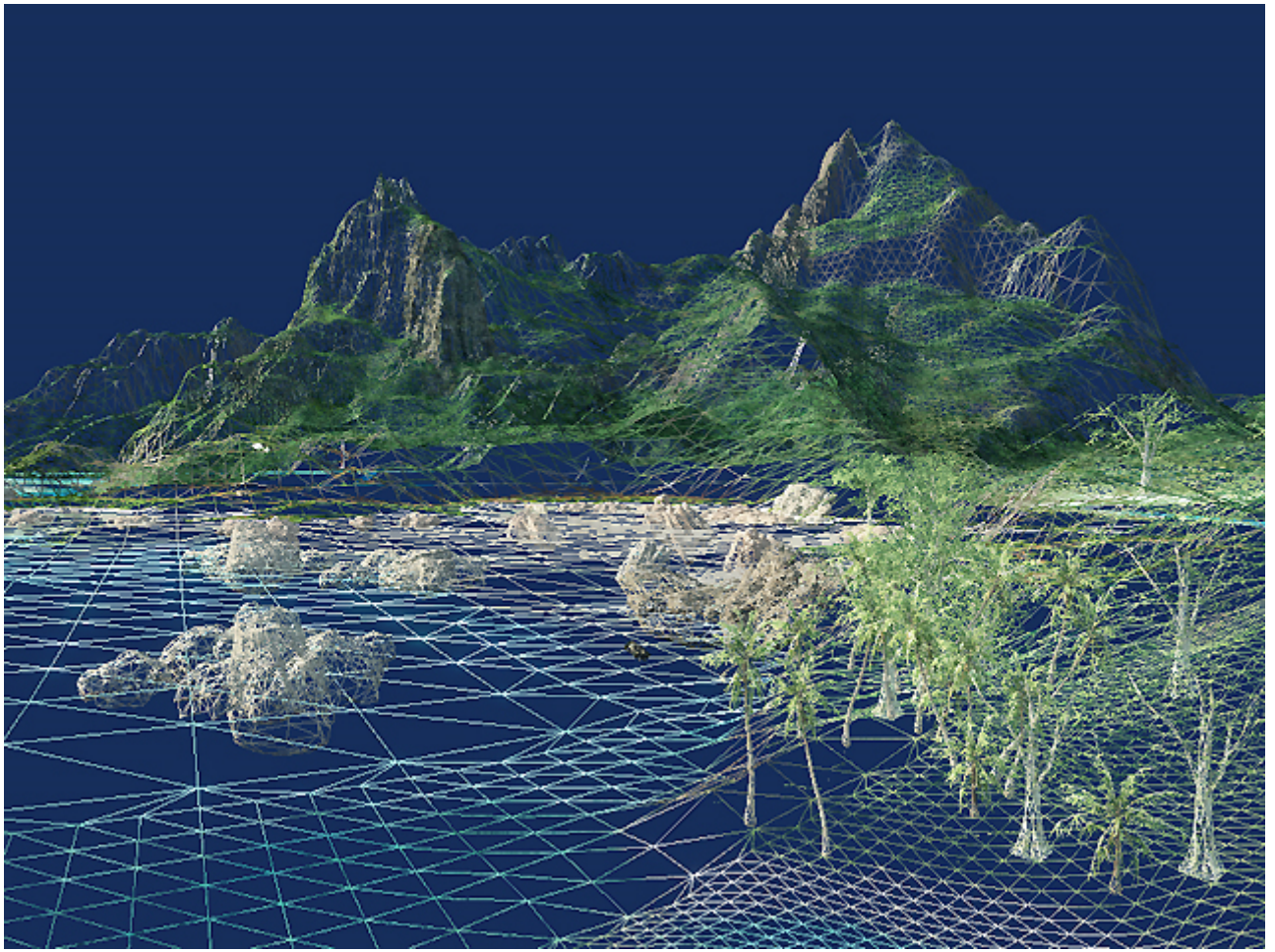
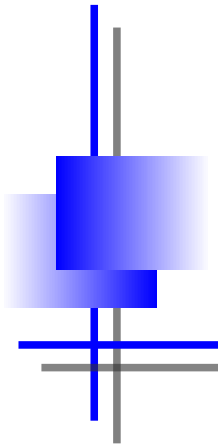
Optimierungsalgorithmen

