Групповой проект. 3 этап

Электрический пробой. Программа

Астафьева Анна Андреевна
Коломиец Мария Владимировна
Жиронкин Павел Владимирович
Паландузян Артем Карапетович
Сурнаков Александр Васильевич
Евдокимова Юлия Константиновна
Группа: НПИбд-01-18

Содержание

Список таблиц

Список иллюстраций

1 Цели и задачи

Цель работы: реализация программы по алгоритму, сотавленному на прошлом этапе для моделирования роста стримерной структуры при электрическом пробое.

Задачи:

1. Реализовать в геометрии «острие – плоскость» однозвенную модель со степенной зависимостью вероятности роста от напряженности поля $p \sim E^{\eta}$.
2. Рассмотреть изменение геометрии стримерной структуры для случаев $\eta = 0, 1, 2$.

2 Реализация алгоритма в программе

Реализовывать алгоритм мы решили на языке Python, с использованием графического модуля graphicps для наглядного изображения пробоя.

2.1 Вычисление потенциала

1. Задаем квадратную сетку 50x50 в качесте области моделирования:

```
# Размер матрицы и размер каждого узла на UI
n, m, nmSize = 50, 50, 8
```

2. Задаем произвольные значения потенциала для внутренних узлов области:

```
# Генерация матрицы потенциалов
#matF = [[rd.randint(1, 50) / 100 for j in range(m)] for i in range(n)]
matF = [[0 for j in range(m)] for i in range(n)]
```

3. Узлам, примыкающим к границе, задаем значение потенциала, равное значению потенциала границы (0 для верхней границы, 1 для нижней для простоты вычислений):

```
# По краям потенциал укажем как 0, а снизу 1
for i in range(0, len(matF)):
matF[i][0],matF[0][i],matF[-1][i],matF[i][-1] = 0,0,0,1
```

4. Вычисляем новые значения потенциала во всех узлах.
   Для вычисления потенциала мы используем формулу:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1})$$

И итерационно просчитываем потенциал для каждого узла сетки (рис. 2.1):

```
# Функция, которая рассчитает матрицу потенциалов итерационным способом с точностью d_min
def getNewMat(mat, streamer, d_min):
    d = 10.0 # Рандомное число больше, чем d_min
    while (d > d_min):
        sum = 0.0 # Сумма отличий в новой и старой матрице
        temp_mat = copy.deepcopy(mat) # Новую матрицу рассчитываем и храним во временной переменной

        # Значение каждого узла становится средним арифметическим из этого узела и восьми прилегающих
        for i in range(1, len(temp_mat) - 1):
            for j in range(1, len(temp_mat[i]) - 1):
                temp_mat[i][j] = (mat[i][j] + mat[i - 1][j - 1] + mat[i - 1][j] + mat[i - 1][j + 1] + mat[i][j - 1] + mat
[i][j + 1] + mat[i + 1][j - 1] + mat[i + 1][j] + mat[i + 1][j + 1]) / 9
                sum += abs(temp_mat[i][j] - mat[i][j])

        d = sum / (len(mat) * len(mat[0])) # Находим среднее арифметическое разницы по всем узлам

        # В точках стримеров меняем усредненное значение на 0
        for i in range(len(streamer)):
            temp_mat[streamer[i][0]][streamer[i][1]] = 0

        mat = copy.deepcopy(temp_mat) # Записываем данные из временной матрицы в основную
    return mat
```

Рис. 2.1: Функция вычисления потенциала

5. Повторяем пункт 4. пока потенциалы не перестанут изменяться.

В результате получаем потенциал (рис. 2.2):

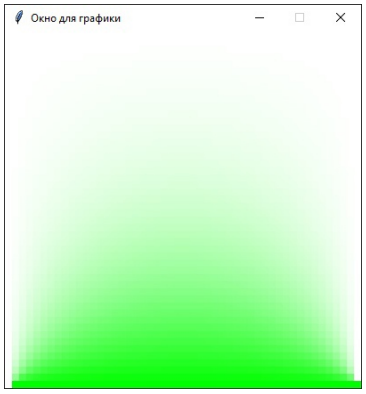


Рис. 2.2: Потенциал поля

Посчитав потенциал поля перейдем к моделированию разряда по модели НПВ.

1. Пробиваем первый узел:

sX, sY = int(n / 2), 10 # Начальная точка стримера

2. Пробегаем по всем узлам, в которые возможем рост и считаем сумму вероятностей роста по формуле:

$$Z = \sum_{k=1}^{M} E_k^{\eta}$$

Здесь $|E| = \phi_B$ (для горизонтальных и вертикальных звеньев),
$|E| = \phi_B/\sqrt{2}$ (для диагональных).

 # Находим сумму всех вероятностей

for i in range(0, len(matProb)):

for j in range(0, len(matProb[i])):

sum+=matProb[i][j]

3. Разыгрываем случайное число ξ, равномерно распределенное от 0 до Z:

 # Берем эту сумму за 100% и генерируем случайное число от 0 до этой суммы

randval = rd.randint(0,100) / 100 ∗ sum

4. Затем повторно шаг за шагом рассчитывается сумма до тех пор, пока текущая сумма не станет больше ξ. Тот узел, для которого сумма стала больше ξ, присоединяется к структуре (рис. 2.3):

```
    # В зависимости от случайного значения randval выбираем, каким будет новый узел стримера. Когда находим, выходим из
цикла
    for i in range(0, len(matProb)):
        for j in range(0, len(matProb[i])):
            randval-=matProb[i][j]
            if randval < 0:
                oX = i
                oY = j
                break
        if randval < 0:
            break
    # Добавляем узел в конец списка с узлами стримера
    streamer.append([oX,oY])

    # Указываем нулевой потенциал для этого узла
    matF[oX][oY] = 0
```

Рис. 2.3: Присоединение нового узла

5. Пересчитываем поле (п. 4-5):

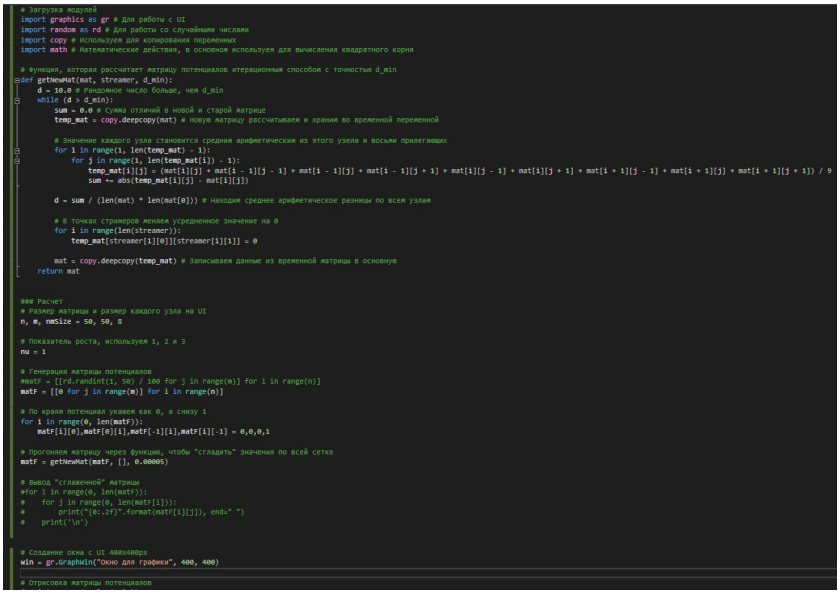
 # Перерассчитываем потенциал после добавления нового узла к стримеру

matF = copy.deepcopy(getNewMat(matF, streamer,0.005))

6. Повторяем пункты 2-5, пока узор не достигнет границы.

