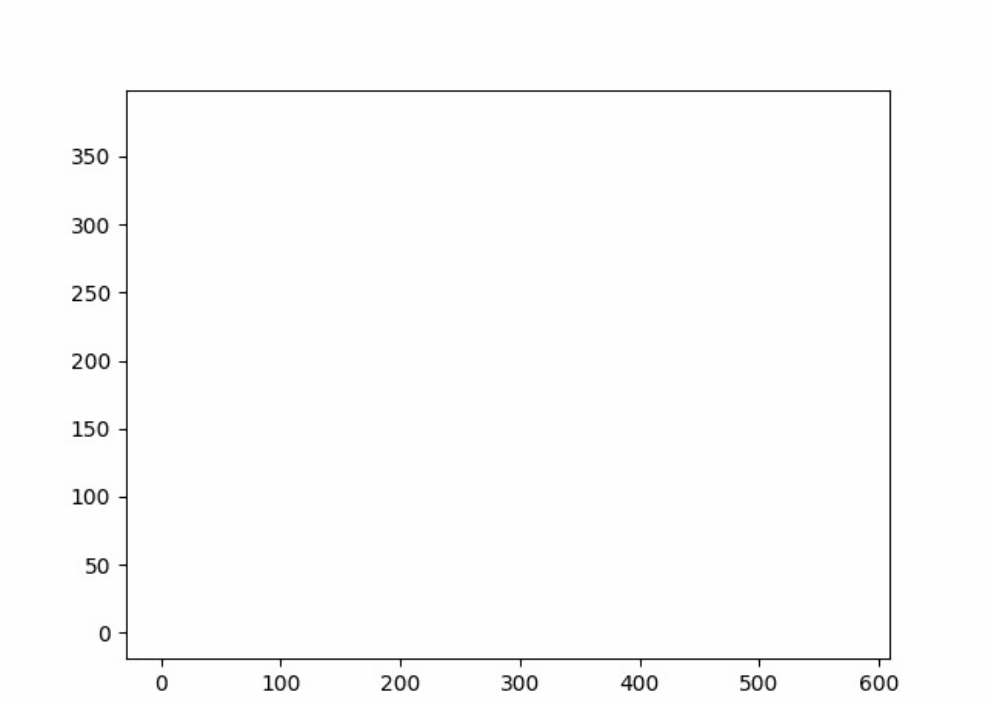
In [ ]:
```
u, v = vectors(perlin_noise, x_window, y_window, 5, 16/18)
plt.quiver(x_grid, y_grid, u, v)
display()
```



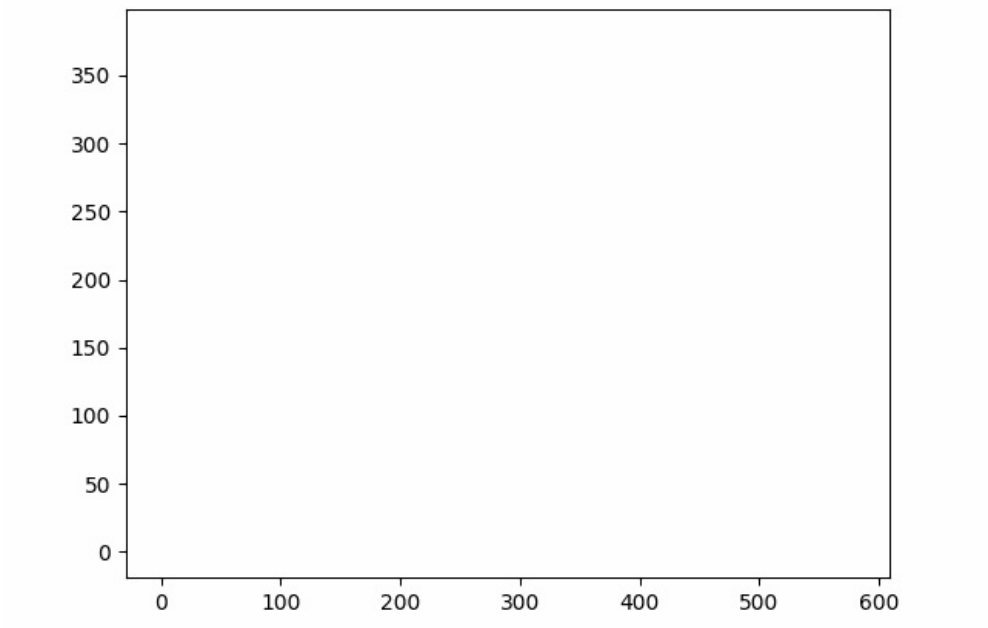
Розглянемо тепер це ж саме поле, але будемо змінювати `z` в шумі Перліна:

In [ ]:
```
# fig, ax = plt.subplots()

# z = np.linspace(0, 4, 256)

# ims = [ax.plot()]
# for z_i in z:
#     perlin_noise = (perlin(x_grid/window_shape[0] * scale, y_grid/window_shape[1] * scale, z_i)+1)/2
#     u, v = vectors(perlin_noise, x_window, y_window, 6, 2)
#     im = ax.quiver(x_grid, y_grid, u, v, animated = True)
#     ims.append([im])

# ani = animation.ArtistAnimation(fig, ims, interval=20, blit=True,
#                                 repeat_delay=10)

# ani.save("gif/vectors_field.gif")
```

`scale_vector_angle` = 2π



`scale_vector_angle` = π



Плаваюче поле

Тепер запустимо частинки на векторне поле, і значення вектора будемо записувати як вектор швидкості частинки

Particle:

Параметри:

- `x` , `y` : нинішні координати частинки
- `px` , `py` : координати попередньої позиції частинки
- `vx` , `vy` : значення вектора швидкості
- `window_limit_x` , `window_limit_y` : значення меж вікна(полотна)

Функції:

- `__init__()` : ініціалізація об'єкту
- `move()` : переміщує частинку згідно з вектором швидкості, після перевіряє граничні умови
- `follow()` : слідкує за данною сіткою векторів

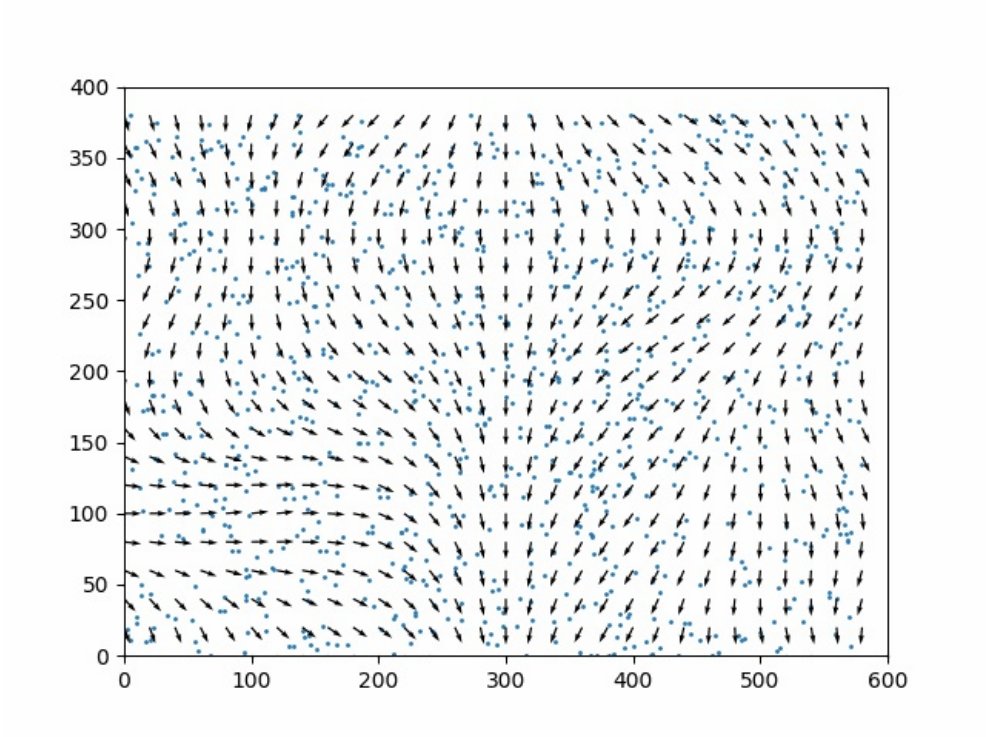
In [ ]:
```
class Particle:

    x: float
    y: float
    px: float
    py: float
    vx: float = 0
    vy: float = 0

    window_limit_x: float
    window_limit_y: float

    def __init__(self, limit_x: float, limit_y: float) -> None:
        self.window_limit_x = limit_x
        self.window_limit_y = limit_y

        self.x = np.random.random()*self.window_limit_x
        self.y = np.random.random()*self.window_limit_y

        self.px = self.x
        self.py = self.y

    def move(self) -> None:
        self.px = self.x
        self.py = self.y

        self.x += self.vx
        self.y -= self.vy

        # граничні умови
        if self.x > self.window_limit_x:
            self.x = 0
            self.px = self.x
        if self.x < 0:
            self.x = self.window_limit_x
            self.px = self.x
        if self.y > self.window_limit_y:
            self.y = 0
            self.py = self.y
        if self.y < 0:
            self.y = self.window_limit_y
            self.py = self.y

        # if self.x > self.window_limit_x or self.x < 0 or self.y > self.window_limit_y or self.y < 0:
        #     self.x = self.window_limit_x
        #     self.y = np.random.random()*self.window_limit_y

    def follow(self, u, v, cell_step) -> None:
        get_pos_x, get_pos_y = int(self.x//cell_step), int(self.y//cell_step)
        # print(f"x:{self.x}, y:{self.y}")
        # print(f"pos_x:{get_pos_x}, pos_y:{get_pos_y}")
        # print(f"shape u:{u.shape}\nshape v:{v.shape}")
        self.vx = u[get_pos_y, get_pos_x]
        self.vy = v[get_pos_y, get_pos_x]
        # print(f"vx:{self.vx}, vy:{self.vy}")
        # print("============================")
        self.move()
```

