f

Aufgabe 2: Verzinkt

Team-ID: 00225

Team-Name: Simon und Benjamins Teams

Bearbeiter/-innen dieser Aufgabe:
Samuele Bradke

20. November 2022

Inhaltsverzeichnis

1 Lösungsidee

In der Aufgabenstellung gab es bereits eine Anspielung darauf, dass man das in einem Array Simulieren könnte. Also habe ich das gemacht. Die Idee ist am Anfang der Simulation die Anzahl der gewünschten Keime schon zu generieren. Diese Keime würden aber erst an zufälligen Zeiten anfangen sich auszubreiten. Das Programm sollte dann immer ein Bild beziehungsweise ein Array generieren, und so schrittweiße bis zum endergebniss kommen. Bei jedem dieser Schritte sollen die Keime, deren Zeit schon gekommen ist weiter wachsen. Das soll das Programm so lange wiederholen, bis es keinen weißen Fleck mehr auf dem ganzen Bild gibt. Dann ist die Simulation fertig. Dann muss das Array noch in ein Bild umgewandelt werden, das wird als Endergebnisse gespeichert.

2 Umsetzung

Um das zu implementieren, bin ich folgt vorgegangen. Zuerst wird abgefragt, wie viele Pixel in Höhe und Weite simuliert und wie viele Keime generiert werden sollen. Außerdem wird die Zeit Variable auf 0 gestellt. Dann erstellt das Programm ein dreidimensionales Array der eingegebenen Größe. Die ersten zwei Dimensionen werden dazu genutzt, um die Position der Keime festzustellen. Die dritte Dimension speichert die Farbe und Ausbreitungsgeschwindigkeit in die verschiedenen Richtungen des Keimes. Vor jedem nächsten Schritt überprüft das Programm, ob es noch leere, also weiße, Zellen gibt. Wenn nicht wird das Array in ein zweidimensionales Array umgewandelt, wobei die Ausbreitungsgeschwindigkeit verloren werdne. Dieses Array kann dann in ein Bild umgewandelt werden.

3 Beispiele

Das hier sind einige Beispiele die ich Generiert habe. Abbildung 1 wurde in 1080*1920 px Simuliert. Die anderen Bilder wurden in 1024*768 px simuliert, wie das Beispielbild. Die Bilder unterscheiden sich jedoch auch in Anzahl der Keime, und dem Raum zwischen dem die Ausbreitungsgeschwindigkeiten lagen.

