# Aufgabe 2: Verzinkt

Team-ID: 00811

Team: Lorian

Bearbeiter/-innen dieser Aufgabe:
Lorian Linnertz

20. November 2022

## Inhaltsverzeichnis



# Lösungsidee

Meine Lösungsidee beruht darauf, dass wir jeden Kristal als eine Klasse definieren und das Bild als eine Matrix darstellen.

### Matrix

Da wir ein Bild mit der Auflösung 720*1280, da es ein guter Kompromis aus Auflösung und Rechenaufwand darstellt, müssen wir ebenfalls eine Matrix mit der Form 720 auf 1280 erstellen, wobei wir für den Anfang überall 0 einsetzen. In der Matrix sind die Pixel der Kristalle mit dem Grauwert(zwischen 1 und 255) welcher dem Kristal zugeordnet wurde, markiert. Somit ist das Bild vollendet sobald keine Pixel mehr mit dem Wert 0 überig bleiben.

### Kristalle

Die Idee hinter den Kristallen ist, dass wir ein nach dem anderen Wachsen lassen, indem wir in die Klasse Kristal eine Funktion integrieren, welche überprüft ob die äußen Pixel erweitert werden können. Sobald ein Kristal keine Möglichkeit mehr hat erweitert zu werden so wird er aus der Liste, in welcher sich alle Kristalle befinden, entfernt und gilt als fertig berrechnet. Man iteriert durch diese Liste solange bis alle Kristalle fertig berechnet sind und somit die Liste leer ist.

# Umsetzung

Für die Umsetzung müssen wir als erstes die Kristalle und die Matrix initalisieren. Dabei ist es wichtig, dass wir für die Kristalle, jeweils eine einzigartigen Farbton bestimmen, sowie eine Zufällige Position auf der Matrix, auf welcher sich noch kein anderer Kristalkeim befindet(dies wird umso wahrscheinlich umso mehr Kristalle man hat), eine Zufällige Wachstumsgeschindigkeit und eine zufällige Verzögerung der Entstehung des Keims

**Benötigte Variabeln:**

crystals_list => In dieser Liste werden alle Kristalle gespeichert

self.seed_color => speichert die Farbe des Kristals

self.grow_speed => speichert die Wachstumsgeschindigkeit

self.delay => speichert die Verzögerung

self.open_pos => speichert die Pixel, welche noch nicht überprüft wurden

self.counter => Zählt die Durchgäng bzw. Wie oft der Pixel aktualisiert wurde

self.complete => Diese Variable ist am Anfang False, sobald der Kristal fertig berechnet wurde ist er Positiv

## Algorithmus

1) Aus crystals_list entimmt man das [0] Element

2) Man ruft die Funktion zur Erweiterung des entnommenen Kristals auf

a) Wir überprüfen ob der counter größer ist als der Delay

I) Sollte dies nicht zutreffen, so wird der counter um 1 erhöht und man geht direkt zu Schritt 3 weiter

b) Im nächsten Schritt berechnen wir die Anzahl der zuerweiternden Pixel mit der Formel len(self.open_pos) * self.grow_speed

c) Wir iterieren jetzt so oft wie vorhin berechnet durch folgenden Algorithmus:

I) Wir wählen zufällig ein Pixel aus open_pos aus und entnehmen diesen

II) Wir suchen alle freien Pixel um den ausgewählten Pixel(wichtig ist zu überprüfen ob die neuen Pixel noch in der Matrix sind)

III) Wir iterieren durch die gefundenen Pixel und malen diese mit der Farbe self.seed_color an und fügen diese Pixel zu open_pos hinzu

IV) Wir überprüfen ob die len(open_pos) == 0 ist.