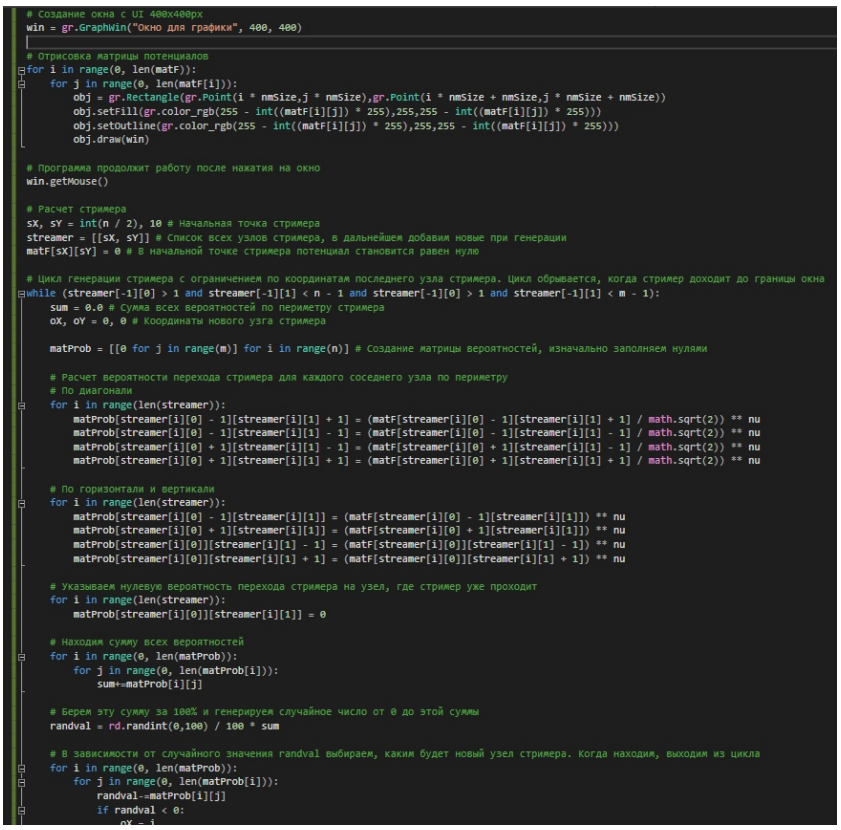
2.2 Программа

В итоге мы получили программу (рис. 2.4, 2.5, 2.6):



