Prof. G. Zachmann
Max Kaluschke
Toni Tan

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
19. November 2018

Wintersemester 2018/19

Übungen zu Computergraphik - Blatt 5

Abgabe am 25.11.2018

Aufgabe 1 (Korrektes Rendern transparenter Objekte mittels BSP-Tree, *4+6 Punkte*)

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

Laden Sie das Projekt `BSPFramework` 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`.

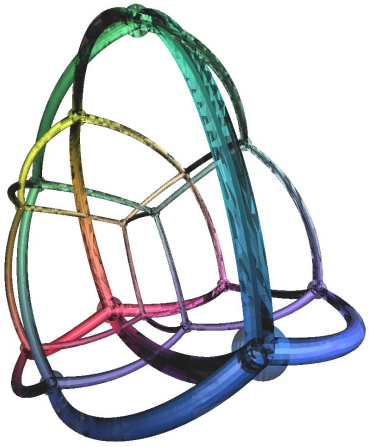
In der Klasse `Mesh` wird eine Szenen in oben genannter Form gespeichert und die BSP-Trees, bisher ohne Splitten, generiert.

a) Implementieren Sie die bereits vorhandene, aber noch leere, Routine `Mesh::drawBSPRek`. Ein `BSPNode` enthält die Normale (`m_planeNormal`), Verweise auf die Teilbäume `m_front` und `n_back` und das Dreieck selbst (`m_triangle`). Die Indices der Punkte des Dreiecks befinden sich im Array `m_triangle.v`, die man benötigt, um die eigentlichen Position im Raum vom Array `m_vertices` abzufragen. Der Parameter data muss an die Methoden `Mesh::drawBSPRek` und `Mesh::drawTriangle` lediglich weitergereicht werden.

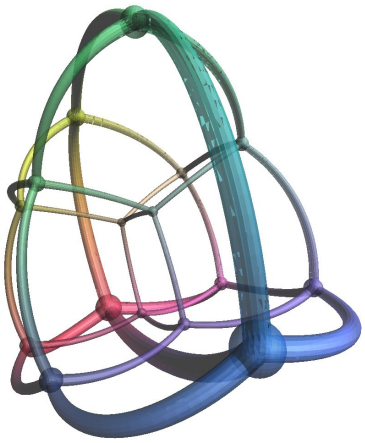
b) Implementieren Sie die bereits vorhandene, aber noch leere, Routine `Mesh::splitTriangle`. Vergessen Sie dabei nicht, die Methode auch in `Mesh::generateBSPRek` zu verwenden. Dreiecke, die von der Ebene geschnitten werden, sollen korrekt unterteilt und die resultierenden Teildreiecke zu dem passenden Kindknoten hinzugefügt werden.

   Um zu entscheiden, ob ein Vertex in der Ebene ist oder nicht, müssen Sie Floatingpointwerte numerisch robust vergleichen, d.h., z.B. nicht `if(a==b)` sondern `if(fabs(a-b)<EPS)`. Verwenden Sie hierzu die vordefinierte Instanzvariable `Mesh::m_EPS` und `Mesh::intersectPlaneLine`. Splitten Sie nur dann, wenn es auch Sinn macht. Bei einem minimalen Eindringen in die Ebene sollte eher nicht weiter gesplittet werden.

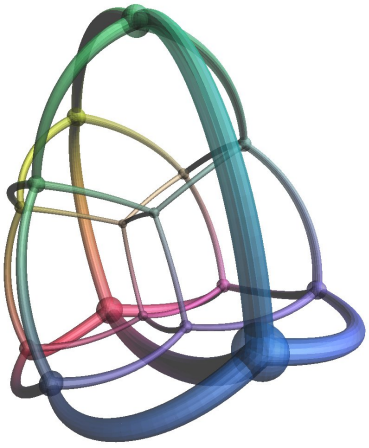
Abbildung 1 zeigt die verschiedenen Ansätze anhand einer Beispielszene.



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

Tipp: 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:

```
std::list<int> liste;
for( std::list<int>::iterator it = liste.begin(); it != liste.end(); ++it )
{
    int aktuelle_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 Elemente.
- Ein neues Element kann mit `push_back()` hinzugefügt werden, z.B.

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