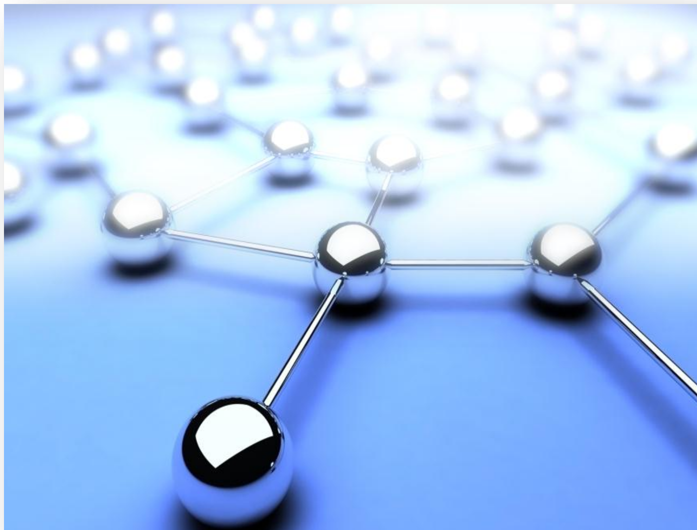
Vorlesung Algorithmen für hochkomplexe Virtuelle Szenen

Sommersemester 2012



Matthias Fischer
mafi@upb.de

Vorlesung 13
3.7.2012

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Übersicht

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Hierarchical Image Caching

- Motivation
- Idee
- Rendering erster Durchlauf
- Rendering zweiter Durchlauf
- Ziele der Partitionierung
- BSP-Patitionierung der virtuellen Szene
- Beispiel
- Fehlermetrik
- Beispiele

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Übersicht

Prüfungsvorleistung

- Prüfungsvorleistung ist ein regelmäßiger Übungsbesuch und ein Übungsgespräch über den Inhalt der Übungen.
- Das Übungsgespräch dauert ca. 5-10 Minuten und wird über den Inhalt einer in den Übungen behandelten Übungsaufgabe geführt.
- In dem Gespräch soll der Studierende nachweisen, dass er mit der Problemstellung und Lösung der Übungsaufgabe vertraut ist.
- Nur wenn dieser Nachweis gelingt, kann der Studierende an der Prüfung teilnehmen.
- Das Übungsgespräch findet gegen Ende der Vorlesung statt.

Termine

- 1. Woche nach Vorlesung: 17.7 / 18.7
- 2. Woche nach Vorlesung 23.7
- weitere nach Vereinbarung

- Termin vereinbaren: per email an mich mafi@upb.de

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Literatur

- Tomas Akenine-Möller, Eric Haines
  Real-Time Rendering
  AK Peters, 2002
- David Luebke, Martin Reddy, Jonathan D. Cohen
  Level of Detail for 3D Graphics
  Morgan Kaufmann Publishers, 2002
- Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, John Snyder
  Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments
  Proc. 23rd Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96), p. 75-82, 1996
  http://doi.acm.org/10.1145/237170.237209

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Walkthrough-System

- Der Benutzer bewegt sich typischerweise auf einem Pfad.
- Aufeinander folgende Bilder verfügen über hohe Kohärenz.

Wie nutzt man die Kohärenz aus?
Warum müssen wir bei jedem Bild von vorne anfangen und alles neu rendern?

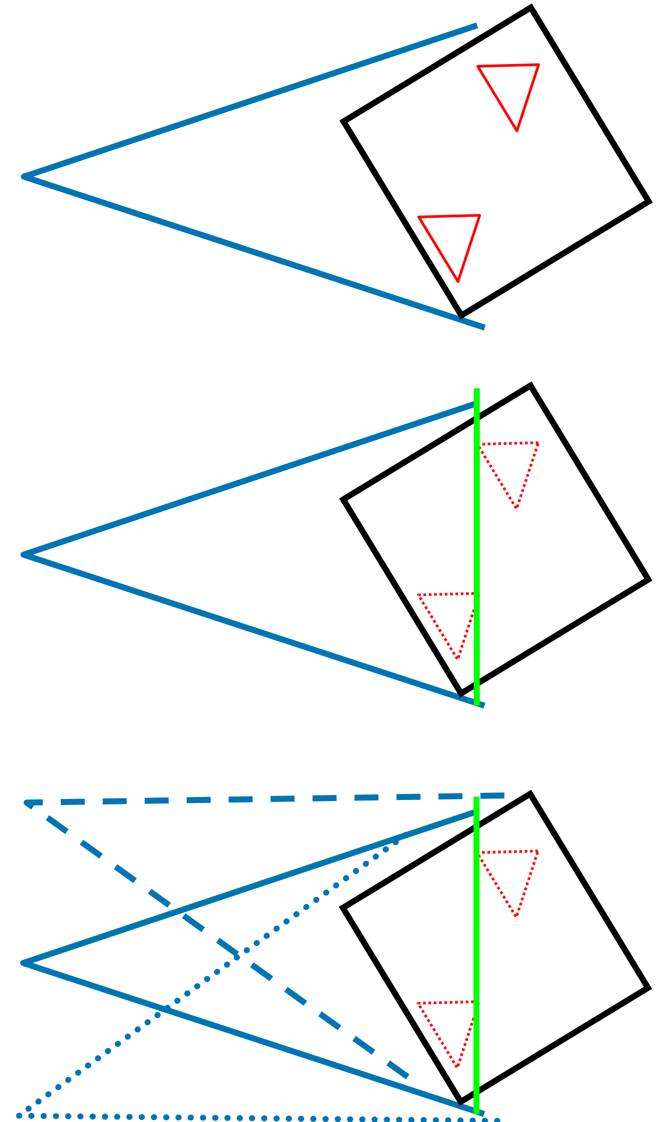
Gibt es eine Möglichkeit, Teile des Bilder wieder zu verwenden?

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Idee des Verfahrens

- Rendere benachbarte Objekte in eine Textur,
- verwende die Textur für nachfolgende Bilder
- und rendere die Objekte erst wieder, wenn sich die Kameraposition von der ursprünglichen Position zu weit entfernt.



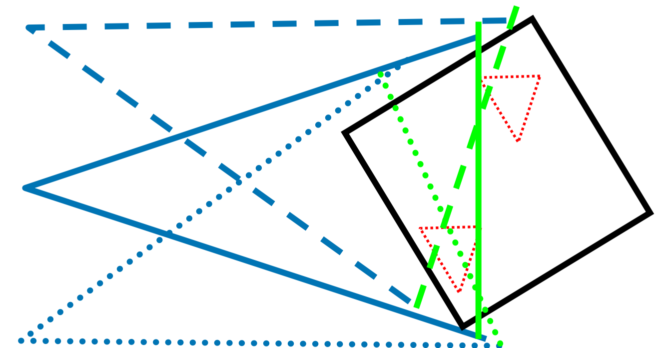
© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching

Idee

Wie können wir die Textur wiederverwenden?

- An Stelle der Objekte wird das gerenderte Bild der Objekte als Textur auf ein Rechteck gelegt.
- Das Rechteck wird mittig in die BBox des zu ersetzenden Szenenteils (später: des Knotens der Hierarchie) gesetzt.
- Bei leichten Bewegungen der Kameraposition wird das Rechteck entsprechend mit bewegt.
- Damit werden Parallaxefehler reduziert, aber nicht verhindert.

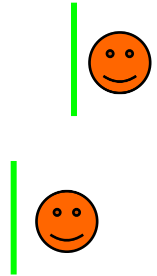


© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Idee

Warum cachen wir nicht einzelne Objekte?