Effektives Rendern in der Computergraphik



Benjamin Schneider



Agenda

1. Einleitung
2. Grundlagen
3. Algorithmen im Detail
4. Weitere Techniken
5. Verbesserungen
6. Fazit

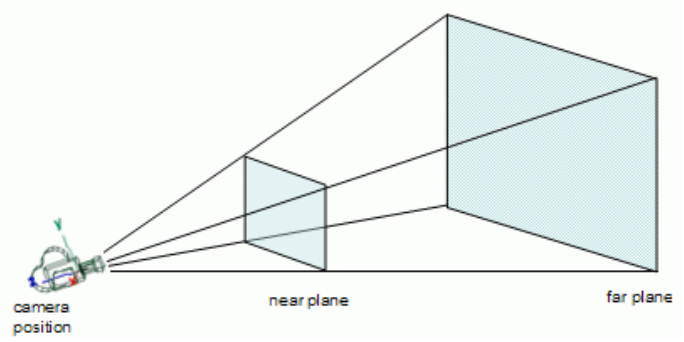
Einleitung

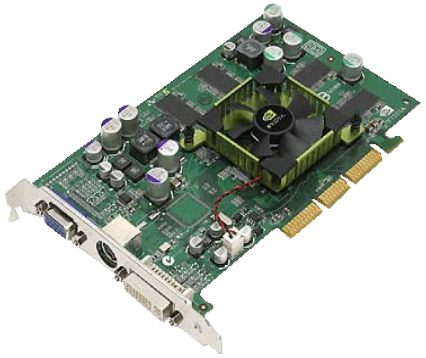
Motivation:

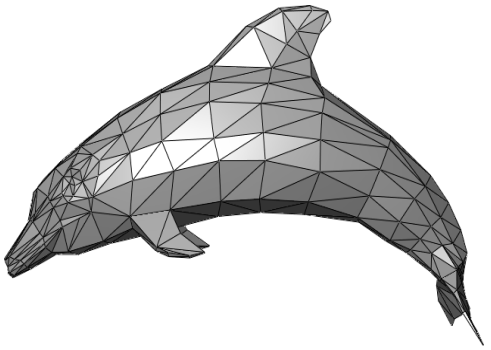
- Renderaufwand minimieren
- Unsichtbare Flächen aussortieren
- Leistung für andere Berechnungen zur Verfügung(z.B KI, Physik, ...)
- Aufteilen der Rechenarbeit auf CPU und GPU

Grundlagen

- View Frustrum (Sichtkegel)
- Abarbeitung in der Grafikkarte (Grafik-Pipeline)
- Geometrie basiert auf Dreiecken