→ Sollte dies Zutreffen, so beendet man die Schleife und setzt dies self.complete auf True

→ Sollte dies nicht zutreffen so beginne wieder bei Schritt I) bis die Schleife fertig ist

3) Wir überprüfen ob der Kristal [0] beendet ist, indem wir die Variable self.complete überprüfen

I) True: bedeutet, wir entfernendiesen Kristal aus der Liste und überprüfen ob crystal_list == 0 ist

a) Sollte dies Zutreffen, so ist der Algorithmus beendet

b) Sollte dies nicht Zutreffen so beginnt man wieder bei Schritt 1)

II) False: bedeutet, dass wir den Kristal ans Ende der Liste verschieben und wieder bei Schitt 1) beginnen)

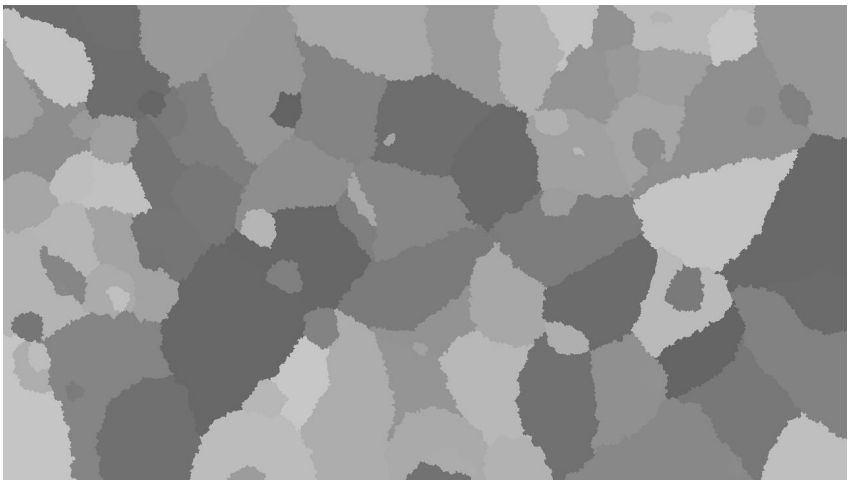
## Erklärung zu meinem Vorgehen:

Ich habe bei Schritt 2)c)I) die Random Funktion benutzt, da dies zu natürlicheren Ergebnissen im Vergleich zu der Methode bei welcher man direkt durch die Liste open_pos iteriert. Damit meine ich, dass ohne diese Random Funktionen die Linien zwischen den Übergang zweier Kristalle eine Gerade Linie ist, und nicht den Eindruck einer Natürlichen Verzinkung erzeugt.

# Beispiele

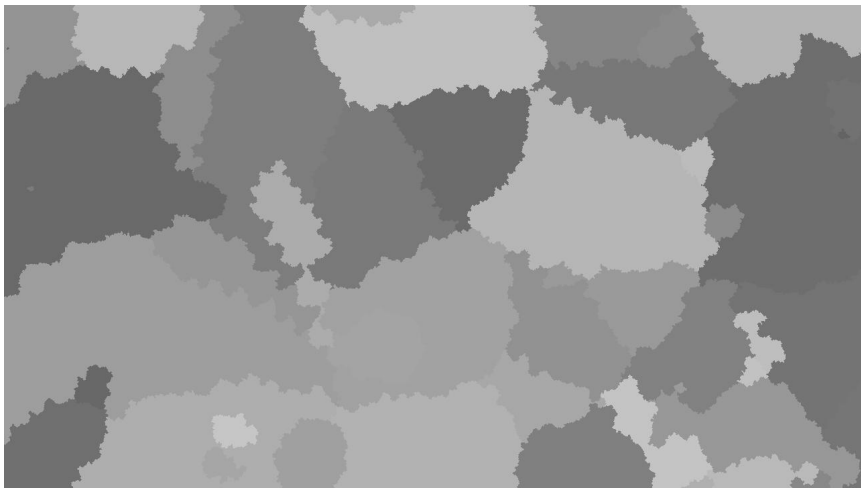
Beispiel 1:

*<u>Konfiguration:</u> Anzahl an Kristallen = 100, Wachstumsgeschwindigkeit: zwischen 1 und 10, Maximale Verzögerung: 0*