Острівець коду для збереження `.gif`

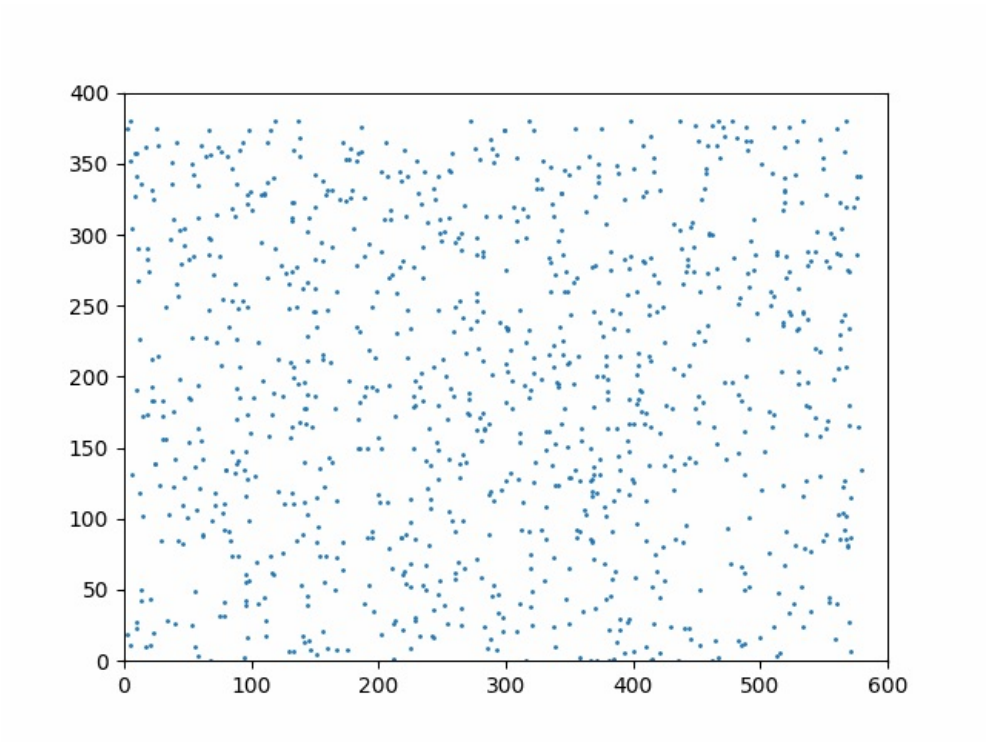
In [ ]:
```
# fig, ax = plt.subplots()

# z = np.linspace(0, 4, 256)
```

```
# particles = []
# population = 1000

# for i in range(population):
#     particles.append(Particle(window_shape[0]-cell_step, window_shape[1]-cell_step))

# def animation_func(i):

#     ax.clear()

#     perlin_noise = (perlin(x_grid/window_shape[0] * scale, y_grid/window_shape[1] * scale, z[i])+1)/2
#     u, v = vectors(perlin_noise, x_window, y_window, 6, 2)
#     scatter_x_= np.zeros(population)
#     scatter_y_= np.zeros(population)

#     for i in range(population):
#         particles[i].follow(u, v, cell_step)
#         scatter_x_[i] = particles[i].x
#         scatter_y_[i] = particles[i].y

#     ax.scatter(scatter_x_, scatter_y_, s=1)
#     ax.set_ylim((0, window_shape[1]))
#     ax.set_xlim((0, window_shape[0]))
#     # ax.quiver(x_grid, y_grid, u, v, animated = True)
#     return ax

# func_ani = animation.FuncAnimation(fig, animation_func, repeat = True, frames = z.size-1, interval = 50)
# writer = animation.PillowWriter(fps=15,
#                                 metadata=dict(artist='BerVol57'),
#                                 bitrate=1800)
# func_ani.save('gif/particles_scale2.gif', writer=writer)
```

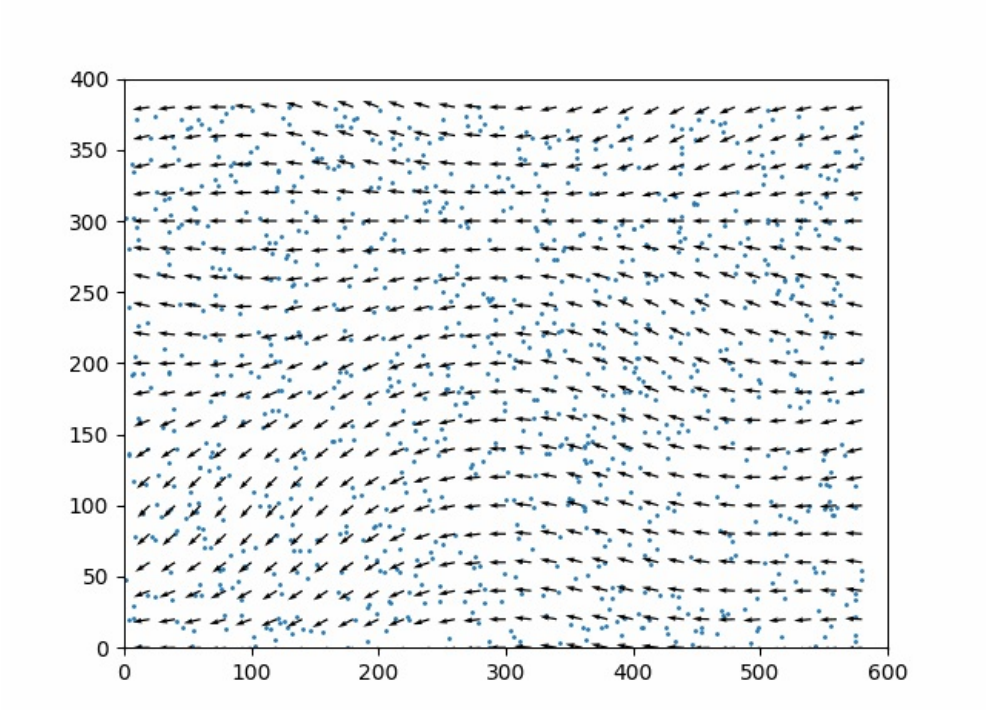
Приклад "плаваючого" поля з частинками, `scale_vector_angle` = 2π