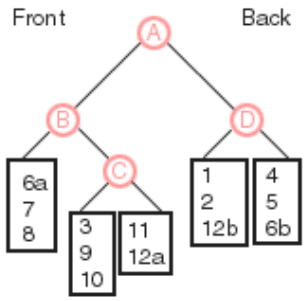


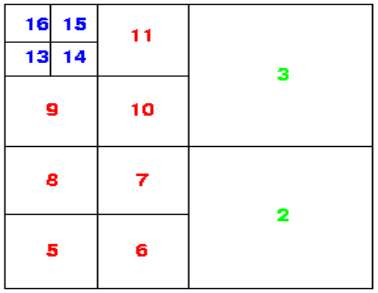


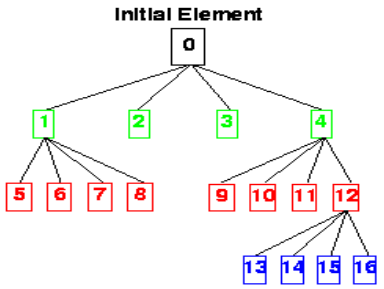


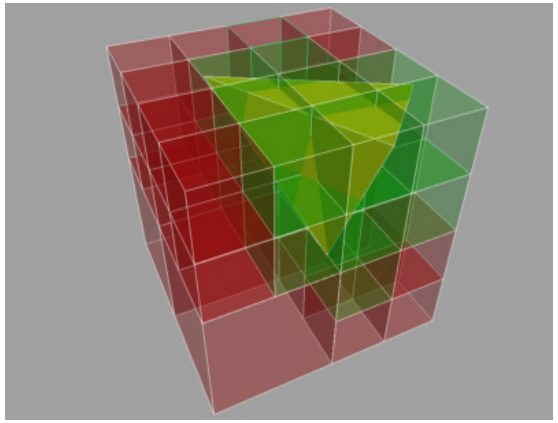
Algorithmen im Detail

- Binary Space Partitioning (BSP)
- Quadtree
- Octree









BSP-Trees

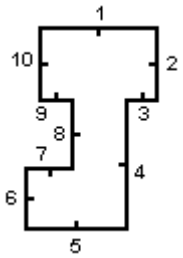
- Schon in Quake und Doom verwendet
- Einsatz im Indoor-Rendering
- Kombiniert mit PVS sehr effektiv für Sichtbarkeitstests
- Ziel: Aufteilung der Geometrie in konvexe Stücke

BSP-Trees

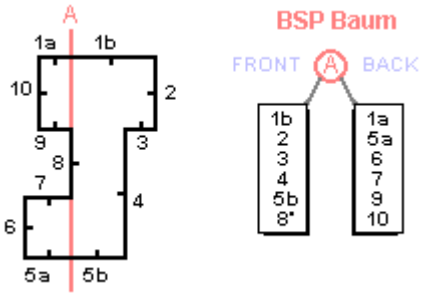
- Ablauf:
  - Raum wird mit Teilungsebene in zwei möglichst gleichgroße Hälften geteilt, ohne zu viele Polygone zu „zerschneiden“
  - Polygone vor der Teilungsebene in Frontliste, Polygone dahinter in Backliste
  - Wiederholung bis keine Teilung mehr möglich – konvexe Räume sind entstanden

BSP-Trees

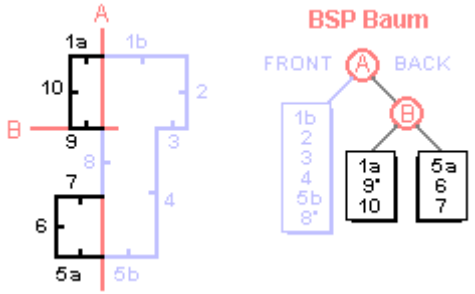
- Beispielablauf:



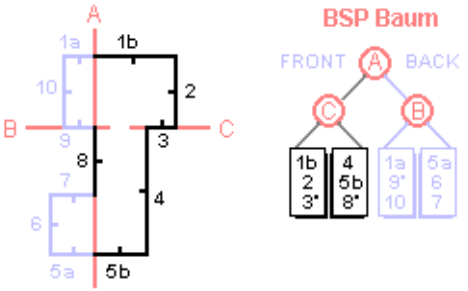
Ausgangsraum



1. Schritt



2.Schritt



3.Schritt

BSP-Trees

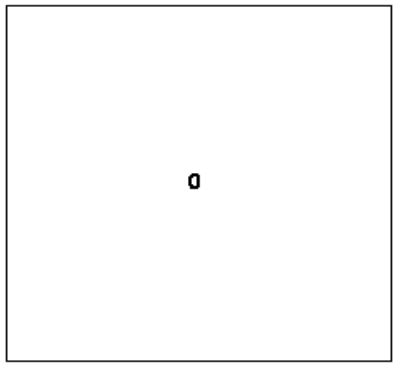
- PVS fügt jedem Leaf eine Sichtbarkeitsliste hinzu
- Durch Bounding-Box Tests werden aus diesen potentiell sichtbaren Teilräumen, nur die im Sichtkegel zum Rendern ausgewählt

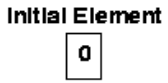
Quadtrees

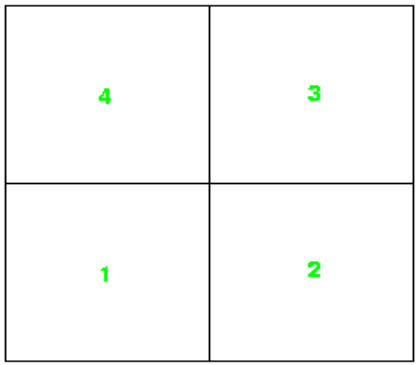
- Einsatz im Outdoor-Bereich
- Hauptsächlich genutzt für das Terrain-Rendering
- Ziel: frühes Aussortieren großer Geometriestücke
- Ablauf:
  - Gesamtes Terrain wird in vier gleichgroße Quadrate zerteilt
  - Unterteilung bis Abbruchkriterium(z.B. gewisse Tiefe des Baums oder Seitenlänge der Quadrate)