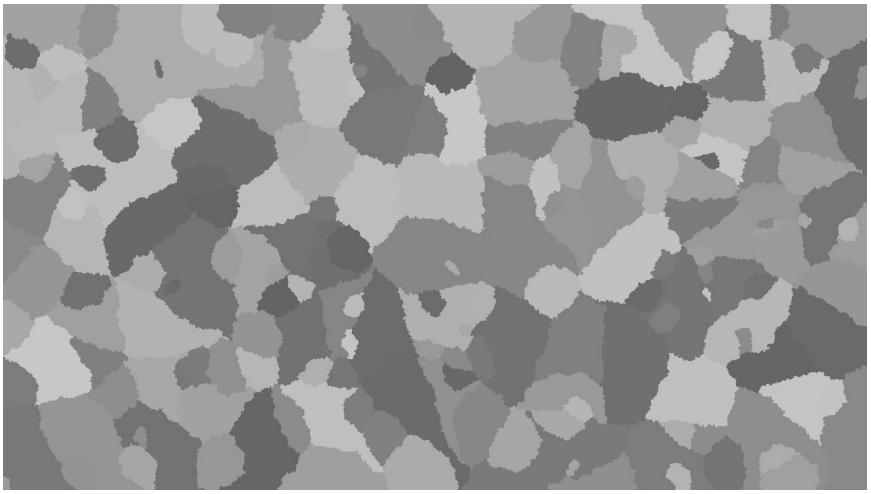
Beispiel 2:

*<u>Konfiguration:</u> Anzahl an Kristallen = 50, Wachstumsgeschwindigkeit: zwischen 3 und 12, Maximale Verzögerung: 5*



Beispiel 3:

*<u>Konfiguration:</u> Anzahl an Kristallen = 200, Wachstumsgeschwindigkeit: zwischen 1 und 2, Maximale Verzögerung:*



