TU BRAUNSCHWEIG
Prof. Dr.-Ing. Marcus Magnor
Institut für Computergraphik
Contact: cgg@cg.cs.tu-bs.de

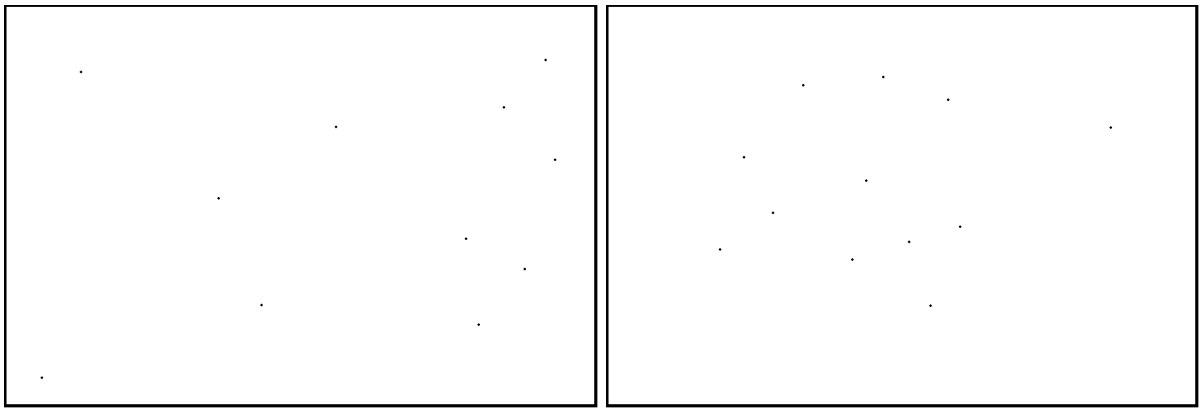
January 20, 2023

Computer Graphics WS 22/23
Assignment 8

This week we will do a bit of theoretical work testing what you have learned so far. Complete the assignments and hand in your solutions to these theoretical tasks (with drawings/formulas). Please use different colors in your drawings. Each group hands in one solution before **Friday, 9:45**.

8.1 Sketch kd-tree from point set (30 Points)

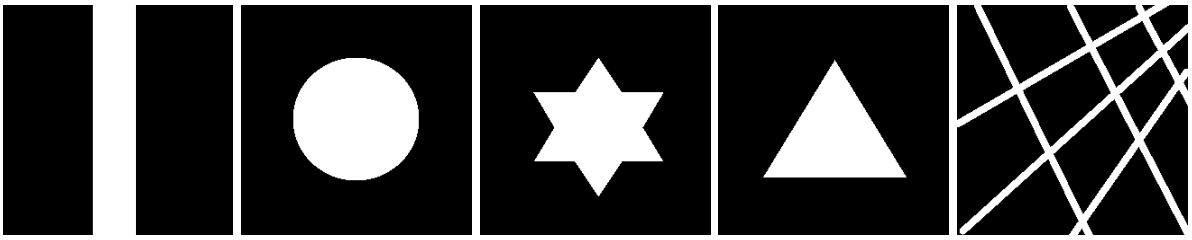
In the following you have to sketch a kd-tree for a given (2d) point set. At first, have a look at `http://homes.ieu.edu.tr/hakcan/projects/kdtree/kdTree.html` to get an overview on kd-tree building for point sets.
In this task we will use the same algorithm for kd-tree building, i.e.: The first axis is vertical. The splitting axis at each level coincides with the median of a given dataset. For a vertical split the left child contains the points left of or on the splitting axis, the right child contains the points right of the splitting axis. For a horizontal split the left child contains the points below the or on the splitting axis, the right child contains the points above the splitting axis. A leaf node contains only one point. Now sketch the kd-trees for the following two point sets:



Also provide the tree structure with nodes and links. Name the points p_1, p_2, the splitting axis l_1, l_2...

8.2 Guess the Fourier images (20 Points)

The following frequency domain images (second row) have been created for spatial domain images (first row) using numpy's `fft2` method. However, the frequency domain images have been mixed up. Use your knowledge about the transformation properties of edges to guess, which frequency domain image belongs to which spatial domain image



(a) (b) (c) (d) (e)

(a) (b) (c) (d) (e)

8.3 Fourier Transformation (20 Points)

The transformation of a signal $f(x)$ to Fourier space is given by

$$F(\omega) = \int_{\infty}^{+\infty} f(x) e^{-2\pi i \omega x} dx$$

Consider the box function

$$B_d(x) = \begin{cases} 0 & \text{for } x \leq -d \\ 1 & \text{for } -d < x < d \\ 0 & \text{for } d \leq x \end{cases}$$

and show that its Fourier transformation is a *sinc* type function. Compare the behavior of the functions as $x \to \pm\infty$.

8.4 Inverse Fourier Transform and Image Processing (10 Points)

A checkerboard image I has been applied the Fourier transform to get $g(x, y) = F\{I\}$. Then in Fourier space a multiplication with a 2d function $f(x, y)$ has been performed, where $f(x, y)$ is defined as

a) $f(x, y) = 1$ for $\sqrt{x^2 + y^2} \leq r$ and $f(x, y) = 0$ otherwise

b) $f(x, y) = 0$ for $\sqrt{x^2 + y^2} \leq r$ and $f(x, y) = 1$ otherwise

Sketch the resulting images when applying the inverse Fourier transform to $f \cdot g$ for both definitions of $f(x, y)$. Describe briefly what type of filtering has been realized for both situations.

8.5 Sampling Theory (20 Points)

Let $f(x)$ be an infinite signal. Consider a regular sampling $f_S(x)$ of $f(x)$ with sample distance T, that fulfills the Nyquist property, thus the highest frequency of the signal is smaller than $\frac{1}{2T}$

a) Is the exact signal reconstruction of $f(x)$ possible? If so, why?

b) How has the reconstruction to be performed in image and Fourier space?

8.6 Diffuse Globale Beleuchtung (Bonus, 20 Punkte)

Ein wichtiges physisches Phänomen, das sich mit Raytracing gut simulieren lässt, ist globaler Lichttransport, d.h. die Verfolgung von Licht, das mehrfach auf dem Weg von der Lichtquelle zur Kamera gestreut wird. Um diesen Effekt zu simulieren, werden mehrere Pfade pro Pixel durch die Szene verfolgt und die über den Pfad transportierte Radiance summiert. Dieser Prozess ist stochastisch, wird also durch eine zufällige Richtungswahl an jedem Szenenschnittpunkt umgesetzt und solange fortgesetzt, bis der Pfad an einer Lichtquelle endet oder die maximale Rekursionstiefe erreicht ist.
Legen Sie hierfür zwei Dateien `shader/lambertgi.h` und `shader/lambertgi.cpp` an, die eine Variante des `LambertShader` implementieren. Anstatt über alle Lichtquellen zu iterieren, wählen Sie eine zufällig aus, die dann wie gewohnt ein *sample* für die direkte Beleuchtung liefert (allerdings skaliert mit der Anzahl der Lichtquellen). Für die indirekte Beleuchtung erzeugen Sie einen neuen Strahl mit einer Richtung, die uniform über die obere Hemisphäre verteilt ist, wie folgt:
Erzeugen Sie zwei uniform verteilte Zufallszahlen $(\xi_1, \xi_2) \in [0,1]^2$ und wenden Sie dann die Transformation $\phi = 2\pi\xi_1$ und $\theta = \cos^{-1}(1 - \xi_2)$ an, um Polarkoordinaten im lokalen Koordinatensystem des Schnittpunktes zu berechnen (mehr dazu finden Sie hier: `https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/global-illumination-path-tracing-practical-implementation`). Nutzen Sie die Normalen-, Tangenten- und Bitangentenvektoren, um dann diese wieder in kartesische Weltkoordinaten umzuwandeln und als neue Strahlrichtung für einen indirekten Strahl zu verwenden. Beachten Sie, dass Sie dabei *nicht*, die Tangenten und Bitangentenvektoren aus der Strahlstruktur nehmen, da diese für die gegebene Szene nicht immer korrekte Ergebnisse liefern. Da die Richtung aber ohnehin zufällig gewählt ist, können Sie dies umgehen, indem Sie einfach ein lokales Orthonormalsystem berechnen (das aber auch deterministisch berechnet werden muss, sonst könnten die Verteilungsfunktionen nicht mit der Gleichverteilung übereinstimmen). Verrechnen Sie beide einfallenden *samples* wie vom `LambertShader` gewohnt und geben Sie das Resultat zurück. *Beachten Sie, dass die Berechnungen hier sehr lange dauern können. Zum Testen empfehlen wir daher, dass Sie entweder die Anzahl der samples in der `task_gi.cpp` oder die maximale Rekursionstiefe in der `CMakeLists.txt` (der Parameter `ICG_RAY_BOUNCES`) verringern. Weitere allgemeine Hinweise für den hier umzusetzenden Algorithmus finden Sie unter dem Begriff* ***Monte-Carlo Integration*** *mit* ***Next Event Estimation***.

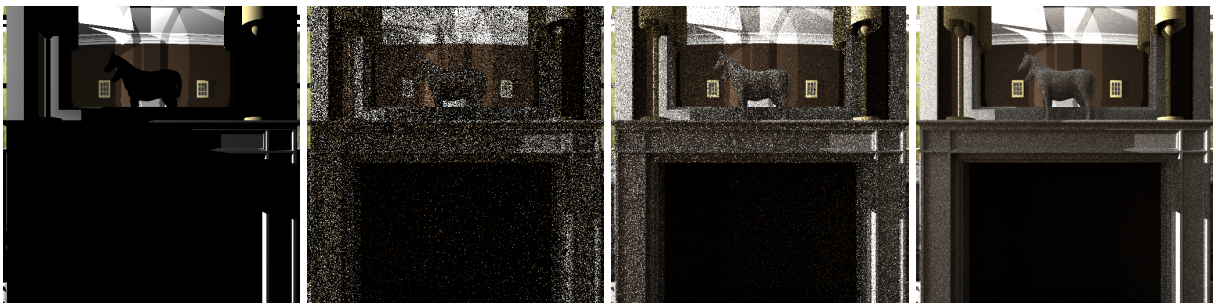


Figure 3: Wenn Sie alles richtig gemacht haben, sollte das Ergebnis von `task_gi` wie ganz rechts aussehen. Von links nach rechts: direkte Beleuchtung, 1 sample pro Pixel, 16 samples pro Pixel, 256 samples pro Pixel.