Prof. G. Zachmann
J. Boeckers
A. Mühlenbrock
N. M. Jadid

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
4. Dezember 2024

Wintersemester 2024/25

Übungen zu Computergraphik - Blatt 8

Abgabe am 15.12.2023, 23:59 Uhr

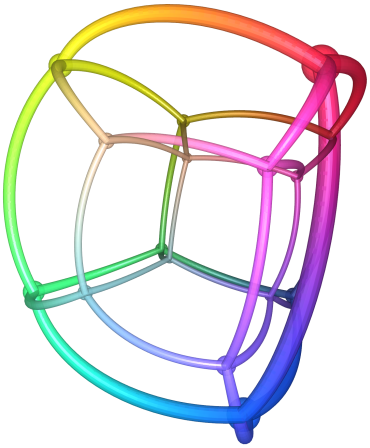

Abbildung 1: **Unsortiert**. Renderergebnis von Transparenz ohne Sortierung. Extreme Fehler, welche die Geometrie kaum erkennbar machen.

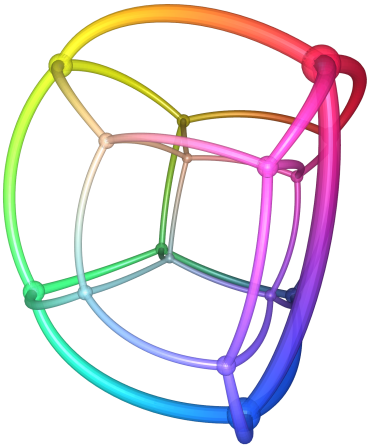

Abbildung 2: **BSP ohne Splitting**. Renderergebnis bei BSP Sortierung ohne Splitting. Die Geometrie ist besser erkennbar, aber kleine Fehler sind weiterhin vorhanden (s. Abbildung 4).

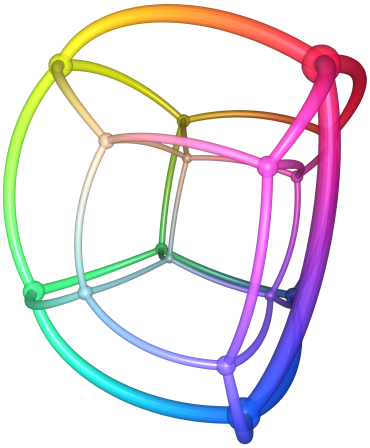

Abbildung 3: **BSP mit Splitting**. Renderergebnis bei BSP Sortierung mit Splitting. Die Transparenz wird korrekt angezeigt.

Aufgabe 1 (Binary Space Partition: Aufbau, *1+6+1 Punkte*)

Zunächst geht es um den Aufbau des BSP-Baumes. Orientieren Sie sich an den Algorithmen die in der Vorlesung vorgestellt wurden!

a) Implementiere dafür die Berechnung einer Normale zu einem 3D Dreieck in der statischen Funktion `BspNode::CreateNormal()`, welche die 3 Eckpunkte nimmt und daraus die Normale des Dreiecks berechnet. Diese Funktion wird bei der Erzeugung des BSP-Baumes genutzt, um die Ebene zu definieren, welche durch ein Dreieck verläuft. Die Normale und ein beliebiger Punkt des Dreiecks definieren eine Ebene implizit. Die Ebene wird später genutzt, um zu entscheiden, auf welcher Seite der Ebene ein Dreieck liegt. Die Normale soll senkrecht auf dem Dreieck liegen und die Länge 1 haben.

b) In dem Konstruktor `BspNode::BspNode()` soll nun der BSP-Baum erzeugt werden. Wie der Name `BspNode` suggeriert, betrachten wir immer nur einen Knoten. Jedoch hat jeder Knoten Unterbäume in `m_front` und `m_back`. Das sind wiederum Pointer auf jeweils eine `BspNode`. Diese Struktur ist rekursiv, somit können wir allein mit der Knotenklasse den gesamten Baum definieren. Der Wurzelknoten dieses Baumes agiert somit als der Baum an sich (dieser wird in `Mesh::bspRootNode` und `Mesh::bspRootNodeNoSplit` gespeichert). Der Baum wird beim

