Games Engines

Realtime Terrain Rendering

RTR – Gliederung

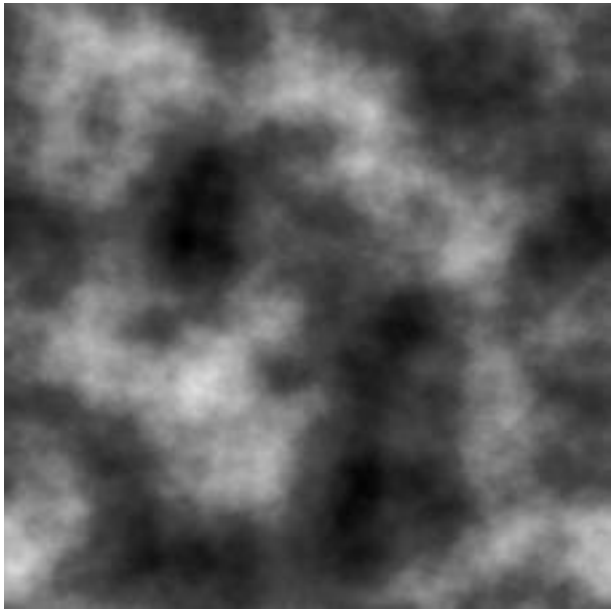
- Probleme & Anforderungen
- Grundlagen
  - Heightmaps und Paging
- Visibility
  - View Frustrum Culling
  - Occlusion Culling/ Occlusion Map
  - Fogging
- Level of Detail
  - Realtime Adaptive Optimizing Meshes (ROAM)
  - Quadtree Terrain
  - Brute Force / der 2. Frühling

RTR – Anforderungen/Probleme

- Weite begehbare Welt
- Keine Unterbrechungen durch Ladezeiten
- Detaillierte Darstellung
- Gute Performance
- Lebendige Ökosysteme

RTR – Heightmaps

- Meist 8-bit Graustufen Bilder, wobei die Pixelkoordinaten zusammen mit dem Farbwert die x,z und y Koordinaten eines Vertex bilden.
- Schon bei kleiner Dimension des Terrains entstehen große Menge an Punkten (1024x1024 heightmap > 1Mio Vertices)
- 1 Vertex ~= 16 byte, bei 1 Mio vertices sind das 16 MByte Grafikkartenspeicher