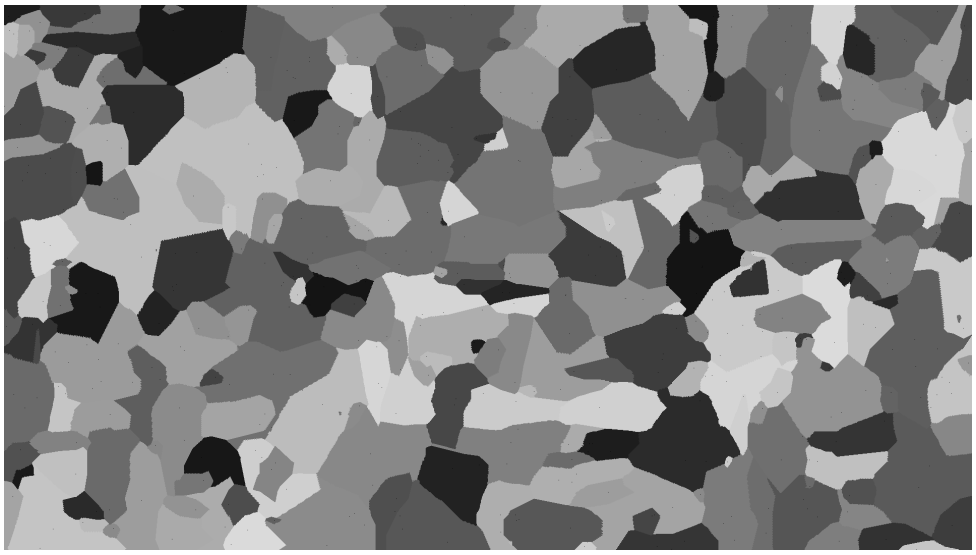

Abbildung 1: 1080*1920 mit 500 Keimen und 1-10 px pro Schritt

Abbildung 2: 1024*768 mit 50 Keimen und 1-10 px pro Schritt

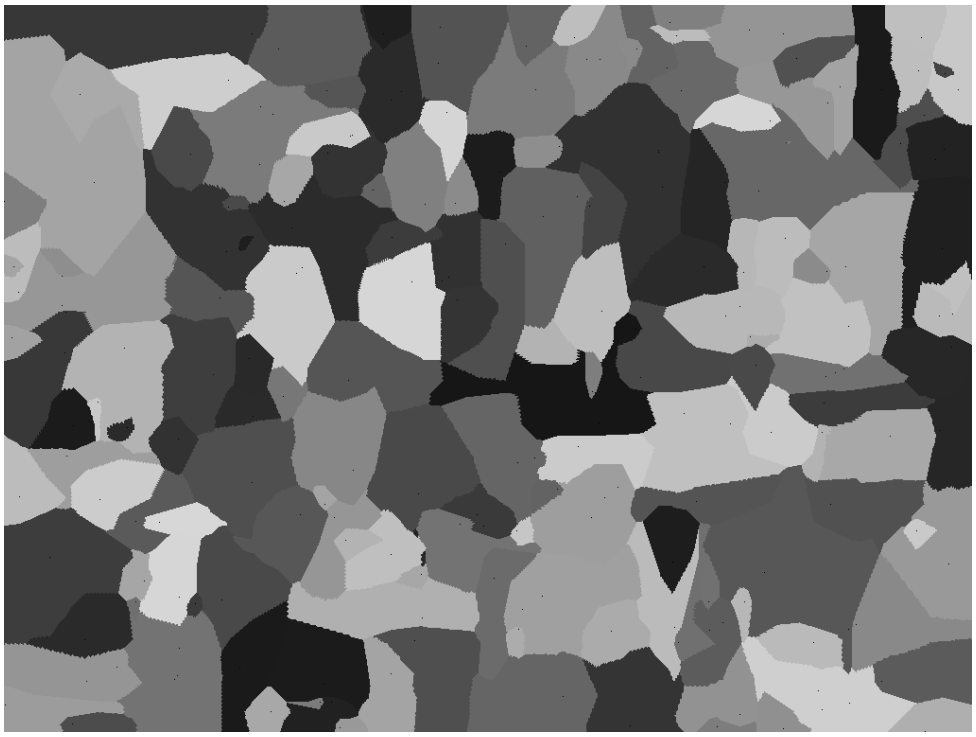

Abbildung 3: 1024*768 mit 200 Keimen und 1-10 px pro Schritt

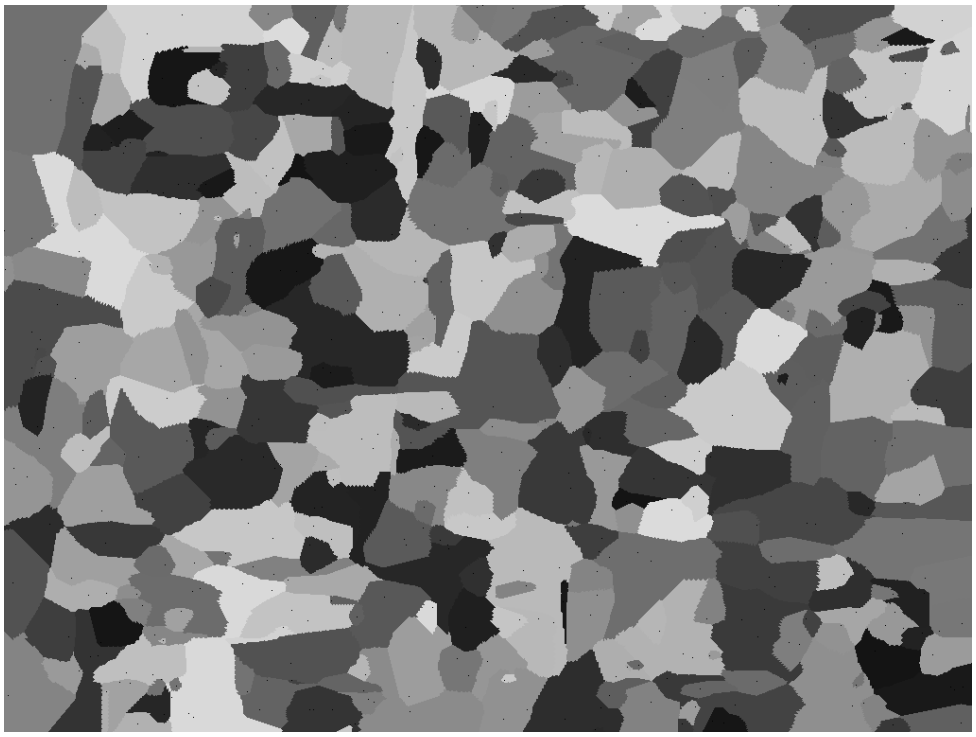

Abbildung 4: 1024*768 mit 300 Keimen und 1-10 px pro Schritt

Abbildung 5: 1024*768 mit 100 Keimen und 1-10 px pro Schritt

Abbildung 6: 1024*768 mit 200 Keimen und 1-10 px pro Schritt

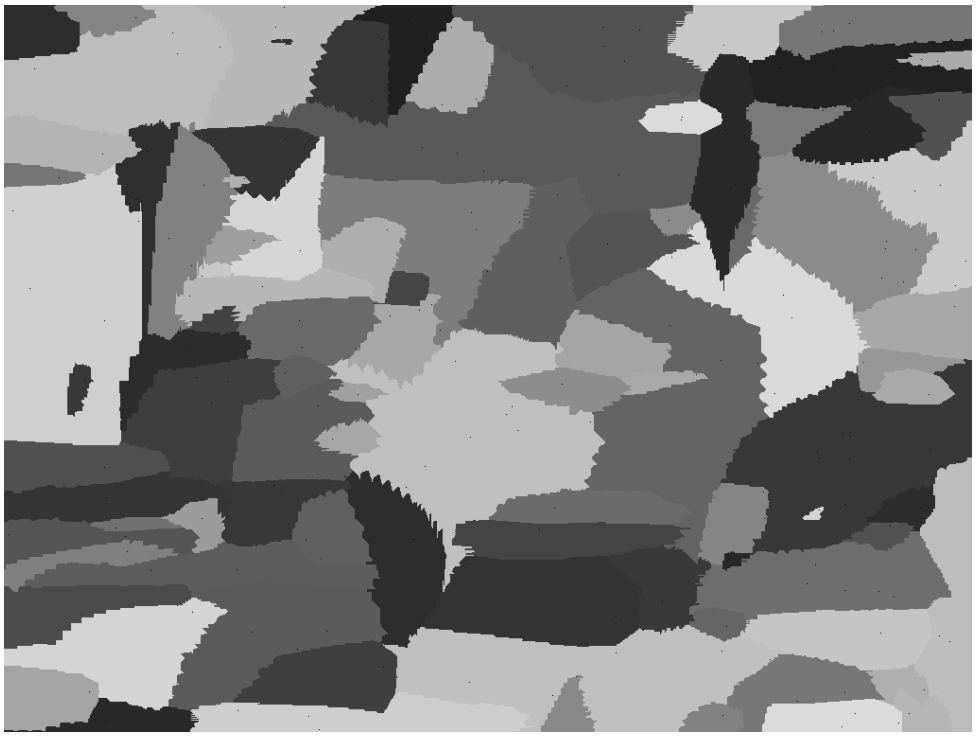

Abbildung 7: 1024*768 mit 100 Keimen und 1-20 px pro Schritt

Abbildung 8: 1024*768 mit 200 Keimen und 1-20 px pro Schritt

4 Quellcode

Die genutzten Bibloteken:

```
from PIL import Image
import numpy as np
import random
```

Die Funktion die schaut ob die Simulation fertig ist:

```
def Space (Array):
    Space = False                           # Die Variable Space wird inizialisiert und auf False
    ↪ gesetzt
    for y in range (0, len(Array)):         # Durch die beiden For-Schleifen geht man durch das
    ↪ gesammte Array
        for x in range (0,len(Array[0])):
            if Array[y][x][0] == 255:       # Die IF-Verzweigung schaut, ob die Farbe, also der
            ↪ erste Wert von dem Kästchen 255 ist, also weiß
                return True                 # Falls der Pixel weiß ist, gibt die Funktion True
                ↪ zurück
    return False                            # Falls kein weißer Pixel gefunden wurde gibt die
    ↪ Funktion False zurück
```