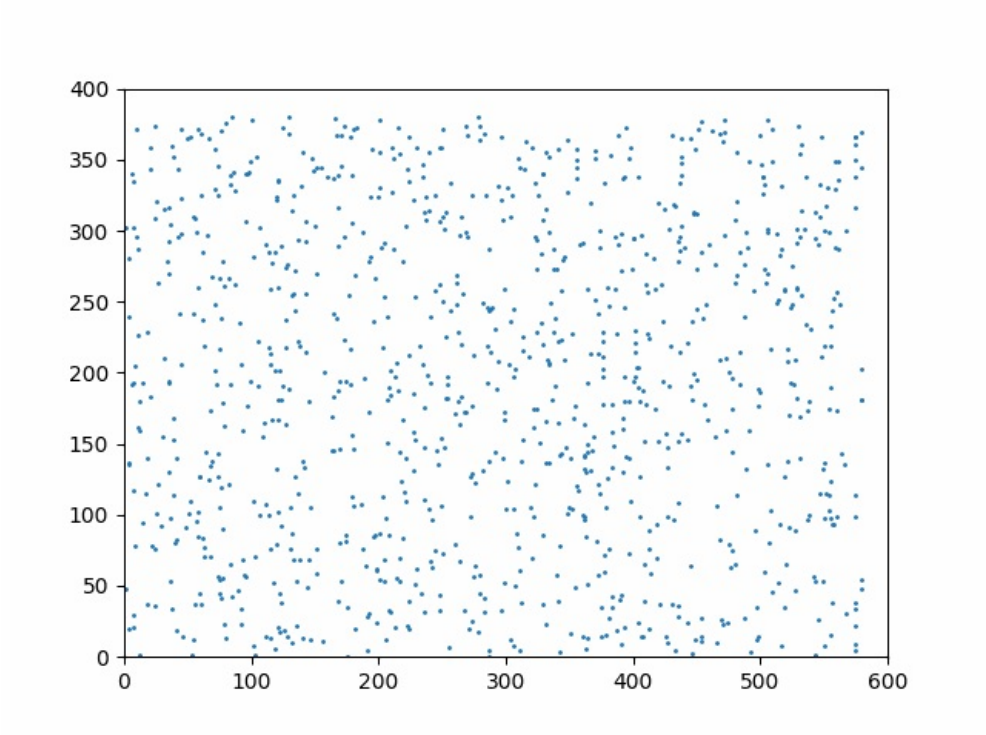
Без векторного поля



3 `scale_vector_angle` = π



Без векторного поля



Відтепер частинки залишатимуть сліди ^w^

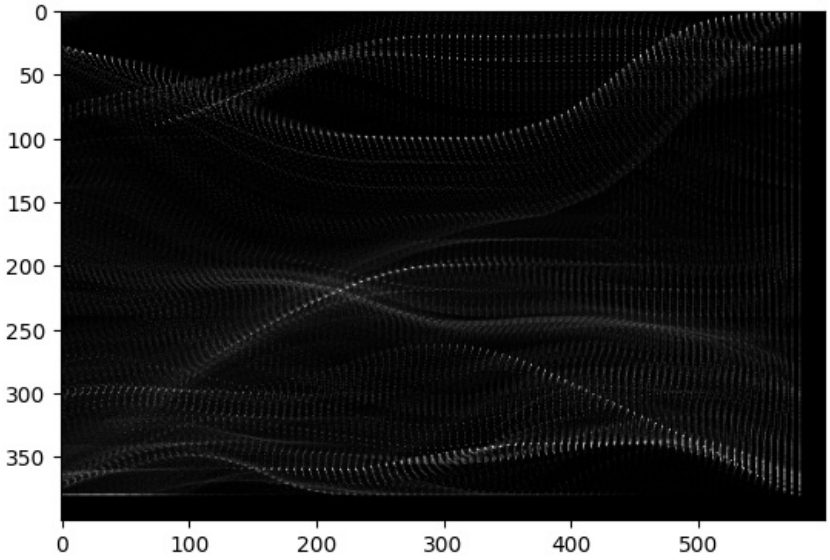
Отже створюємо нове зображення із шириною та висотою як у нашого полотна і тепер з кожним рухом частинка залишатиме свій відбиток на зображенні. Наприклад збільшуватиме значення пікселя на 1

In [ ]:
```
from IPython.display import clear_output
import time

%matplotlib inline

z = np.linspace(0, 4, 256)

particles = []
population = 10_000

img_path_particles = np.zeros(window_shape[::-1])

for i in range(population):
    particles.append(Particle(window_shape[0]-cell_step, window_shape[1]-cell_step))


for z_i in z:
    perlin_noise = (perlin(x_grid/window_shape[0] * scale, y_grid/window_shape[1] * scale, z_i)+1)/2
    u, v = vectors(perlin_noise, x=x_window, y=y_window, length=6, scale_vector_angle=1)

    scatter_x_ = np.zeros(population)
    scatter_y_ = np.zeros(population)

    for i in range(population):
        particles[i].follow(u, v, cell_step)
        scatter_x_[i] = particles[i].x
        scatter_y_[i] = particles[i].y
        img_path_particles[int(particles[i].y), int(particles[i].x)] += 1
        if img_path_particles[int(particles[i].y), int(particles[i].x)] > 255:
            img_path_particles[int(particles[i].y), int(particles[i].x)] = 255

    # plt.scatter(scatter_x_, scatter_y_, s=1)
    # plt.quiver(x_grid, y_grid, u, v)
    # clear_output(wait=True)
    # plt.show()
    # time.sleep(.01)
```

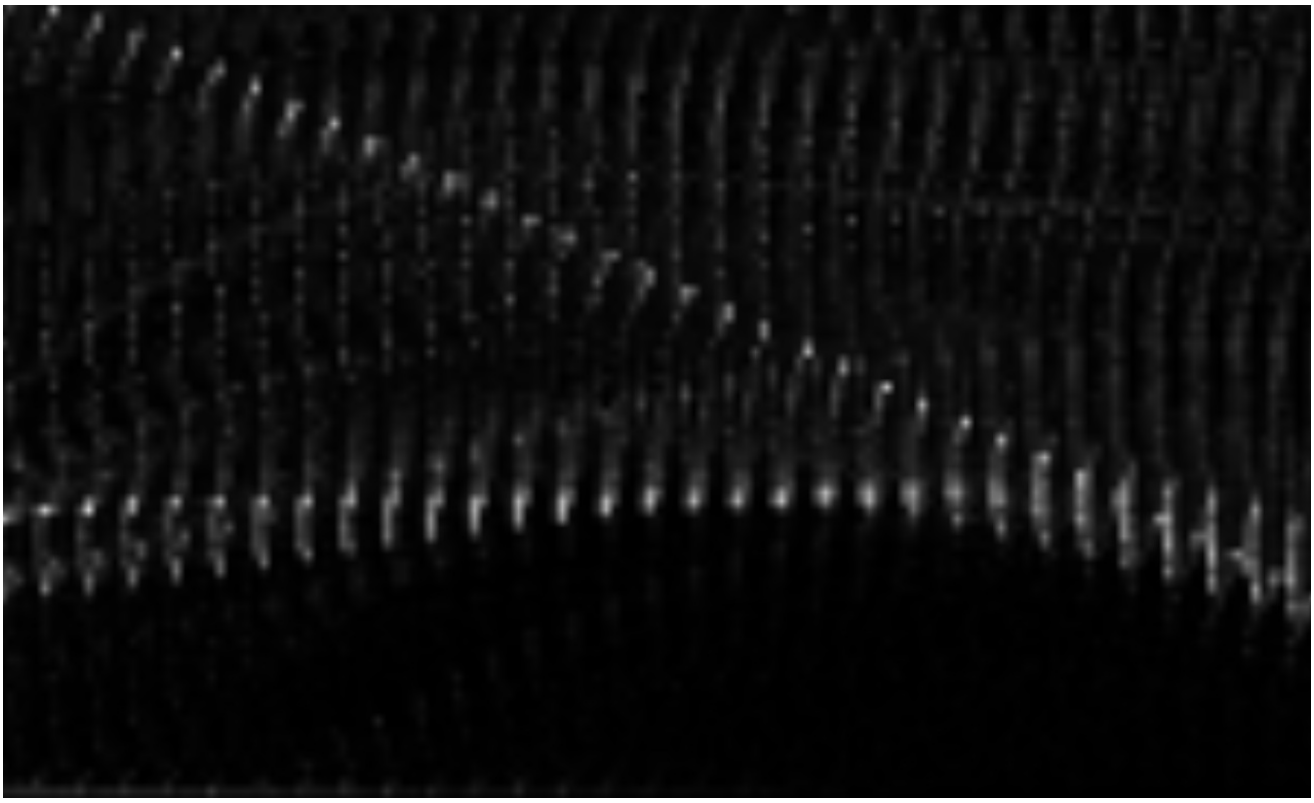
Маємо ось такий результат

In [ ]:
```
plt.imshow(img_path_particles, cmap="gray")
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a2552fade0>
```



