Prof. G. Zachmann
David Mainzer

TU Clausthal
Institut für Informatik
26. Oktober 2011

Wintersemester 2011/12

Übungen zu Virtuelle Realität und Simulation - Blatt 3

Abgabe am 11. 11. 2011

Aufgabe 1 (Progressive Meshes, *5 Punkte*)

Skizzieren Sie — in Form von Pseudo-Code — den Algorithmus, der die Erstellung eines Progressive Meshes leistet. Gehen Sie dabei von dem in der Vorlesung angegeben Kostenmaß für einen Edge-Collapse aus. Sie dürfen weiterhin davon ausgehen, dass alle Nachbarschaftsinformationen innerhalb eines Meshes gegeben / bekannt sind.

Aufgabe 2 (Kollisionsdetektion, 3D-Gitter, *3 Punkte*)

Gegeben sei ein 3D Gitter mit $k \times l \times m$ Zellen. Dieses werde in einem 3D Array `G[k][l][m]` gespeichert, d.h., die Indizes laufen von $0, \ldots, k-1$, etc. Gegeben sei weiterhin eine achsenparallele Bounding-Box (AABB) $(x_1, x_2, y_1, y_2, z_1, z_2)$. Das Universum geht von 10 bis 100 für die X-Achse, von 20 bis 200 für die Y-Achse und von 10 bis 100 für die Z-Achse reichen.

Geben Sie eine Rechenvorschrift an, mit der man *schnell* alle Array-Indizes bestimmen kann, die zu Zellen gehören, die von der AABB "belegt" werden.

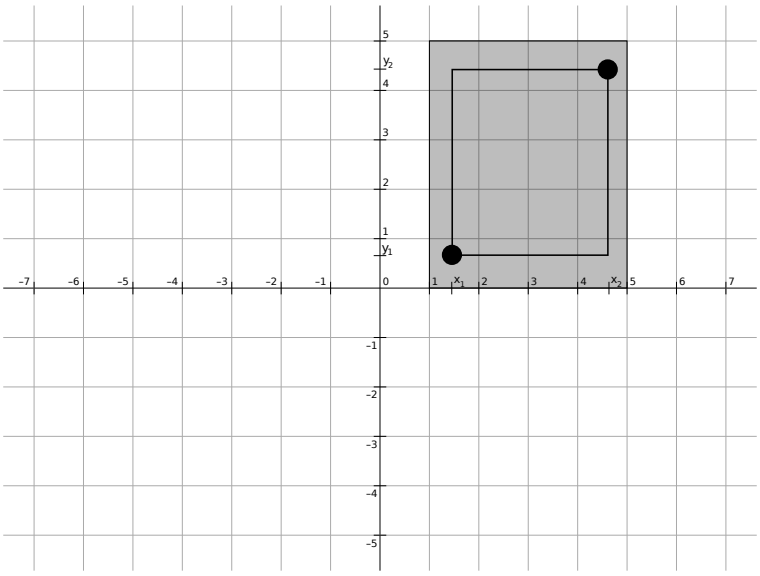

Abbildung 1: AABB innerhalb eines 2D-Gitters. Das Universum würde von -7 bis 7 für die X-Achse und -5 bis 5 für die Y-Achse reichen.

Aufgabe 3 (Kollisionsdetektion, BBoxes, *3 Punkte*)

Geben Sie einen Algorithmus für einen Überlappungstest zweier achsenparalleler Bounding-Boxes (AABBs) an. (Pseudo-Code genügt.) Versuchen Sie, Ihren Algorithmus so schnell wie möglich zu machen.

Aufgabe 4 (Kollisionserkennung, *10 Punkte*)

Für die meisten Interaktionen mit Gegenständen einer Umgebung ist eine Kollisionserkennung erforderlich. Für die Kollision zwischen dem Betrachter (Viewpoint) und Objekten existiert in *VRML* bereits ein Kollisionsknoten. Die Kollision zwischen Objekt und Objekt muss hingegen "von Hand" berechnet werden. Die Umgebung (*Rollercoaster*) wurde mit einem Tuch, welches auf die Fahrbahn der Achterbahn ragt, versehen. Nun besteht ihre Aufgabe darin, eine Kollisionsüberprüfung zwischen dem *carriage* (Fahrgestell) und dem Tuch durchzuführen.

Hierbei ist darauf zu achten, dass sich das Fuhrwerk bewegt. Es muss somit auf die Koordinaten des Dreiecksnetzes noch die entsprechende Transformation angewandt werden. Der SFNode carriageTrans enthält die Transformation des Fuhrwerks.

Da es sich bei dem Fuhrwerk um eine etwas komplizierte Geometrie handelt, ist diese in 2 Gruppen unterteilt. Die SFNodes carriage-maingeom und carriage-geom enthalten die Punkte des Dreiecknetzes.

Die Kollisionserkennung kann mit der Taste "<" aktiviert beziehungsweise deaktiviert werden.

```
DEF coll Script {
  # collision objects carriage vs. rag
  # geometries
  field SFNode carriageGeo1 USE carriage-maingeom
  field SFNode carriageGeo2 USE carriage-geom2
  field SFNode ragGeo USE rag
  # transformation
  field SFNode carriageTrans USE carriageT
  field SFNode ragTrans USE ragT

  # if TRUE run collision detection check
  field SFBool collCheck FALSE

  # run in a loop
  eventIn SFTime cycle
  # handle key events
  eventIn SFString actionPress
  field SFNode time USE time
  field SFNode sound USE ac

  url ["Collision.class"]
}
```

Quellcode 1: Informationen zur Kollisionserkennung

Der Quellcode 1 zeigt das Ausgangsskript, welches die Kollisionserkennung lädt. Sie müssen die Kollisionserkennung in der entsprechenden Java Datei Collision.java noch vervollständigen. Ein Test Strahl-Dreieck (Math.testRayTriangle(...)) ist bereits implementiert und in der Funktion check() ist ein kleines Beispiel zu sehen. Die SFNodes carriageGeo1, carriageGeo2 und ragGeo zeigen auf die Geometrie der auf Kollision zu überprüfenden 2 Objekte (Fuhrwerk und Tuch). Es ist zu beachten, dass carriageGeo1 und carriageGeo2 zu einem Objekt gehören und somit zwischen diesen beiden keine Kollisionserkennung durchgeführt werden muss. Findet eine Kollision statt, ist wie im Beispiel ein Ton abzuspielen oder eine Ausgabe auf der Konsole auszuführen.