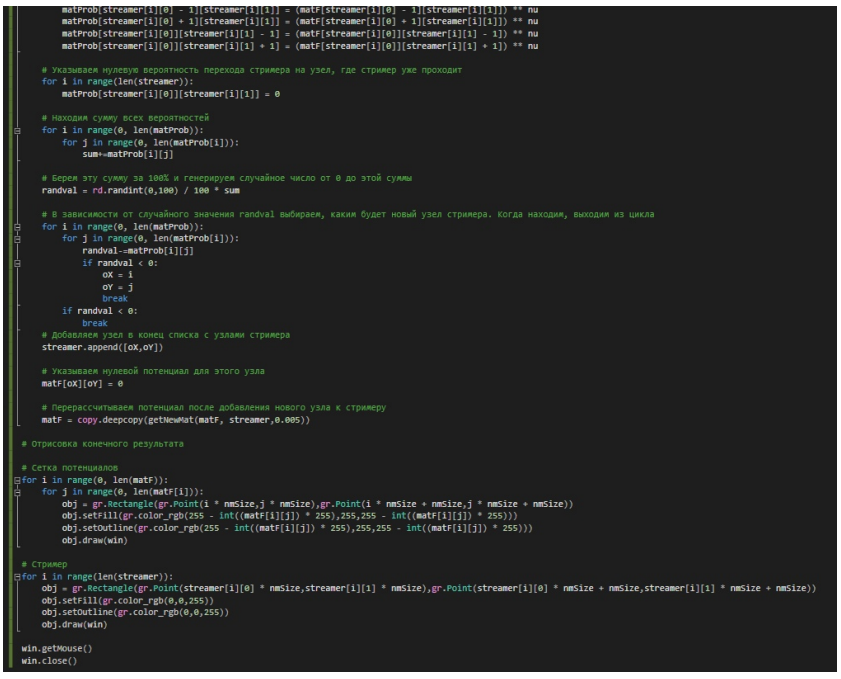
```
# Загрузка модулей
import graphics as gr # Для работы с UI
import random as rd # Для работы со случайными числами
import copy # Используем для копирования переменных
import math # Математические действия, в основном используем для вычисления квадратного корня

# Функция, которая рассчитает матрицу потенциалов итерационным способом с точностью d_min
def getNewMat(mat, streamer, d_min):
    d = 10.0 # Рандомное число больше, чем d_min
    while (d > d_min):
        sum = 0.0 # Сумма отличий в новой и старой матрице
        temp_mat = copy.deepcopy(mat) # Новую матрицу рассчитываем и храним во временной переменной

        # Значение каждого узла становится средним арифметическим из этого узла и восьми прилегающих
        for i in range(1, len(temp_mat) - 1):
            for j in range(1, len(temp_mat[i]) - 1):
                temp_mat[i][j] = (mat[i][j] + mat[i - 1][j - 1] + mat[i - 1][j] + mat[i - 1][j + 1] + mat[i][j - 1] + mat[i][j + 1] + mat[i + 1][j - 1] + mat[i + 1][j] + mat[i + 1][j + 1]) / 9
                sum += abs(temp_mat[i][j] - mat[i][j])

        d = sum / (len(mat) * len(mat[0])) # Находим среднее арифметическое разницы по всем узлам

        # В точках стримеров меняем усредненное значение на 0
        for i in range(len(streamer)):
            temp_mat[streamer[i][0]][streamer[i][1]] = 0

        mat = copy.deepcopy(temp_mat) # Записываем данные из временной матрицы в основную
    return mat


### Расчет
# Размер матрицы и размер каждого узла на UI
n, m, nmSize = 50, 50, 8

# Показатель роста, используем 1, 2 и 3
nu = 1

# Генерация матрицы потенциалов
#matF = [[rd.randint(1, 50) / 100 for j in range(m)] for i in range(n)]
matF = [[0 for j in range(m)] for i in range(n)]

# По краям потенциал укажем как 0, а снизу 1
for i in range(0, len(matF)):
    matF[i][0],matF[0][i],matF[-1][i],matF[i][-1] = 0,0,0,1

# Прогоняем матрицу через функцию, чтобы "сгладить" значения по всей сетке
matF = getNewMat(matF, [], 0.00005)

# Вывод "сглаженной" матрицы
#for i in range(0, len(matF)):
#    for j in range(0, len(matF[i])):
#        print("{0:.2f}".format(matF[i][j]), end=" ")
#    print('\n')

# Создание окна с UI 400x400px
win = gr.GraphWin("Окно для графики", 400, 400)

# Отрисовка матрицы потенциалов
```

