Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
Dipl.-Inform. Johannes Meng, Dipl.-Inform. Florian Simon,
M.Sc. Emanuel Schrade

4. Übungsblatt zur Vorlesung Computergraphik im WS 2016/17

Abgabe bis Montag, 09.01.2017, 11:00 Uhr.

In diesem Übungsblatt sollen Sie eine Bounding Volume Hierarchy implementieren um effizient eine große Anzahl an Dreiecken in einer Szene handhaben zu können.

Außerdem soll ein einfacher Gauß-Filter implementiert werden mit dem Texturen vorgefiltert werden können.

Die Ausgaben einer kompletten Implementierung sind in Abbildung 2 dargestellt. Ihre Implementierung beschränkt sich wieder auf die Datei `exercise_04.cpp`. Lesen Sie bitte das Übungsblatt sorgfältig und machen Sie sich mit dem Code des Frameworks vertraut.

1 *Gauß-Filter* — **8 Punkte**

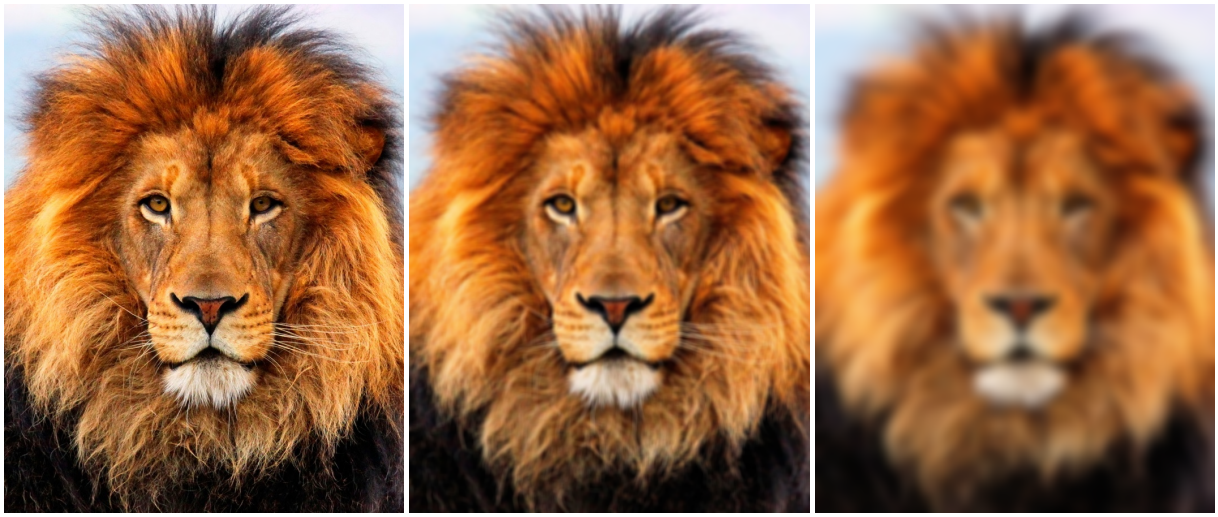


Abbildung 1: Effekte des Weichzeichnens mit einem Gauß-Filter. Links: Orginalbild 707×900. Mitte: $\sigma = 5$, Kernelgröße 9×9. Rechts: $\sigma = 10$, Kernelgröße 49×49.

Für das *Weichzeichnen* eines Bildes verwendet man in der Computergrafik sehr häufig einen Gauß-Filter. Dabei wird für jeden Pixel im Bild seine lokale Nachbarschaft gemäß einer zweidimensionalen Gauß-Glocke der Breite σ gewichtet:

$$g(x, y) = \frac{1}{2\pi\sigma^2} \, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

wobei x und y der horizontale und vertikale Abstand in Pixeln ist.

