Prof. G. Zachmann
D. Mohr

TU Clausthal
Institut für Informatik
30. November 2011

Wintersemester 2011/12

Übungen zu Computergraphik I - Blatt 6

Abgabe am 14. 12. 2011

Aufgabe 1 (BSP-Trees, *6+4 Punkte*)

In dieser Aufgabe soll eine Szene mit (semi-)transparenten Objekten korrekt dargestellt werden. Hierfür müssen die Objekte bzgl. Augkoordinaten von hinten nach vorne gerendert werden. (Das wurde in der Vorlesung noch nicht bewiesen, ist aber so.)

Laden Sie das Projekt `bsp_framework` von der Vorlesungs-Homepage herunter. Über den Menüeintrag *Files→Open Model* können Sie Szenen im Wavefront Obj-Format laden. Einige Beispielszenen stehen im Unterverzeichnis `./models` zur Verfügung. Der Obj-Loader kann Szenen aus Vier- und Dreiecken laden, und konvertiert diese in Dreiecke. Die Dreiecke sind in der Instanzvariable `Mesh::m_triangles` gespeichert; jedes Dreieck enthält 3 Indices auf seine Eckpunkte. Die dazugehörigen Vertexkoordinaten stehen in der Instanzvariablen `Mesh::m_vertices`.

a) In der Klasse `Mesh` wird eine Szenen in oben genannter Form gespeichert. Bauen Sie aus der Menge von Dreiecken einen BSP-Tree auf. Um zu enscheiden, auf welcher Seite der Splitplane ein Dreieck ist, testen Sie der Einfachheit halber einen beliebigen der drei Eckpunkte des Dreiecks, d.h. Splitting ist in dieser Teilaufgabe nicht gefordert.

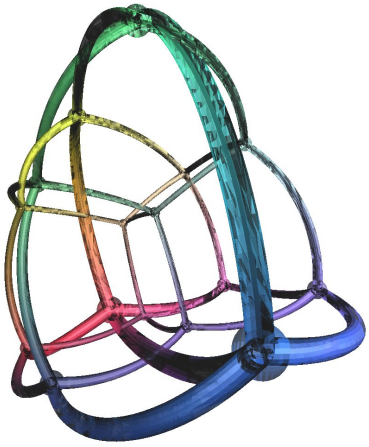
   Implementieren Sie hierzu die bereits vorhandene, aber noch leere, Routine `Mesh::generateBSP()`. Verwenden Sie die vordefinierte Klasse `Mesh::BspNode`. Die Wurzel des BSP-Baumes soll der Variable `Mesh::m_bspRoot_splitsDisabled` zugewiesen werden. Die Traversierung inklusive Rendering des BSP-Baumes sind bereits implementiert. Sie können also sofort das Ergebnis Ihrer Implementierung betrachten. Vergessen Sie nicht, den entsprechenden Radiobutton oben zu aktivieren.

b) Erweitern Sie ihre Methode dahingehend, dass Dreiecke, die die Splitplane schneiden, korrekt unterteilt und die resultierenden Teildreiecke zu dem passenden Kindknoten hinzugefügt werden. Weisen Sie die Wurzel des resultierenden BSP-Baumes der Variablen `Mesh::m_bspRoot_splitsEnabled` zu. Um zu entscheiden, ob ein Vertex in der Ebene ist oder nicht, müssen Sie Floatingpointwerte numerisch robust vergleichen, d.h., nicht `if(a==b)` sondern `if(fabs(a-b)<EPS)`. Verwenden Sie hierzu die vordefinierte Instanzvariable `Mesh::m_EPS`.

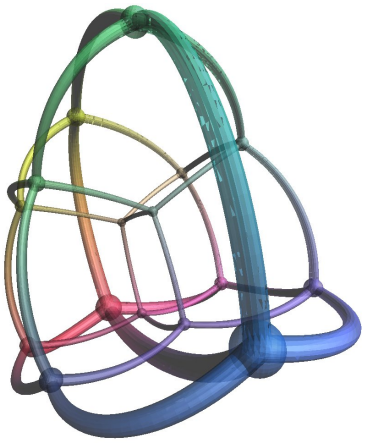
Abbildung 1 zeigt die verschiedenen Ansätze anhand einer Beispielszene.

Tip: Die folgenden Funktionen der STL-Klassen `std::list` und `std::vector` können für Ihre Implementierung hilfreich sein:

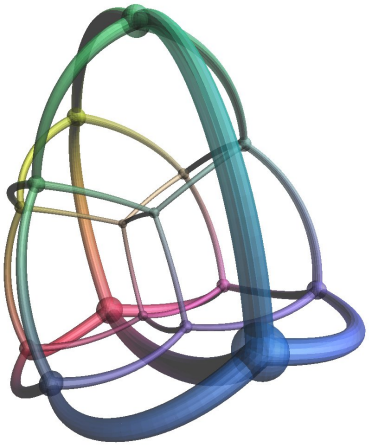
- Um auf Elemente sequentiell zuzugreifen, gibt es die sogenannten “Iteratoren” (eine Art intelligenter Pointer), die wie folgt verwendet werden können:



Naives Rendering, d.h., die Polygone werden in einer zufälligen Reihenfolge gerendert (nämlich in der Reihenfolge, in der sie im Array stehen)



Rendering von hinten nach vorne mit *BSP ohne Splitting*



Rendering von hinten nach vorne mit *BSP mit Splitting*

Abbildung 1: Rendering eines transparenten Objektes

```
for( std::list<int>::iterator it = liste1.begin(); it != liste.end(); ++it )
{
    int akuelle_elem = *cit;
}
```

- `begin()` gibt einen Iterator auf das erste Element einer Liste bzw. eines Vectors zurück,
- `end()` das letzte Element, und `size()` die Anzahl der Elementen.
- Ein neues Element kann mit `push_back()` hinzugefügt werden, z.B.

```
std::vector<Triangle> dreiecksliste;
dreiecksliste.push_back( Triangle( index0, index1, index2 ) );
```