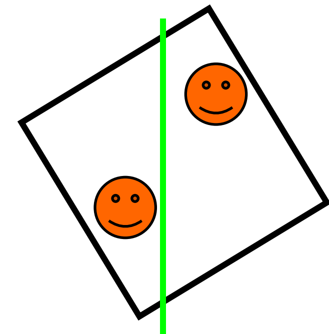
- Anzahl Objekte typischerweise zu groß
- zu viele Texturen (Rechtecke) müssten gerendert werden
- bringt keine Entlastung der Hardware, verlagert das Problem nur



Es ist effizienter, Gruppen von Objekten zu Clustern zusammenzufassen.

Räumliche hierarchische Datenstruktur zum Clustern von Objekten

- In der räumlichen Hierarchie können für beliebige Knoten Bilder gespeichert werden (Caching).
- Die Bilder werden in den inneren Knoten gespeichert.
- Jedes Bild ist ein Blick auf die im Teilbaum des Knotens enthaltenen Objekte.



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Idee

Welche Hierarchie verwenden wir?

- 3D-BSP-Baum
- Die Blätter des BSP-Baums stellen eine konvexe Region des Raumes dar.
- In den Blättern sind die Objekte (Geometrie) gespeichert.
- Objekte können durch Splittingebenen zerschnitten sein.

Wie wird gerendert?

- Wir rendern entweder:
  - normale Geometrie,
  - oder gecachte Texturen, die in vorhergehenden Frames gerendert wurden.
- Texturen unterschiedlicher Knoten können sich überlappen
- back-to-front order
  (notwendig, da die Texturen teilweise transparent sind)

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Welche Fragen sind zu klären?

- Wie partitionieren wir die Szene (räumliche Datenstruktur)?
- Wie speichern wir die Texturen?
- Wie rendern wir die Szene?
- Wie lange verwenden wir die Textur?
- Wie messen wir die auftretenden Bildfehler?

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Rendering

- back-to-front order
- besteht in jedem Frame aus zwei Durchläufen

1. Durchlauf

- Culling von Knoten außerhalb des Frustum.
- Aktualisiere den Bilder-Cache für die sichtbaren Knoten.
- Aktualisiert werden alle Knoten, für die die Fehler der Textur zu groß sind.

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Algorithmus UpdateCaches (node, viewpoint)

```
if „node ist außerhalb Frustum“  then
    node.status := CULL
else if node.cache ist für viewpoint noch „genügend gut“ then
    node.status := DRAWCACHE
else if „node is a leaf“ then
    UpdateNode( node, viewpoint)
else   // erst Kinder aktualisieren, dann Eltern
    UpdateCaches( node.back, viewpoint )
    UpdateCaches( node.front, viewpoint )
    UpdateNode( node, viewpoint )
```

UpdateNode()
berechnet,

- ob die Geometrie gezeichnet wird,
- oder ob es sich lohnt, einen Cache anzulegen.

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Rendering erster Durchlauf

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wir betrachten jetzt: UpdateNode()

Berechnet:

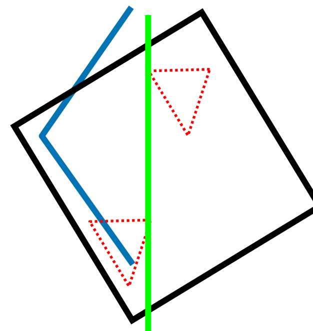
- ob die Geometrie gezeichnet wird,
- oder ob es sich lohnt einen Cache anzulegen?

Was benötigen wir für die Entscheidung?

- Wir benötigen die Renderingkosten für
  „cost to draw geometry“ + „cost to create cache“
- beides wird experimentell auf jedem Rechner neu ermittelt
  (im Preprocessing)

Beachte:

die Geometrie wird immer gezeichnet,
wenn der Betrachter im Knoten steht.



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Algorithmus UpdateNode(node, viewpoint)

```
if „viewpoint ist innerhalb node“
 then
    if „node ist ein Blatt“ then
        node.status := DRAWGEOM
    else
        node.status := RECURSE
        // im BSP-Baum absteigen und die Kinder einzeln behandeln
    return
```

k := EstimateCacheLifeSpan(node, viewpoint)
 // schätzt die Anzahl der Frames ab, für die sich das Caching des Bildes lohnt

amortizedCost := „cost to create cache“ / k + „cost to draw cache“
 // Amortisierte Kosten über die Anzahl Frames (=k), für die das Bild aus dem
 // Cache zum Rendern verwendet wird.

…… nächste Folie …..

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Algorithmus UpdateNode(node, viewpoint)

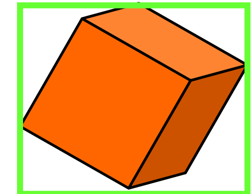
… Fortsetzung ….

```
if amortizedCost < „cost to draw geometry“ then          // Caching lohnt sich!
    CreateCache( node, viewpoint )
    node.status := DRAWCACHE
    node.drawingCost := „cost to draw cache“
    // Kosten für das Zeichnen der Textur
else                                                     // Caching lohnt sich nicht!
    if „node ist ein Blatt“ then
        node.status := DRAWGEOM
        node.drawingCost := „cost to draw geometry“
        // Kosten für die eigene Geometrie
    else
        node.status := RECURSE
        node.drawingCost := node.back.drawingCost + node.front.drawingCost
        // Kosten der Kinder
```

Hierarchical Image Caching
Rendering erster Durchlauf

Wie wird ein Cache angelegt „CreateCache(node, viewpoint)“ ?

- Berechne achsenparallele **BBox** vom **Knoten** im Bildraum.
- Projiziere dazu Eckpunkte des Knotens in den Bildraum und verwende die Minima/Maxima.
- Zeichne den Knoten
  - für Blätter: rendere die Geometrie
  - für innere Knoten: zeichne die Kinder
- Speichere Bild als Textur im Cache.



Innere Knoten …

- werden nicht ausschließlich durch Zeichnen der Geometrie dargestellt.
- können Texturen von Kindern im Teilbaum mitverwenden!

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Rendering

- back-to-front order
- besteht in jedem Frame aus zwei Durchläufen

1. Durchlauf

….

2. Durchlauf

Durchlaufe den BSP-Baum ein zweites mal in back-to-front order:

- zeichne die Geometrie,
- oder die Textur des Knotens im Cache.

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Rendering zweiter Durchlauf

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Algorithmus Render(node, viewpoint)

```
if node.status == CULL then
    return

else if node.status ϵ {DRAWCACHE, DRAWGEOM} then
    Draw( node ); // zeichnet die Geometrie oder gecachte Textur

else if „viewpoint is vor der node.splittingPlane“ then
    Render( node.back, viewpoint );
    Render( node.front, viewpoint );

else
    Render( node.front, viewpoint );
    Render( node.back, viewpoint );
```

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

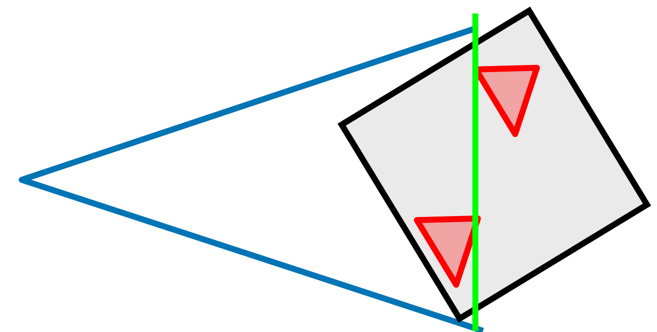
Partitionierung der Szene in Zellen

Fragen

- Wie partitionieren wir die Szene (räumliche Datenstruktur)?
- Wie speichern wir die Texturen?

