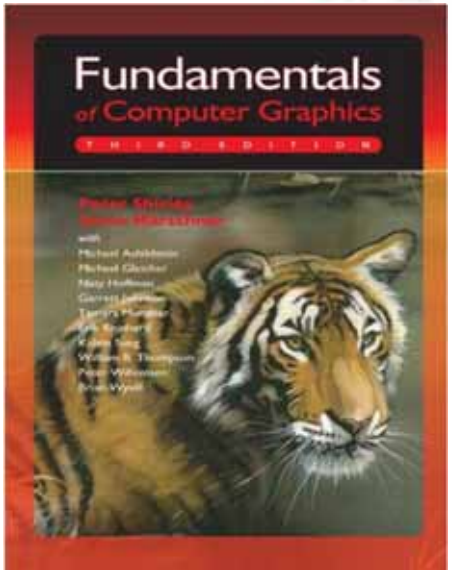
Computergrafik

Vorlesung im Wintersemester 2014/15
Kapitel 5: Räumliche Datenstrukturen

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

Literatur

- **Fundamentals of Computer Graphics**,
  P. Shirley, S. Marschner, 3rd Edition, AK Peters
  → Kapitel 12 (Data Structures for Graphics)
  → insb. Kapitel 12.3 (Spatial Data Structures)



Was haben wir bisher gelernt...

- Farbe, Bilder und ein wenig Perzeption
- Ray Tracing: Schnittpunktberechnung, Beleuchtungsberechnung, ...
- Texturierung

- und wie lange dauert es ein Bild mit diesem Dreiecksnetz zu berechnen?
- Path Tracing (ähnlich Distributed Ray Tracing)
  - 32× stochastisches Supersampling
  - 256^2 Pixel, 480000 Dreiecke
  - Brute Force: 176462sec ≈ 49h
  - nach diesem Kapitel: <80sec
    (BVH 18.3sec, Rendering 59sec)

Inhalt: Räumliche Datenstrukturen

- Analyse der Kosten bei Ray Tracing
- Ansätze zur Beschleunigung von Ray Tracing
- Bildraumverfahren
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - **Bounding Volume Hierarchies**
  - **reguläre und adaptive Gitter**
  - **kD-Bäume** und BSP-Bäume

- nebenbei: dieselben Datenstrukturen werden auch eingesetzt für
  - Kollisionserkennung und Bahnplanung in der Robotik
  - Szenengraphen, Culling und Physiksimulationen
  - als Suchbäume (z.B. Nächster Nachbar in (hochdim.) Räumen)

Ray Tracing Pseudocode

```
class Ray {
  vec3   origin, direction; // Startpunkt und Richtung des Strahls
  float  t;                 // Strahlparameter
  void * object;            // Zeiger auf evtl. getroffenes Objekt
  ...
};

for ( y = 0; y < height; y++ ) {
  for ( x = 0; x < width; x++ ) {

    generateRay( ray, x, y );

    color = raytrace( ray, ... );

  }
}
```

Ray Tracing Pseudocode

```
vec3 raytrace( Ray *ray, ... ) {
  vec3 color = 0.0f;

  if ( !cast( ray, FLOAT_MAX ) )
    return color;

  for ( jede Lichtquelle )
    color += computeDirectLight( ... );

  if ( Fläche ist spiegelnd ) {
    // berechne Reflexionsstrahl
    Ray reflect = ...;
    color += i.kr * raytrace( reflect, ... );
  }

  if ( Fläche ist transparent ) {
    // berechne Transmissionsstrahl
    Ray refract = ...;
    color += i.kt * raytrace( refract, ... );
  }

  return color;
}
```

Ray Tracing Pseudocode

```
class Ray {
  vec3   origin, direction; // Startpunkt und Richtung des Strahls
  float  t;                 // Strahlparameter
  void * object;            // Zeiger auf evtl. getroffenes Objekt
  ...
};

bool cast( Ray *ray, float maxDist )
{
  intersection = NULL;
  float t = maxDist;

  for ( each object ) {

    t' = intersect( object, ray->origin, ray->direction );

    if ( t' > 0 && t' < t ) {
      intersection = object;
      t = t';
    }
  }

  ray->t = t;
  ray->object = intersection;
  return intersection != NULL;
}
```

