Echtzeit Culling Algorithmen
Eine Übersicht

Peter Grundmann

pegr@hrz.tu−chemnitz.de

Chemnitz, den 24. April 2002

Gliederung

- Einleitung
- Die Render Pipeline
- Backface Culling
- View Frustum Culling
- Quadtrees
- Portalisierung
- BSP Bäume

Einleitung

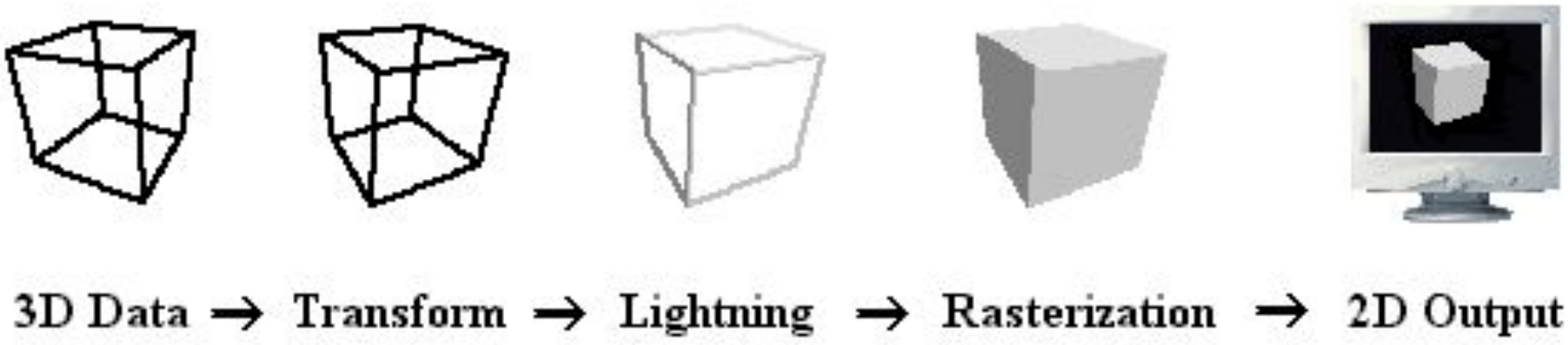
Warum Culling?

- Hardware für komplexe Szenen nicht ausreichend
- Beschleunigung der Darstellung durch Culling

  => Komplexere Szenen möglich

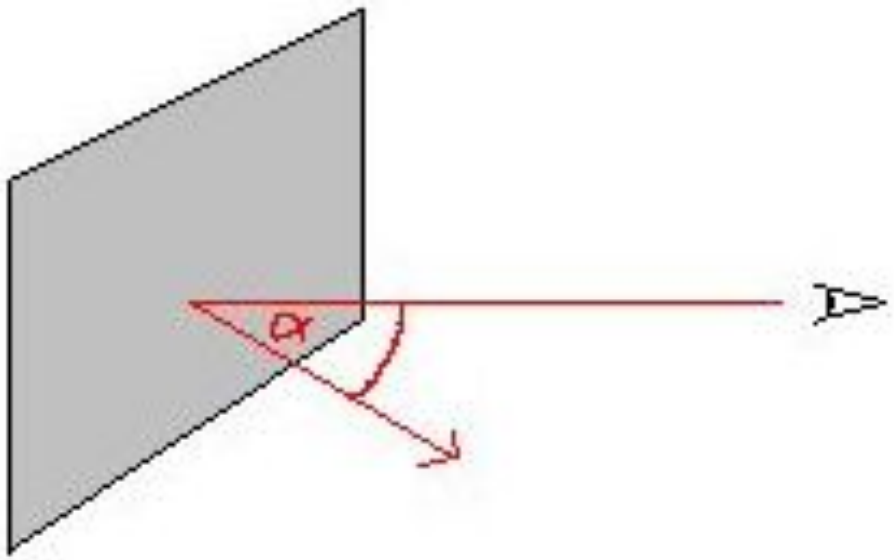
- Anspruch an Grafik steigt ständig

Die Render Pipeline



- Berechnen => Beleuchten => Rendern
- Nicht sichtbare Vertices sollten aussortiert werden, bevor sie diese Operationen durchlaufen

Backface Culling



- Winkel zwischen Sichtstrahl und Normalenvektor berechnen
- Ist dieser größer als 90° => nicht sichtbar

Backface Culling

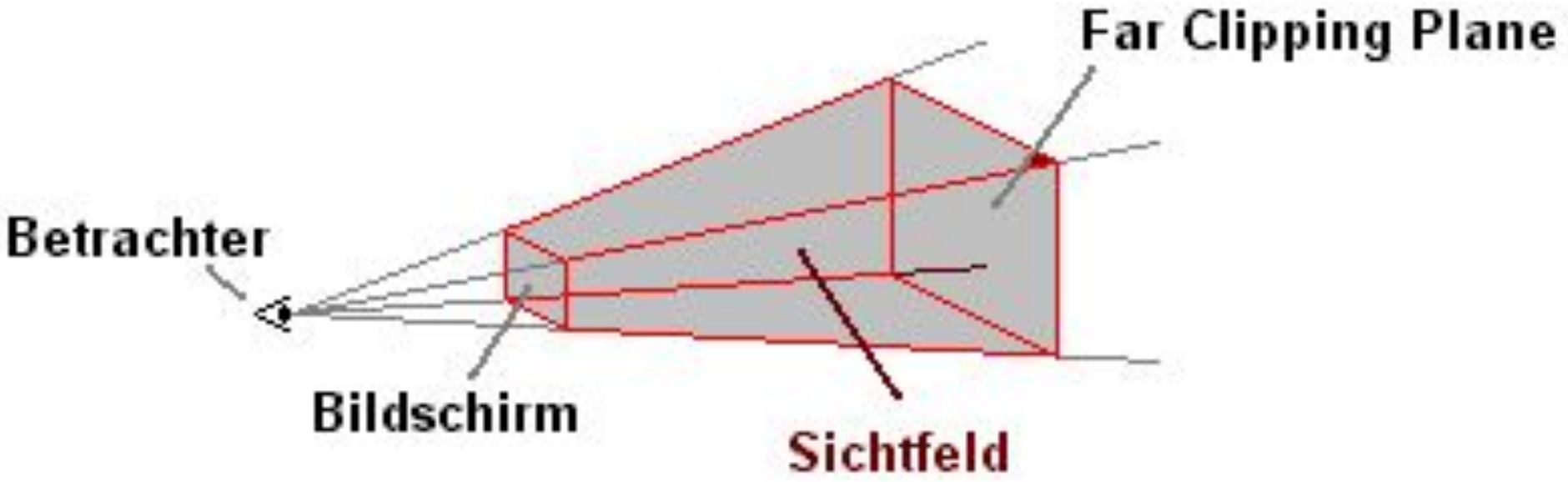
Vorteil:

- Effektivität ~ 50%
- schnelle Berechnung

Nachteil:

- muss für jedes Polygon einzeln duchgeführt werden

View Frustum Culling



- View Frustum = Sichtfeld des Betrachters
- Begrenzt durch 6 Ebenen

View Frustum Culling

- Test aller Vertices eines Polygons auf Sichtbarkeit
- 3 mögliche Fälle

a) alle Punkte hinter einer Ebene
   => nicht sichtbar

b) alle Punkte vor jeder Ebene
   => komplett sichtbar

c) Nur einige Punkte vor jeder Ebene
   => teilweise sichtbar

View Frustum Culling

- lokale Ansammlung an Polygone zu Objekte zusammenfassen
- Optimierung durch Bounding Boxen
- Dessen Seitenflächen/Eckpunkte testen
- Effektivität ~70%

Quadtrees

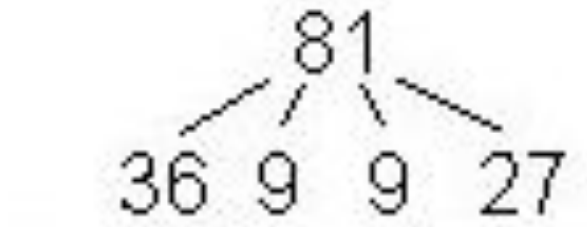
- Erstellen einer Baumstruktur zur Ladezeit
- Einordnen aller Polygone in diesen Baum
- Ziel: Effektive "Eliminierung" durch wenige Tests
- Nur für Outdoor Szenen
- Effizienz je nach Betrachterposition