Рис. 2.4: Программа



```
# Создание окна с UI 400x400px
win = gr.GraphWin("Окно для графики", 400, 400)

# Отрисовка матрицы потенциалов
for i in range(0, len(matF)):
    for j in range(0, len(matF[i])):
        obj = gr.Rectangle(gr.Point(i * nmSize,j * nmSize),gr.Point(i * nmSize + nmSize,j * nmSize + nmSize))
        obj.setFill(gr.color_rgb(255 - int((matF[i][j]) * 255),255,255 - int((matF[i][j]) * 255)))
        obj.setOutline(gr.color_rgb(255 - int((matF[i][j]) * 255),255,255 - int((matF[i][j]) * 255)))
        obj.draw(win)

# Программа продолжит работу после нажатия на окно
win.getMouse()

# Расчет стримера
sX, sY = int(n / 2), 10 # Начальная точка стримера
streamer = [[sX, sY]] # Список всех узлов стримера, в дальнейшем добавим новые при генерации
matF[sX][sY] = 0 # В начальной точке стримера потенциал становится равен нулю

# Цикл генерации стримера с ограничением по координатам последнего узла стримера. Цикл обрывается, когда стример доходит до границы окна
while (streamer[-1][0] > 1 and streamer[-1][1] < n - 1 and streamer[-1][0] > 1 and streamer[-1][1] < m - 1):
    sum = 0.0 # Сумма всех вероятностей по периметру стримера
    oX, oY = 0, 0 # Координаты нового узла стримера

    matProb = [[0 for j in range(m)] for i in range(n)] # Создание матрицы вероятностей, изначально заполняем нулями

    # Расчет вероятности перехода стримера для каждого соседнего узла по периметру
    # По диагонали
    for i in range(len(streamer)):
        matProb[streamer[i][0] - 1][streamer[i][1] + 1] = (matF[streamer[i][0] - 1][streamer[i][1] + 1] / math.sqrt(2)) ** nu
        matProb[streamer[i][0] - 1][streamer[i][1] - 1] = (matF[streamer[i][0] - 1][streamer[i][1] - 1] / math.sqrt(2)) ** nu
        matProb[streamer[i][0] + 1][streamer[i][1] - 1] = (matF[streamer[i][0] + 1][streamer[i][1] - 1] / math.sqrt(2)) ** nu
        matProb[streamer[i][0] + 1][streamer[i][1] + 1] = (matF[streamer[i][0] + 1][streamer[i][1] + 1] / math.sqrt(2)) ** nu

    # По горизонтали и вертикали
    for i in range(len(streamer)):
        matProb[streamer[i][0] - 1][streamer[i][1]] = (matF[streamer[i][0] - 1][streamer[i][1]]) ** nu
        matProb[streamer[i][0] + 1][streamer[i][1]] = (matF[streamer[i][0] + 1][streamer[i][1]]) ** nu
        matProb[streamer[i][0]][streamer[i][1] - 1] = (matF[streamer[i][0]][streamer[i][1] - 1]) ** nu
        matProb[streamer[i][0]][streamer[i][1] + 1] = (matF[streamer[i][0]][streamer[i][1] + 1]) ** nu

    # Указываем нулевую вероятность перехода стримера на узел, где стример уже проходит
    for i in range(len(streamer)):
        matProb[streamer[i][0]][streamer[i][1]] = 0

    # Находим сумму всех вероятностей
    for i in range(0, len(matProb)):
        for j in range(0, len(matProb[i])):
            sum+=matProb[i][j]

    # Берем эту сумму за 100% и генерируем случайное число от 0 до этой суммы
    randval = rd.randint(0,100) / 100 * sum

    # В зависимости от случайного значения randval выбираем, каким будет новый узел стримера. Когда находим, выходим из цикла
    for i in range(0, len(matProb)):
        for j in range(0, len(matProb[i])):
            randval-=matProb[i][j]
            if randval < 0:
                oX = i
```

Рис. 2.5: Программа



```
        matProb[streamer[i][0] - 1][streamer[i][1]] = (matF[streamer[i][0] - 1][streamer[i][1]]) ** nu
        matProb[streamer[i][0] + 1][streamer[i][1]] = (matF[streamer[i][0] + 1][streamer[i][1]]) ** nu
        matProb[streamer[i][0]][streamer[i][1] - 1] = (matF[streamer[i][0]][streamer[i][1] - 1]) ** nu
        matProb[streamer[i][0]][streamer[i][1] + 1] = (matF[streamer[i][0]][streamer[i][1] + 1]) ** nu

    # Указываем нулевую вероятность перехода стримера на узел, где стример уже проходит
    for i in range(len(streamer)):
        matProb[streamer[i][0]][streamer[i][1]] = 0

    # Находим сумму всех вероятностей
    for i in range(0, len(matProb)):
        for j in range(0, len(matProb[i])):
            sum+=matProb[i][j]

    # Берем эту сумму за 100% и генерируем случайное число от 0 до этой суммы
    randval = rd.randint(0,100) / 100 * sum

    # В зависимости от случайного значения randval выбираем, каким будет новый узел стримера. Когда находим, выходим из цикла
    for i in range(0, len(matProb)):
        for j in range(0, len(matProb[i])):
            randval-=matProb[i][j]
            if randval < 0:
                oX = i
                oY = j
                break
        if randval < 0:
            break
    # Добавляем узел в конец списка с узлами стримера
    streamer.append([oX,oY])

    # Указываем нулевой потенциал для этого узла
    matF[oX][oY] = 0

    # Перерассчитываем потенциал после добавления нового узла к стримеру
    matF = copy.deepcopy(getNewMat(matF, streamer,0.005))

# Отрисовка конечного результата

# Сетка потенциалов
for i in range(0, len(matF)):
    for j in range(0, len(matF[i])):
        obj = gr.Rectangle(gr.Point(i * nmSize,j * nmSize),gr.Point(i * nmSize + nmSize,j * nmSize + nmSize))
        obj.setFill(gr.color_rgb(255 - int((matF[i][j]) * 255),255,255 - int((matF[i][j]) * 255)))
        obj.setOutline(gr.color_rgb(255 - int((matF[i][j]) * 255),255,255 - int((matF[i][j]) * 255)))
        obj.draw(win)

# Стример
for i in range(len(streamer)):
    obj = gr.Rectangle(gr.Point(streamer[i][0] * nmSize,streamer[i][1] * nmSize),gr.Point(streamer[i][0] * nmSize + nmSize,streamer[i][1] * nmSize + nmSize))
    obj.setFill(gr.color_rgb(0,0,255))
    obj.setOutline(gr.color_rgb(0,0,255))
    obj.draw(win)

win.getMouse()
win.close()
```