# Quellcode

```
def get_seeds(num,grow_speed_range,maxdelay,color_range=(100,200)): #Diese Funktion soll die Kristal_Klassen initalisieren; num => gibt die Anzahl an Seeds an

    resolution=(720,1280) #(x-Achse,y-Achse)

    crystals_list = [] #In dieser Liste werden die unterschiedlichen Kristale gespeichert

    seeds_list = [] #Diese Liste hilft zu überprüfen, ob eine Koordinaten Paar bereits benutzt wurde

    delta_color = (color_range[1]-color_range[0])/num #delta_color gibt die benötigte differenz an,

    #um einer num-Anzahl an Pixel eine gleichmäßig aber unterschiedliche Anzahl an Grautöne in einer Farbbreite von color_range[0] bis color_range[1] zu geben

    for i in range(num):

        while True: #wiederholt, so oft den Prozess, bis ein Seed gefunden wurden, welcher noch nicht benutzt wurde. (Ist vor allem bein höheren Werten für num wichtig)

            seed_pos = (random.randint(0,resolution[0]-1),random.randint(0,resolution[1]-1)) #Gibt ein zufälliges Koordinaten Paar, zwischen den Grenzen von x -> {0,720} und y -> {0,1280} wieder

            if seed_pos not in seeds_list: #überprüft ob das Koordinaten-Paar nicht bereits verwendet wurde

                seed_color = i*delta_color+color_range[0] #berechtnet den Grauton für den i.ten Kristal

                grow_speed = random.randint(grow_speed_range[0],grow_speed_range[1]) #Gibt eine zufällige Wachstumsgeschwindigkeit zwischen 1 und einem Wert an.

                delay = random.randint(0,maxdelay) #Gibt eine Zufällige Zahl für die Verzögerung des Kristals

                seed = Crystal(seed_pos,seed_color,grow_speed,delay) #und initalisiert diesen

                crystals_list.append(seed) #Dieser neue Kristal wird zur liste hinzugefügt

                seeds_list.append(seed_pos) #Und zum Schluss ebenfall das Koordinaten Paar

                break

    return crystals_list #Gibt die Liste mit den initalisierten Kristalen zurück

class Crystal: #Diese Klasse representiert jeden Kristal einzel und ermöglicht es uns einem einzelnen Kristal eine Wachstumsgeschwindigkeit zu gebe

    def find_pixel(self,matrix,position): #Diese Funktion soll Pixel finden, welche noch von keinem anderen Kristal besetzt sind

        new_pos = [] #In diese Liste werden die gefunden Pixel gespeichert

        conditions =[(1,0),(-1,0),(0,1),(0,-1)] #gibt an welcher Wert auf welche Achse hinzu gerechnet wird (x,y)

        borders = [(0,720),(0,1280)] # Definiert die Grenzen des Bildes => [(x_min,x_max),(y_min,y_max)]

        for c in conditions:

            x = position[0] + c[0] #Durch das Auf addieren von der Condition für die x-Achse erhalten wir die x-Achse des neu zubetrachtenden Pixels

            y = position[1] + c[1] #Genau das gleiche nur für die y-Achse

            if x < borders[0][0] or x >= borders[0][1] or y < borders[1][0] or y >= borders[1][1]: #Dieses if_statement überprüft, ob der betrachtete Pixel außerhalb des Bildes liegt

                continue #Falls dies der Fall ist, so fährt man sofort mit der nächsten Condition fort

            i = matrix.item((x,y)) #Entnimmt den Wert des neu zubetrachtenden Pixels

            if i == 0: #Falls dieser Wert == 0 sein sollt, bedeutet das dass dieser Pixel noch leer/undefiniert ist

                new_pos.append((x,y))#Somit wird dieser neu gefundene Pixel zur liste hinzugefügt

        return new_pos #Am ende wird diese Liste noch zurückgegeben
```

```
    def grow_crystal(self,matrix): #Diese Funktion soll die Kristalle wachsen lassen
        if self.counter < self.delay: #Falls das Limit vom Delay noch nicht ereicht wurde,so
            self.counter += 1 #erhöht man den Zähler
            return matrix #Und beendet den Durchlauf
        expansions = len(self.open_pos) * self.grow_speed #Gibt an wie viele Pixel erweitert werden
        for _ in range(expansions):
            pixel = self.open_pos.pop(random.randint(0,len(self.open_pos)-1)) #Entnimmt einen zufälligen Pixel aus der Liste der noch zu bearbeitenden Pixel
            to_change = Crystal.find_pixel(self,matrix,pixel) #Die gefunden Pixel werden in der Liste to_change abgespeichert.

            for p in to_change: #iteriert durch die gefunden Pixel
                matrix.itemset(p,self.seed_color) #und gibt ihnen den Grauton, des Kristals
                self.open_pos.append(p) #Fügt diesen Pixel in die Liste der zu überprüfenden Pixel

            if len(self.open_pos) == 0: #Überprüft ob die Liste open_pos leer ist
                self.complete = True #Sollte dies der Fall sein, so ist dieser Kristal fertig berechnet und die Variable complete wird auf True gesetzt
                return matrix #beendet die Schleife und die Funktion
        return matrix #Gibt die Matrix mit den neu definierten Pixeln zurück

def running(crystals_list): #Diese Funktion ist die Hauptfunktion, welche die unterfunktionen aufruft und koordiniert
    matrix = np.full([720, 1280],0,dtype=np.uint8) #im ersten Schritt wird unser Matrix erstellt
    while True:
        matrix = crystals_list[0].grow_crystal(matrix) #Startet die Funktion welche den Kristal[0] wachsen lässt
        if crystals_list[0].complete == True: #Falls der betrachtete Kristal bereits fertig ist,
            crystals_list.pop(0) #So wird er aus der Liste entfernt
            if len(crystals_list) == 0: #Überprüft ob die Liste mit den Kristalen leer ist, sollte dies der Fall sein, so ist da Bild vollendet
                return matrix #gibt die fertig angefärbte Matrix wieder
            continue #Und man macht mit dem nächsten Kristal weiter
        else:
            crystals_list.append(crystals_list.pop(0)) #Zum schluss wird der Kristal an der Stelle[0] wieder ganz nach hinten in der Liste verschoben
```