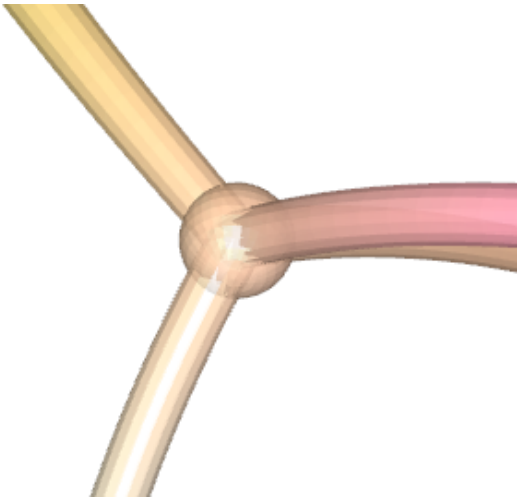

Abbildung 4: **Artefakte ohne Splitting**. Ohne Splitting entstehen Renderartefakte, weil das ganze Polygon entweder vor oder hinter der Splittingebene eingeordnet wird.

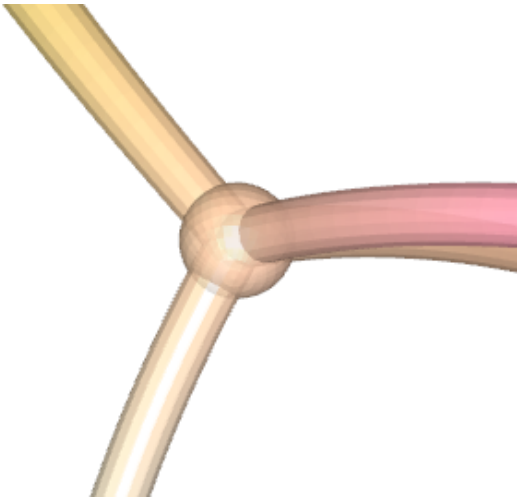

Abbildung 5: **Keine Artefakte mit Splitting**. Mit korrektem Splitting verschwinden die Artefakte.

Start des Programms einmalig erzeugt. Splitting von Dreiecken soll nur durchgeführt werden, sofern der Parameter `split` den Wert `true` hat. Abbildung 4 und Abbildung 5 zeigen, welche Artefakte entstehen, wenn man Splitting nicht nutzt.

Achte darauf, die Rekursion zu beenden, wenn einer der Teilbäume keine Dreiecke mehr enthält, ansonsten entsteht ein Stackoverflow.

Bonus *(1 Punkt)*: Überlege dir eine sinnvolle Methode, die Dreiecke zu sortieren, falls diese (im Fall ohne Splitting) die Ebene schneiden, so dass die Seite gewählt wird, auf welcher mehr vom Dreieck liegt. Damit kann das Renderergebnis leicht verbessert werden(s. Abbildung 2 & Abbildung 4; die Bilder zeigen den bereits verbesserten Fall).

c) In dem Konstruktor wird per `new` Speicher auf dem Heap reserviert, welcher im Dekonstruktor `BspNode::~BspNode` wieder freigegeben werden soll. Der Speicher soll rekursiv freigegeben werden, d.h. vom gesamten Baum. Den Dekonstruktor eines anderen Objekts kann man per `delete` aufrufen.

Solltest du `std::shared_ptr` verwendet haben, dann ist hier vermutlich nichts zu tun.

Aufgabe 2 (Binary Space Partition: Traversierung, *3 Punkte*)

Nachdem der Baum erstellt wurde, soll er in dieser Aufgabe traversiert werden. Dazu wird in der Funktion `BspNode::gather()` eine Liste mit `Triangle`-Objekten gefüllt. Die so erzeugte Liste wird beim Rendern genutzt, um die Reihenfolge der zu rendernden Dreiecke zu bestimmen. Nutze den Algorithmus aus der Vorlesung. Dieser traversiert den Baum, je nachdem auf welcher Seite die Kameraposition ist, zunächst auf der gegenüberliegenden Seite. Danach das Dreieck aus dem Knoten, und letzlich der Teilbaum auf der Seite der Kamera. Somit wird garantiert, dass wir sortiert von Hinten nach Vorne rendern.

Achte darauf, die Rekursion nur weiterzuführen, wenn der Teilbaum existiert (der Wert `nullptr` in einem der Pointer `m_front` oder `m_back` markiert, dass dieser Teilbaum leer ist und somit nicht existiert). Beim Aufbau des BSP-Baumes (s. Aufgabe 1b) muss entsprechend `nullptr` in solche Pointer geschrieben werden.