Partitionierung

- Wir verwenden zur Aufteilung in Zellen einen 3D BSP-Baum.
- Das Bild für jede Zelle wird als Textur gespeichert
- und auf ein Rechteck gerendert.

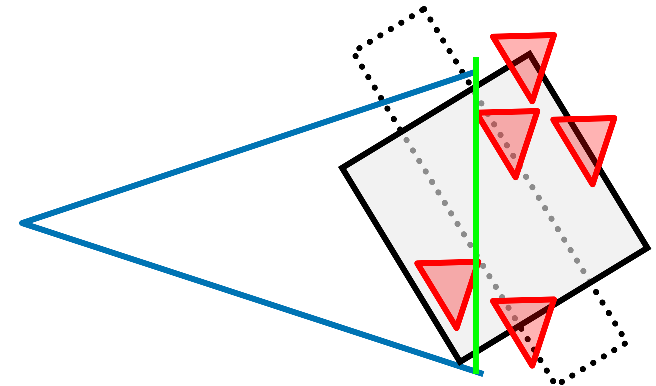


© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Ziele der Partitionierung

1. Teile so wenig Objekte wie möglich.
2. Balanciere die hierarchische Aufteilung bestmöglich.
3. Das Seitenverhältnis der BBox jedes Knotens sollte bei 1 liegen.

Warum benötigen wir diese Ziele?



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Warum benötigen wir diese Ziele?

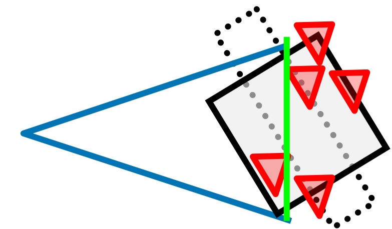
1. Teile so wenig Objekte wie möglich.
2. Balanciere die hierarchische Aufteilung bestmöglich.
3. Das Seitenverhältnis der BBox jedes Knotens sollte bei 1 liegen.

1:

- Vermeidung von visuellen Artefakten,
  geteilte Objekte werden ggfs. durch mehrere Texturen dargestellt.

2,3:

- bessere Performance
- balancieren ermöglicht besseres Frustum Culling
- BBox mit ausgewogenem Seitenverhältnis sind länger gültig



Hierarchical Image Caching

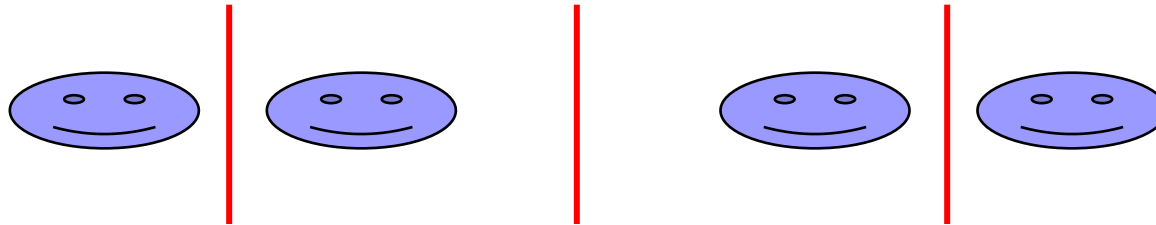
BSP-Partitionierung der virtuellen Szene

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Wie partitionieren wir den 3D BSP-Baum?

- Wir legen die Splittingebenen zwischen die Objekte,
- möglichst ohne Objekte zu schneiden!
- Mit der Menge der geteilten Objekte verfahren wir rekursiv weiter.



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

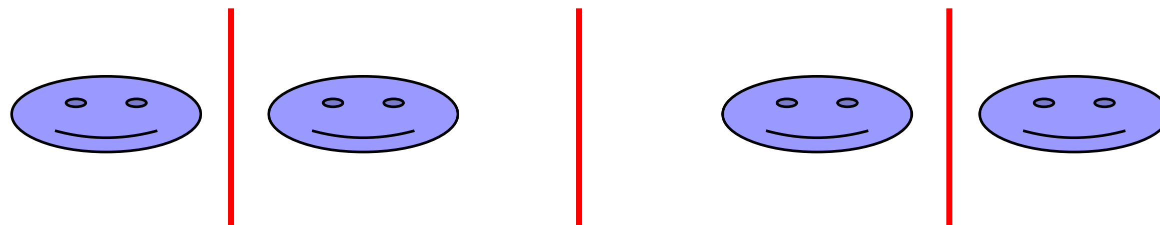
Welche Splittingebenen wählen wir?

- Wir wählen n verschiedene Richtungen und verwenden die Ebenen senkrecht zu den Richtungen als Splittingebenen.

Beispiel

- 3 Richtungen: x, y, z-Achse
- Splittingebenen: yz, xz, xy-Ebene

Die Richtungen wählen wir so, das möglichst wenig Schnitte entstehen.



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Welche Charakteristika besitzt unsere Szene?

Unsere Zielszenen sind Landschaftsszenen, d.h.:

- wir haben ein großes Oberflächennetz (Höhenfeld),
- auf der Oberfläche sind Objekte wie Bäume und Häuser aufgestellt.

d.h. alle Objekte befinden sich auf der XZ-Ebene mit unterschiedlichen Höhenwerten (y-Koordinate)

Wie wählen wir bei diesem Szenentyp die Splittingebenen?

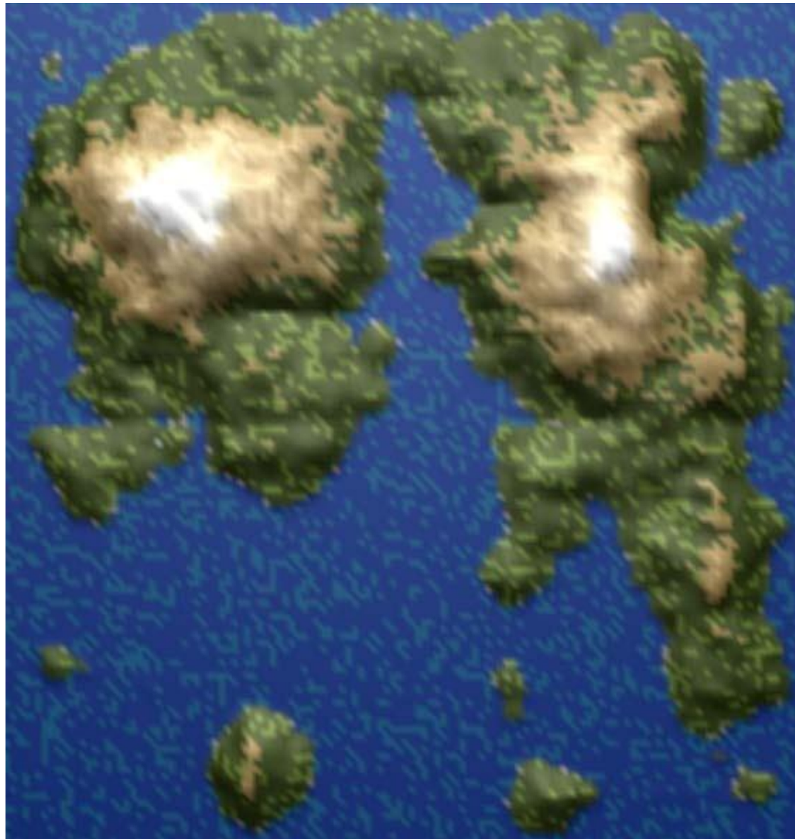
- alle Splittingebenen stehen senkrecht auf der XZ-Ebene
- d.h. wir verwenden zwei Richtungen (x- und z-Achse)