Quadtrees

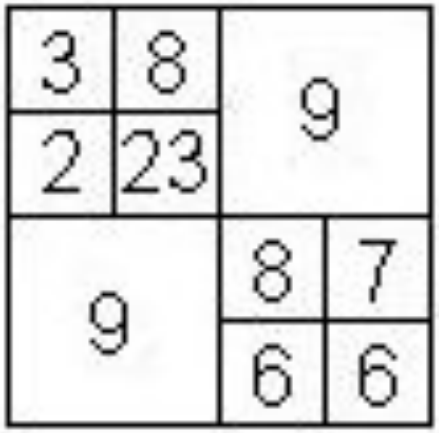
81

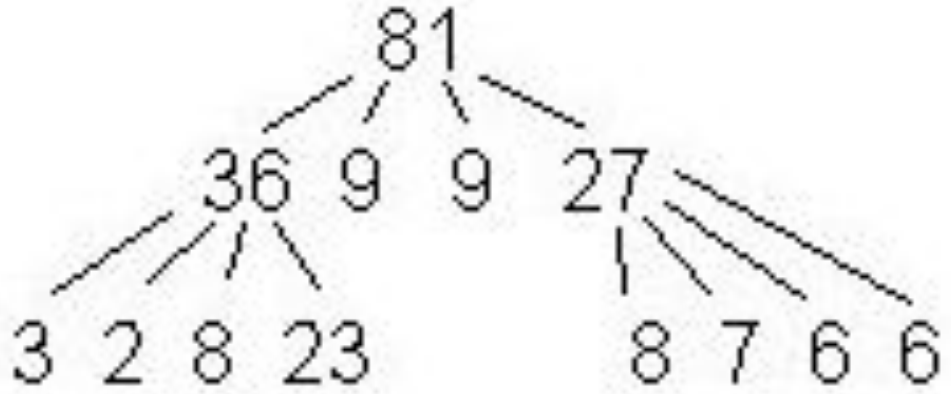
Quadtrees

36	9
9	27

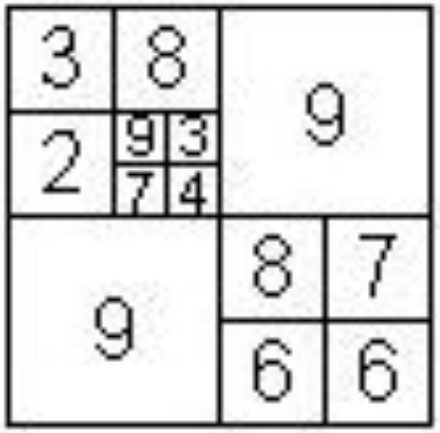


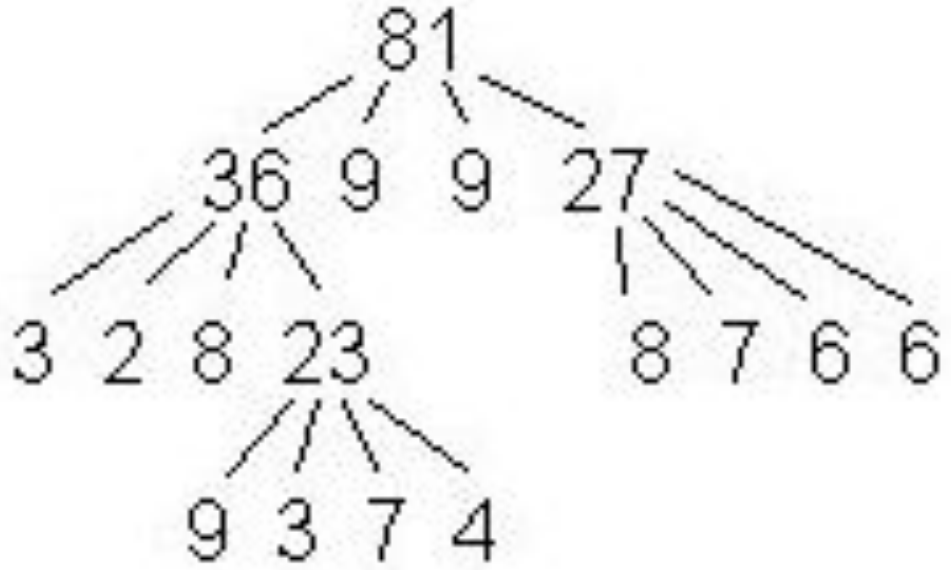
Quadtrees



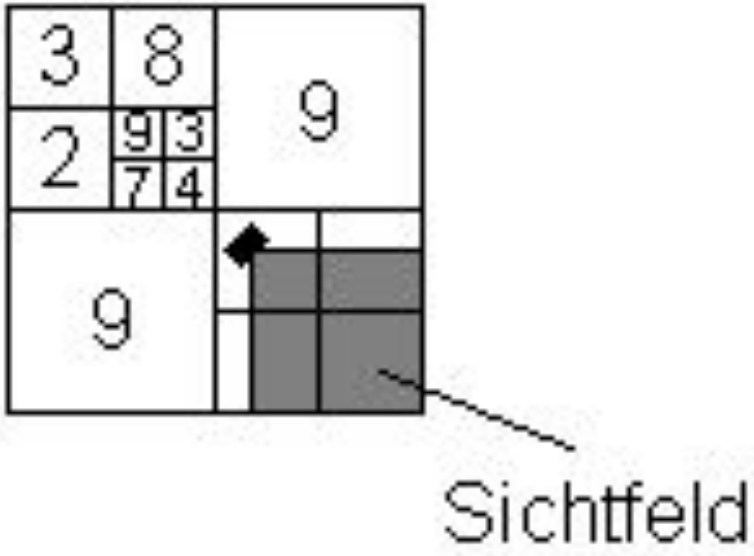