Рис. 2.6: Программа

2.3 Результат работы программы

При вероятности с показателем роста $\eta = 1$:

$p \sim E$

Получаем стримерную структуру (рис. 2.7):

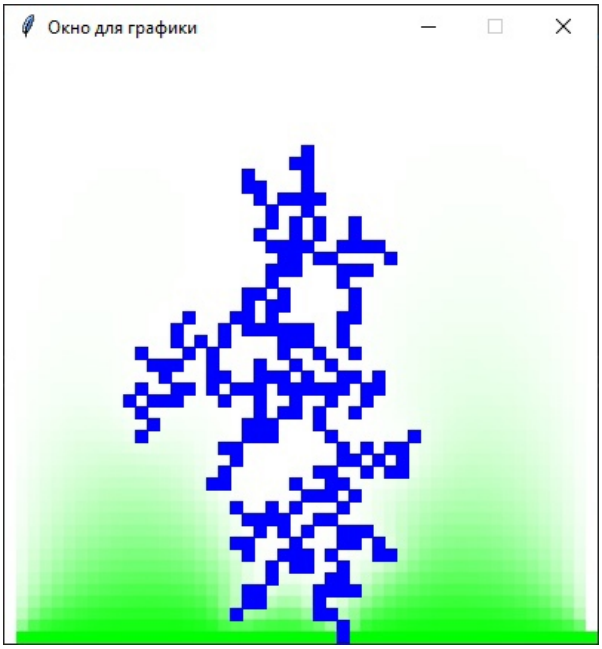


Рис. 2.7: Рост стримерной структуры при электрическом пробое

2.4 Изменение геометрии стримерной структуры в зависимости от показателя роста η

Рассмотрим изменения стримерной структуры при увеличении показателя роста η.

2.4.1 1. $\eta = 1$ (рис. 2.8, 2.9, 2.9):