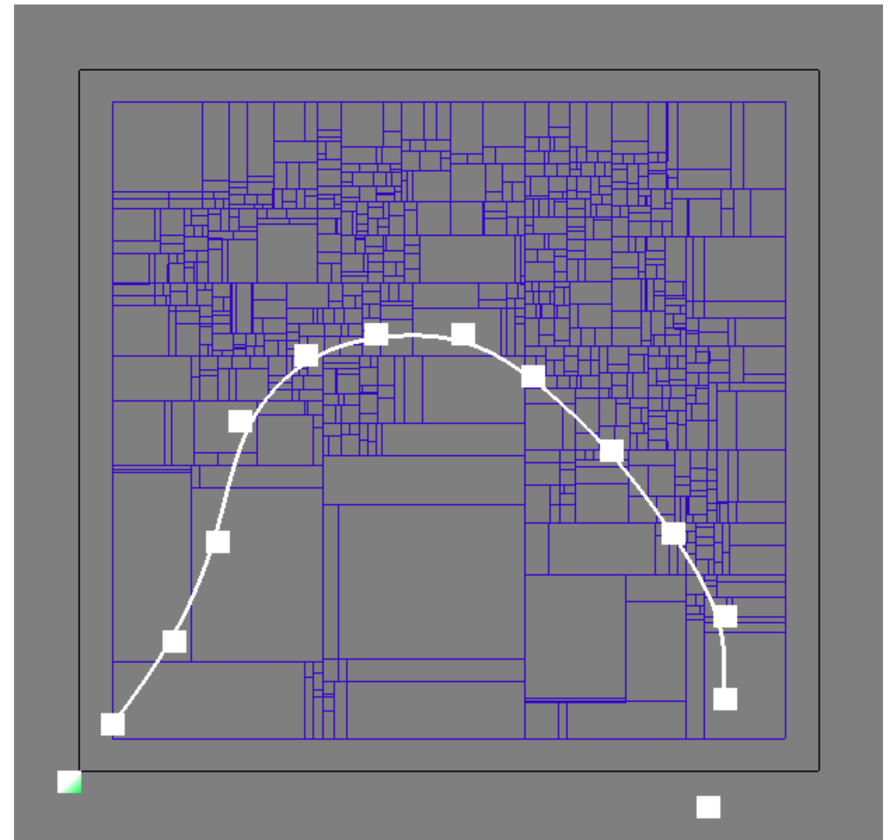
Hierarchical Image Caching
BSP-Partitionierung der virtuellen Szene

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Beispiel:

Szene mit einer Inselgruppe, Partitionierung durch den BSP-Baum





© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Objekte und Typ der Szene

- Das Verfahren benötigt keine Information über die semantische Bedeutung von Objekten.
- Auch die Charakteristik Szene kann beliebig sein.

Aber:

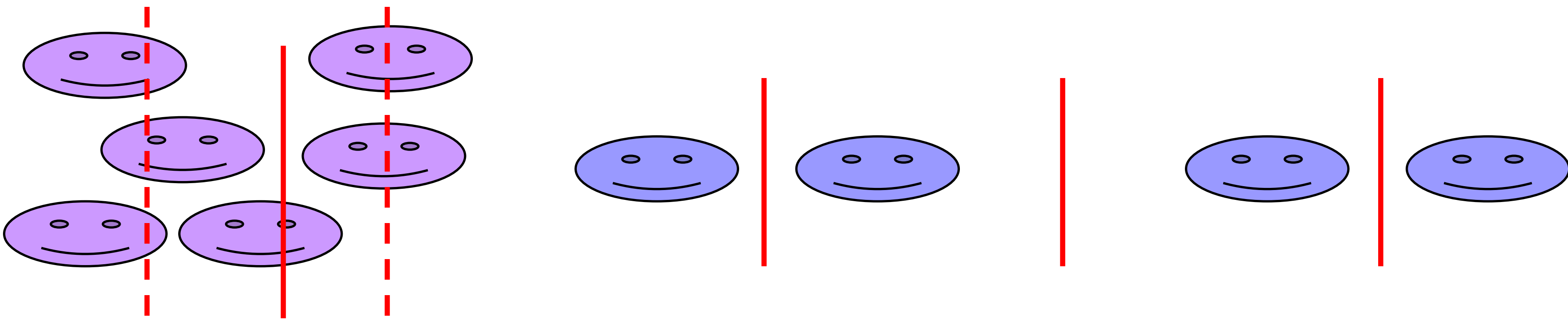
Ist die Charakteristik der Szene bekannt,
so ist eine „bessere“ Partitionierung mit dem BSP-Baum möglich!

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wie berechnen wir die Splittingebenen?

Was passiert, wenn wir keine Splittingebene finden, ohne Objekte zu schneiden?



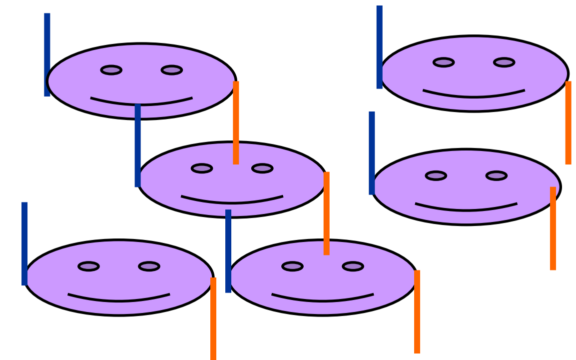
© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Heuristik

Für jede der n Richtungen (bei uns 2, x und z Achse!):

- Erzeuge zwei sortierte Listen,
- eine speichert die untere Begrenzung des Objektes,
- die Zweite speichert die obere Begrenzung des Objektes.



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

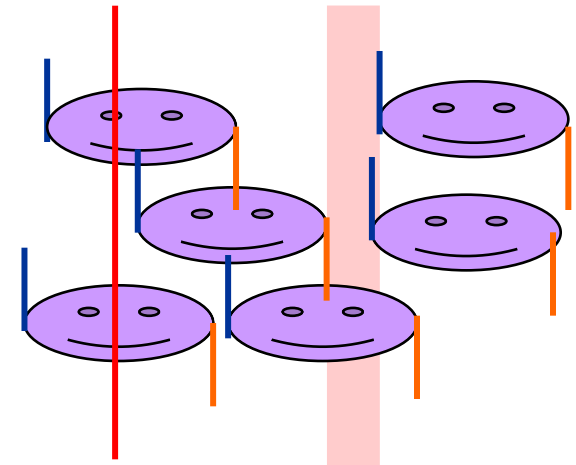
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wie finden wir die Ebene mit den wenigsten Schnitten?

- Im Scan-Line-Verfahren laufe jeweils über eine Liste.
- Zähle die bereits geschnitten Objekte (= „aktive Objekte").
- Berechne alle Intervalle mit den wenigsten „aktiven Objekten"

Zusätzlich sollte die Aufteilung in zwei gleich große Mengen erfolgen.

Das ist jedoch nicht immer möglich!

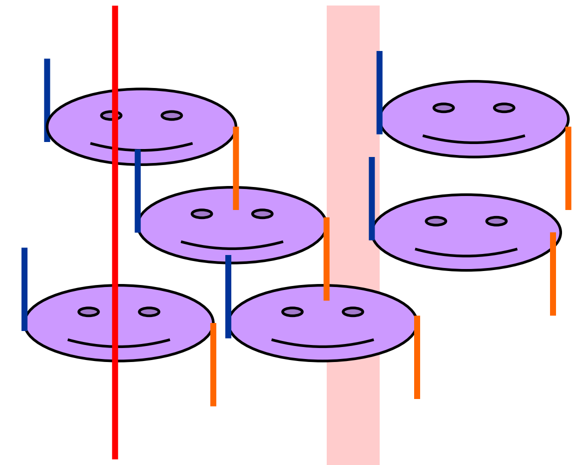


© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wie wählen wir die Splitebene?

- Berechne eine Funktion aus Anzahl „aktiver“ Objekte und dem Verhältnis der Aufteilung.
- Wähle für jede Richtung die Splittingebene mit minimalem Funktionswert (Heuristik).



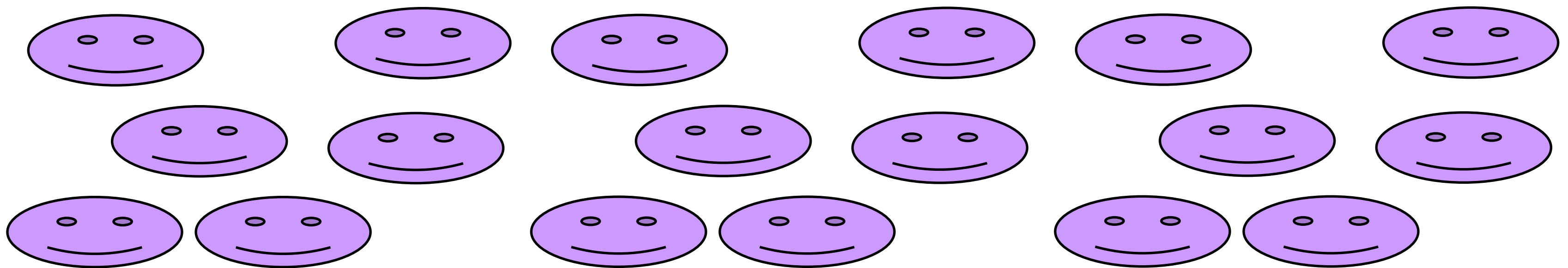
© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Wie berücksichtigen wir ein gutes Seitenverhältnis der Aufteilung?

- teile immer die längste Seite



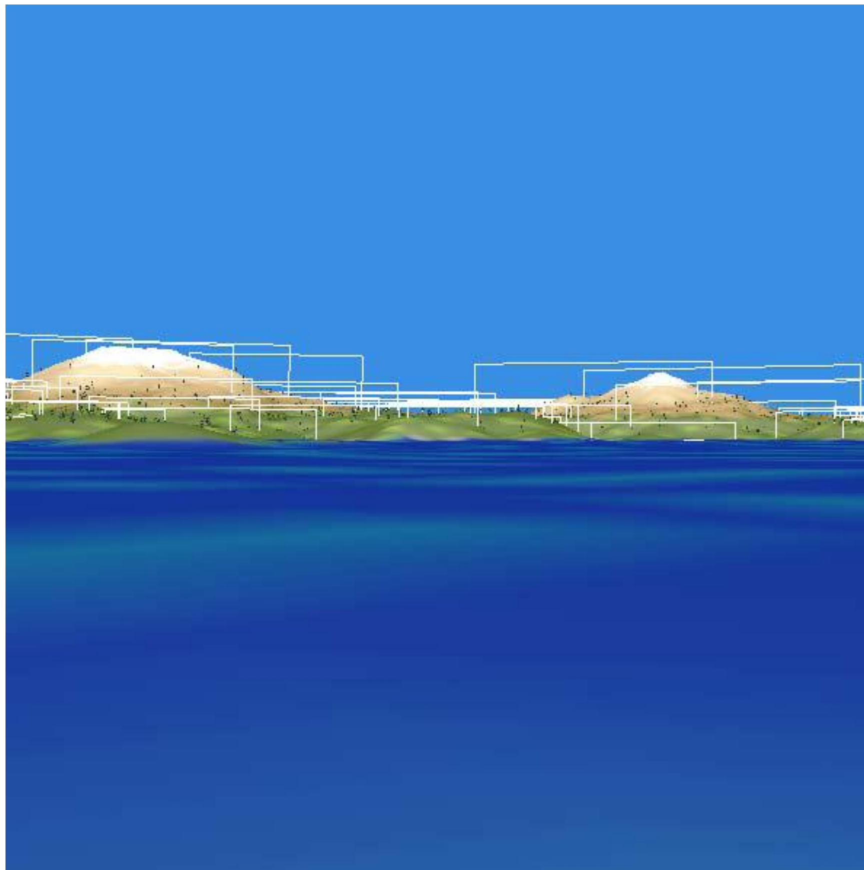
Hierarchical Image Caching
Beispiel

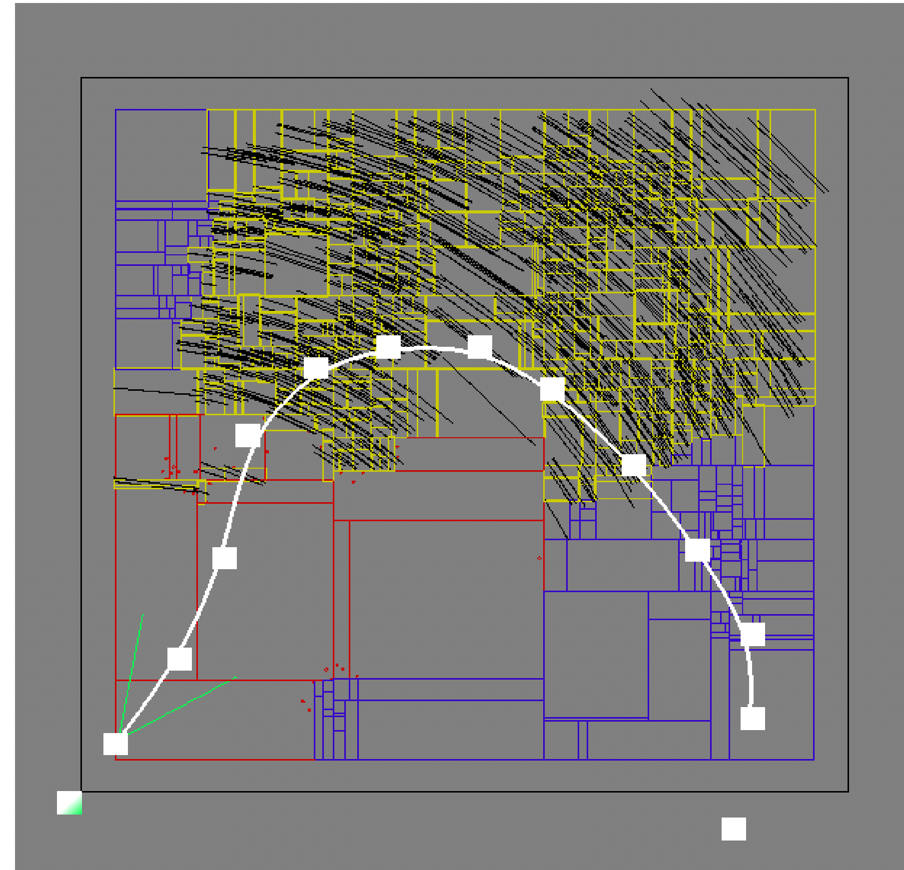
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Beispiel: Bewegung durch die Inselgruppe

links: aus Sicht der Kamera mit eingezeichneten texturierten Bildern

rechts: Sicht von oben auf die BSP-Partitionierung mit eingezeichneten Bildern





© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Fehlermetrik

Zwei Fragen müssen noch beantwortet werden:

Gegeben sei ein Knoten in der Hierarchie und seine Textur im Cache:

- Ist diese Textur auch für die aktuelle Position eine gute Approximation?