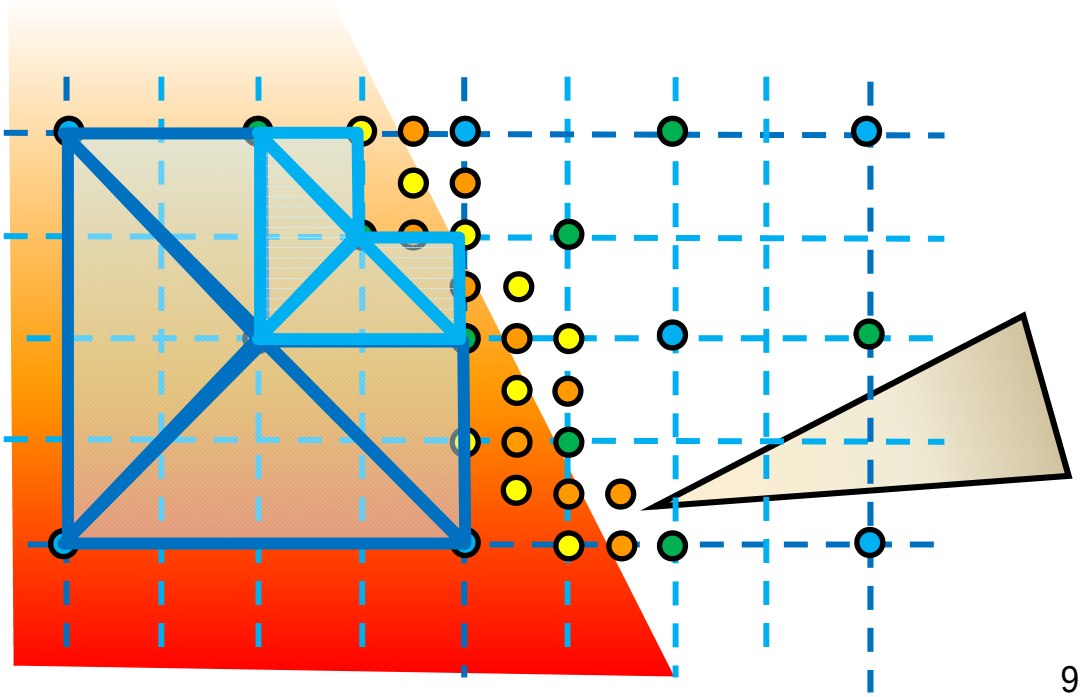
Optimierung des Ray Tracing

- der wesentlicher Teilalgorithmus von RT ist:
  - finde Strahlschnittpunkt mit nächstem Objekt der Szene
  - bisheriger Ansatz: Schnitt mit allen Objekten → >90% der Rechenzeit!
    - ...und es skaliert schlecht mit der Anzahl der Objekte
- Optimierungsansätze
  - schnellere Schnittalgorithmen? wenig Spielraum zur Optimierung
  - weniger Strahlen?
    - weniger Primär- und Sekundärstrahlen
    - Bsp. adaptives statt uniformes Supersampling, Tree Pruning
  - weniger Schnittberechnungen!
    - vermeide Berechnungen mit weit vom Strahl entfernten Objekten
    - Raumunterteilung, Richtungsunterteilung
  - Schnittpunkte mit „dicken“ Strahlen (Strahlenkegel)
    - Cone- und Beam-Tracing, behandeln wir nicht!

