Aufgabe 2: Verzinkt

Team-ID: 01103

Team: Skadi Baumgarte

Bearbeiter/-innen dieser Aufgabe:
Skadi Baumgarte

17. November 2022

Inhaltsverzeichnis

Lösungsidee

Es wird eine Matrix mit Pixel Objekten gefühlt die später als ein Pixel in dem fertigen Bild zu sehen sind. Jeder Pixel hat Attribute, wie Farbe, Status also, ob er sich ausbreiten kann, oder Ausbreitungsgeschwindigkeit. Ich habe entschieden, alles randomisiert ablaufen zu lassen. Die Parameter der Wahrscheinlichkeiten können im Code angepasst werden, zum Beispiel Größe des Bildes, Anzahl der Startpixel, Wahrscheinlichkeit, dass später noch ein neuer Pixel hinzukommt, minimale Ausbreitungsgeschwindigkeit. Die Ausbreitungsgeschwindigkeit ist eigentlich eine Wartezeit wie viele Runden ein Pixel warten muss, bis er sich ausbreiten darf.

Am Anfang wird dann eine festgelegte Zahl der Pixel mit zufälligen Werten in Ausbreitungsgeschwindigkeit, Position und Farbe festgesetzt und auf *aktiv* gesetzt. Dann wird die Simulation begonnen. In dieser wird nacheinander für jeden Pixel überprüft, ob er *aktiv* ist und wenn ja, ob er sich in eine Richtung ausbreiten darf, weil die schon abgelaufene Zeit durch seine Ausbreitungsgeschwindigkeit eine Division ohne Rest ist. Ist dies der Fall, breitet er sich in diese Richtung aus, wofür all seine Werte auf den neuen Pixel übertragen werden. Sind alle Pixel gefärbt, wird ein Bild mit den Farben der Pixel ausgegeben.

Umsetzung

Pixel Klasse

die Pixel Klasse enthält die Attribute:

-active(boolean): gibt den Zustand an in dem sich der Pixel befindet.

-color(integer): zwischen 0 und 255 wobei 0 schwarz entspricht und 255 weiß.

-pos_x(intiger): gibt die Ausbreitungsgeschwindigkeit in positiv-x-Richtung also nach rechts an.

-neg_x(intiger): gibt die Ausbreitungsgeschwindigkeit in negative-x-Richtung also nach links an.

-pos_y(intiger): gibt die Ausbreitungsgeschwindigkeit in positiv-y-Richtung also nach unten an.

-neg_y(intiger): gibt die Ausbreitungsgeschwindigkeit in negative-y-Richtung also nach oben an.

All diese Attribute haben ihre eigenen Getter und Setters sowie eine get bzw. set für alle gleichzeitig.

Main

check

Die check Methode wird im späteren Verlauf benutzt, um zu überprüfen ob der Timer durch die Ausbreitungsgeschwindigkeit ohne Rest existiert und der Pixel, der gefärbt werden soll nicht schon eine Farbe hat. Falls beides funktioniert, wird der neue Pixel gesetzt und auf eine Warteliste gepackt mit Pixeln, die aktiviert werden müssen.