Die Ausbreiung der Keime am Beispiel von der rechtserweiterung:

```
def colour_right(color,x,y, Array,t):                       # Diese Funktion bekommt die Position
↪ des Ursprünglichen Pixels mit einer Farbe dem ganzen Array und t für die Rekursionstiefe
    if Array[y][x][1]>t:                                    # erst kontrolliert die Funktion ob t
    ↪ kleiner ist als die wachsgeschwindigkeit pro Schritt in die Richtung
        if color == -5:                                     #Falls die übergebene Farbe -5 ist wird
        ↪ die Farbenvariable von dem Aktuellen Pixel übernommen.
            color = Array[y][x][0]
        if x+1 < len(Array[0]):                             #Dann überprüft die Funktion, ob der
        ↪ nächste Pixel rechts noch im Array ist
            if Array[y][x+1][0] == 255:                     # Wenn ja, wird als nächstes überprüft,
            ↪ dass die Farbe des Pixels weiß ist, um keine schon Simulierten Pixel zu
            ↪ überstreichen
                Array[y][x+1][0] = color - (color*2)        #Hier werden dann alle Parameter des
                ↪ ursprünglichen Pixel an den nächsten Pixel übertragen. Als farbe wird der wert
                ↪ mit einem - davor angegeben. Das wird am Ende des Schrittes wieder weggetan.
                ↪ Dadurch werden diese Pixel nicht nochmal vergrößert in diesem Schritt
                Array[y][x+1][1] = Array[y][x][1]
                Array[y][x+1][2] = Array[y][x][2]
                Array[y][x+1][3] = Array[y][x][3]
                Array[y][x+1][4] = Array[y][x][4]
                Array = colour_right(color,x+1,y,Array,t+1) #Hier werden die Ausbreitungsfunktionen
                ↪ Rekursiv aufgerufen, dabei wird t erhöht, und die Position des neuen Pixels
                ↪ angegeben
                Array = colour_left(color,x+1,y,Array,t+1)
                Array = colour_up(color,x+1,y,Array,t+1)
                Array = colour_down(color,x+1,y,Array,t+1)
    return Array
```

Umwandlung von Temporerär farbe zu fester Farbe am ende von jedem Schritt:

```
def colour_temp(Array):                 # Am Ende von jedem Schritt, wird diese Gunktion
↪ aufgerufen. Sie dreht die Negativen Zahlen zu Positiven, damit sie im nächsten Schritt beachtet
↪ werden.
    for x in range (0,len(Array[0])):   #Damit wird wieder das ganze Array abgegangen
        for y in range (0, len(Array)):
            if Array[y][x][0] < 0:      # Wird ein Pixel mit einer Farbe kleiner als 0 gefunden
            ↪ wird das - wieder entfernt
                temp = Array[y][x][0]
                Array[y][x][0] = -temp
    return Array                        #Zum Schluss gibt die Funktion das Array zurück
```

Erzugung der Keime:

```
class Keim:                                          #Das ist die Keim Klasse
    def __init__(self,x,y):                          #Sie benötigt lediglich die größe des Array mit
    ↪  x und y
        self.speed_right = int(random.random()*20+1)#Die geschwindigkeiten in jede Richtung werden
        ↪  zufälligfestgelegt zwischen 1 und 21
        self.speed_left = int(random.random()*20+1)
        self.speed_up = int(random.random()*20+1)
        self.speed_down = int(random.random()*20+1)
        self.time = int(random.random()*10)          #Time, bzw. wann der Keim anfängt zu wachsen
        ↪  ist eine zufallszahl zwischen 0-10
        self.color = int(random.random()*200+20)     #Die Farbe wird auch zufällig festgelegt,
        ↪  zwischen 20-220
        self.x = int(random.random()*(x))            #Die positionen werden auch Zufällig festgelegt
        ↪  zwischen 0-x, bzw 0-y
        self.y = int(random.random()*(y))
```