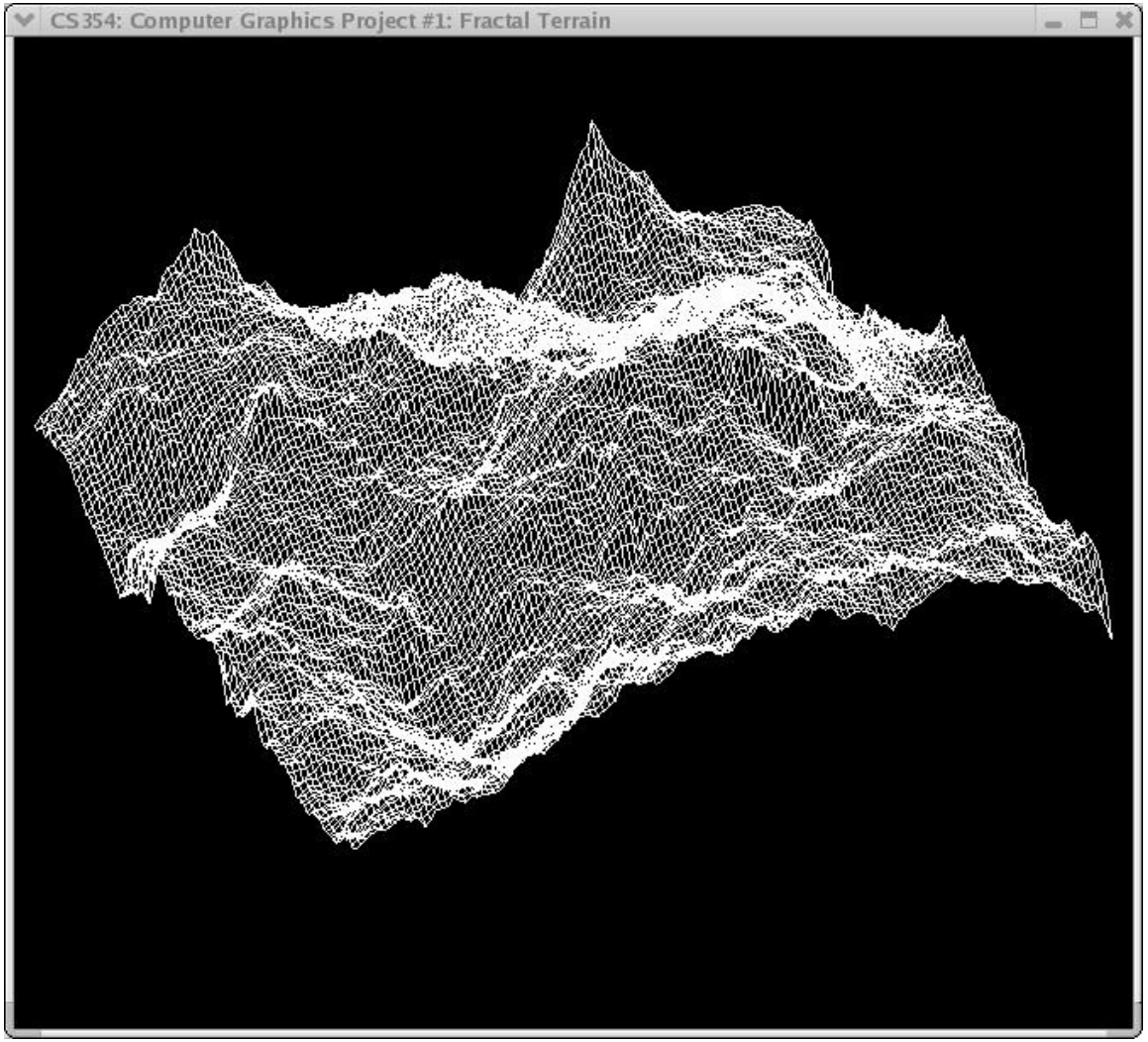


Graustufen Heightmap (256x256)

RTR – Paging

- Um keine Unterbrechung durch Ladezeiten zu bekommen wendet man Paging an.
- Gleichmäßige Unterteilung des Terrains in Pages.
- Beispiel: 4 Pages á 512x512 für eine 1024x1024 Heightmap.
- Einzelne Pages werden bei Bedarf in den VRAM geladen oder aus ihm entfernt.

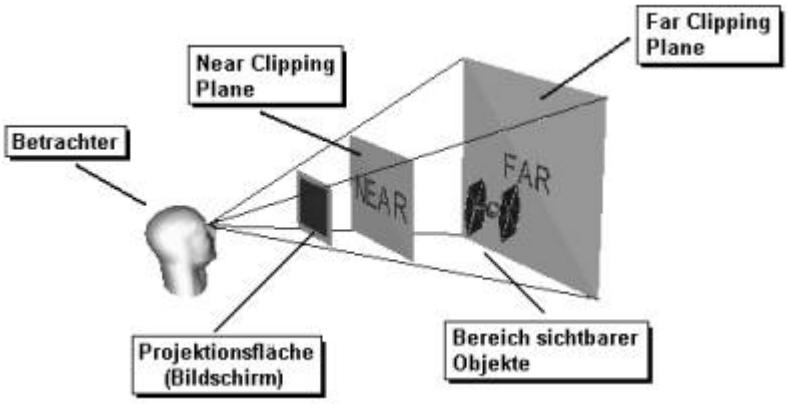
RTR – Paging



RTR – Visibility

- Für Realismus muss ein Spiel 3-4km Sichtweite simulieren
- Problem: Auf diese Weise entstehen Polygonzahlen, die nicht in Echtzeit von der Grafikkarte berechnet werden können.
- Visibility umfasst Techniken zur Reduzierung der Polygonzahl.

RTR – View Frustrum



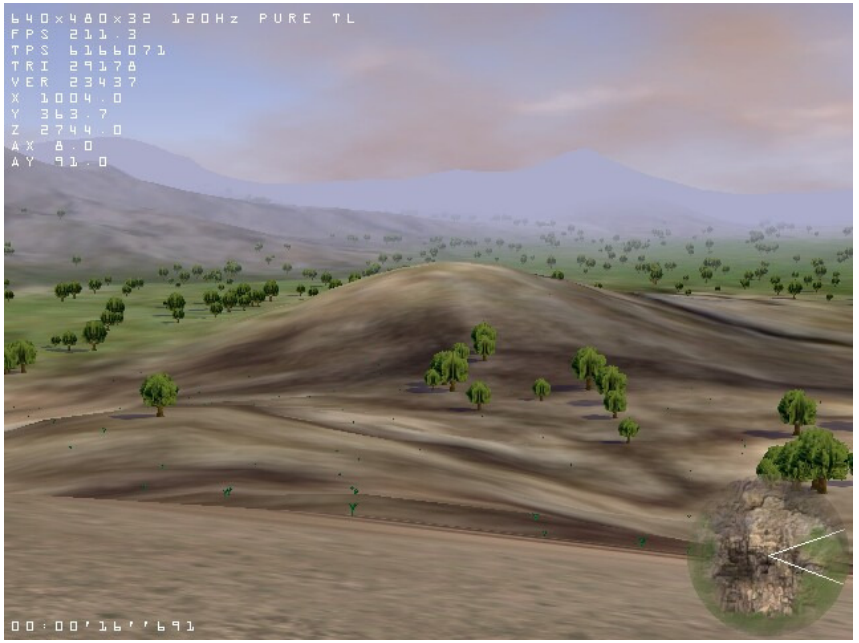
- Der View Frustrum ist Bestandteil der 3D-Renderpipeline von z.b. OpenGL und DirectX.
- Dient zur Elimination (Clipping) von Polygonen, die nicht zwischen der Near und Far Clipping Plane liegen.

Mitttels View Frustrum Culling kann man ganz einfach feststellen welche Dreiecke nicht im Bild sind und diese erst gar nicht an die Grafikkarte schicken. Bei einem Sichtfeld von 180° (meist kleiner) sind das 50% Performancegewinn

RTR – Occlusion Culling

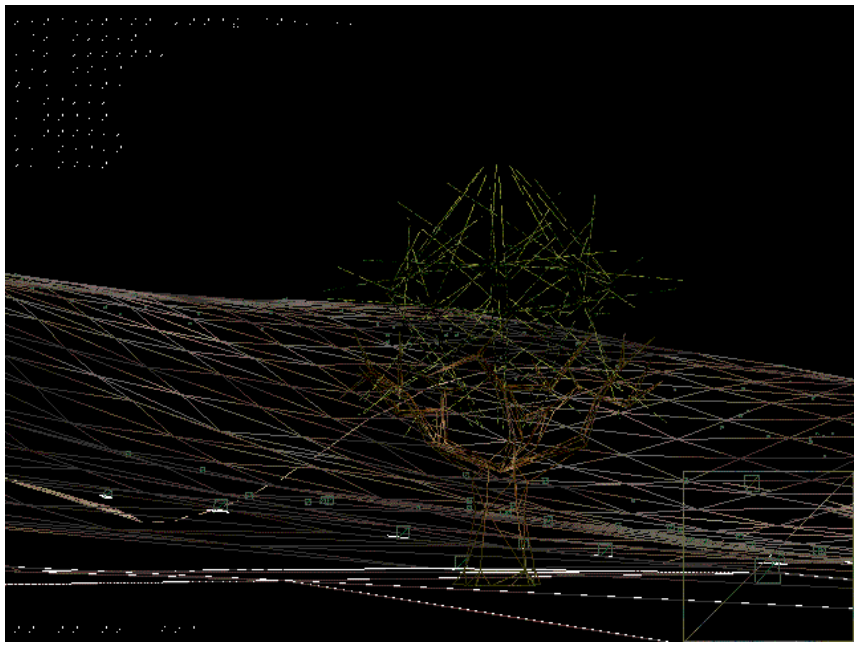
Occluder

- Idee umgekehrt zum Frustrum Culling.
- Man platziert Rechtecke (Occluder) in Bergen, die evtl. viel dahinterliegendes Terrain verdecken.
- Man macht ein Frustrum Culling, wobei die Near Clipping Plane der Occluder ist

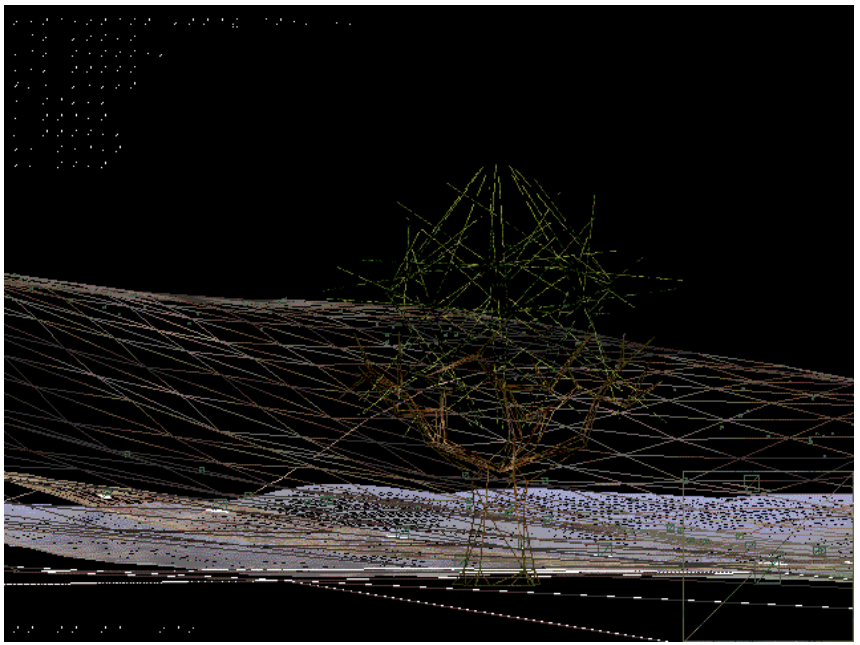


Alles was sich im View Cone befindet kann weggelassen werden

RTR – Occlusion Culling



mit occlusion culling

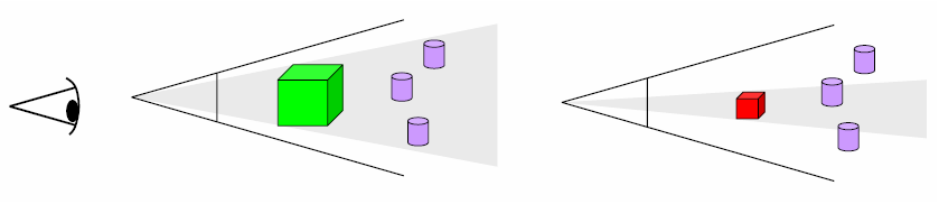


ohne occlusion culling

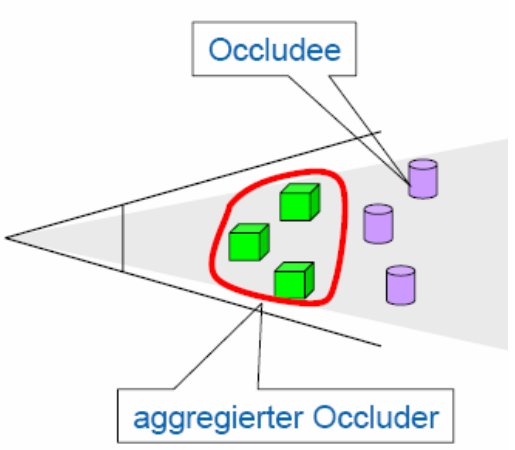
RTR – Occlusion Culling

Occlusion Maps

- Problem: kleine Occluder (rot) eignen sich schlecht für OC, da nur wenige Objekte verdeckt sind und entfernt werden können.



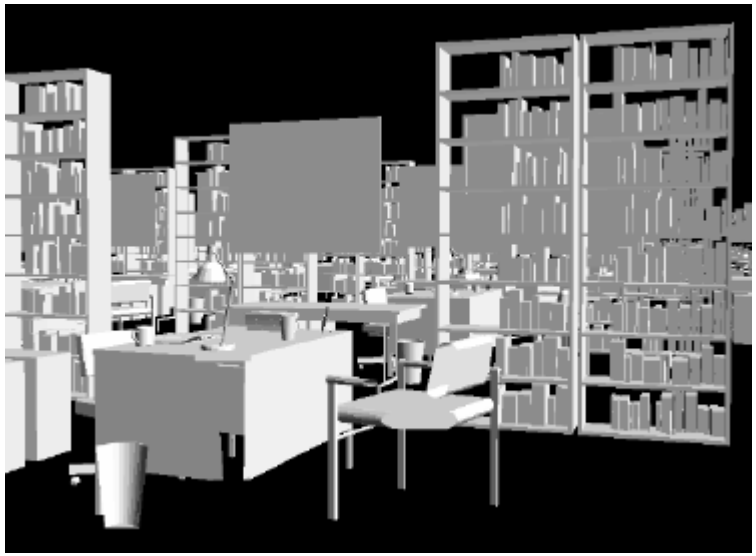
- Idee: Aggregation der Occluder
- Erstellung der Occlusion Map = 2D Projektion der der Szene, Occluder (weiß)
- Zur Beschleunigung werden mehrere kleinere Abstufungen der Map berechnet (Hierarchical Occlusion Map)



RTR – Occlusion Culling

Visibility Culling

- Bounding Box jedes Objects der Szene mit der Occlusion Map abgleichen
- Wenn die Bounding Box komplett in einem weißen Bereich der Occlusion Map liegt und das Object sich hinter den Occludern der Szene befindet, dann ist es nicht sichtbar und kann weggelassen werden.



3D - Szene

Hierarchical Occlusion-Map

RTR – Fogging

- einfachste Art die Polygonzahl einer Szene zu verringern
- in OpenGL und DirectX implementiert.
- lineare, quadratische Zunahme der Nebelintensität ausgehend von der Position Kamera (unrealistisch), neuerer Ansatz Volumetric Fogging
- beliebige Farbe des Nebels möglich, meist Farbe des Himmels

Drakan – Order of the Flame

Shadow Warrior

RTR – Level of Detail

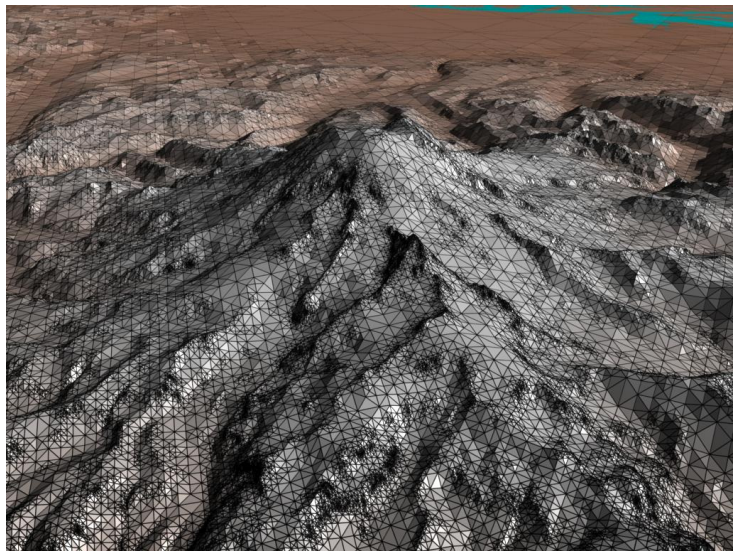
Motivation

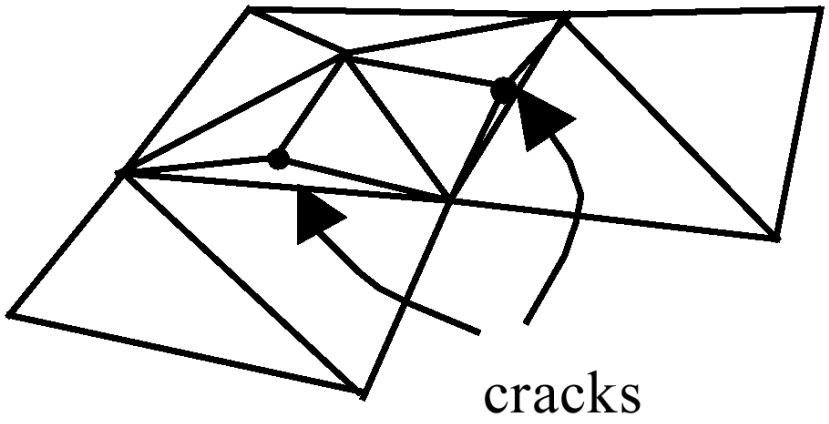
- **Ziel:** Reduzierung der Polygonzahl
- **Idee:** Weit entfernte Teile der Landschaft weniger detailliert rendern
- Beispiel: Eine Raumstation gerendert mit 40.000 Polygonen sieht aus einer Entfernung von 1km für den Spieler genau so aus wie dieselbe Raumstation mit 100 Polygonen.
- 2 verbreitete Varianten: continious LOD, discrete LOD

Wegen der Bedeutung von interaktiver Umgebung in heutigen Spielen haben cLOD Algorythmen an Bedeutung gewonnen, da auf diese Weise Terrainverformungen z.B. durch Granateneinschlag einfach umgesetzt werden können.

RTR – ROAM

- Real-Time Optimally Adapting Meshes (Mark Duchaineau, 1997)
- Grundlage für viele heute aktuelle Terrain Engines
- Crack Problem einfach gelöst

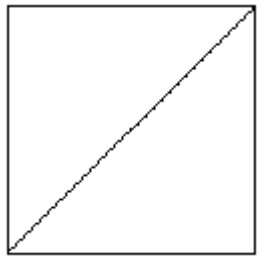




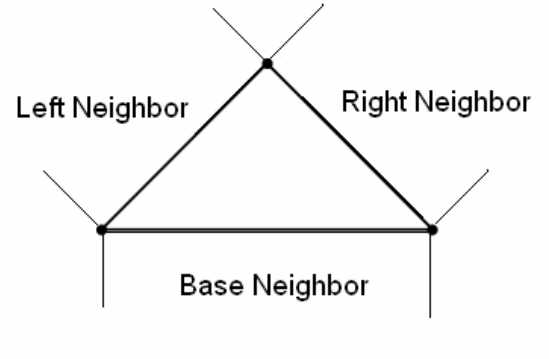
RTR – ROAM

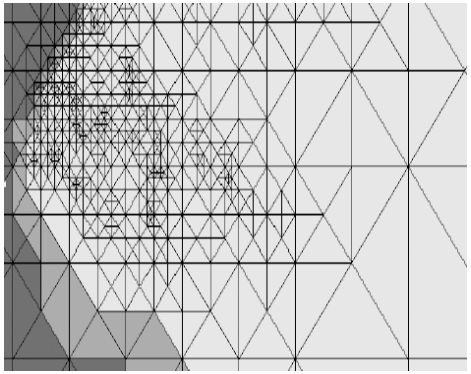
Idee: Ausgehend von einer groben Auflösung des Terrains wird abhängig von der Kameraposition und einer speziellen Heuristik weiter verfeinert.

1. Gleichmäßige Einteilung der Terrain Page in Patches
2. Verknüpfen der Patches untereinander durch einen Triangle Bintree
3. Triangulation der Patches, forced Splits



Patch
mit 2 Basisdreiecken





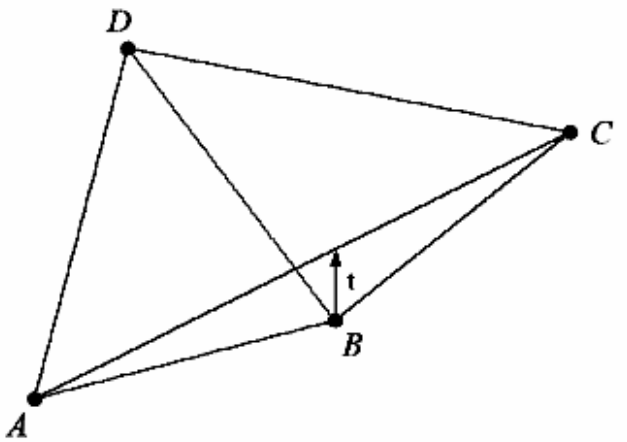
RTR – ROAM

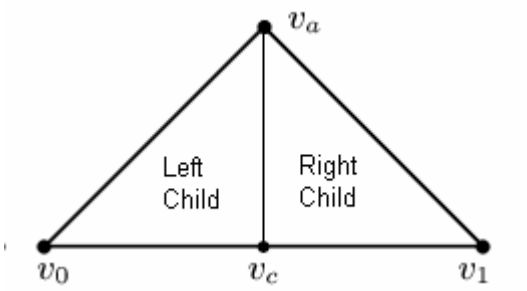
Triangulation / Splitting

- Für jedes der 2 Basisdreiecke eines Patches wird geprüft, ob die Abweichung t größer als der Grenzwert ist. Ist das der Fall muss gesplittet werden

- Die 2 neuen child-Dreiecke werden Nachbarschaftlich korrekt verlinkt

- Triangulation wird rekursiv auf den Childs fortgesetzt



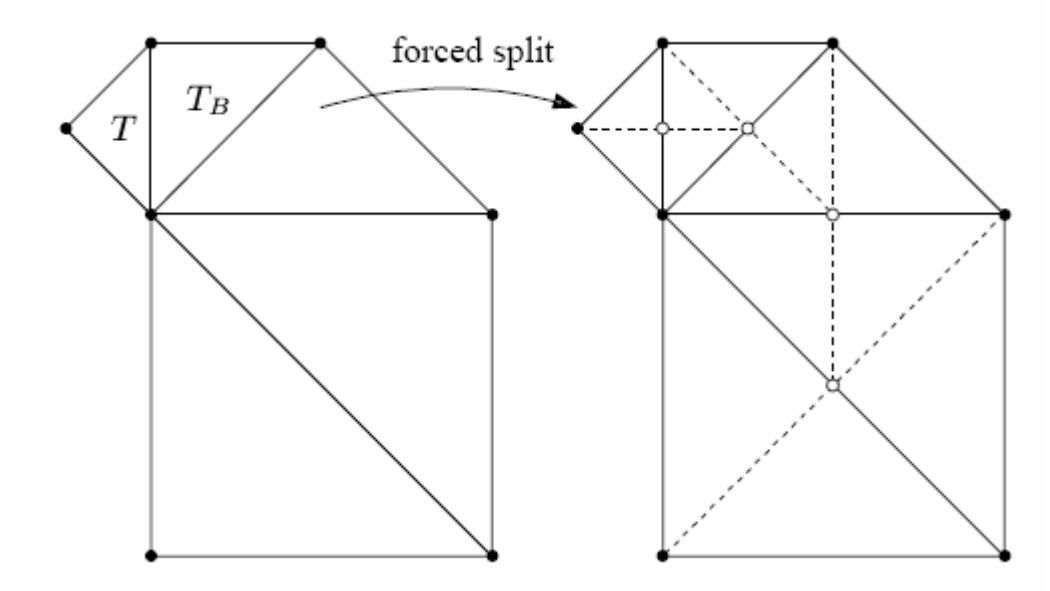


RTR – ROAM

Forced Splits

Zur Vermeidung von Cracks müssen benachbarte Dreiecke oft auch gesplittet werden.

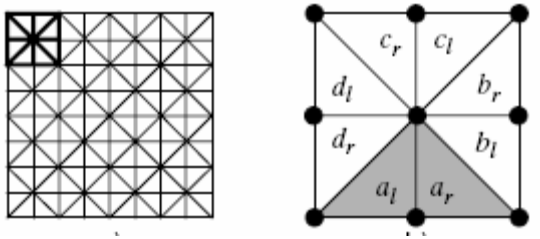
Bsp: Dreieck T hat eine zu große Abweichung und muss gesplittet werden.