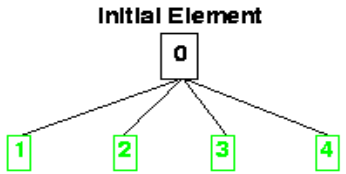
Quadtrees

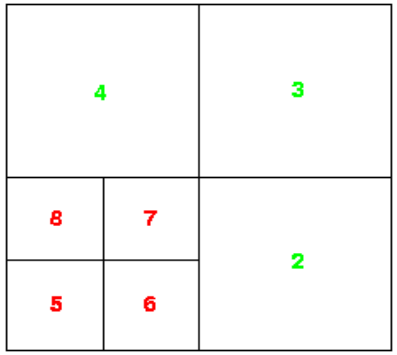
- Beispielablauf:

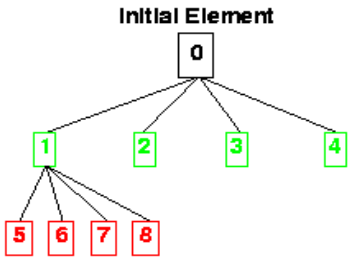


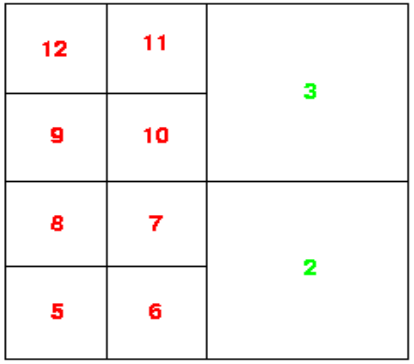


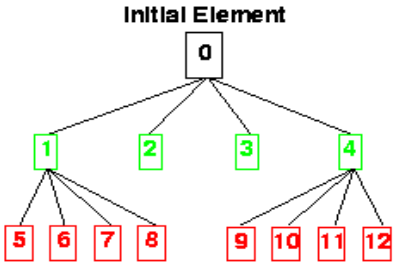






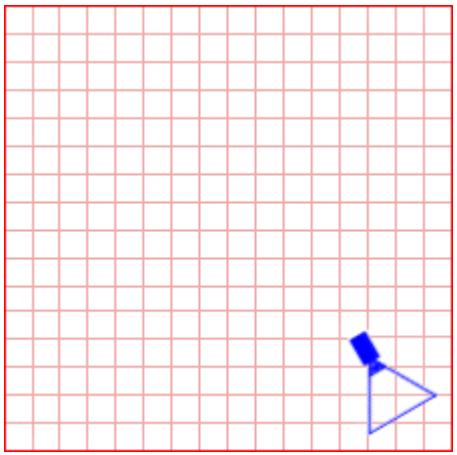


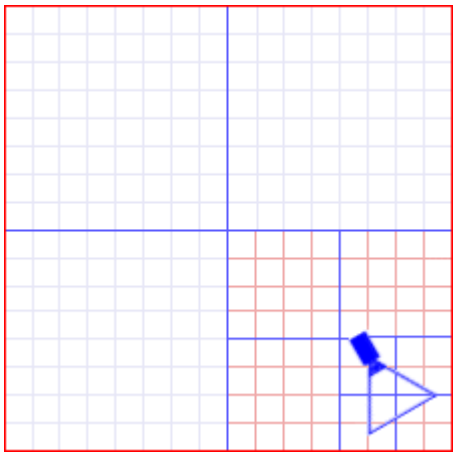




Quadtrees

- View Frustum Culling am Quadtree:

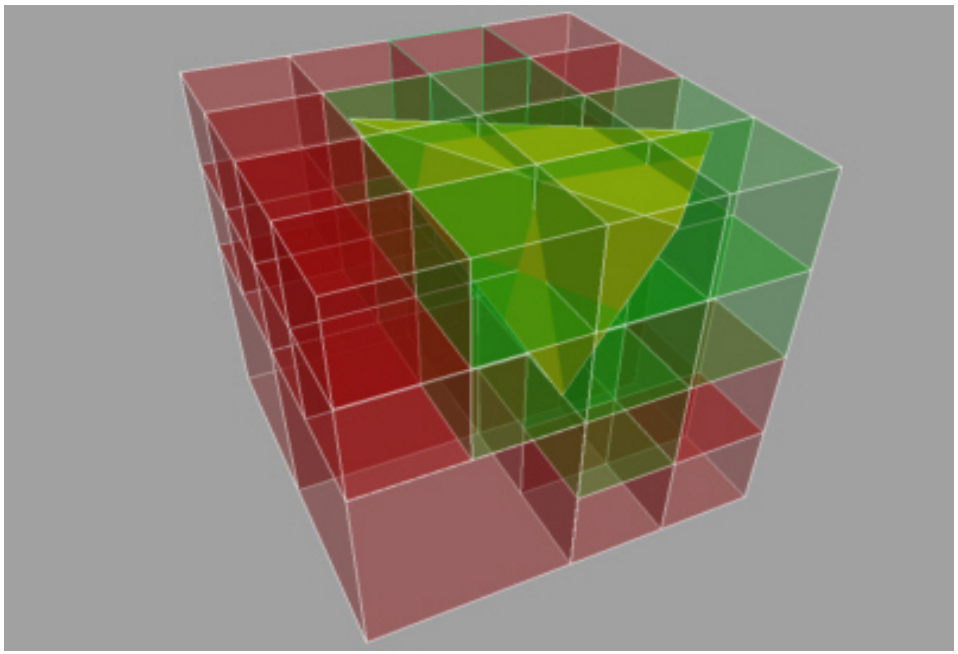




- Große Teile können schnell aussortiert werden
- Kollisionsabfrage mit Terrainoberfläche wird beschleunigt

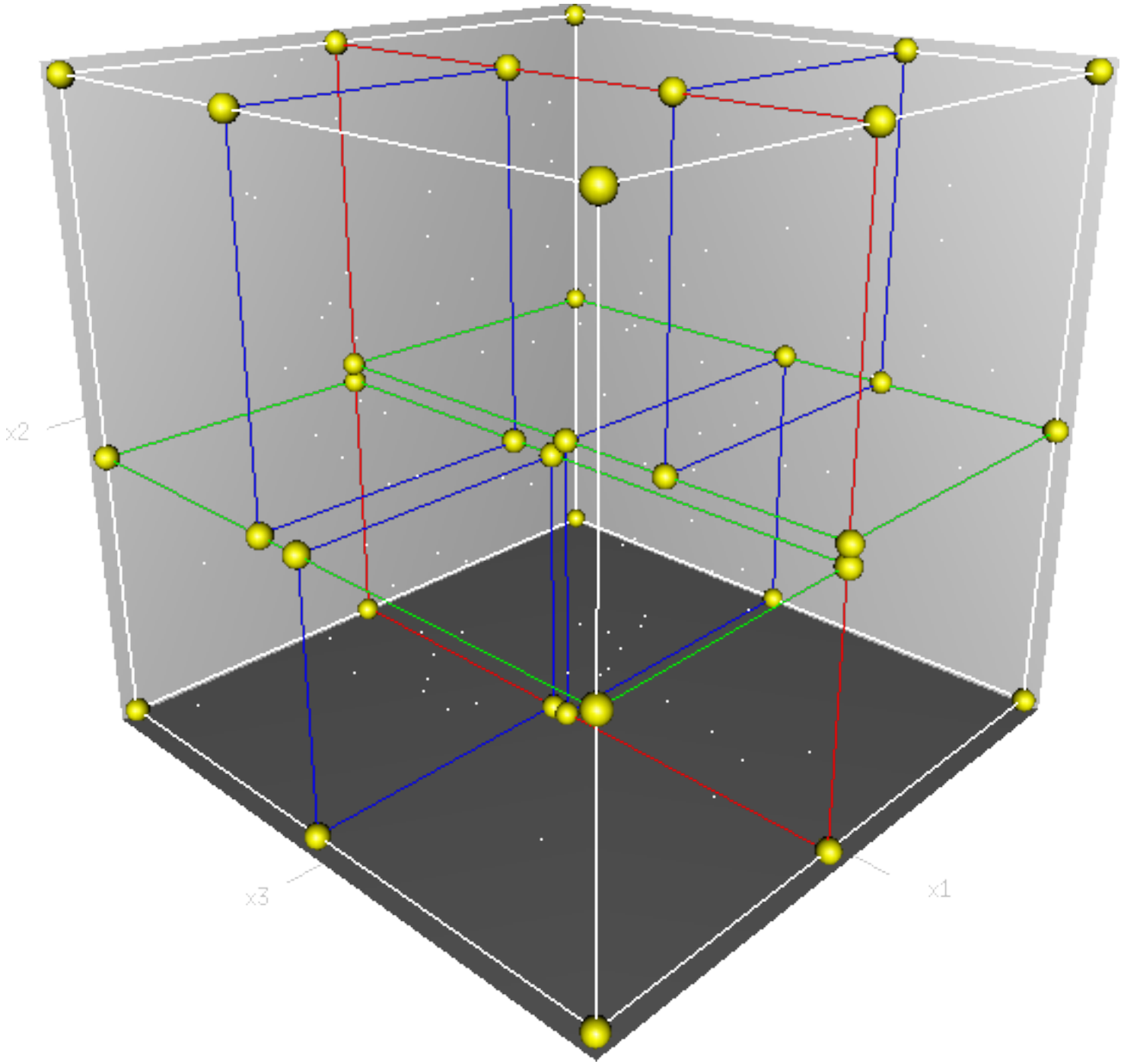
Octrees

- 3-dimensionaler Quadtree
- Unterteilung in Würfel statt Quadtrate
- Sehr universell einsetzbar (sog. Nodoor-Rendering)



Weitere Techniken

- ROAM (Realtime Optimally Adapting Meshes)
  - Alte Technik, nicht mehr gebräuchlich
  - Dreiecksstruktur wird in jedem Frame neu erzeugt
- Geometrical Clipmapping
  - Neuere Technik, GPU lastig
  - Ähnlich dem Texture-Mipmapping
- kd-Tree


x2
x3
x1

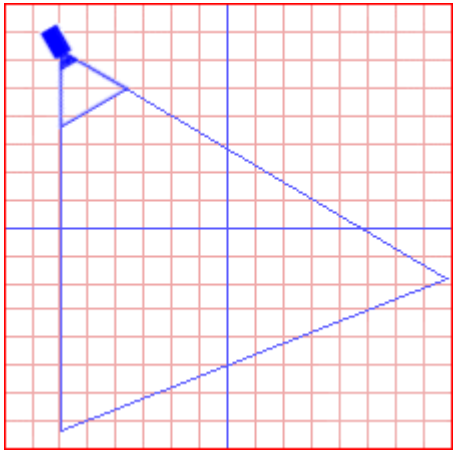
Verbesserungen

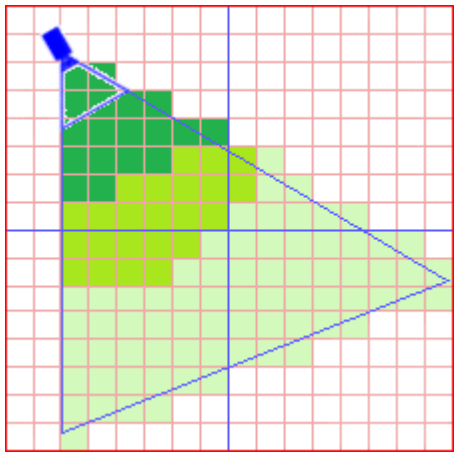
Kombination mit anderen Techniken möglich:

- Occlusion Culling
  - Potentially Visible Set (PVS), Portals
- Contribution Culling (meist mit Fog)
- Backface Culling
- Early Z Culling (unterstützt durch front-to-back Rendern)
- Level of Detail (LOD)

GeoMipMapping

- Leafs werden in mehreren Detailgraden gespeichert
- Abhängig von der Kameradistanz wird Detailgrad ausgewählt





Fazit

- Welcher Algorithmus ist wann angebracht?
- Aufwand vs. Nutzen
- Optimierungsalgorithmen sollten unsichtbar sein

Software is a gas; it expands to fill its container.

- Nathan's First Law

Quellen

[1] http://www.vterrain.org

[2] http://www.globalnerdy.com/2007/07/18/laws-of-software-development/

[3] http://old.zfx.info

[4] Zerbst, Stefan: 3D Spieleprogrammierung mit DirectX in C/C++ - Band II

[5] http://www.gamedev.net

[6] Microsoft DirectX SDK (August 2007)

[7] en.wikipedia.org/kd-tree/

[8]www.keepcoding-development.ch.vu, Reto da Forno