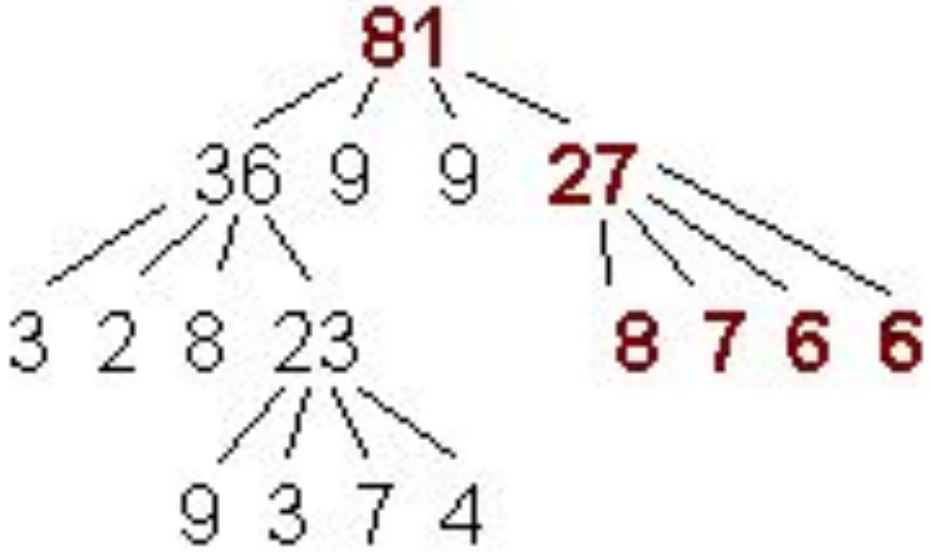
Quadtrees





Quadtrees





- In diesem Beispiel schon beim 1. Test eine Ersparniss von ~67%

Octrees

Octrees:

- Erweiterte Form der Quadtrees
- Aufteilung in Würfel
- Statt 4 hier 8 Baumäste

Quadtrees/Octrees

Vorteil:

- teilweise sehr schnelle Eliminierng hoher Polygonzahlen

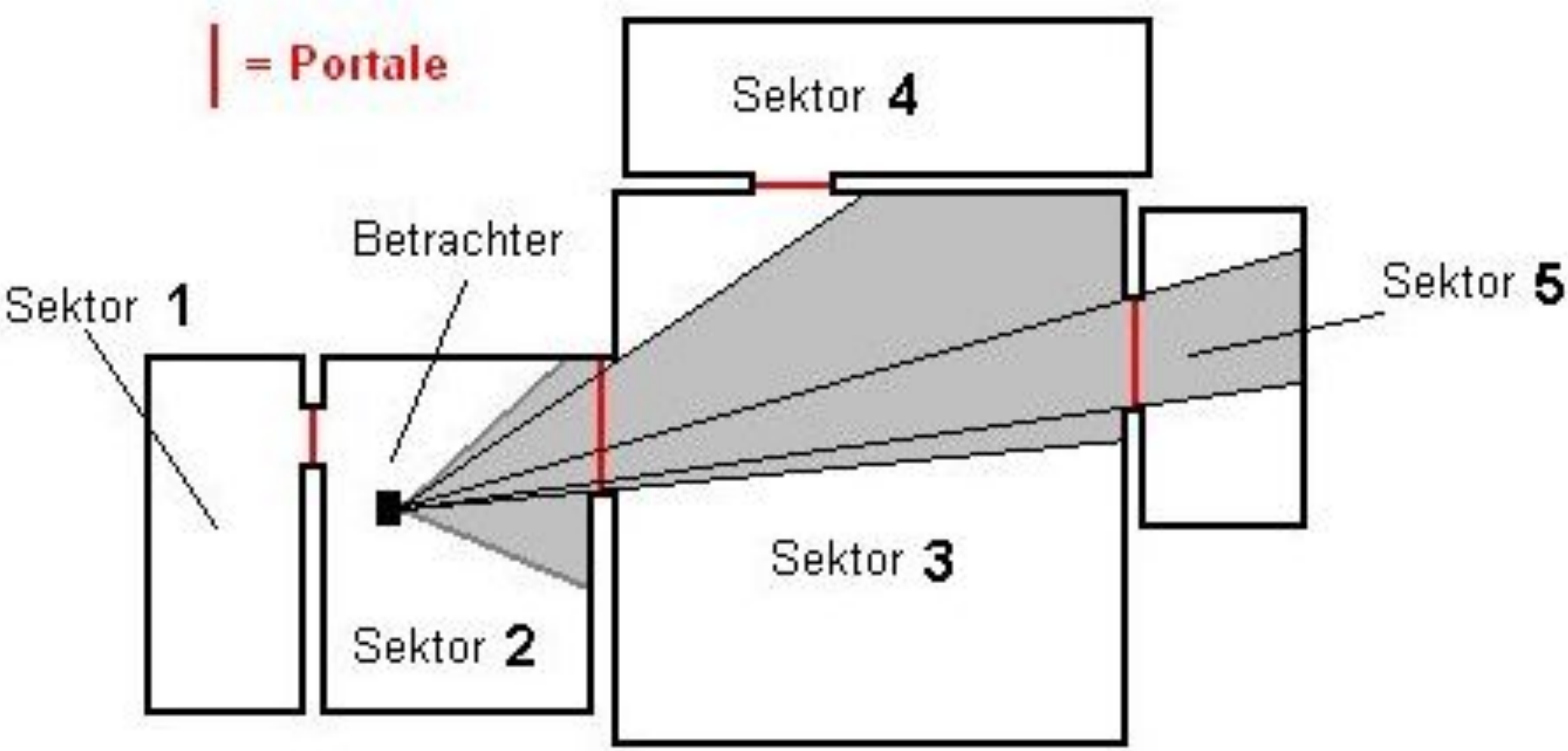
Nachteil:

- hinzufügen von Polygonen
- nur für statische Szenen

Portalisierung

- Weltaufteilung in Sektoren
- Verbinden dieser Sektoren durch Portale
- Portale besitzen Quell und Zielsektor
- Rendern des Quellsektors und der Zielsektoren aller sichtbaren Portale
- Rekursiver Algorithmus

Portalisierung



- Anpassen des Sichtfeldes an die Portale
- Rendern beim Erstellen oder Abarbeiten des Baumes

Portalisierung

Vorteil:

- Dynamische Welten möglich
- Erlaubt Mix aus Indoor und Outdoor
- Einfache Integration von Spiegelungen
- Keine unnötigen Tests auf nicht sichtbare Sektoren

Nachteil:

- Sektoraufteilung "von Hand"

BSP – Bäume

- Binary Space Partitioning
- Erstellen von konvexen Partitionen
- Ordnen der Polygone nach Tiefe in binären Baum
- BSP Compiler: Algorithmus zum Erstellen des Baumes
- Prinzip: Rekursive Teilung der Szene in 2 Hälften

BSP – Bäume

Vorteil:

- Rendern ohne Z−Buffer (hohe Geschwindigkeit)
- korrekte Transparenzdarstellung

Nachteil:

- Erstellung zusätzlicher Polygone
- statische Welten
- nur Indoor
- Baum wächst schnell auf enorme Größe an

Schlussbemerkung

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Ausarbeitung und Folien unter: **http//www.pscg−culling.de.vu**

Kontakt: **pegr@hrz.tu−chemnitz.de**