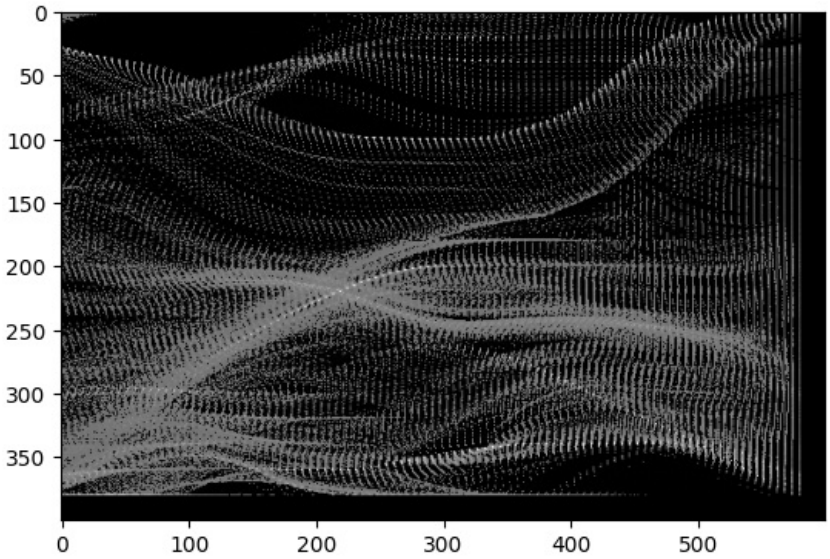
Отже на рисунку присутні ось такі "прогалини"



In [ ]:
```
test = img_path_particles.copy()

print(np.max(test))
test /= np.max(test)
test[np.where(test>.04)] += 0.2
test /= np.max(test)
test[np.where(test>.05)] += 0.2
test /= np.max(test)
test[np.where(test>.06)] += 0.2

plt.imshow(test, cmap="gray")
```

```
255.0
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a25580bfe0>
```



Висвітливши зображення можемо побачити, що прогалини є по всьому рисунку. Для того, щоб забрати ці прогалини замість того, щоб тільки фарбувати піксель де знаходиться частинку, будемо проводити лінію, між попереднім місцезнаходженням частинки та теперішнім.

Малюємо лінії

Алгоритм Брезенхема

Деталі алгоритму не бачу сенсу розподвідати, коли є чудова стаття на вікіпедії із псевдокодом тик. Хоча алгоритм трохи модифікований, так щоб не малювати кінцеву точку

In [ ]:
```
def LowLine(matrix, x0, y0, x1, y1, set_value=1, value_limit = 255):
    dx = x1 - x0
    dy = y1 - y0
    yi = 1
    if dy < 0:
        yi = -1
        dy = -dy
    D = 2 * dy - dx
    y = y0

    for x in range(x0, x1+1):
        matrix[y, x] += set_value
        if matrix[y, x] > value_limit:
            matrix[y, x] = value_limit
        if D > 0:
            y += yi
            D += 2 * (dy - dx)
        else:
            D += 2 * dy

    # if matrix[y0, x0] - set_value < 0:
    #     matrix[y0, x0] = 0
    # else:
    #     matrix[y0, x0] -= set_value


def HighLine(matrix, x0, y0, x1, y1, set_value=1, value_limit = 255):
    dx = x1 - x0
    dy = y1 - y0
    xi = 1
    if dx < 0:
        xi = -1
        dx = -dx
    D = 2 * dx - dy
    x = x0

    for y in range(y0, y1+1):
        matrix[y, x] += set_value
```

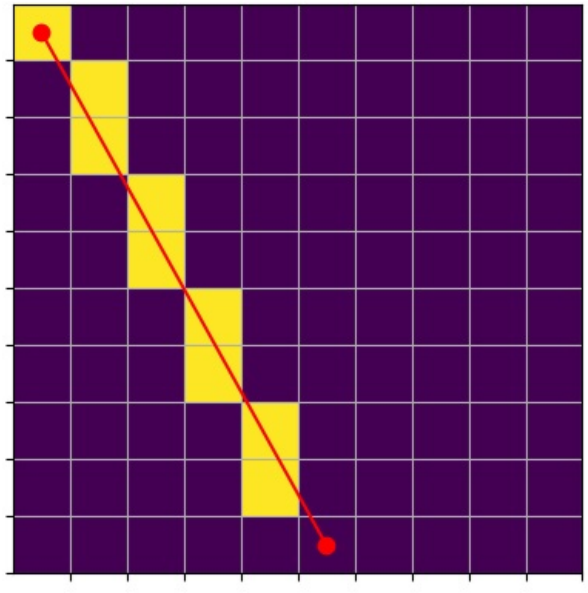
```
        if matrix[y, x] > value_limit:
            matrix[y, x] = value_limit
        if D > 0:
            x += xi
            D += 2 * (dx - dy)
        else:
            D += 2 * dx

    # if matrix[y0, x0] - set_value < 0:
    #     matrix[y0, x0] = 0
    # else:
    #     matrix[y0, x0] -= set_value


def Line(matrix, x0, y0, x1, y1, set_value=1, value_limit = 255):
    if abs(y1 - y0) < abs(x1 - x0):
        if x0 > x1:
            LowLine(matrix, x1, y1, x0, y0, set_value, value_limit)
        else:
            LowLine(matrix, x0, y0, x1, y1, set_value, value_limit)
    else:
        if y0 > y1:
            HighLine(matrix, x1, y1, x0, y0, set_value, value_limit)
        else:
            HighLine(matrix, x0, y0, x1, y1, set_value, value_limit)
    matrix[y1, x1] -= set_value
    # if matrix[y1, x1] < 0:
    #     matrix[y1, x1] = 0
```

Перевіряємо роботу алгоритму

In [ ]:
```
test_area_matrix = np.zeros((10, 10))
fig, ax = plt.subplots()
print(test_area_matrix)
print("-----------------------------------")
Line(test_area_matrix, 0, 0, 5, 9)

ax.set_xticks(np.arange(0.5, 10, 1), labels="")
ax.set_yticks(np.arange(0.5, 10, 1), labels="")
plt.grid()
ax.plot([0, 5], [0, 9], marker = ".", markersize = 15, color = "red")
ax.imshow(test_area_matrix)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
-----------------------------------
```

Out[ ]: <matplotlib.image.AxesImage at 0x1a27480b7a0>



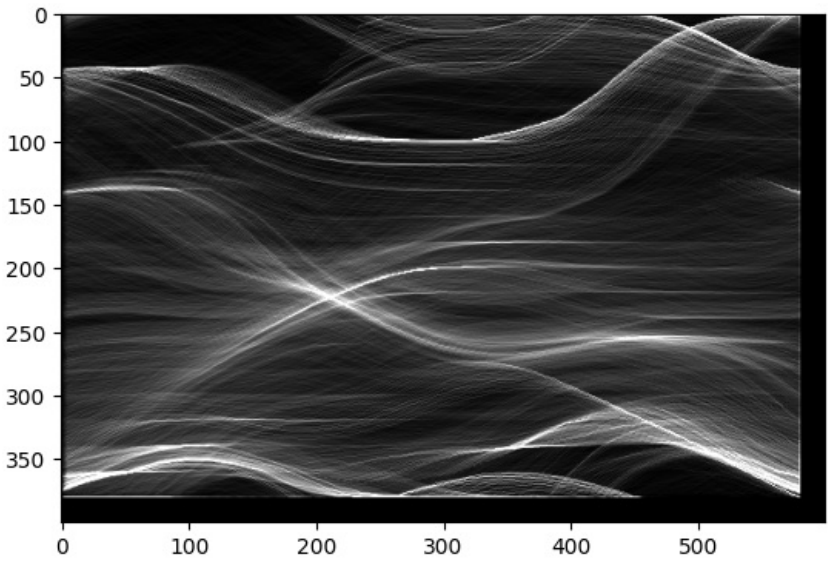
Малюємо лінії на зображенні

In [ ]:
```
z = np.linspace(0, 4, 256)
```

```
particles = []
population = 10_000

img_path_particles_line = np.zeros(window_shape[::-1])

for i in range(population):
    particles.append(Particle(window_shape[0]-cell_step, window_shape[1]-cell_step))

for z_i in z:
    perlin_noise = (perlin(x_grid/window_shape[0] * scale, y_grid/window_shape[1] * scale, z_i)+1)/2
    u, v = vectors(perlin_noise, y_window, x_window, 5, 1)

    scatter_x_= np.zeros(population)
    scatter_y_= np.zeros(population)

    for i in range(population):
        particles[i].follow(u, v, cell_step)
        # scatter_x_[i] = particles[i].x
        # scatter_y_[i] = particles[i].y
        # img_path_particles_line[int(particles[i].y), int(particles[i].x)] += 1
        Line(img_path_particles_line, int(particles[i].px), int(particles[i].py),
             int(particles[i].x), int(particles[i].y))

    # plt.scatter(scatter_x_, scatter_y_, s=1)
    # plt.quiver(x_grid, y_grid, u, v)
    # clear_output(wait=True)
    # plt.show()
    # time.sleep(.01)
```

In [ ]:
```
plt.imshow(img_path_particles_line, cmap="gray")
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a25602b470>
```



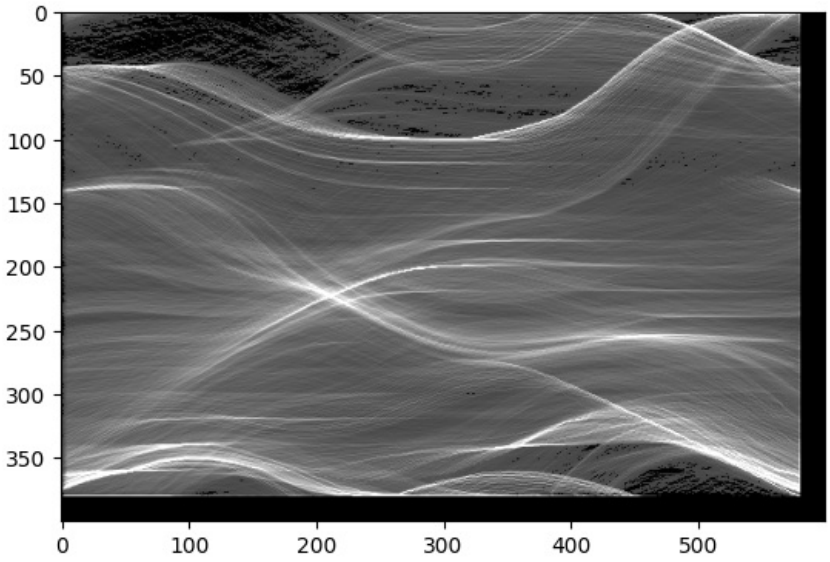
In [ ]:
```
# fig, ax = plt.subplots()

# z = np.linspace(0, 4, 256)

# particles = []
# population = 10_000

# img_path_particles_line = np.zeros(window_shape[::-1])

# for i in range(population):
#     particles.append(Particle(window_shape[0]-cell_step, window_shape[1]-cell_step))

# def anim_beuty_line_img(i):
#     ax.clear()

#     perlin_noise = (perlin(x_grid/window_shape[0] * scale,
#                            y_grid/window_shape[1] * scale, z[i])+1)/2
#     u, v = vectors(perlin_noise, x_window, y_window, 6, 1)

#     for i in range(population):
#         particles[i].follow(u, v, cell_step)
#         Line(img_path_particles_line, int(particles[i].x), int(particles[i].y),
#              int(particles[i].x - particles[i].vx), int(particles[i].y - particles[i].vy))

#     ax.imshow(img_path_particles_line, cmap="gray")
#     # ax.quiver(x_grid, y_grid, u, v, animated = True)
```

```
#     return ax

# func_anim_beuty_line = animation.FuncAnimation(fig, anim_beuty_line_img, repeat = True, frames = z.size-1, in
# writer = animation.PillowWriter(fps=15,
#                                 metadata=dict(artist='BerVol57'),
#                                 bitrate=1800)
# func_anim_beuty_line.save('gif/anim_beuty_line_img.gif', writer=writer)
```

In [ ]:
```
test_line = img_path_particles_line.copy()

print(np.max(test_line))
test_line /= np.max(test_line)
test_line[np.where(test_line>.03)] += 0.1
test_line /= np.max(test_line)
test_line[np.where(test_line>.03)] += 0.1
test_line /= np.max(test_line)
test_line[np.where(test_line>.03)] += 0.1

plt.imshow(test_line, cmap="gray")
```

```
255.0
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a25616e6f0>
```



Отже маємо вже набагато кращий результат

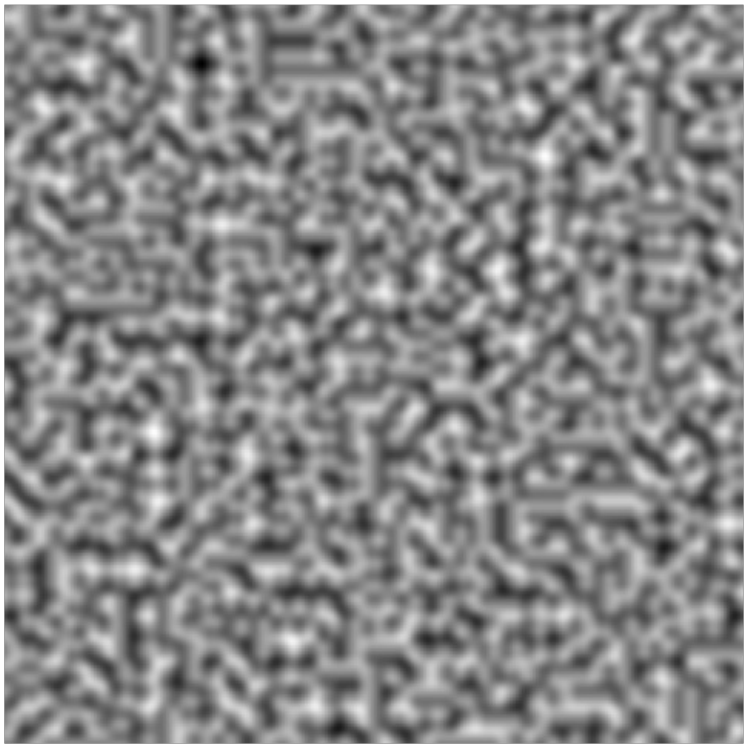
Фрактальний Броунівський рух

Результат звичайно вже набагато кращий, але спробуємо зробити лінії менш плавними. Для цього будемо використовувати FractalBrownianMotion

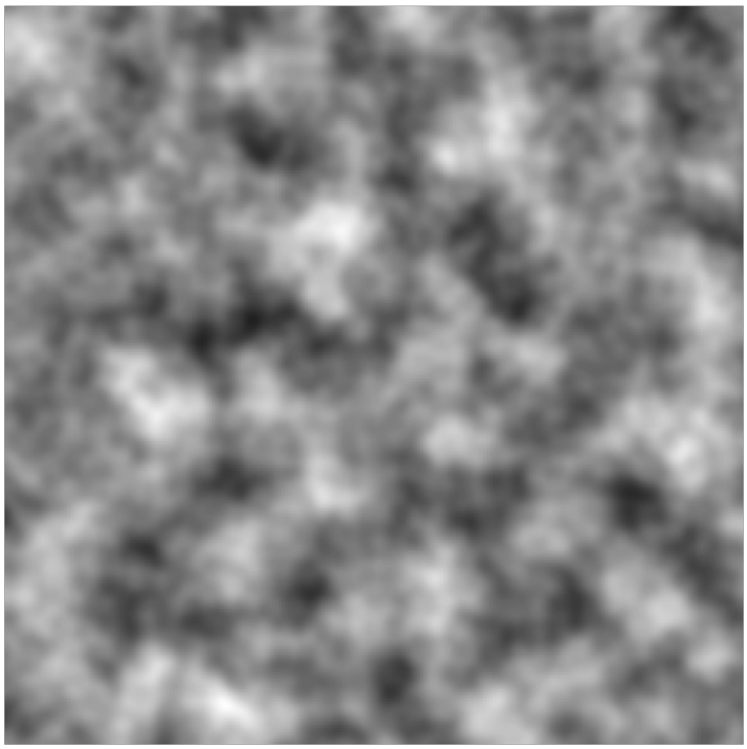
Ідея цього руху полягає в тому щоб дотати шуми Перліну з різними частотами, щоб отрмати менш плавні значення

+

+



=

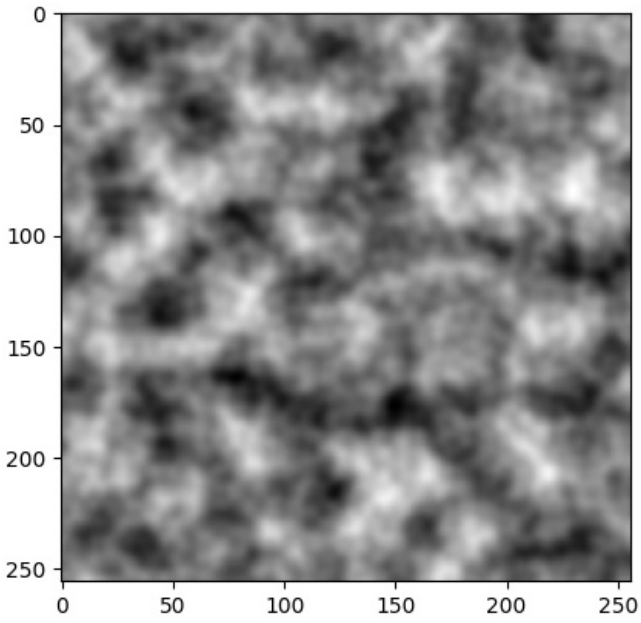


In [ ]:
```
def FractalBrownianMotion(gridspace_x, gridspace_y, numOctaves,
                          amplitude = 1, frequency = 1, z = 0, seed = 0):
    result = 0

    for _ in range(0, numOctaves):
        n = perlin(gridspace_x * frequency, gridspace_y * frequency, z = z, seed = seed) * amplitude
        result += n

        amplitude *= .5
        frequency *= 2.

    return result
```

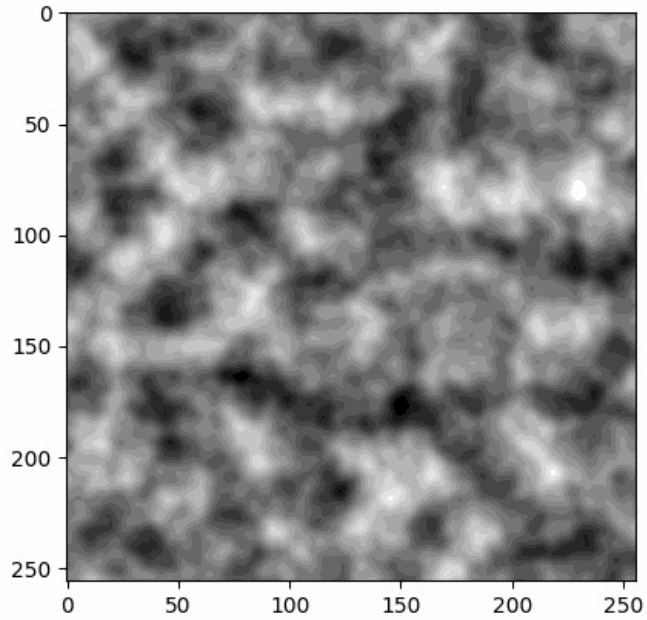
Приклад 2D

In [ ]:
```
fig, ax = plt.subplots()

test_space = np.linspace(0, 8, 256)
test_x, test_y = np.meshgrid(test_space, test_space)
def test_fractal_motion_func(i):
    ax.clear()
    test_fractal_motion = FractalBrownianMotion(test_x, test_y, z=test_space[i], numOctaves=3, seed=1)
```

```
    ax.imshow(test_fractal_motion, cmap="gray")
    return ax

func_test_fractal_motion_func = animation.FuncAnimation(fig, test_fractal_motion_func, repeat = True, frames =
writer = animation.PillowWriter(fps=15,
                                metadata=dict(artist='BerVol57'),
                                bitrate=1800)
func_test_fractal_motion_func.save('gif/func_test_fractal_motion_func.gif', writer=writer)
# print(test_fractal_motion.shape)
# plt.imshow(gif/func_test_fractal_motion_func, cmap="gray")
```