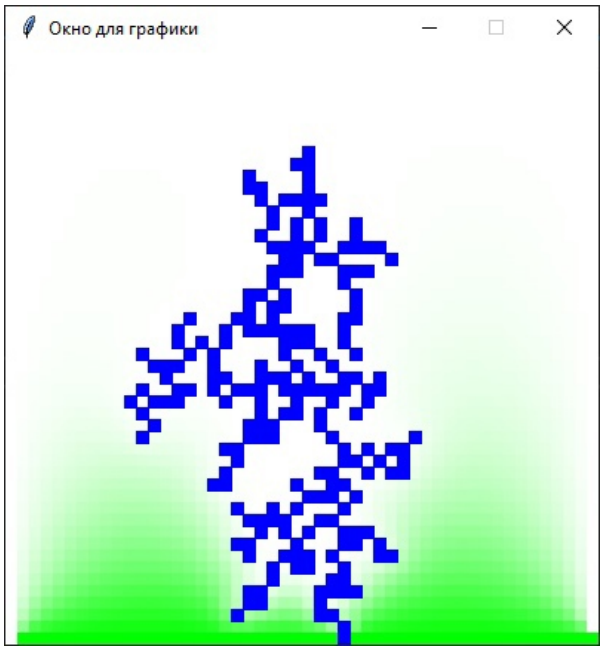


Рис. 2.8: Рост стримерной структуры при $\eta = 1$

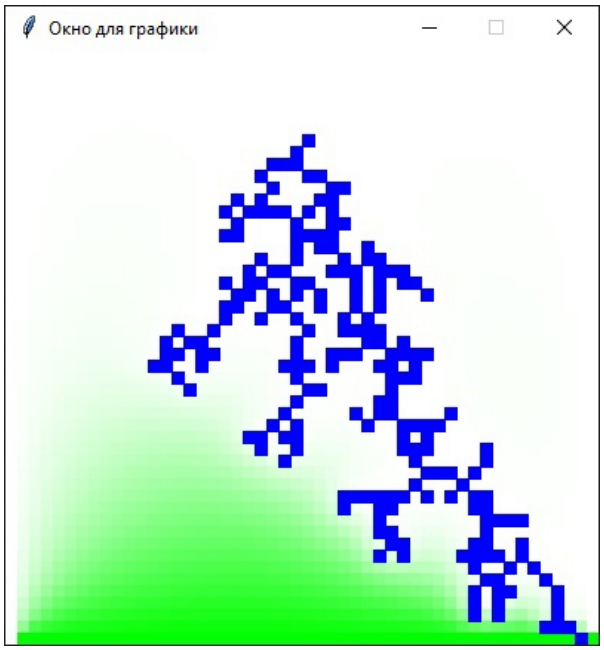


Рис. 2.9: Рост стримерной структуры при $\eta = 1$

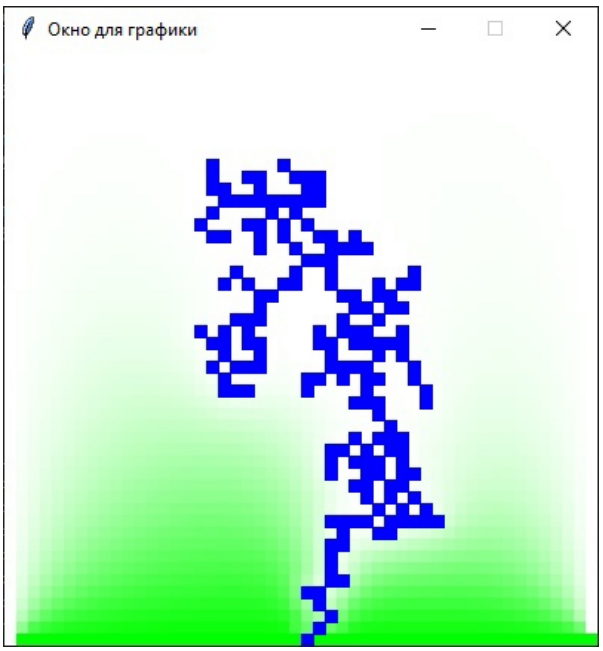


Рис. 2.10: Рост стримерной структуры при $\eta = 1$

2.4.2 2. $\eta = 2$ (рис. 2.11):

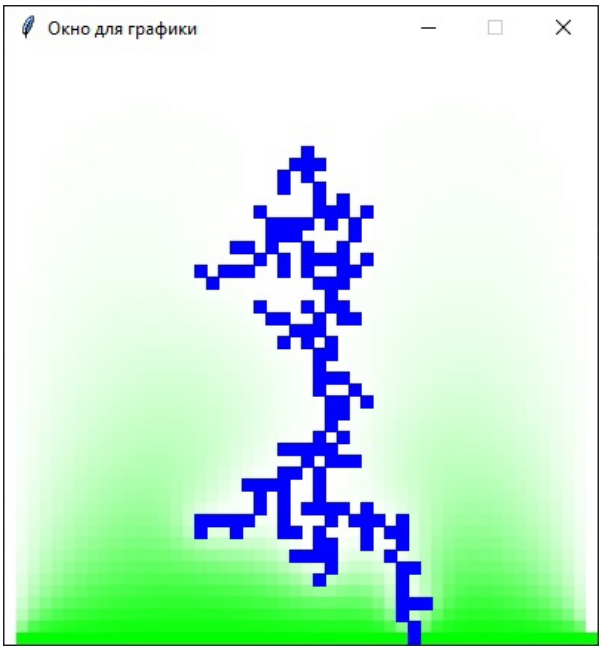


Рис. 2.11: Рост стримерной структуры при $\eta = 2$

2.4.3 3. $\eta = 3$ (рис. 2.12):

