Aufgabe 3 (Raytracing-Implementierung in Python)
In `raytracing.zip` auf der Webseite zur Vorlesung[1] findet sich die Vorlage einer (einfachen) Python-Implementierung des Raytracing-Algorithmus (basierend auf "Ray Tracing in One Weekend" von Peter Shirley).

(a) Machen Sie sich mit der Implementierung vertraut, und vervollständigen Sie den Code in `material.py` und `utils.py` an den (mit `#### ... ####`) angegebenen Stellen. [4 Punkte]

(b) Rendern Sie nun ein Bild mit Tiefenschärfe, indem Sie die Variable `aperture` (d.h. die Linsenöffnung) in `main_depth_of_field.py` auf `1.0` setzen. [1 Punkt]

Hinweis: Der Befehl `python main_depth_of_field.py` bzw. `python3 main_depth_of_field.py` führt das Programm unter Python aus; die Laufzeit kann mehrere Minuten betragen. Das gerenderte Bild wird in der Datei `depth_of_field.png` abgespeichert.

(c) Je mehr Lichtstrahlen (rays) pro Pixel des gerenderten Bilds gemittelt werden, desto geringer ist das statistische Rauschen. Untersuchen Sie diesen Effekt, indem Sie `main_sampling.py` mit unterschiedlichen Werten des `ns` Parameters ("number of samples") laufen lassen, etwa `ns = 1`, `ns = 10` und `ns = 100`. [1 Punkt]

Aufgabe 4 (Monte-Carlo-Integration)

(a) Das Volumen V_4 der vierdimensionalen Einheitskugel beträgt $\pi^2/2$. Ausgedrückt durch die Indikatorfunktion

$$f_{\mathsf{S}} : \mathbb{R}^4 \to \mathbb{R}, \quad f_{\mathsf{S}}(x) = \begin{cases} 1, & \|x\| \leq 1 \\ 0, & \text{sonst} \end{cases}$$

gilt also

$$V_4 = \int_{\mathbb{R}^4} f_{\mathsf{S}}(x)\,\mathrm{d}^4 x = \frac{\pi^2}{2}.$$

Schreiben Sie ein Python-Programm, das dieses Integral durch Monte-Carlo-Integration approximiert. Generieren Sie hierzu n (unabhängige) zufällige Punkte $x_i \in [-1,1]^4$ (gleichverteilt), und berechnet Sie

$$\tilde{V}_4^{(n)} = 2^4 \frac{1}{n} \sum_{i=1}^{n} f_{\mathsf{S}}(x_i).$$

(Der Faktor $2^4 = 16$ berücksichtigt das Volumen des Würfels mit Kantenlänge 2, aus dem die x_i gewählt werden.) Visualisieren Sie nun den Fehler $|\tilde{V}_4^{(n)} - V_4|$ als Funktion von n in einem doppelt-logarithmischen Plot. Es genügt, für n die Werte 2^j mit $j = 0, 1, \ldots, 19$ zu wählen. [3 Punkte]

Hinweis: Der NumPy-Aufruf `2*np.random.rand(4)-1` liefert einen vierdimensionalen Zufallsvektor aus $[-1,1]^4$. Die `loglog`-Funktion der Matplotlib-Bibliothek eignet sich zum Erstellen eines doppelt-logarithmischen Plots.

(b) Zur Monte-Carlo-Integration von

$$I_c = \int_{-\infty}^{\infty} \cos(2x) \frac{\mathrm{e}^{-\frac{x^2}{2}}}{\sqrt{2\pi}}\,\mathrm{d}x$$

ist es hilfreich, den Term $\frac{\mathrm{e}^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$ im Integranden als Wahrscheinlichkeitsdichte der Gauß-Standardnormalverteilung zu identifizieren und entsprechende Zufallsvariablen zu verwenden. Generieren Sie also n zufällige, normalverteilte Punkte $x_i \in \mathbb{R}$ und approximieren Sie das Integral durch

$$\tilde{I}_c^{(n)} = \frac{1}{n} \sum_{i=1}^{n} \cos(2x_i).$$

Visualisieren Sie nun analog zu (a) den Fehler $|\tilde{I}_c^{(n)} - I_c|$ als doppelt-logarithmischen Plot, und vergleichen Sie die Konvergenzrate mit $1/\sqrt{n}$. Der exakte Wert beträgt $I_c = \mathrm{e}^{-2}$. [3 Punkte]

Hinweis: Sie erhalten eine standardnormalverteilte Zufallszahl mittels `np.random.standard_normal()`.