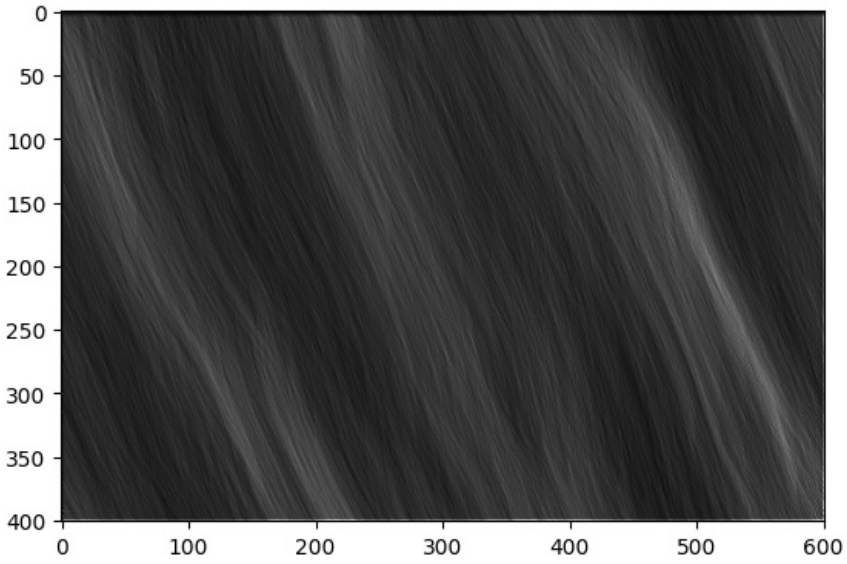
Приклад 3D



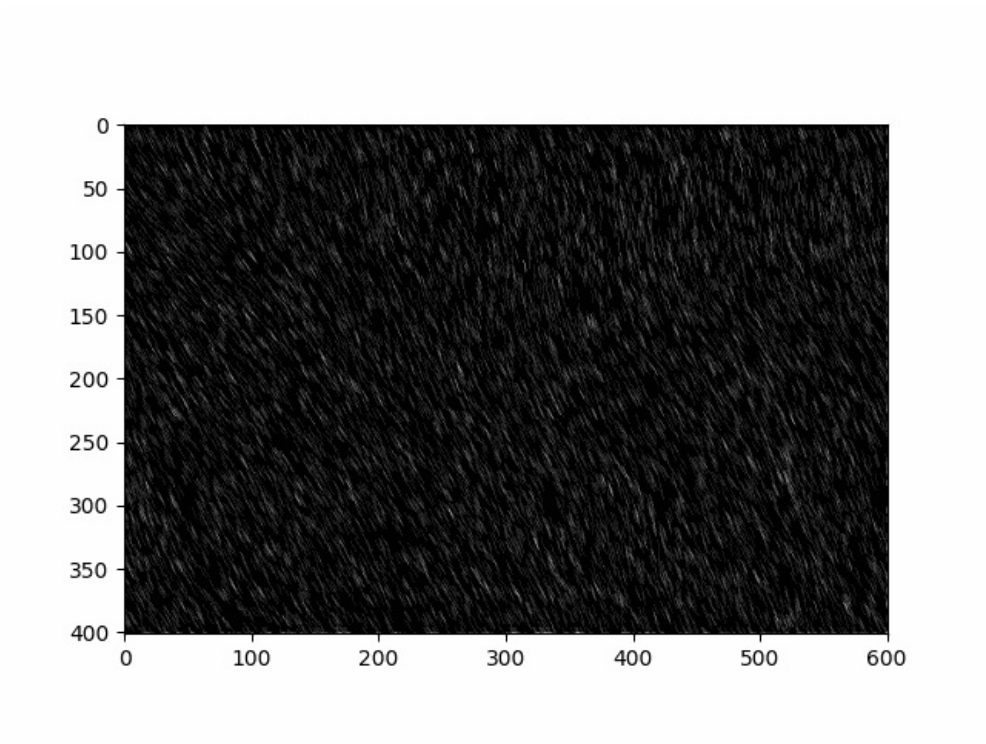
Малюємо вже з новим шумом

In [ ]:
```
fig, ax = plt.subplots()

z = np.linspace(0, 4, 256)

particles = []
population = 10_000

img_w = np.zeros((np.array(window_shape)+1)[::-1])

for i in range(population):
    particles.append(Particle(window_shape[0], window_shape[1]))

def anim_w(i):
```

```
    ax.clear()

    FractalBrownianMotion_noise = (FractalBrownianMotion(x_grid/window_shape[0] * scale,
                                                         y_grid/window_shape[1] * scale,
                                                         z = z[i], numOctaves = 8, seed=1011321)+1)/2
    # ax.imshow(FractalBrownianMotion_noise, cmap="gray")
    # return ax
    u, v = vectors(FractalBrownianMotion_noise,
                   x=x_window, y=y_window, length=5,
                   scale_vector_angle=1/4)

    # scatter_x_ = np.zeros(population)
    # scatter_y_ = np.zeros(population)

    for i in range(population):
        particles[i].follow(u, v, cell_step)
        # scatter_x_[i] = particles[i].x
        # scatter_y_[i] = particles[i].y
        Line(img_w, int(particles[i].px), int(particles[i].py),
             int(particles[i].x), int(particles[i].y))
        # ax.scatter(x_grid[int(particles[i].y//cell_step), int(particles[i].x//cell_step)],
        #            y_grid[int(particles[i].y//cell_step), int(particles[i].x//cell_step)],
        #            c="blue")


    # ax.scatter(scatter_x_, scatter_y_, s=1, c="red")
    ax.imshow(img_w, cmap="gray")
    # ax.quiver(x_grid, y_grid, u, v)
    return ax

func_anim_w = animation.FuncAnimation(fig, anim_w, repeat = True, frames = z.size-1, interval = 50)
writer = animation.PillowWriter(fps=15,
                                metadata=dict(artist='BerVol57'),
                                bitrate=1800)
func_anim_w.save('gif/anim_beuty_line_with_fractal_motion.gif', writer=writer)
```



Ось гіфка процесу малювання :)

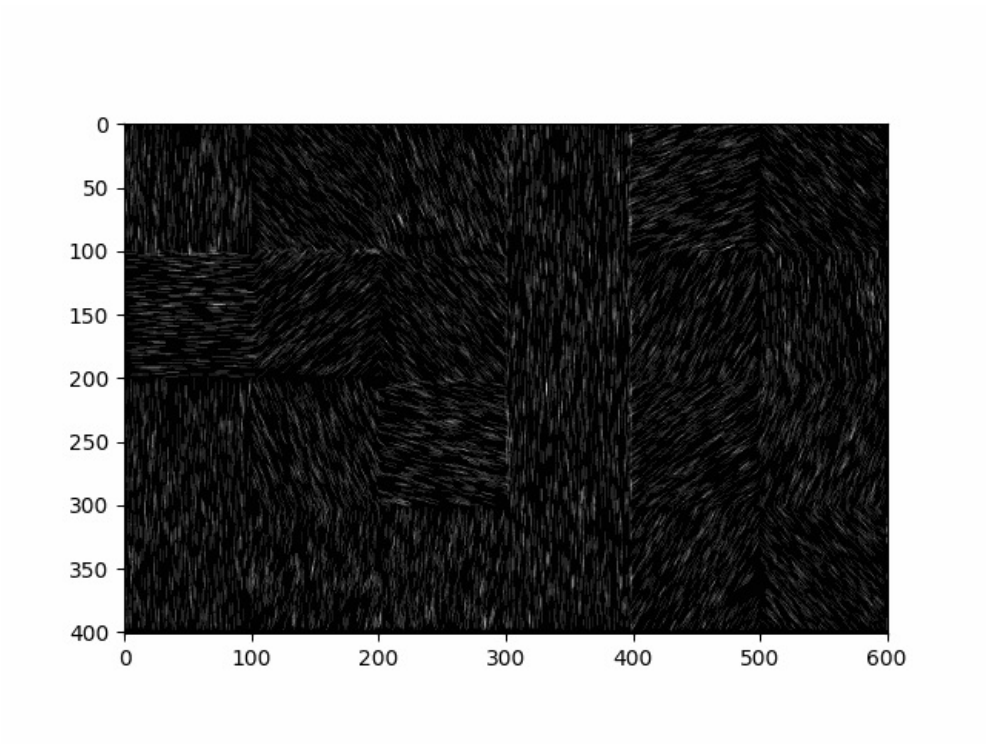


Майже кінець

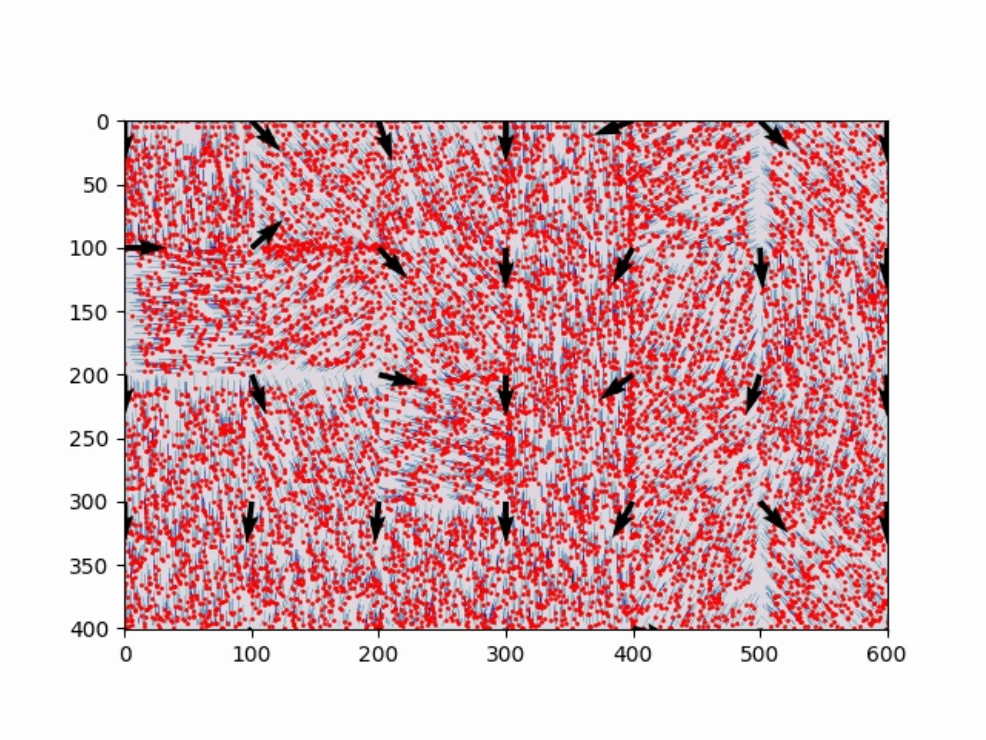
Підсумуємо увесь процес імітації псевдовипадкових ліній і запишемо все однією функцією, яка повертає рисунок в `grayscale`.

Отже на граничних пікселях зображення інколи з'являється артефакт у вигляді білої лінії, для того, щоб його забрати, просто забираємо ті пікселі, тому початкове зображення і полотно збільшуємо на крок векторів зі всіх сторін, а потім обрізаємо ці сторони

Формування білих ліній чітко видно ось тут



Тепер розглянемо із рухом частинок та із відображення векторів



Отже білі лнії формуються коли напрямки двох сусідніх векторів не є плавними, що викликає таке явище, коли частинки застрягають на межі дій цих векторів. Природа появи білих ліній на краях зображення та ж сама

`FlowFieldPseudoRandLine()`

- **`window_width`** : ширина вікна
- **`window_height`** : висота вікна
- **`vector_step`** : крок векторів
- **`noise_mode`** : вибір шума або звичайний Перліна, або із фрактальним рухом
- **`particles_population`** : кількість частинок
- **`set_value`** : значення, що буде додаватися як слід частинок
- **`zoom_noise_coef`** : коефіцієнт зближення площини шуму
- **`z_space`** : проміжок в межах якого буде розбиття `z`
- **`z_step`** : крок з яким буде розбито проміжок для `z`
- **`angle_scale`** : коефіцієнт повороту векторів
- **`vector_length`** : довжина векторів
- **`numOctaves`** : кількість остав для фрактального руху
- **`seed`** : насіння рандомності

In [ ]:
```
from typing import Literal

def FlowFieldPseudoRandLine(window_width: int, window_height: int, vector_step: int,
                            noise_mode: Literal["Perlin", "FractalBrownianMotion"], particles_population: int =
                            set_value: float = 1, zoom_noise_coef: float = 1., z_space: tuple = (0, 4),
                            z_step: int = 256, angle_scale: float = 1., vector_length: float = 5., numOctaves:

    x_window = np.arange(0, window_width + 2 * vector_step, vector_step)
    y_window = np.arange(0, window_height + 2 * vector_step, vector_step)

    x_grid, y_grid = np.meshgrid(x_window, y_window)

    particles = []

    img = np.zeros((window_height + 2 * vector_step,
                    window_width + 2 * vector_step))

    for _ in range(particles_population):
        particles.append(Particle(window_width + vector_step, window_height + vector_step))

    z = np.linspace(z_space[0], z_space[1], z_step)

    for z_i in range(z.size):

        if noise_mode == "Perlin":
            noise_map = (perlin(x_grid/window_width * zoom_noise_coef,
                                y_grid/window_height * zoom_noise_coef,
                                z=z[z_i], seed=seed) + 1) / 2
        elif noise_mode == "FractalBrownianMotion":
            noise_map = (FractalBrownianMotion(x_grid/window_width * zoom_noise_coef,
```

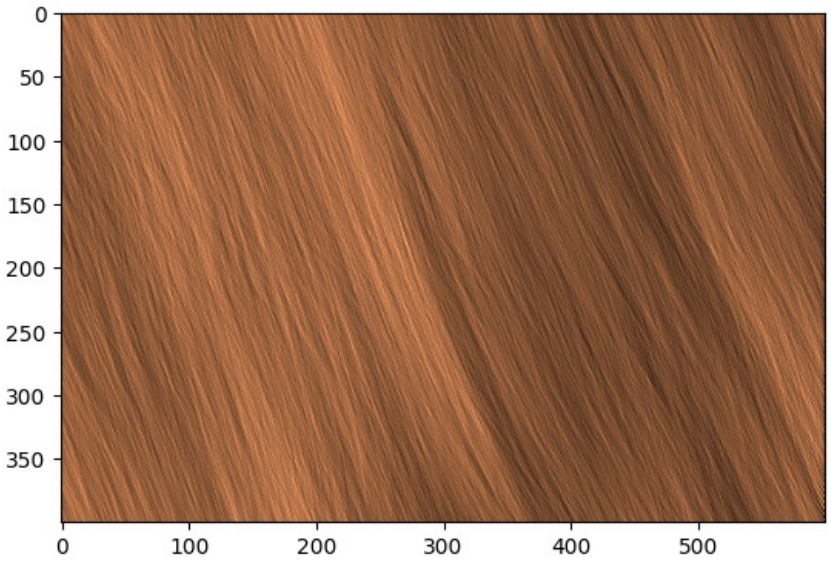
```
                                            y_grid/window_height * zoom_noise_coef,
                                            z=z[z_i], numOctaves=numOctaves, seed=seed) + 1) / 2
        else:
            raise f"Invalid noise mode: {noise_mode}"

        u, v = vectors(noise_map=noise_map, x=x_window, y=y_window,
                       length=vector_length, scale_vector_angle=angle_scale)

        for i in range(particles_population):
            particles[i].follow(u, v, vector_step)

            Line(img, int(particles[i].px), int(particles[i].py),
                 int(particles[i].x), int(particles[i].y), set_value=set_value)

    # img = img.clip(0, 255)
    img = img.astype("uint8")

    return img[vector_step:window_height + vector_step,
               vector_step:window_width + vector_step]
```

In [ ]:
```
FlowFieldPseudoRandLine_example = \
    FlowFieldPseudoRandLine(600, 400, 20,
                            "FractalBrownianMotion",
                            vector_length=5, angle_scale=1/4)
```

Використовуючи різні палітри можна отримати більш корисніші/гарніші результати

In [ ]:
```
plt.imshow(FlowFieldPseudoRandLine_example, cmap="copper")
FlowFieldPseudoRandLine_example.shape
```

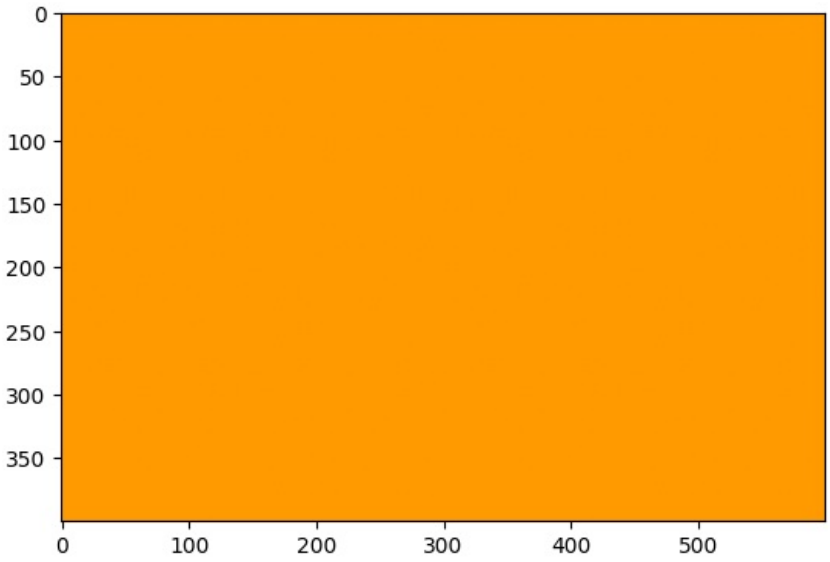
Out[ ]:
```
(400, 600)
```



Або ж можна накласти на зображення як значення кольору в *HSV*

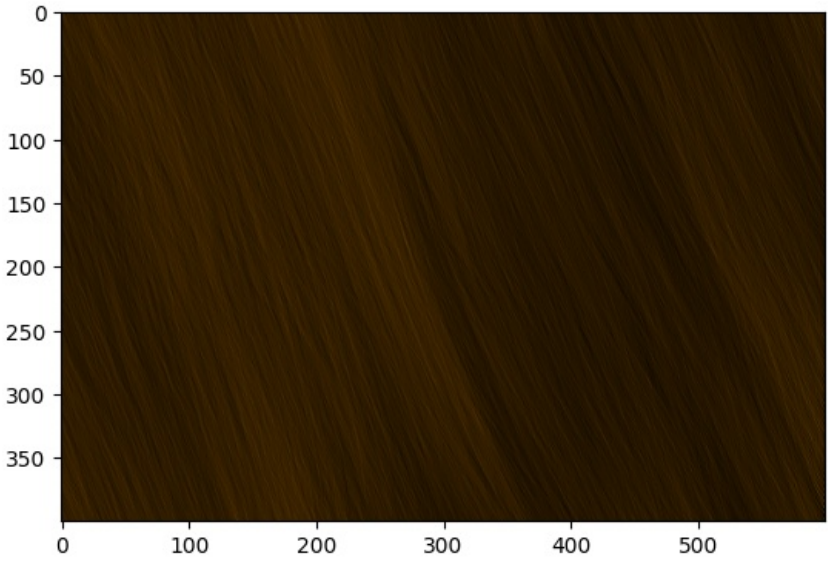
In [ ]:
```
wood_img = np.zeros((400, 600, 3))
wood_img[:, :, 0] = 0/255
wood_img[:, :, 1] = 154/255
wood_img[:, :, 2] = 255/255

plt.imshow(wood_img[:, :, ::-1])
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1a26d7a45c0>
```



In [ ]:
```
hsv_wood_img = cv.cvtColor((wood_img*255).astype("uint8"), cv.COLOR_BGR2HSV)
hsv_wood_img[:, :, 2] = FlowFieldPseudoRandLine_example
return_to_bgr = cv.cvtColor(hsv_wood_img, cv.COLOR_HSV2BGR)

plt.imshow(return_to_bgr[:, :, ::-1])
```

Out[ ]: `<matplotlib.image.AxesImage at 0x1a26d720200>`



Джерела, що використовувалися

- **Perlin noise and Fractal Brwonian Motion** | YT video | medium | Blog
- **Bresenham's line algorithm**
- **Flow Field**

Кінець) Дякую за увагу