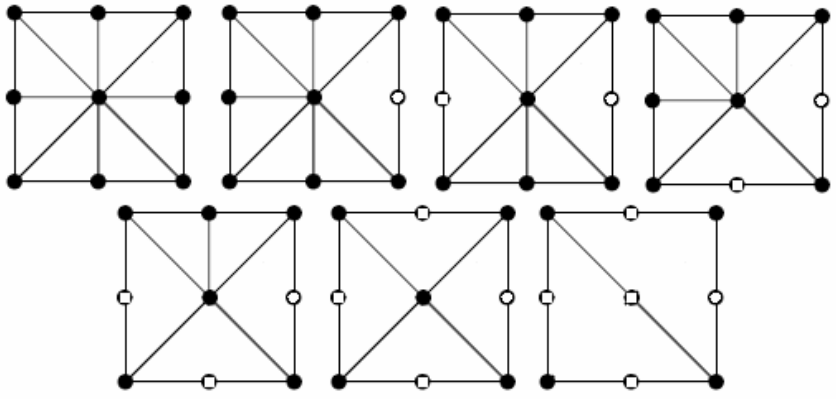
RTR – Quadtree Terrain

Idee: Es wird die feinste Auflösung genommen und in jedem Frame geprüft, welche Vertices weggelassen werden können.

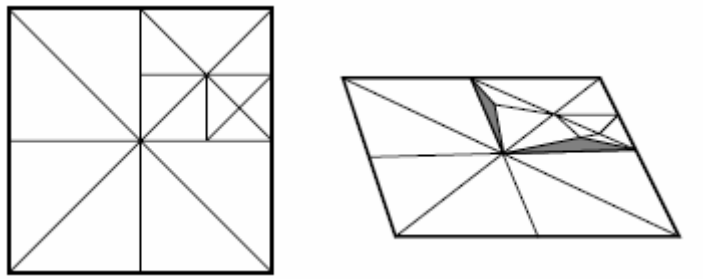
- Aufbau des Quadtrees rekursiv, anfangend bei der Größe der Page bist auf die Größe von 3x3 Vertices Blöcken.



- Deaktivierung von Vertices aufgrund einer speziellen Heuristik (meist Entfernung des Blocks zur Kamera)

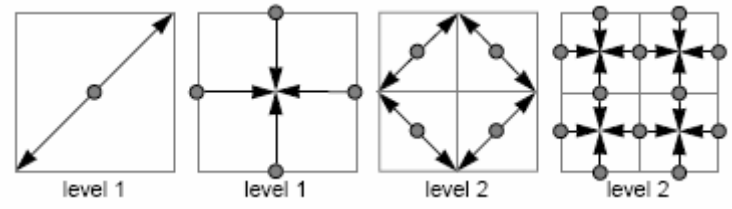


RTR – Quadtree Terrain

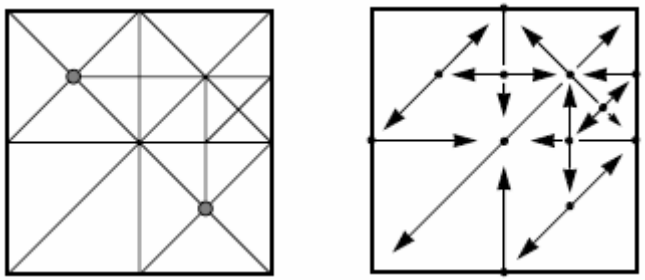


Uneingeschränkte Quadtree Triangulation

- Wahlfreies Deaktivieren von Vertices führt jedoch zu unschönen Cracks im Terrain



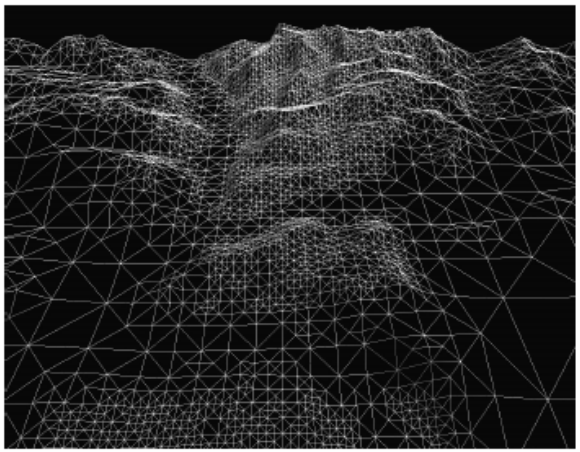
- Zur Lösung dieses Problems werden Abhängigkeits- Graphen benutzt