a) **(4 Punkte)** In dieser Aufgabe soll in den Funktionen `Image::create_gaussian_kernel_1d` bzw. `Image::create_gaussian_kernel_2d` ein 1-dimensionalen bzw. 2-dimensionalen Gauß-Filterkernel erstellt werden. Die normalisierten Gauß-Filterkernel der Größe 3 mit $\sigma = 1$ (bis zur dritten Nachkommastelle) sind:

0.274	0.452	0.274

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

b) **(2 Punkt)** Implementieren Sie nun die Methode `Image::filter`, die eine gefilterte Version des Bildes berechnet und zurückgibt. Verwenden Sie dazu den zweidimensionalen, als Parameter übergebenen Filterkernel.

c) **(2 Punkt)** Gauß-Filter sind *separierbar*, d.h. ein zweidimensionaler Gauß-Filter lässt sich realisieren, indem man das Bild zuerst horizontal und danach vertikal (oder umgekehrt) mit einem eindimensionalen Gauß-Filter filtert. In `Image::filter_separable` soll eine separiert gefilterte Version des Bildes berechnet und zurückgegeben werden. Verwenden Sie dazu den eindimensionalen, als Parameter übergebenen Filterkernel.

Die gefilterten Bilder können Sie erzeugen, indem Sie auf der Konsole das Framework mit `--gauss` aufrufen. Alternativ können Sie in der GUI den Render Mode `View image` aktivieren um im Image Mode `Image Filtering` das gefilterte Bild anzuzeigen.

Das Ergebniss des naiven und des separierten Gauß-Filters ist dasselbe. Welche Variante würden Sie in der Praxis bevorzugen? Wenn Sie in der GUI die Einstellung `Use Filtered Envmap` aktivieren sollten Sie die gefilterte Environment Map sehen können.

2 *Bounding Volume Hierarchy* — 12 Punkte

Abbildung 2: Ergebnisbilder des Übungsblattes.

a) **(6 Punkte)** Nun soll eine Hüllkörperhierarchie (BVH) mit *Object Median Split* aufgebaut werden. Mit einer BVH wird es möglich werden, komplexe Geometrie in Form von Dreiecksnetzen im Renderer zu nutzen. Als Hüllkörper sind dabei an den Achsen ausgerichtete Quader zu wählen, die mit der Klasse `AABB` (`cglib/include/cglib/rt/aabb.h`) dargestellt werden.

In dieser Aufgabe müssen Sie die Methoden `build_bvh` und `reorder_triangles_median` der `BVH`-Klasse implementieren.

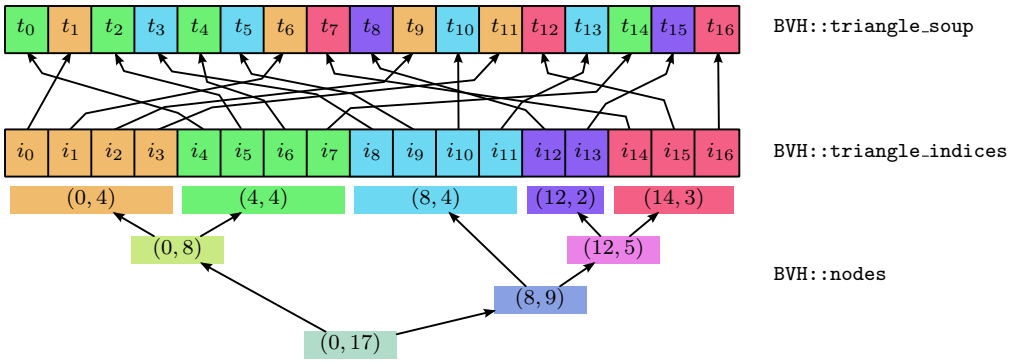


Abbildung 3: Dreiecke liegen in einer ungeordneten *triangle soup* vor (oben). Die BVH speichert zusätzlich eine durch den Aufbaualgorithmus sortierte Indexliste (Mitte). Zu jedem Knoten der BVH gehört ein Teilstück dieser Liste und somit ein Teil der gespeicherten Dreiecke. In Klammern angegeben sind die Werte von `Node::triangle_idx` und `Node::num_triangles` des jeweiligen Knotens.

