1 Pixelbasiertes Inpainting

In dieser Aufgabe implementieren Sie das pixelbasierte Inpainting-Verfahren.

1.1 Pattern bestimmen

Bestimmen Sie zunächst die sparse Maske für einen Pixel. Implementieren Sie hierzu die Funktion `create_pattern( origin, mask, k )`. Die Funktion akzeptiert ein tupel `origin (x,y)`, eine maske sowie eine Konstante k, z.B. 5. Sie bestimmt dann die k nächsten Nachbarn von `origin`, also die Koordinaten der definierten Pixel in der Umgebung von `origin`. Die Koordinaten werden dann relativ zu `origin` zurückgegeben.

Ist beispielsweise der Pixel 2 links von `origin` definiert, so enthält das Ergebnis die Koordinate $(-2, 0)$. Verwenden Sie zur Implementierung Ihre Lösung des letzten Aufgabenblattes.

a) Implementieren Sie Unittests sowie eine Visualisierung, was vor sich geht.

b) Implementieren Sie die Funktion.

Für die Bearbeitung der Aufgabe habe ich folgende Zeit benötigt:

1.2 Auswahl eines Pixels

Als nächstes wird eine Liste von Kandidaten erstellt. Gültige Kandidaten haben folgende Eigenschaften: a) der Pixel ist definiert. b) Wird die Maske auf den Pixel gelegt, so sind alle Pixel innerhalb der Maske ebenfalls definiert. Beispiel: die Maske ist `[(-2,0)(0,1)]`. Damit `(100/100)` ein gültiger Kandidate ist, müssen die folgenden Pixel definiert sein: `(100/100),(98,100),(100,101)`.

a) Implementieren Sie Unittests .

b) Implementieren Sie die Funktion.

Nun werden die Kandidaten bewerted. Die Distanz eines Pixels ist die sparse L_2-Norm der Farbwerte zwischen dem Zielpixel und dem Kandidaten, ausgewerted jeweils an den Positionen der sparsen Maske. Verwenden Sie den Kandidaten mit der geringsten Distanz.

c) Implementieren Sie Unittests.

d) Implementieren Sie die Funktion.

Für die Bearbeitung der Aufgabe habe ich folgende Zeit benötigt:

1.3 Inpainting-Schleife

Nun implementieren Sie die Inpainting-Schleife. Diese besteht aus folgenden Schritten:

1. Wähle zufällig einen nicht definierter Pixel aus. Dieser heisst nun `origin`.
2. Erstelle eine Liste mit n gültigen Kandidaten. n ist ein wichtiger Qualitäts vs. Performanz Parameter. Typische Werte sind 25 (sehr grob), 100 (grob), 1000 (gut) oder 10000 (sehr gut).
3. Wählen den besten Kandidaten aus.
4. Kopieren den Farbwert des besten Kandidaten nach `origin`. Nun ist `origin` definiert.
5. Iteriere, bis keine Pixel mehr undefiniert sind.

Für die Bearbeitung der Aufgabe habe ich folgende Zeit benötigt:

1.4 Performance Optimierung

a) Profilen Sie Ihre Anwendung. Welche Funktionen lohnt es sich zu optimieren?

b) Optimieren Sie Ihre Anwendung.

Für die Bearbeitung der Aufgabe habe ich folgende Zeit benötigt: