Aufgabe 2: Verzinkt

Team-ID: 00464

Team: Enrique Lopez

Bearbeiter/-innen dieser Aufgabe:
Enrique Lopez

6. November 2022

Inhaltsverzeichnis

Lösungsidee

Die Bildfläche wird als zwei-dimensionales Array interpretiert, welches zunächst die Werte 1 enthält. In diesem Array werden zufällig eine dem Programm übergebene Anzahl an Keimen gestreut (Werte werden verändert), wobei diese Keime einen Wert von 100-220 annehmen können. Diese Werte werden später als Grautöne interpretiert.

Nun wird über das Array iteriert. Wenn ein Kristall (Wert ungleich null) entdeckt wird, so wird der Grauwert (Zahlenwert) als Orientierung interpretiert und je nach Orientierung verändert das Programm links, rechts, oben und unterhalb des Keims den Pixel zur Farbe des Keims. Wie viele Pixel in jede Richtung gefärbt werden hängt zusätzlich von einem dem Programm übergebenen Wert ab.

Es wird solange über das Array iteriert, bis kein Wert im Array mehr 0 ist. Dann wird das Array mithilfe einer Bibliothek als Bild mit Grautönen interpretiert.

Umsetzung

Zur Implementierung des Kernprogramms werden zwei Bibliotheken verwendet: Numpy und die Python Image Library. Numpy erlaubt die Verwendung von Arrays in Python. Die Python Image Library kann Numpy Arrays als Bild mit Grautönen Interpretieren. Weiterhin erlaubt die Bibliothek Random die Generierung von zufälligen Zahlen für die Keimstreuung und die Bibliothek time ist hilfreich zur Messung der Programmlaufzeit

Das Programm selbst besteht aus der Funktion verzinkt(). Diese nimmt 3 Argumente:

1. quantity – Wie viele Keime bzw. Kristalle sollen gestreut werden

2. resolution – Dimensionen des Bildes, muss aufgrund der Funktionsweise des Programms Quadratisch sein → Tupel mit 2 gleichen Werten

3. sharpness – Wie scharf sollen die Kristalle sein bzw: Wie schnell wächst ein Kristall in seine Hauptorientierung im Vergleich zu den anderen 3 Richtungen.

Warnung: Die Effizienz des Programms ist unglaublich schlecht, da meine Lösung für das Problem vermutlich nicht die eleganteste ist. Die Wartezeiten können bei hohen Resolutionen wie (1000,1000) gerne mal die 7 Minuten Marke überschreiten. Aufgrund der langen Laufzeit wird später nach jeder Iteration eine Nachricht ausgegeben, damit der Benutzer weiß, dass das Programm nicht abgestürzt ist.

In der Funktion wird eine weitere Funktion definiert, create_seeds(), welche später in der Funktion die Argumente quantity und resolution entgegennimmt. In ihr wird ein mit dem Wert "1" gefülltes Numpy Array mit den vorgegebenen Dimensionen erstellt und es werden **ungefähr** (siehe Quellcode Kommentar für Erklärung) die Anzahl an übergebenen Keimen gestreut. Zudem wird eine Kopie des Arrays erstellt, auf welchem später die Änderungen vorgenommen werden.

In der Hauptfunktion werden zunächst einige Checks durchgeführt um den Benutzer auf gewisse Limitationen hinzuweisen. Daraufhin werden die Keime mit der Funktion create_seeds gestreut.

Nun beginnt der eigentliche Doppel-Loop. Ein äußerer While Loop wiederholt den inneren for-loop so lange, bis es den Wert "1" im Array nicht mehr gibt (alle Pixel wurden gefärbt). Der innere for-loop iteriert über jedes Objekt im Array und ändert je nachdem, wo der Wert in dem Wertespektrum 100-220 liegt, die Werte in 4 Richtungen um das aktuelle Objekt (so geht aus der Orientierung der Grauton hervor (oder andersherum!)). Dabei nehmen die Objekte um den Wert herum den Wert an.

Da alle Werteänderungen an der flachen Version des Array vorgenommen werden, muss mithilfe der Dimensionen des Arrays die Position des späteren Pixel oberhalb des Wertes ausgerechnet werden. Dabei wird vor jeder Änderung geprüft, ob der zu ändernde Pixel auch wirklich den Wert "1" hat, damit keine Kristallwerte erneut umgefärbt werden. Um jede Änderung wird mit try und except zusätzlich dafür gesorgt werden, dass das Programm IndexErrors ignoriert, da diese im Programm sehr häufig vorkommen, da zum Beispiel der Wert links von dem linksäußersten Wert nicht existiert. Durch Conditional Statements wird zusätzlich das "sharpness" Argument, welches Werte von 1-4 annehmen kann, implementiert. Wenn sharpness = 3 ist, so werden besipielsweise 3 Pixel in der Hauptorientierungsrichtung eingefärbt. Der Kristall wächst in die anderen 3 Richtungen um 1 Pixel pro Iteration. Zum Ende einer Schleife wird die Kopie des Arrays, auf welchen die Änderungen stattfinden, aktualisiert und der Loop beginnt von neuem, bis alle Pixel eingefärbt sind.