Die Methode `build_bvh` steuert den rekursiven Aufbau der BVH. Sie bearbeitet immer einen Knoten und alle zu diesem Knoten gehörigen Dreiecke. Der aktuelle Knoten wird in `build_bvh` einmal unterteilt, falls er mehr als `BVH::MAX_TRIANGLES_IN_LEAF` Dreiecke enthält. Die Unterteilung wird dann rekursiv auf den Kindknoten fortgesetzt.

Die Methode `reorder_triangles_median` teilt die zu einem Knoten gehörigen Dreiecksindices in zwei Teile, wobei ein Teil maximal einen Index mehr haben darf, als der andere. Dazu werden alle Dreiecke, die vor dem Objektmedian liegen, in die erste Hälfte der Indexliste einsortiert. Die Reihenfolge wird dabei durch die Mittelpunkte der Dreiecks-AABB und die gegebene Unterteilungsachse definiert. Die Dreiecke innerhalb eines Blattknotens müssen nicht sortiert sein.

Vor der Implementierung werden Sie sich über die Schnittstelle der `BVH`-Klasse informieren müssen. Sie finden diese in `cglib/include/cglib/rt/bvh.h`. Eine schematische Darstellung der Datenstruktur sehen Sie in Abbildung 3. In der Funktion `intersect_recursive` können Sie nachsehen, wie das Datenlayout funktioniert und wie auf die Dreiecksdaten zugegriffen werden kann.

Das Framework stellt verschiedene Visualisierungsmodi bereit, die Sie zur Fehleranalyse verwenden können. Mit dem Modus ,,BVH Intersection Time" wird pro Pixel die Zeit, die zum berechnen des Schnittpunktes benötigt wurde, visualisiert. Der Modus ,,AABB Intersection Count" schneidet einen Strahl mit sämtlichen Hüllkörpern der BVH. Dieser Modus soll dazu dienen die Struktur der Hierarchie zu analysieren.

Stellen Sie folgende Punkte sicher:

- Die AABB eines Knoten soll immer kleinstmöglich sein, aber alle Dreiecke des jeweiligen Teilbaumes umschließen. Um degenerierte AABBs zu vermeiden werden diese aber um ein Epsilon in jeder Dimension vergrößert (siehe `AABB::extend` in `aabb.h`).

- Ein innerer Knoten soll stets genau die Dreiecke enthalten, die in seinen Kindern enthalten sind. Insbesondere enthält der Wurzelknoten alle Dreiecke.

- Bei inneren Knoten sollen beide Kindzeiger `left` und `right` auf gültige Elemente des Vektors `BVH::nodes` verweisen. Bei Blattknoten sollen beide Kindzeiger den Wert -1 haben. Eine Mischform (ein Zeiger gültig, der andere -1) ist nicht erlaubt.

- Zu einem Knoten gehören immer `Node::num_triangles` Dreiecke, deren Indices in einem zusammenhängenden Stück von `BVH::triangle_indices` liegen und bei Element `Node::triangle_idx` beginnen.
- Blattknoten sollen maximal `BVH::MAX_TRIANGLES_IN_LEAF` Dreiecke enthalten.
- Kein Blattknoten darf weniger als ein Dreieck enthalten.
- Der Split soll in der Reihenfolge $X, Y, Z, X, Y, Z, \ldots$ entlang je einer der Hauptachsen erfolgen. Der Wurzelknoten soll entlang der X-Achse geteilt werden.
- Ein Dreieck soll vor einem anderen Dreieck einsortiert werden, wenn der Mittelpunkt seiner AABB vor dem Mittelpunkt der AABB des anderen Dreiecks liegt.
- Der BVH-Aufbau muss für alle möglichen Eingaben terminieren, also nicht in einer Endlosschleife enden.
- Für die volle Punktzahl darf der Aufbau der BVH für die Sponza-Szene (in der VM) nicht länger als 2 Minuten dauern. Benutzen Sie daher einen effizienten Algorithmus zum Sortieren. (z.B. `std::sort` oder `std::nth_element`).