Gegeben sei ein Knoten in der Hierarchie und eine Kameraposition:

- Wird eine neue Textur für den Cache berechnet?
- Wie lange ist diese gültig?

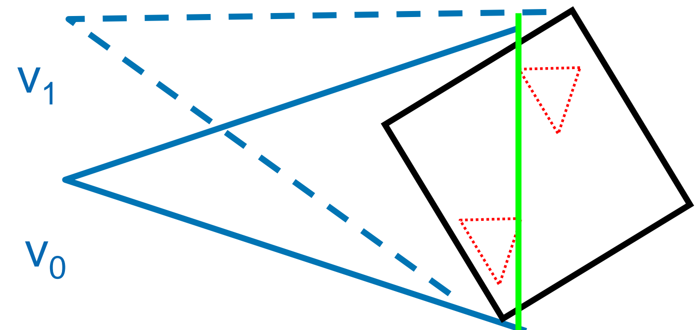
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Fehlermetrik

- Wir müssen eine Fehlermetrik finden, die den Unterschied zwischen der Textur im Cache und der gerenderten Geometrie quantifiziert.
- Ist dieser Unterschied kleiner als ein Schwellwert ε, dann ist die Textur noch gültig, anderenfalls muss sie verworfen werden.

Wichtig:

- Die Berechnung muss schnell erfolgen (zur Laufzeit).
- Eine genaue zeitaufwendige Berechnung möchte man vermeiden.



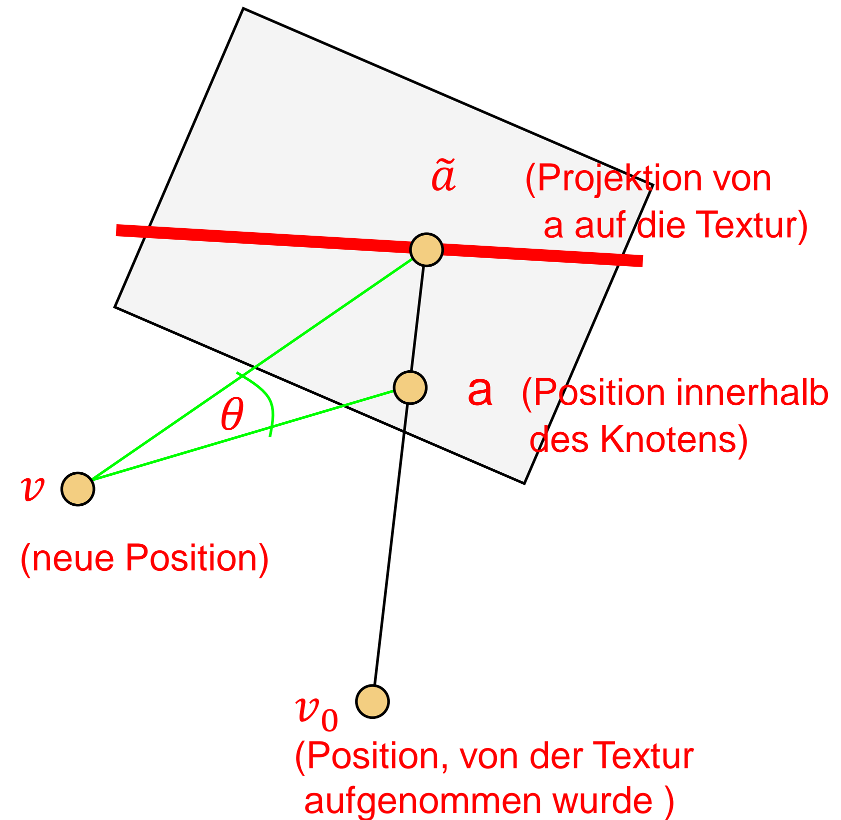
© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Fehlermetrik

Unser Fehlermaß „$Error()$“ misst das Maximum des Winkels θ über alle Positionen a innerhalb des Knotens (BBox)

$$Error(v, v_0) = max_a \theta(a, v, \tilde{a})$$



Wie können wir für $Error()$ schnell eine Näherung berechnen?

- Wir berechnen $Error()$ für alle 8 Ecken der BoundingBox des Knotens.
- Davon nehmen wir das Maximum.

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Fehlermetrik

Gültigkeit eines Fehlers

Wie schätzen wir ab,

- wie lange ein Fehler zwischen der Textur und
- der Geometriedarstellung nicht überschritten wird?

Annahme: Geschwindigkeit und Beschleunigung ist bekannt und begrenzt

→ Der Benutzer legt in kleinen Zeiträumen dann nur kleine Entfernungen zurück.

Diese Annahme ist elementar für die Gültigkeitsdauer einer Textur.

„Beamen", also sprunghafte Positionswechsel, sind nicht erlaubt.

Sicherheitszone

Wir definieren als Sicherheitszone den Bereich um eine Position, in dem der Fehler kleiner als ε ist.

$$SafteyZone(v_0) \subseteq \{v \mid Error(v, v_0) \leq \varepsilon\}$$

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Wo liegen die Punkte,

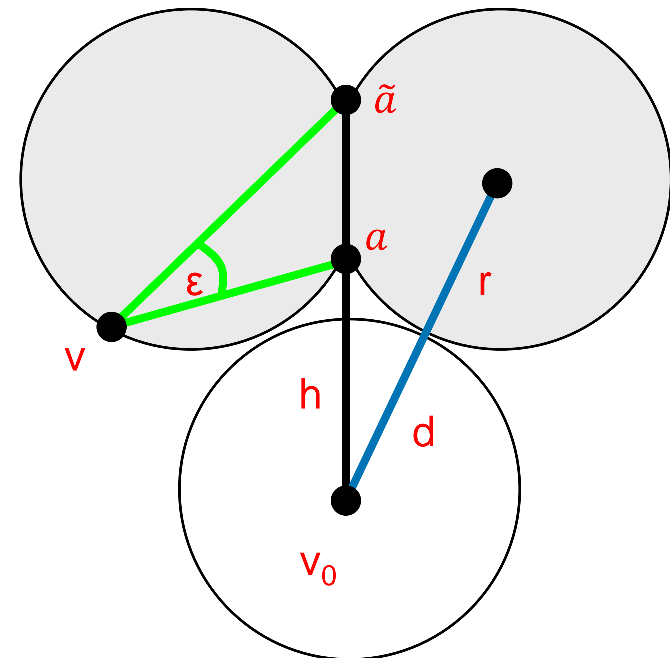
- die einen Fehler erzeugen,
- der größer als ein Schwellwert ε ist?

Alle Punkte gleich ε liegen auf zwei Kreisen mit Radius r:

$$r = \frac{\|a - \tilde{a}\|}{2\sin(\varepsilon)}$$

Alles außerhalb der beiden Kreise,
also auch der Kreis mit Radius d und Mittelpunkt v_0,
ist ein Teil der Sicherheitszone:

$$d = \sqrt{h^2 + r^2 + 2hr\sin(\varepsilon)} - r$$



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

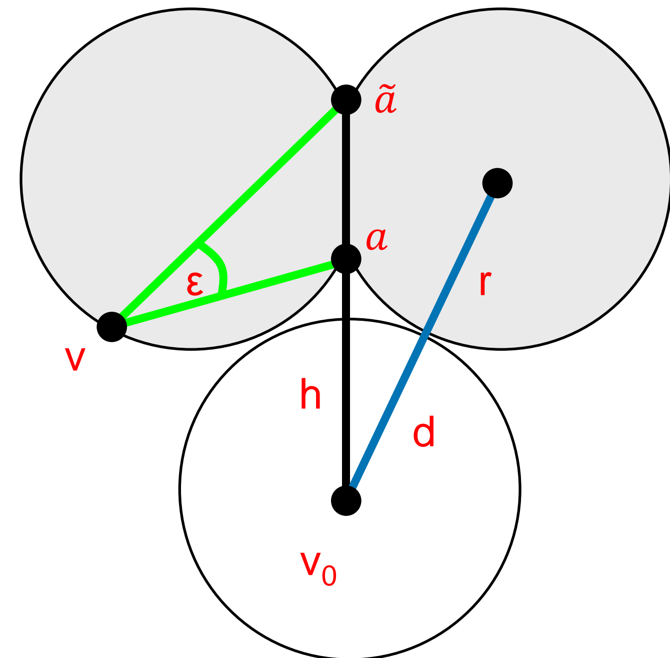
Wie berechnen wir d?

Blätter des BSP-Baums:

Verwende die 8 Ecken der Boundingbox (gleich a)

Innere Knoten:

Verwende die 8 Ecken der BBox des Knotens, schneide diese Sicherheitszone mit denen der Kinder.



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Beispiele von gerenderten Bildern

- Welche Artefakte treten auf?

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Beispiele

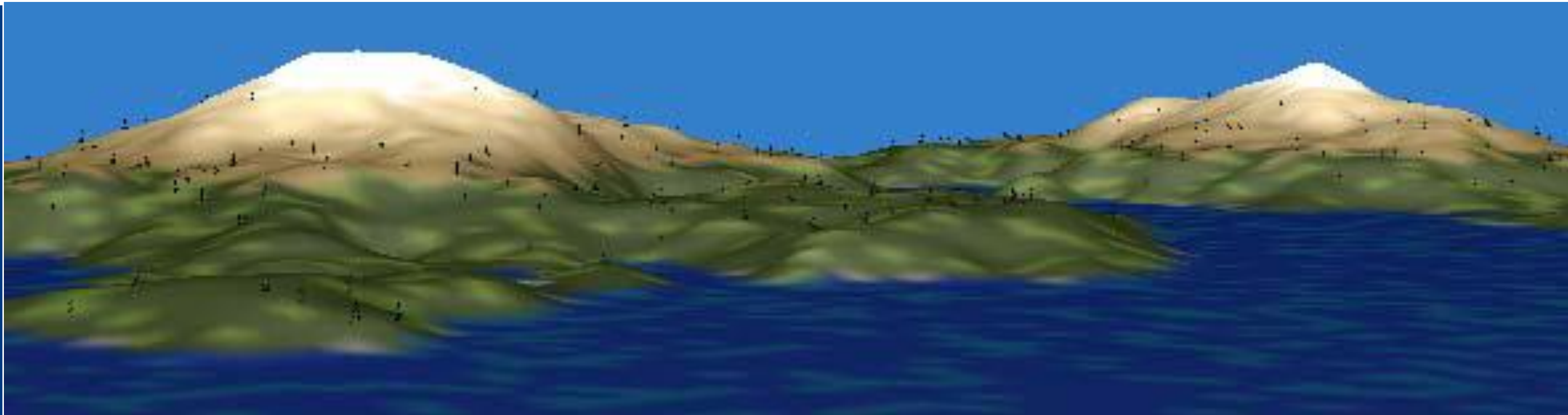


Bild mit Geometrie gerendert

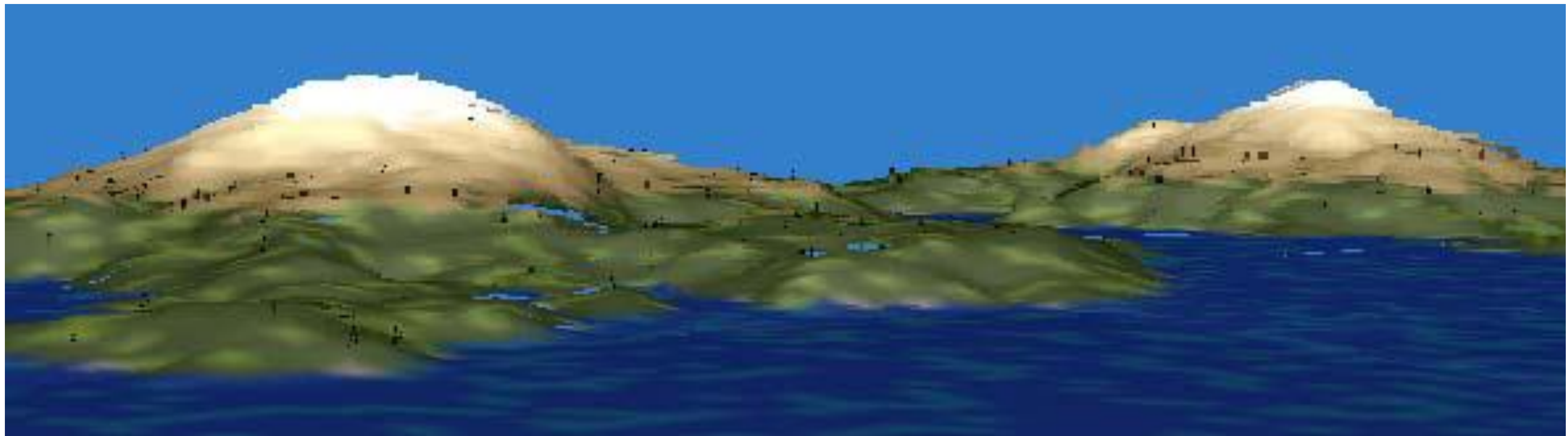


Bild mit Image Cache gerendert : Fehlerschwellwert von 4 Pixeln
Fehler an den Bergkuppen und in der grünen Ebene scheint Wasser durch

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Beispiele

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Bild gerendert
mit Image Cache

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

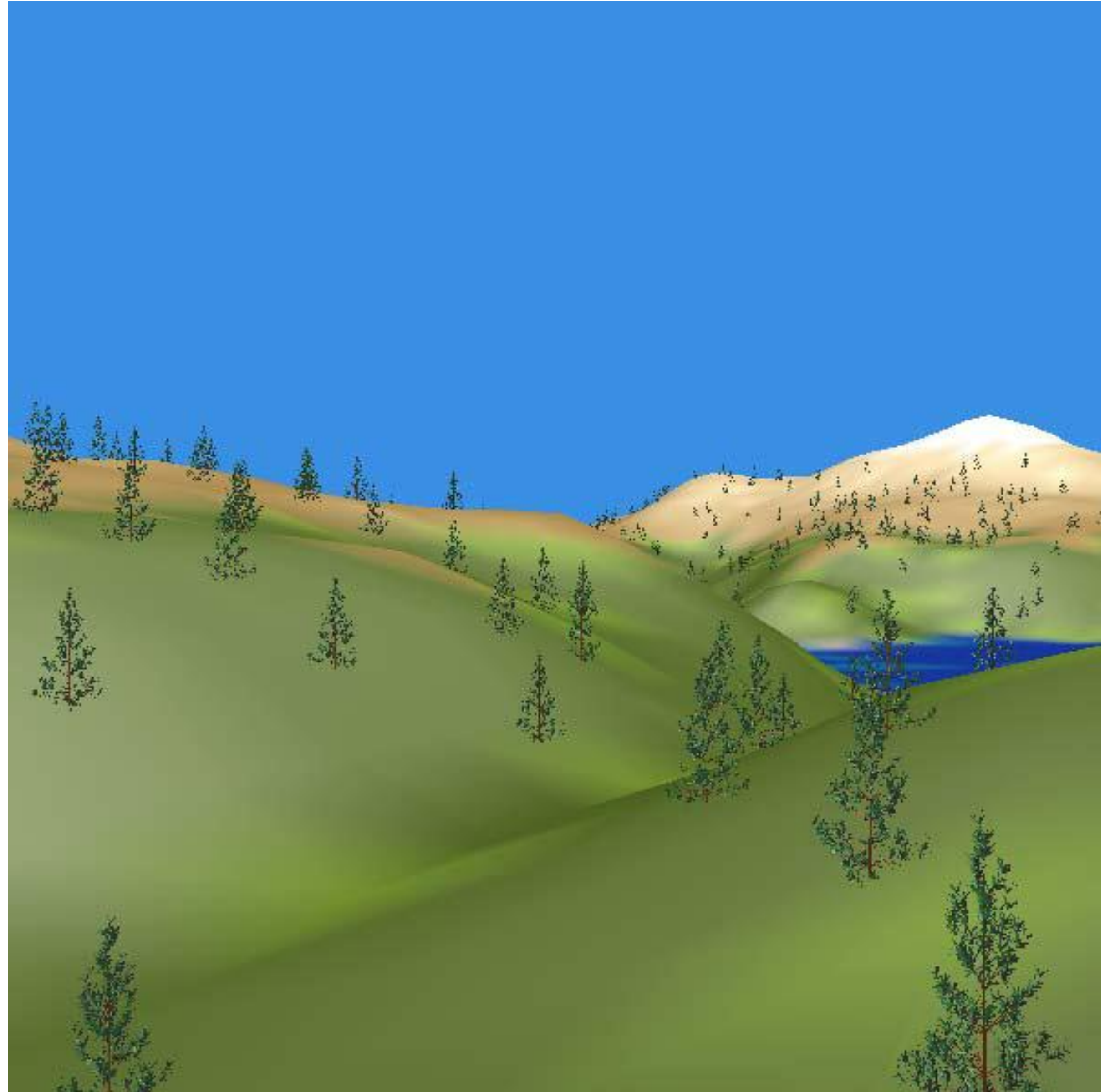
Hierarchical Image Caching

Beispiele

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Bild gerendert mit Geometrie

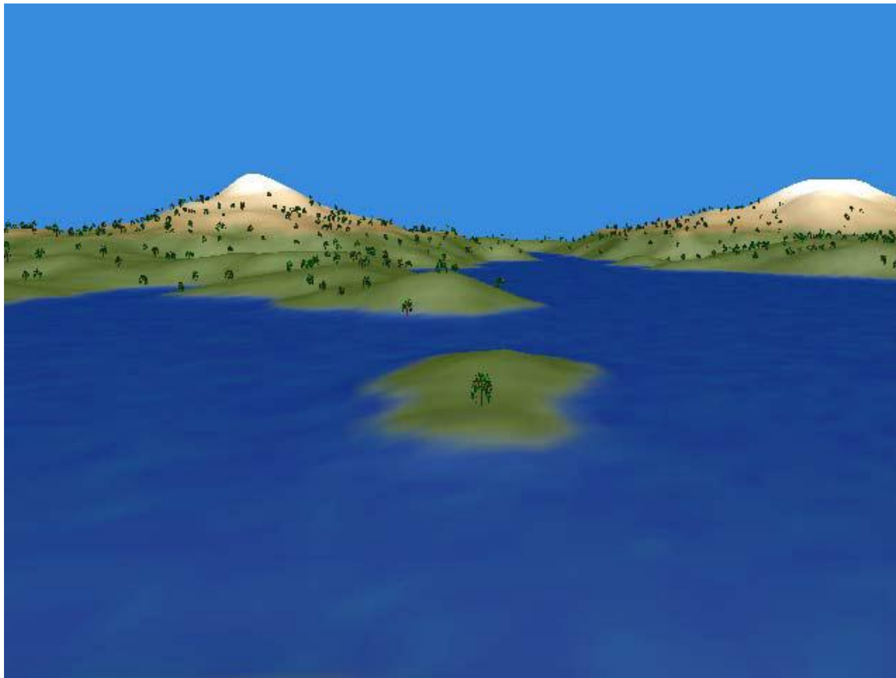
- Die Bäume am hellbraunen Hügel sehen „voller“ aus
- Hier treten im normalen Bild allerdings auch schon Aliasing Artefakte auf
- Bäume im Vordergrund ohne erkennbare Unterschiede



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Beispiele

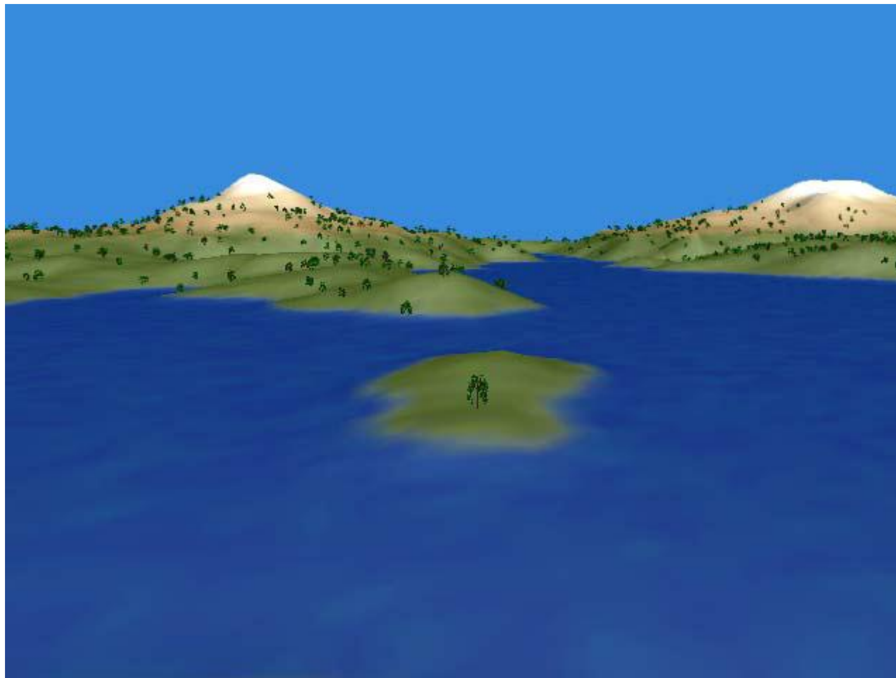
Bild mit Geometrie gerendert



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Bild mit Image Cache gerendert

Fehlerschwellwert: 2 Pixel



© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Hierarchical Image Caching
Beispiele

© Prof. Dr. math. F. Meyer auf der Heide, Heinz Nixdorf Institut, Universität Paderborn, Bild: © Fotolia, tom

Bild mit Image Cache gerendert

Fehlerschwellwert: 8 Pixel

Am Ufer im grünen Bereich scheint der Hintergrund durch