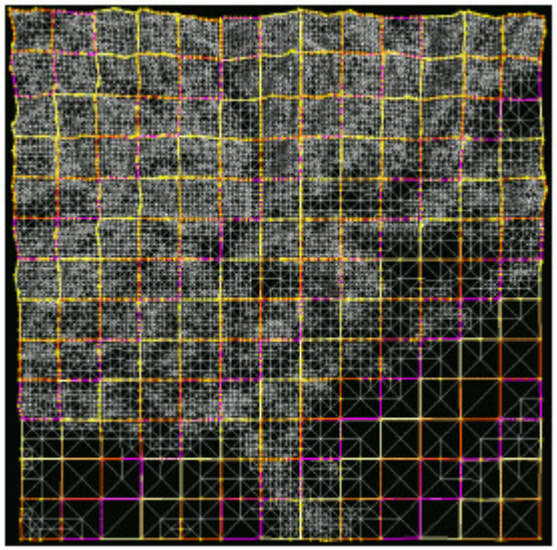


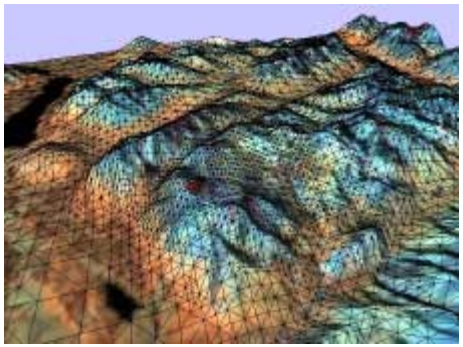
Eingeschränkte Quadtree Triangulation

- Jeder Vertex hängt von 2 anderen ab. Soll er enfternt werden müssen die anderen 2 auch entfernt sein.

RTR – Quadtree Terrain







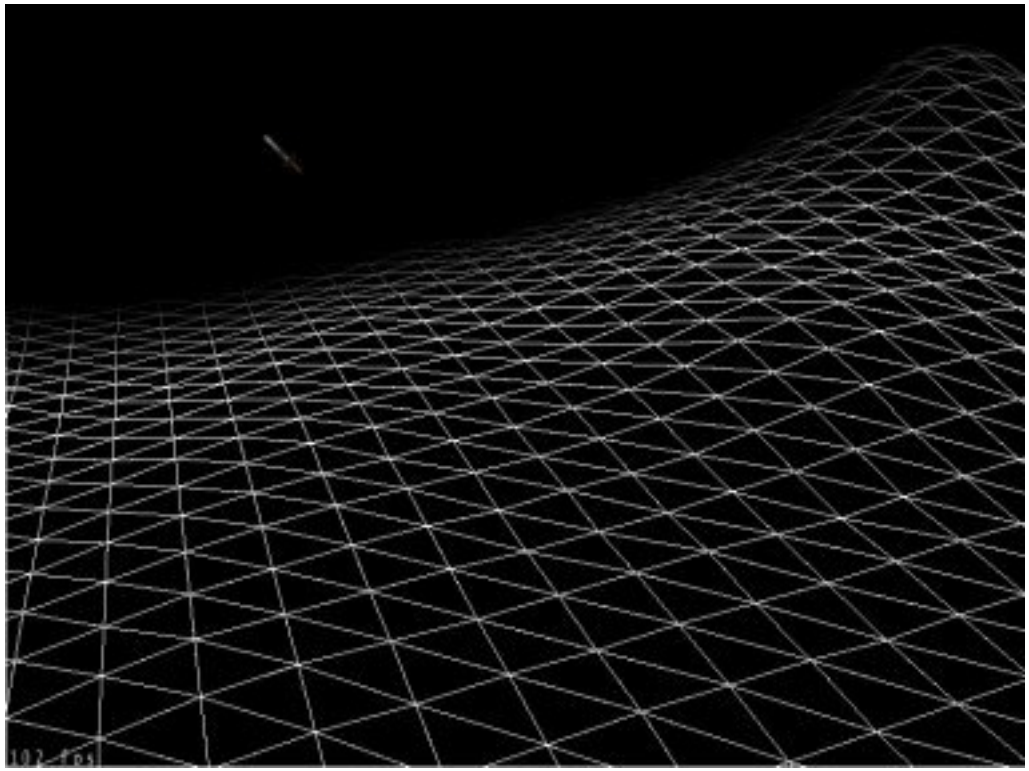
RTR – Brute Force Terrain

- ältestes und einfachstes Rendering Verfahren
- mit immer schnelleren Grafikkarten und größerem VRAM wird es interessanter die gesamte Landschaft in hoher Auflösung zu rendern, als aufwändige cLOD Berechnungen auf der CPU durchzuführen.

- Vorteile
  - Spart CPU Zeit
  - Dreiecke ändern sich nicht und können von der Grafikkarte für schnelles Rendern vorbereitet werden.

- Nachteile
  - Veränderung des Terrains schwierig
  - Nicht so detailliert wie cLOD Terrain

RTR – Brute Force Terrain



- Wichtig: Akkurates Frustrum & Occlusion Culling

Quellenangaben

- Stefan Zerbst, *3D-Spieleprogrammierung mit DirectX*, Band 2
- David Scherfgen, *3D-Spieleprogrammering*, Hanser
- *ROAMing Terrain: Real-time Optimally Adapting Meshes*, Mark Duchaineau, '97
- *Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation*, 1998, Renato Pajarola, IEEE Visualization '98
- Linksammlung: *Terrain LOD Published Papers*, http://www.vterrain.org/LOD/Papers/index.html
- *Terrain Occlusion Culling*, http://www.flipcode.com/articles/article_vystoc.shtml
- *Hierarchical Occlusion Maps*, Heinz Nixdorf Institut, Universität Paderborn 2006
- GameDev.net, *High Speed Brute Force*, http://www.gamedev.net/reference/programming/features/bruteforce/page2.asp