Sind alle Werte geändert, so wird das Array mit der Funktion Image.fromarray() zu einem Bild mit Grautönen konvertiert und ausgegeben.

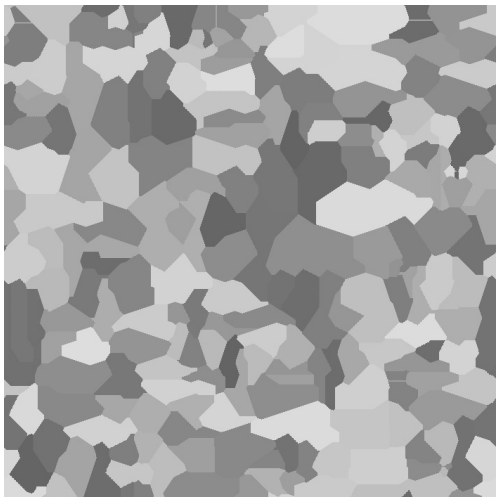
Beispiele

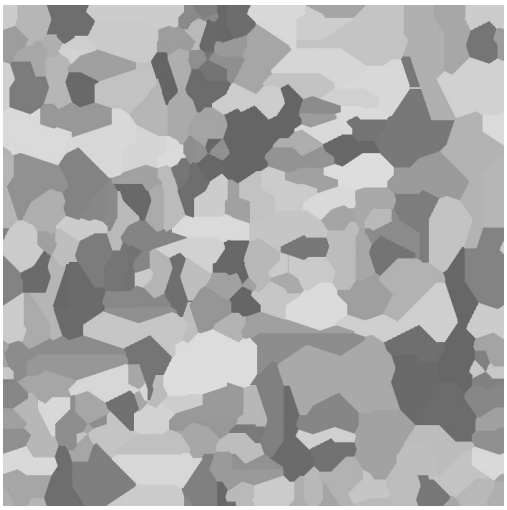
Da die Keimstreuung komplett zufällig ist, variiert die Ähnlichkeit zum Beispielbild je nach Glück.

Es ist schwer, mit meinem Programm eine Ähnlichkeit zum Beispielbild des Bundeswettbewerbsbilds zu generieren, jedoch ähneln die Muster eher diesem Muster auf einem feuerverzinkten Rohr.

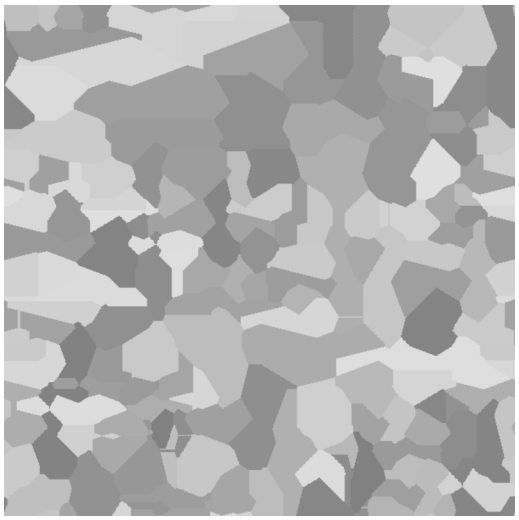
verzinkt(200,resolution = (700,700),sharpness = 3)

Die gelegentlich auftretenden langen Striche ausgehend von einem Kristall sind ungewollte Nebenprodukte der Funktionsweise des Programms. Das Problem wird mit steigendem sharpness Wert schlimmer.