Der Start und Definition

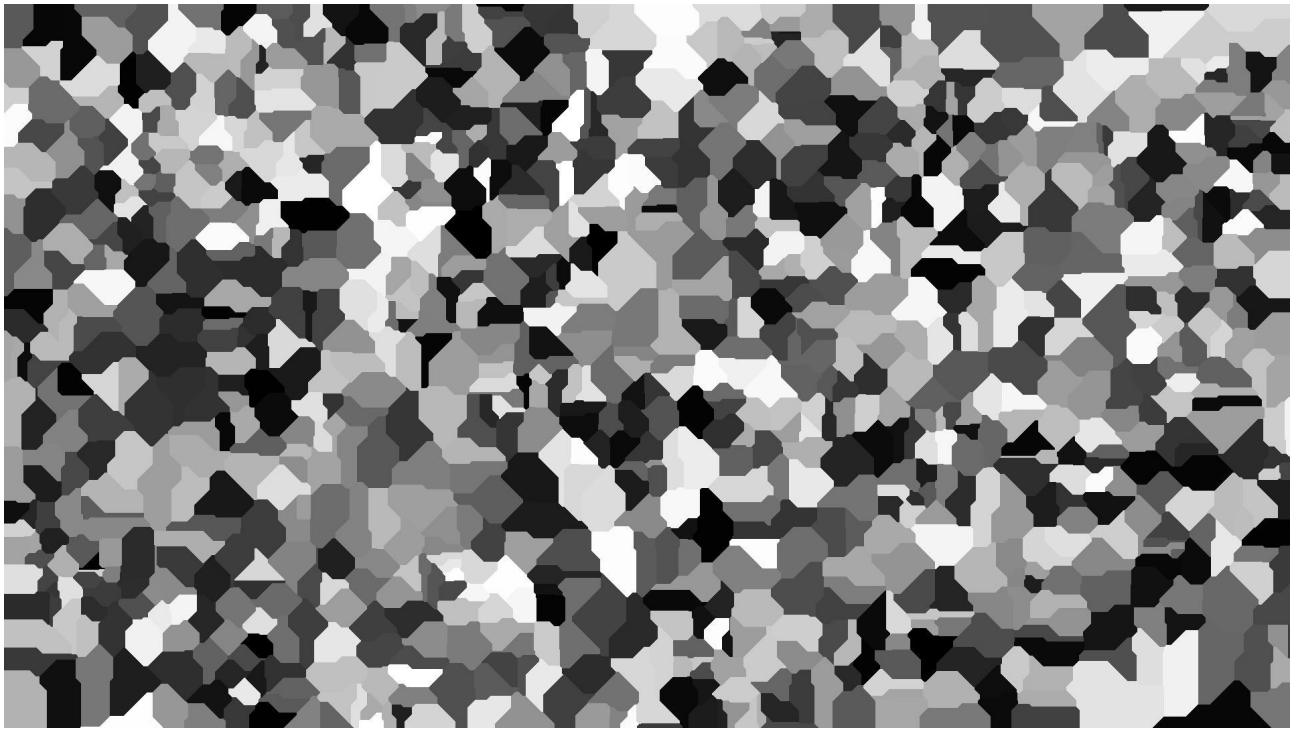
Noch bevor die Simulation los geht, müssen ein paar Dinge definiert werden. Hier kann man auch Einstellungen vor nehmen und zwar die maximale Ausbreitungsgeschwindigkeit (wie langsam es maximal sein soll), die Größe des Bildes, die Anzahl der Start- bzw. späteren Start-Pixel und die Wahrscheinlichkeit für einen Pixel, dass dieser ein neuer späterer Start-Pixel wird (1 /new_pixel_prozent). Daraufhin werden die Start-Pixel festgelegt, indem ihnen x und y Koordinaten und auch die restlichen Werte in den festgelegten Bereichen zufällig zugewiesen werden.