Beachten Sie, dass durch eine richtige Implementierung der Vektor `triangle_indices` umsortiert wird.

b) **(6 Punkte)** In der Methode `BVH::intersect_recursive` soll nun die BVH rekursiv traversiert werden. Die Behandlung von Blattknoten ist schon implementiert. Sie müssen sich daher nur um die Behandlung von inneren Knoten kümmern. Schneiden Sie dafür den Strahl mit den AABBs der zwei Kindknoten. Im Fall, dass beide Kindknoten geschnitten werden sollen Sie zuerst in den näheren Kindknoten absteigen. Ein Kindknoten ist dabei näher, wenn der Schnittpunkt näher am Strahlurprung liegt.

3 *Bonus: Fourier Transformation* — 8 Punkte

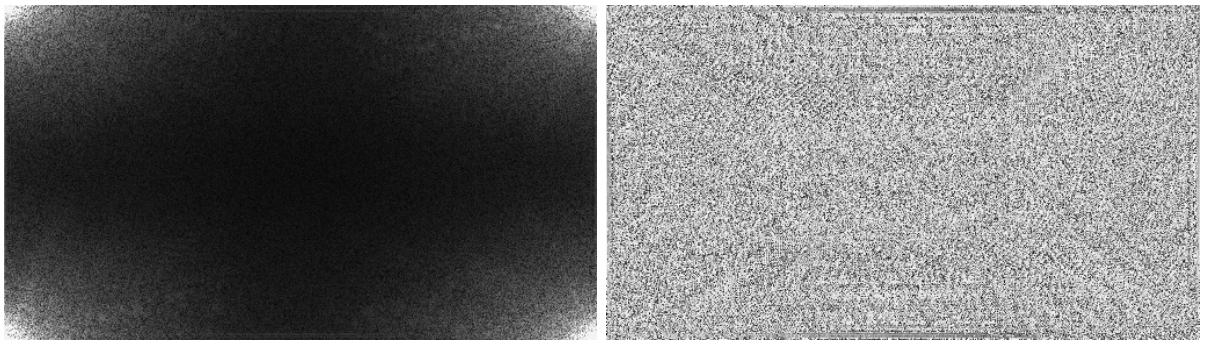


Abbildung 4: Amplitude (links) und Phase (rechts) des Fourierspektrums eines unbekannten Graustufenbildes.

In dieser Bonusaufgabe sollen Sie ein geheimes Graufstufenbild $\{x_{kl}\}, (k = 0, \ldots, M-1, l = 0, \ldots, N-1)$ aus einem Fourierspektrum rekonstriueren. Das Fourierspektrum ist in der Datei `assets/mistery.pfm` als Portable Float Map (pfm) gegeben. Dies ist ein sehr einfaches Bildformat, dass aber leider nur wenige Image Viewer anzeigen können. Der Real- bzw. Imaginärteil der Fourierkoeffizienten befindet sich in der R- bzw G-Komponente des Bildes.

Die Fourierkoeffizienten $\{\hat{x}_{kl}\}, (k = 0, \ldots, M-1, l = 0, \ldots, N-1)$ wurden über die Fouriertransformation

$$\hat{x}_{kl} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{mn} e^{-2\pi i \frac{mk}{M}} e^{-2\pi i \frac{nl}{N}} \tag{1}$$

berechnet. Die inverse Transformation ist

$$x_{kl} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \hat{x}_{mn} e^{2\pi i \frac{mk}{M}} e^{2\pi i \frac{nl}{N}} \tag{2}$$

$$\tag{3}$$

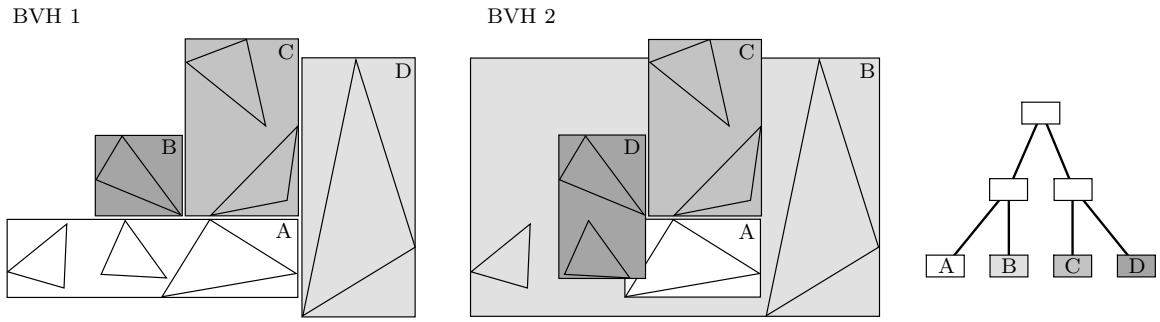
Das Fourierspektrum und das ursprüngliche Bild haben die selbe Auflösung. Über die Funktionen `complex_to_image` und `image_to_complex` der Klasse `Image` können Fourierspektren und Bilder ineinander umgewandelt werden. In der Funktion `fourier` in der Datei `main.cpp` wird das Fourierspektrum geladen und die inverse Fouriertransformation aufgerufen. Implementieren Sie diese in der Funktion `DiscreteFourier2D::transform` in `exercise_04.cpp`. Um die Rekonstruktion durchzuführen müssen Sie das Framework mit `--fourier` aufrufen. Dabei wird eine Datei `reconstruction.png` im Ordner `assignment_images` erzeugt, die Sie in der GUI im Render Modus `Image Viewer` durch Auswahl des Images Mode `Fourier Transform` anzeigen können. Hier können Sie auch die Amplitude und Phase des Fourierspektrums ansehen 4.

Tipp: Die Fouriertransformation ist, wie der Gaußfilter, separierbar. Nutzen Sie dies aus, um die Rekonstruktion zu beschleunigen.

Hinweis: Bitte spoilen Sie die Aufgabe nicht für ihre Kommilitonen, indem Sie das rekonstruierte Bild im Forum o.ä. veröffentlichen.

4 Beschleunigungsstrukturen und Hüllkörper (Theorie, keine Abgabe)

a) Die folgende Abbildung zeigt die Blattknoten zweier Hüllkörperhierarchien (BVH). Beide Hierarchien beinhalten dieselben Dreiecke und besitzen je vier Blattknoten A, B, C, D. Die Topologie beider BVH ist gleich und als Baum rechts dargestellt.



Welche der beiden Hierarchien kann von einem Raytracer effizienter durchlaufen werden? Begründen Sie dies in *Stichpunkten*!

BVH 1 ☐ BVH 2 ☐

b) Bewerten Sie die folgenden Aussagen, indem Sie *Wahr* oder *Falsch* ankreuzen!

Aussage	**Wahr**	**Falsch**
Ein kD-Baum kann für N Primitive den Aufwand für Strahlschnitte von $O(N)$ auf $O(\log(N))$ reduzieren.	☐	☐
Die Surface Area Heuristic sorgt dafür, dass beiden Kindknoten gleich viele Primitive zugeteilt werden.	☐	☐
Für die Suche der Trennebene (Split Plane) mit Objektmittel (Object Median) ist *immer* eine vollständige Sortierung der Primitive notwendig.	☐	☐
Teilen eines Knotens am Objektmittel (Object Median) führt stets zu den effizientesten Hüllkörperhierarchien.	☐	☐
Die Surface Area Heuristic macht die Annahme, dass Strahlen stets außerhalb des zu unterteilenden Hüllkörpers starten.	☐	☐
Es gibt Szenen, in denen ein kD-Baum keinen Vorteil bringt.	☐	☐

c) Kreuzen Sie jeweils die passenden Kästchen an!Sie erhalten für jede vollständig richtige Zeile einen Punkt.