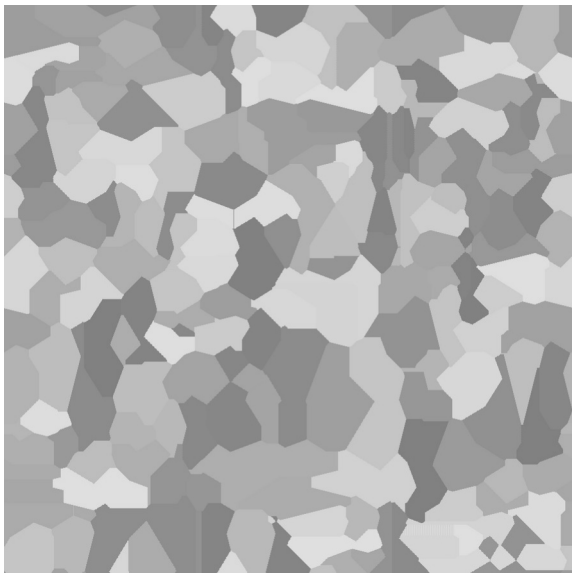


verzinkt(150, resolution = (600,600), sharpness = 4)

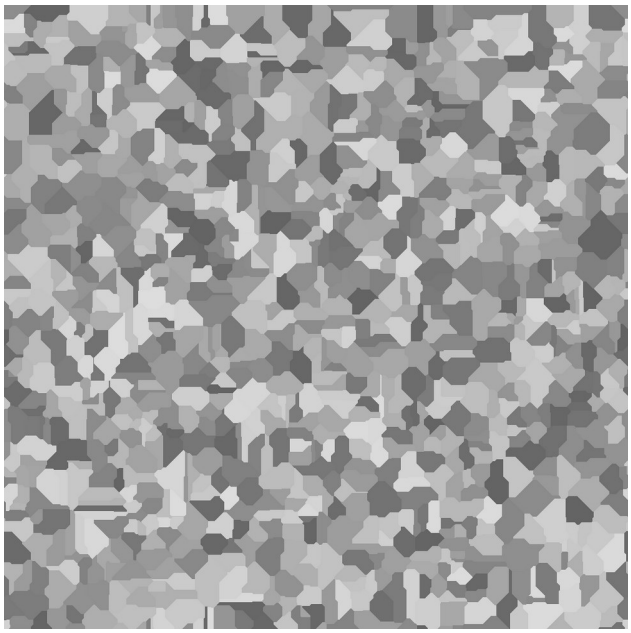


verzinkt(10,resolution = (200,200),sharpness = 1)

verzinkt(180,resolution=(1000,1000),sharpness=5)



verzinkt(1000,resolution=(1000,1000),sharpness=1)



Quellcode

November 6, 2022

[9]:
```
import random
import numpy as np
from PIL import Image
import time


def verzinkt(quantity,resolution = (256,256),sharpness = 3):

    #sharpness = How fast does the crystal grow in it's fastest direction␣
 ↪compared to the other 3 directions

    def create_seeds(quantity,resolution):

        #quantity = Menge der Kristallkeime
        #resolution = Tupel mit Bilddimensionen (Beispiel: (720,720))
        #sharpness = How fast does the crystal grow in it's fastest direction␣
 ↪compared to the other 3 directions
        '''BILD MUSS QUADRAT SEIN -> TUPEL muss 2 mal den gleichen Wert haben'''
        #Bei hohen Bilddimensionen wird die Wartezeit sehr viel länger

        #Array gefüllt mit Einsen
        array = np.ones(resolution)

        #Kristallkeime erstellen - n mal einen Zufälligen Pixel mit Wert␣
 ↪zwischen 100 und 220 (Grautöne) überschreiben
        '''Bei einer sehr hohen Anzahl von Keimen weicht der quantity Wert␣
 ↪stark von der Anzahl der überschriebenen Werte ab,
        da gegen Ende der Schleife mit höher Wahrscheinlichkeit bereits␣
 ↪überschriebene Werte jochmal überschrieben werden.
      Jedoch ist diese Abweichung von Originalwert für kleinere Werte von␣
 ↪quantity bis etwa 1000 vernachlässigbar. Da im
      Beispielbild weitaus weniger Kristalle zu sehen sind, lässt sich ein␣
 ↪solches Bild auch erstellen, ohne die Laufzeit
      der Simulation durch einem Check für jeden Pixel unnötig zu erhöhen.'''

        for i in range(quantity):
```

```
            array.flat[np.random.choice(np.prod(array.shape),1,replace =␣
↪False)] = np.random.randint(100,221)

        return array

    if resolution[0] != resolution[1]:
        return print('ERROR: Bild muss Quadrat sein -> Für Resolution␣
↪übergebener Tupel muss 2 mal den selben Wert haben')
    if not(0 < sharpness <5) or not(type(sharpness) == int):
        return print('ERROR: Gültige Werte für Sharpness: 1,2,3,4')
    print('Programm startet!')
    start_time = time.time()
    #Kristallkeime erstellen
    vorlage = create_seeds(quantity,resolution)
    kristallmuster = vorlage.copy()
    print('Keime gestreut!')

    timer = 1
    #Solange es Werte gibt die nicht ersetzt wurden
    while np.count_nonzero(kristallmuster == 1) != 0:

        for i in range(len(vorlage.flat)):

            ####################################
            if 100 <= vorlage.flat[i] <= 130:

                #Errechnete Array Position könnte out of bounds sein ->␣
↪Ignorier Error mit try / except
                try:
                    if vorlage.flat[i-1] == 1:
                        kristallmuster.flat[i-1] = vorlage.flat[i]
                except:
                    pass

                try:
                    if vorlage.flat[i+1] == 1:
                        kristallmuster.flat[i+1] = vorlage.flat[i]
                except:
                    pass

                try:
                    if vorlage.flat[i-np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i-np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass
```

```
                try:
                    if vorlage.flat[i+np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i+np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass

                if sharpness > 1:
                    try:
                        if vorlage.flat[i-2*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i-2*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]

                    except:
                        pass

                if sharpness > 2:
                    try:
                        if vorlage.flat[i-3*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i-3*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]
                    except:
                        pass

                if sharpness > 3:
                    try:
                        if vorlage.flat[i-4*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i-4*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]
                    except:
                        pass

            ################################################

            if 131 <= vorlage.flat[i] <= 160:
                try:
                    if vorlage.flat[i-1] == 1:
                        kristallmuster.flat[i-1] = vorlage.flat[i]
                except:
                    pass

                try:
                    if vorlage.flat[i+1] == 1:
                        kristallmuster.flat[i+1] = vorlage.flat[i]
                except:
                    pass
```

```
                try:
                    if vorlage.flat[i-np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i-np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass


                try:
                    if vorlage.flat[i+np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i+np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass

                if sharpness > 1:
                    try:
                        if vorlage.flat[i+2] == 1:
                            kristallmuster.flat[i+2] = vorlage.flat[i]
                    except:
                        pass

                if sharpness > 2:
                    try:
                        if vorlage.flat[i+3] == 1:
                            kristallmuster.flat[i+3] = vorlage.flat[i]
                    except:
                        pass

                if sharpness > 3:
                    try:
                        if vorlage.flat[i+4] == 1:
                            kristallmuster.flat[i+4] = vorlage.flat[i]
                    except:
                        pass

            ##############################################
            if 161 <= vorlage.flat[i] <= 190:
                try:
                    if vorlage.flat[i-1] == 1:
                        kristallmuster.flat[i-1] = vorlage.flat[i]
                except:
                    pass

                try:
                    if vorlage.flat[i+1] == 1:
                        kristallmuster.flat[i+1] = vorlage.flat[i]
```

```
                except:
                    pass

                try:
                    if vorlage.flat[i-np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i-np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass


                try:
                    if vorlage.flat[i+np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i+np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass

                if sharpness > 1:
                    try:
                        if vorlage.flat[i+2*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i+2*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]
                    except:
                        pass

                if sharpness > 2:
                    try:
                        if vorlage.flat[i+3*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i+3*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]
                    except:
                        pass

                if sharpness > 3:
                    try:
                        if vorlage.flat[i+4*(np.shape(vorlage)[0])] == 1:
                            kristallmuster.flat[i+4*(np.shape(vorlage)[0])] =␣
↪vorlage.flat[i]
                    except:
                        pass
            ################################################
            if 190 <= vorlage.flat[i] <= 220:
                try:
                    if vorlage.flat[i-1] == 1:
                        kristallmuster.flat[i-1] = vorlage.flat[i]
                except:
```

```
                    pass

                try:
                    if vorlage.flat[i+1] == 1:
                        kristallmuster.flat[i+1] = vorlage.flat[i]
                except:
                    pass

                try:
                    if vorlage.flat[i-np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i-np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass


                try:
                    if vorlage.flat[i+np.shape(vorlage)[0]] == 1:
                        kristallmuster.flat[i+np.shape(vorlage)[0]] = vorlage.
↪flat[i]
                except:
                    pass

                if sharpness > 1:
                    try:
                        if vorlage.flat[i-2] == 1:
                            kristallmuster.flat[i-2] = vorlage.flat[i]
                    except:
                        pass

                if sharpness > 2:
                    try:
                        if vorlage.flat[i-3] == 1:
                            kristallmuster.flat[i-3] = vorlage.flat[i]
                    except:
                        pass

                if sharpness > 3:
                    try:
                        if vorlage.flat[i-4] == 1:
                            kristallmuster.flat[i-4] = vorlage.flat[i]
                    except:
                        pass



        print('Iteration',timer)
```

```
        timer += 1
        vorlage = kristallmuster.copy()


    print('Fertig nach %s Sekunden!' % (round(time.time() - start_time)))
    time.sleep(2)
    img = Image.fromarray(np.uint8(kristallmuster))
    img.show()
```