Die Simulation

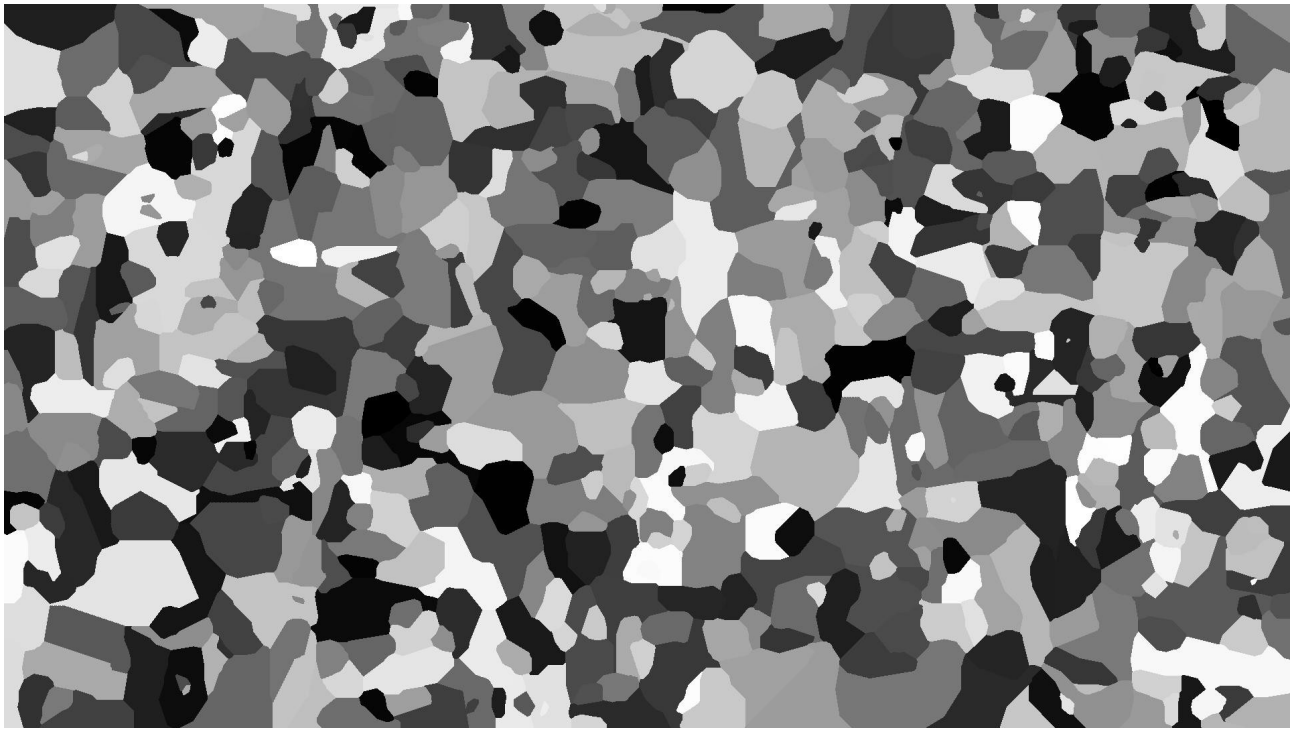
Für jede Runde und damit Zeiteinheit, im weiteren Tick genannt, wird für jeden Pixel erst überprüft ob er zu einem neuen Start-Pixel werden kann. Falls ja, wird dasselbe gemacht, wie beim Start und den Attributen zufällige Werte gegeben, und im Anschluss überprüft ob der Pixel aktiv ist. Falls ja wird, wenn sich die angrenzenden Pixel innerhalb der Matrix befinden, für jeden Pixel drum herum die check Methode ausgeführt. Danach wird noch bei jedem Tick für jeden aktiven Pixel überprüft ob alle Pixel außenrum gefärbt sind, damit man den Pixel wieder *passiv* setzen kann. Danach wird

für jeden Pixel überprüft, ob er gefärbt ist. Am Ende werden noch alle Pixel die neu gefärbt wurden auf aktiv gesetzt und überprüft ob alle Pixel gefärbt sind. Sollte dies der Fall sein wird die Simulation beendet. Genauso wird das Programm beendet wenn die Anzahl der Vergangen Ticks der Anzahl der Spalten entspticht