Aussage	**AABB**	**OBB**	**Kugel**
Der Hüllkörper kann einen beliebig orientierten Würfel optimal, also ohne freien Raum umschliessen.	☐	☐	☐
Es gibt *orthonormale* Transformationen des eingeschlossenen Objektes, die das Volumen des optimalen Hüllkörpers verändern.	☐	☐	☐
Es gibt *affine* Transformationen des eingeschlossenen Objektes, die das Volumen des Hüllkörpers verändern.	☐	☐	☐
Gegegeben sind zwei Objektmengen und zu jeder Menge ihr optimaler Hüllkörper. Der Aufwand, den optimalen Hüllkörper für *alle* Objekte zu bestimmen, ist unabhängig von der Anzahl der Objekte.	☐	☐	☐

Abgabe

Laden Sie die Datei `exercise_04.cpp` in Ilias hoch.

Framework

Wir werden für jedes Übungsblatt ein Framework bereitstellen, das Sie im Ilias-Kurs unter `https://ilias.studium.kit.edu/goto.php?target=crs_607961&client_id=produktiv` herunterladen können. Das Framework nutzt C++ 11 und wird unter Linux getestet. Es ist allerdings auch unter Windows mit Visual Studio 2013 lauffähig.

Sie können das heruntergeladene Archiv unter Linux mit dem Befehl

```
$ unzip archiv.zip
```

entpacken, wobei Sie `archiv.zip` durch den jeweiligen Dateinamen ersetzen müssen.

Grundsätzlich wird das Framework immer ein Unterverzeichnis `cglib` enthalten. Sie dürfen und sollen den Quellcode in diesem Unterverzeichnis lesen.

Je nach Aufgabe wird es auch ein zweites Unterverzeichnis geben. Für das vorliegende Übungsblatt heißt dieses `04_bvh`. Hier werden Sie Ihre Lösung programmieren.

Die Dateien `VirtualMachine.txt` und `Kompilieren.txt` enthalten Informationen darüber, wie sie die Virtuelle Maschine zur Übung installieren und das Framework kompilieren. Bitte lesen Sie diese Informationen.

Achtung: Abgegebene Lösungen müssen in der VM erfolgreich kompilieren und lauffähig sein, ansonsten vergeben wir 0 Punkte. Insbesondere darf Ihre Lösung nicht abstürzen.

Allgemeine Hinweise zur Übung:

- Scheinkriterien: Sie benötigen 60% der Punkte aus den Praxisaufgaben.
- Die theoretischen Aufgaben bedürfen üblicherweise *keiner* elektronischen Abgabe.
- Die Aufgaben müssen in Ilias bis spätestens Montag, 09.01.2017, 11:00 Uhr abgegeben werden.
- Die Abgabe muss im Ordner `build` mit `cmake ../ && make` in der bereitgestellten VirtualBox VM kompilieren, andernfalls wird die Aufgabe mit 0 Punkten bewertet.
- Da nur einzelne Dateien abgegeben werden, müssen diese kompatibel zu unserer Referenzimplementation bleiben. **Verändern Sie daher wirklich nur die Dateien, die auch abgegeben werden müssen**, bzw. *nicht* die mitgelieferten Funktionsdeklarationen! Sie können allerdings in in den *abzugebenden* Dateien gerne Hilfsfunktionen definieren und benutzen.
- Sie dürfen sehr gerne untereinander die Aufgaben diskutieren, allerdings muss jeder die Aufgaben *selbst* lösen, implementieren und abgeben. Plagiate bewerten wir mit 0 Punkten.
- Wenden Sie sich bei Fragen an einen Übungsleiter. Unsere Büros sind in Gebäude 50.34.

Johannes Meng	Raum 142	`meng@kit.edu`
Emanuel Schrade	Raum 140	`schrade@kit.edu`
Florian Simon	Raum 142	`florian.simon@kit.edu`