Hauptprogramm:

```
Border_y = int(input("Enter rows: "))           # Hier kann der Benutzer eingeben, wie hoch die
↪  Simulation sein soll
Border_x = int(input("Enter colums: "))         # Hier kann der benutzer eingeben, wie weit die
↪  Simulation sein soll
Anzahl = int(input("Enter Quantities of Keims: "))  #Hier kann der Benutzer eingeben, wie viele
↪  Keime Simuliert werden sollen
Verzinkt_array= np.array([[[255,0,0,0,0]]*Border_x]*Border_y)   # Hier wird das Array mit den
↪  gewünschten maßen erzeugt, wobei alle Pixel weiß sind und keine Ausbreitungsgeschwindigkeit
↪  haben
Keime = {}                                      # Hier wird eine Lekikon erstellt, in dem gleich
↪  alle Keime gespeichert werden können
for Keime_Anzahl in range (0,Anzahl):           # Hier geht eine For-Schleife von 0 bis zu der
↪  gewünschten Anzahl an Keimen alle Zahlen durch
    Keim_temp = Keim(Border_x,Border_y)         # Hier wird dann immer ein Temporäre Variable
    ↪  erstellt, die auf den neuen Keim Verweißt, der mit der Classe erstellt wurde
    Keime[Keime_Anzahl] = Keim_temp             # hier wird der Keim dem Lexikon hinzugefügt
    Verzinkt_array[Keim_temp.y][Keim_temp.x][0] = 0                     #Hier werden alle Parameter
    ↪  in die Position im Array übertragen, wobei die Farbe 0 also Schwarz ist, damit man ihn
    ↪  besser sieht später
    Verzinkt_array[Keim_temp.y][Keim_temp.x][1] = Keim_temp.speed_right
    Verzinkt_array[Keim_temp.y][Keim_temp.x][2] = Keim_temp.speed_left
    Verzinkt_array[Keim_temp.y][Keim_temp.x][3] = Keim_temp.speed_up
    Verzinkt_array[Keim_temp.y][Keim_temp.x][4] = Keim_temp.speed_down
while Space(Verzinkt_array):                    #Diese Schleife wird so lange wiederholt, wie Platz
↪  im Array ist.
    for wachsen in range (0, len(Keime)):       #Diese Forschleife geht das gesamte Lexicon durch
        if Keime[wachsen].time == 0:            # ist bei einem der Keime die Zeit bei 0 werden die
        ↪  Folgenen Schritte durchgeführt:
            Verzinkt_array =
            ↪  colour_right(Keime[wachsen].color,Keime[wachsen].x,Keime[wachsen].y,Verzinkt_array,0)
            ↪  # Die Funktion colour wird aufgerufen, und die Paramerter des Keimes werden
            ↪  übrgeben. Der Keim beginnt sich auszubreiten
            Verzinkt_array =
            ↪  colour_left(Keime[wachsen].color,Keime[wachsen].x,Keime[wachsen].y,Verzinkt_array,0)
            Verzinkt_array =
            ↪  colour_up(Keime[wachsen].color,Keime[wachsen].x,Keime[wachsen].y,Verzinkt_array,0)
            Verzinkt_array =
            ↪  colour_down(Keime[wachsen].color,Keime[wachsen].x,Keime[wachsen].y,Verzinkt_array,0)
        Keime[wachsen].time = Keime[wachsen].time- 1    # bei allen Keimen wird die time Variable
        ↪  um 1 heruntergesetzt
    for y in range(0, Border_y):                #diese For-Schleifen gehen wieder das gesamte Array
    ↪  ab.
        for x in range(0,Border_x):
            if (0 < Verzinkt_array[y][x][0] < 255 ):                        #Hier wird dann
            ↪  geschaut, ob der Pixel bereits eine Farbe zwischen 1 und 254 hat
                    Verzinkt_array=colour_right(-5,x,y,Verzinkt_array,0)    #Wenn ja, so breitet
                    ↪  sich der Pixel durch die Funktuionen aus
                    Verzinkt_array=colour_left(-5,x,y,Verzinkt_array,0)
                    Verzinkt_array=colour_up(-5,x,y,Verzinkt_array,0)
                    Verzinkt_array=colour_down(-5,x,y,Verzinkt_array,0)
    Verzinkt_array = colour_temp(Verzinkt_array)                            # Wenn alles
    ↪  durchgegangen wurde, wird diese Funktion aufgerufen. Dadurch werden die änderungen fest
    ↪  gemacht.
    # Wenn die While-Schleife fertig durchlaufen wurde, muss das Array in eine Bild umgewandelt
    ↪  werden, das passiert hier
final_array = np.array([[255]*Border_x]*Border_y)   #Hier wied ein neues Array angelegt, das nur
↪  zwei dimensionen hat. Hier werden nur die Farbinformationen der Pixel gespeichert.
for y in range(0, Border_y):                        #Diese beiden For-Schleifen, führen uns durch
↪  die Arrays
    for x in range(0,Border_x):
        final_array[y][x] = Verzinkt_array[y][x][0] #Hier werden dann die Farbinformationen vom
        ↪  alten Array zum neuen übertragen
im = Image.fromarray(final_array.astype('uint8'))   #Hier wird dann das array in ein Bild
↪  umgewandelt, und in der im Variable gespeichert
im.save("./Aufgabe 2 Samuele/result.png")           #Hier wird dann das Bild in einem Ordner
↪  gespeichert
im.show()                                           #zuletzt wird das Bild hiermit dann noch
↪  angezeigt
```