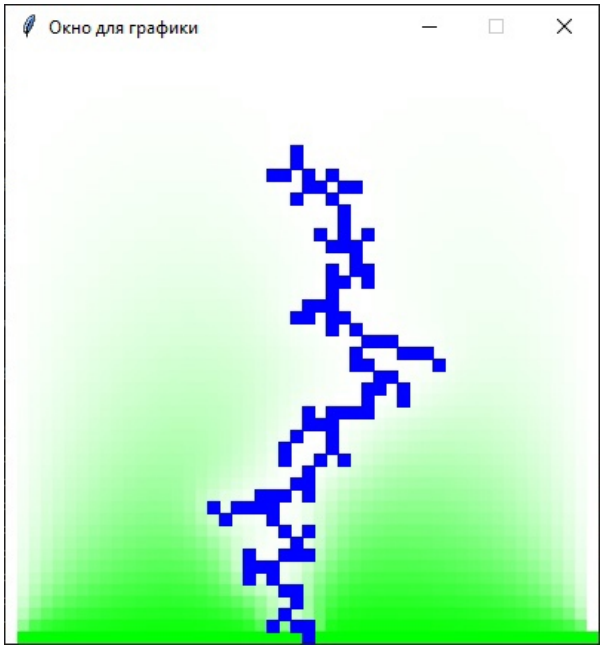


Рис. 2.12: Рост стримерной структуры при $\eta = 3$

2.4.4 4. $\eta = 4$ (рис. 2.13):

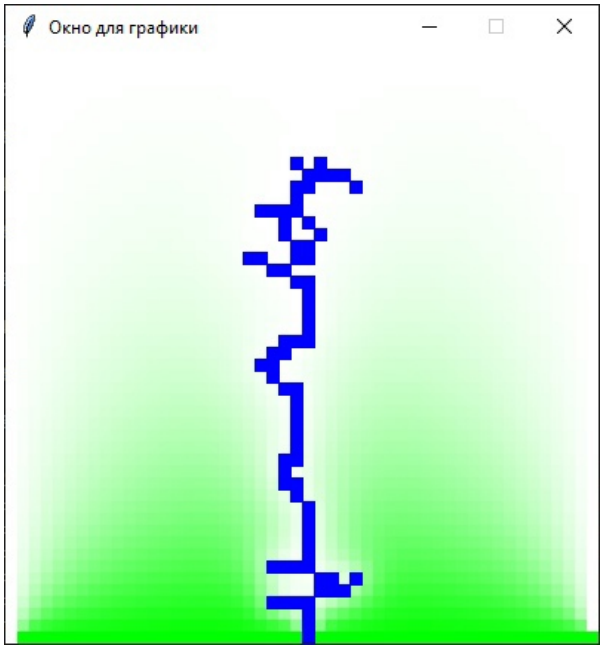


Рис. 2.13: Рост стримерной структуры при $\eta = 4$

При увеличении η уменьшается ветвистость стримерной структуры.

3 Вывод

Написана программа реализующая в геометрии «острие – плоскость» однозвенную модель со степенной зависимостью вероятности роста от напряженности поля $p \sim E^{\eta}$.

Рассмотренно изменение геометрии стримерной структуры для случаев $\eta = 1, 2, 3, 4$: при увеличении η уменьшается ветвистость стримерной структуры.

4 Список литературы

1. Д. А. Медведев, А. Л. Куперштох, Э. Р. Прууэл, Н. П. Сатонкина, Д. И. Карпов - МОДЕЛИ-РОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ И ЯВЛЕНИЙ НА ПК
2. Niemeyer L., Pietronero L., Wiesmann H. J. Fractal dimension of dielectric breakdown // Physical Review Letters. 1984. V. 52, N 12. P. 1033–1036
3. Biller P. Fractal streamer models with physical time // Proc. 11th Int. Conf. on Conduction and Breakdown in Dielectric Liquids, IEEE N 93CH3204-5. Baden-D¨attwil, Switzerland, 1993. P. 199–203.