[1] https://tu-dresden.de/mn/math/wir/mendl/studium/courses/mosim_2018_wise

Tutoraufgabe 2 (Spiralmuster)
Spiralmuster in der Natur, wie z.B. der Blütenstand einer Sonnenblume, lassen sich oft durch Iterationsvorschriften beschreiben.[2] Wir bezeichnen die Positionen der einzelnen Bestandteile (etwa der Samenkerne bei der Sonnenblume) mit $p_n \in \mathbb{R}^2$, $n = 0, 1, 2, \ldots$. Im einfachsten Fall ist die Iterationsvorschrift von der Form

$$p_{n+1} = T(p_n)$$

mit einer Funktion $T : \mathbb{R}^2 \to \mathbb{R}^2$, d.h. die nächste Position hängt nur von der aktuellen Position ab. Wir verwenden im Folgenden Polarkoordinaten $p_n = r_n(\cos(\psi_n), \sin(\psi_n))$ mit Radius $r_n \geq 0$ und Winkel ψ_n, und untersuchen

$$\psi_{n+1} = \psi_n + \Delta\psi, \quad r_{n+1} = a\sqrt{\psi_{n+1}}$$

mit Startwerten $\psi_0 = 0$, $r_0 = 0$. Der "Divergenzwinkel" $\Delta\psi$ zwischen zwei aufeinanderfolgenden Punkten ist also konstant.

(a) Bei der Anordnung von Blättern um eine Achse werden Divergenzwinkel $\Delta\psi = 2\pi q$ mit

$$q = \frac{F_k}{F_{k+2}}$$

beobachtet, wobei F_k die k-te Fibonacci-Zahl $(0, 1, 1, 2, 3, 5, \ldots)$ bezeichnet. Begründen Sie allgemein für rationales $q \in \mathbb{Q}$, dass ein $\ell \in \mathbb{N}$, $\ell \geq 1$ existiert, so dass die Punkte p_n, $p_{n+\ell}$, $p_{n+2\ell}$, ... (mit $n \in \mathbb{N}_0$ beliebig) auf einem vom Zentrum ausgehenden Strahl liegen.

(b) Das Muster einer Sonnenblume lässt sich mittels (irrationalem) Divergenzwinkel

$$\Delta\psi_s = 2\pi \lim_{k\to\infty} \frac{F_k}{F_{k+2}}$$

erzeugen. Berechnen Sie den Grenzwert ausgehend von der Formel $F_k = (\varphi^k - (1-\varphi)^k)/\sqrt{5}$, wobei $\varphi = (1+\sqrt{5})/2$ der Goldene Schnitt ist.

(c) Für Divergenzwinkel $\Delta\psi_s$ und einer fest gewählten, nicht zu kleinen Fibonacci-Zahl ℓ werden Punkte p_n, $p_{n+\ell}$, $p_{n+2\ell}$, ... visuell als zusammengehörig empfunden (sogenannte Spiralarme). Begründen Sie dies anhand der entsprechenden Winkel.

(d) Das folgende Python-Programm

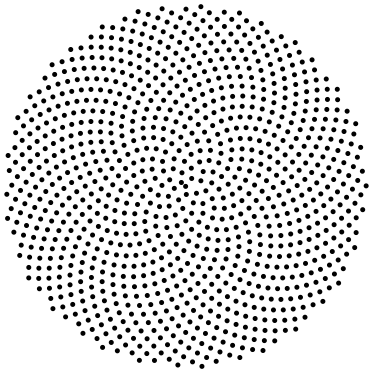
```
import numpy as np
import matplotlib.pyplot as plt

n = 1000    # number of points

# difference angle
dpsi = 2*np.pi * (3 - np.sqrt(5))/2
psi  = dpsi * np.arange(n)

a = np.sqrt(n / (2*np.pi))
r = a*np.sqrt(psi)

plt.plot(r*np.cos(psi), r*np.sin(psi), 'k.', markersize=3.5)
plt.axis('equal'); plt.axis('off')
```



generiert das nebenstehende Spiralmuster. Erweitern Sie das Programm, so dass ein Spiralarm p_0, p_ℓ, $p_{2\ell}$, ... mit einer von Ihnen frei wählbaren Fibonacci-Zahl $\ell \geq 21$ eingezeichnet wird. Wie wirkt sich die "archimedische" Vorschrift $r_n = a\psi_n$ (anstatt $r_n = a\sqrt{\psi_n}$) auf das Muster aus?

[2] siehe z.B. http://www.science.smith.edu/phyllo/