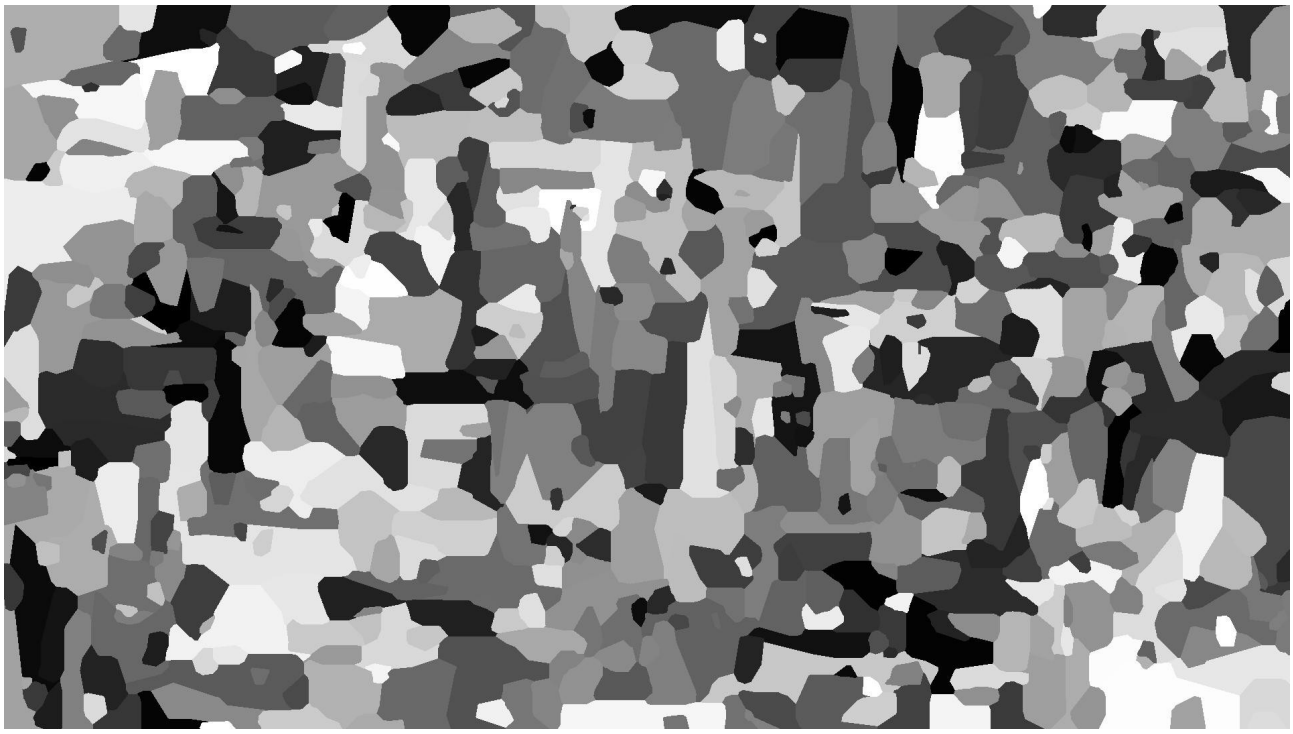
Beispiele



Ein einfaches Beispiel um mal etwas simples zu generieren alle 1000 Pixel wurden am Anfang festgelegt und die Ausbreitungsgeschwindigkeit war überall bei 1.



Ein schon etwas komplexeres Beispiel 600 Pixel am Anfang keine später.
Ausbreitungsgeschwindigkeit bis zu 5 in eine Richtung.



Beispiel mit noch höherer möglichen Ausbreitungsgeschwindigkeit bei bis zu 10

Interessantes Beispiel:
- 300 Direkt am Anfang
- Ausbreitungsgeschwindigkeit wie oben
- 700 Pixel später mit 0.1 % pro Pixel

Und noch ein Letztes
-300 Start-Pixel
-Ausbreitungsgeschwindigkeit bei 30
- 700 Pixel mit 0.0001% pro Pixel

um ein besseres noch ähnlicheres Bild als das Dritte zu bekommen müsste man wahrscheinlich die Startpunkte in Cluster stecken dies wurde hier jedoch nicht bedacht

Quellcode

```
from numpy import *
import random
import pixel
from PIL import Image

def check(way,test,time,time_this):#definition der check methode zum überprüfen der ausbreitung eines Kristalls
    if time%time_this == 0:#timer durch ausbreitungsgeschwindigkeit ohne Rest heißt ausbreitung
        if way.get_color() == None: # nur ausbreitung zu einem freien Pixel

way.set_all(test.get_pos_x(),test.get_pos_y(),test.get_neg_x(),test.get_neg_y(),test.get_color())#ausbreitung
            return True
        else:
            return False
    else:
        return False

status = "definiton"#definition einer Matrix in der Größe des Bilds und anzahl der Kristallkerne
start_pixel_number = 300
random_start_pixel = 700
max_tempo = 10
new_pixel_prozent = 1000000
#höhe und breite des Bildes
num_rows = 1920
num_cols = 1080

matrix_pixel = [[pixel.Pixel() for i in range(num_cols)]]
matrix_pixel = reshape(matrix_pixel,(1,num_cols))
for i in range((num_rows)-1):
    row = [pixel.Pixel() for i in range(num_cols)]
    matrix_pixel = append(matrix_pixel,[row],0)

status = "start"

while start_pixel_number > 0:#werte zuweisen für jeden startpixel
    x = random.randint(0,num_cols)
    y = random.randint(0,num_rows)
    if matrix_pixel[y-1][x-1].get_active() != True:
        matrix_pixel[y-1][x-1].set_color(random.randint(10,255))
        matrix_pixel[y-1][x-1].set_active_true()
        matrix_pixel[y-1][x-1].set_pos_x(random.randint(1,max_tempo))
        matrix_pixel[y-1][x-1].set_pos_y(random.randint(1,max_tempo))
```

```
        matrix_pixel[y-1][x-1].set_neg_x(random.randint(1,max_tempo))
        matrix_pixel[y-1][x-1].set_neg_y(random.randint(1,max_tempo))

        start_pixel_number = start_pixel_number-1

status = "simmulieren"#starten der Simmulation
time = 0
while status == "simmulieren":
    time = time +1
    y = -1
    used_pixel = 0
    liste_forced = []
    for line in matrix_pixel:
        x = -1
        y= y + 1
        for new_pixel in line:#durchlaufen jedes Pixels für jedee Zeit
            x= x + 1
            if new_pixel.get_color() == None and random_start_pixel > 0:# Möglichkeit zum einfügen
späterer Pixel  mit einer zufälligen Zeit
                if random.randint(1,new_pixel_prozent) == 1:
                    matrix_pixel[y][x].set_color(random.randint(0,255))
                    matrix_pixel[y][x].set_active_true()
                    matrix_pixel[y][x].set_pos_x(random.randint(1,max_tempo))
                    matrix_pixel[y][x].set_pos_y(random.randint(1,max_tempo))
                    matrix_pixel[y][x].set_neg_x(random.randint(1,max_tempo))
                    matrix_pixel[y][x].set_neg_y(random.randint(1,max_tempo))
                    random_start_pixel = random_start_pixel-1

            if new_pixel.get_active():#für jeden aktiven Pixel
                if y-1 > -1 :# Überprüfung des Randes des Bild
                    top = matrix_pixel[y-1][x]
                    if check(top,new_pixel,time,new_pixel.get_neg_y()):#nutzen der Checkmethode für
den Pixel über dem ausgewählten Pixel
                        liste_forced = liste_forced +[(y-1,x)]#speichern des neuen Pixels

                if y+1 <= num_rows-1:#Wiederholung für dem Pixel darunter
                    bottom = matrix_pixel[y+1][x]
                    if check(bottom,new_pixel,time,new_pixel.get_pos_y()):
                        liste_forced = liste_forced +[(y+1,x)]

                if x-1 > -1:#Wiederholung für den Pixel links
                    left = matrix_pixel[y][x-1]
                    if check(left,new_pixel,time,new_pixel.get_neg_x()):
                        liste_forced = liste_forced +[(y,x-1)]

                if x+1 <= num_cols-1:#Wiederholung für den pixel rechts
                    right = matrix_pixel[y][x+1]
                    if check(right,new_pixel,time,new_pixel.get_pos_x()):
```

```
                liste_forced = liste_forced + [(y,x+1)]


        if new_pixel.get_color() !=None:#jeden Pixel überprüfen ob er gefärbt ist
            used_pixel = used_pixel+1

    if used_pixel == (num_cols)*(num_rows):#wenn alle gefärbt sind enden der simmulation
        status = "finish"
    for i in liste_forced:# alle neuen Pixel auf aktiv setzen
        y = i[0]
        x = i[1]
        matrix_pixel[y,x].set_active_true()
    if time == num_rows:#not stop der simmulation
        status = "stop"
    i = 0

if status == "finish":#auswertung der simmulation
    ausgabebild=Image.open("test.jpg")
    for y in range(0,num_cols):
        for x in range(0,num_rows):
            color = matrix_pixel[x,y].get_color()
            ausgabebild.putpixel((x, y), (color,color,color))#umwandeln der Matrix in eine Jpg
    name = str(random.randint(1000,9999))
    ausgabebild.save(name +".jpg")
else:
        print("simmulation wurde abgebrochen")
```