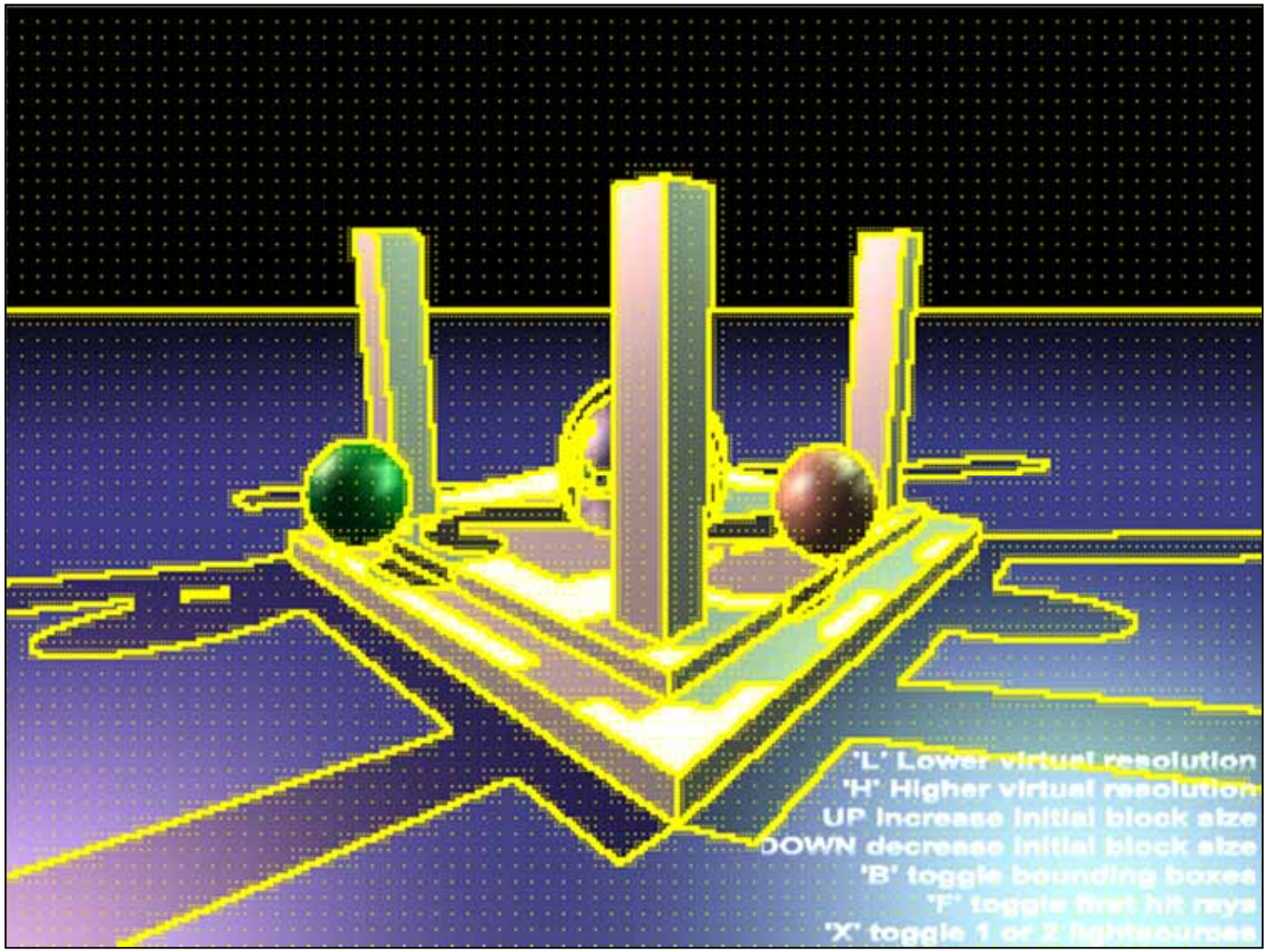
Optimierung: Weniger Strahlen

Weniger Primärstrahlen: Pixel-Selected-RT

- schieße Strahlen im groben Raster, z.B. Eckpunkte und Mittelpunkt eines 4×4 Pixel-Blocks
- vergleiche Eckpunkte mit Mittelpunkt
  - z.B. im Hinblick auf Farbe, Differenzen im Strahlenbaum, etc.
- falls große Differenzen: verfeinere
- falls ähnlich: fülle Bereich durch Interpolation der Farben der Eckpunkte
- Probleme
  - Verlust von Details: z.B. kleine Dreiecke oder kleine Glanzlichter
  - Unschärfe: Interpolation von Farben
  - Texturen: interpoliere Texturkoordinaten



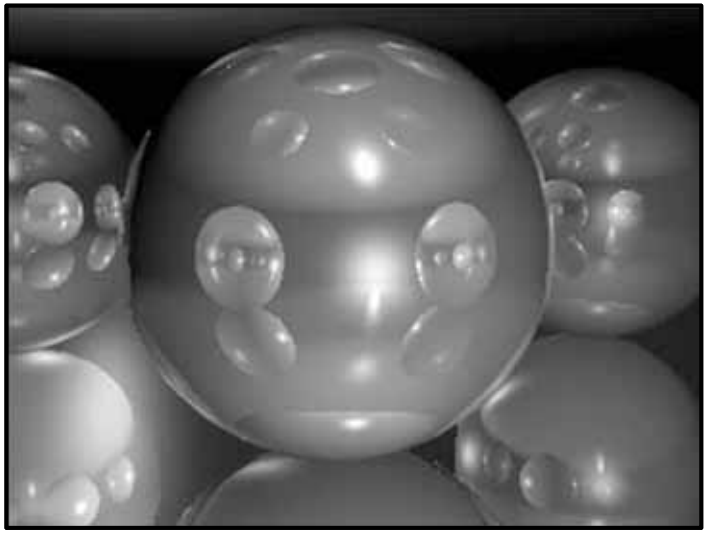
Optimierung: Weniger Strahlen



Optimierung: Weniger Strahlen

Weniger Sekundärstrahlen

- einfaches Tree-Pruning: gebe maximale Rekursionstiefe vor
- Adaptive Tree-Pruning
  - kleine Beiträge von Strahlen werden vernachlässigt
  - akkumuliere Abschwächung des Beitrags von mehrfach reflektierten/gebrochenen Strahlen
  - Produkt der Reflexions- bzw. Transmissionskoeffizienten
  - Abbruch der Rekursion, falls Beitrag des Strahls zu klein



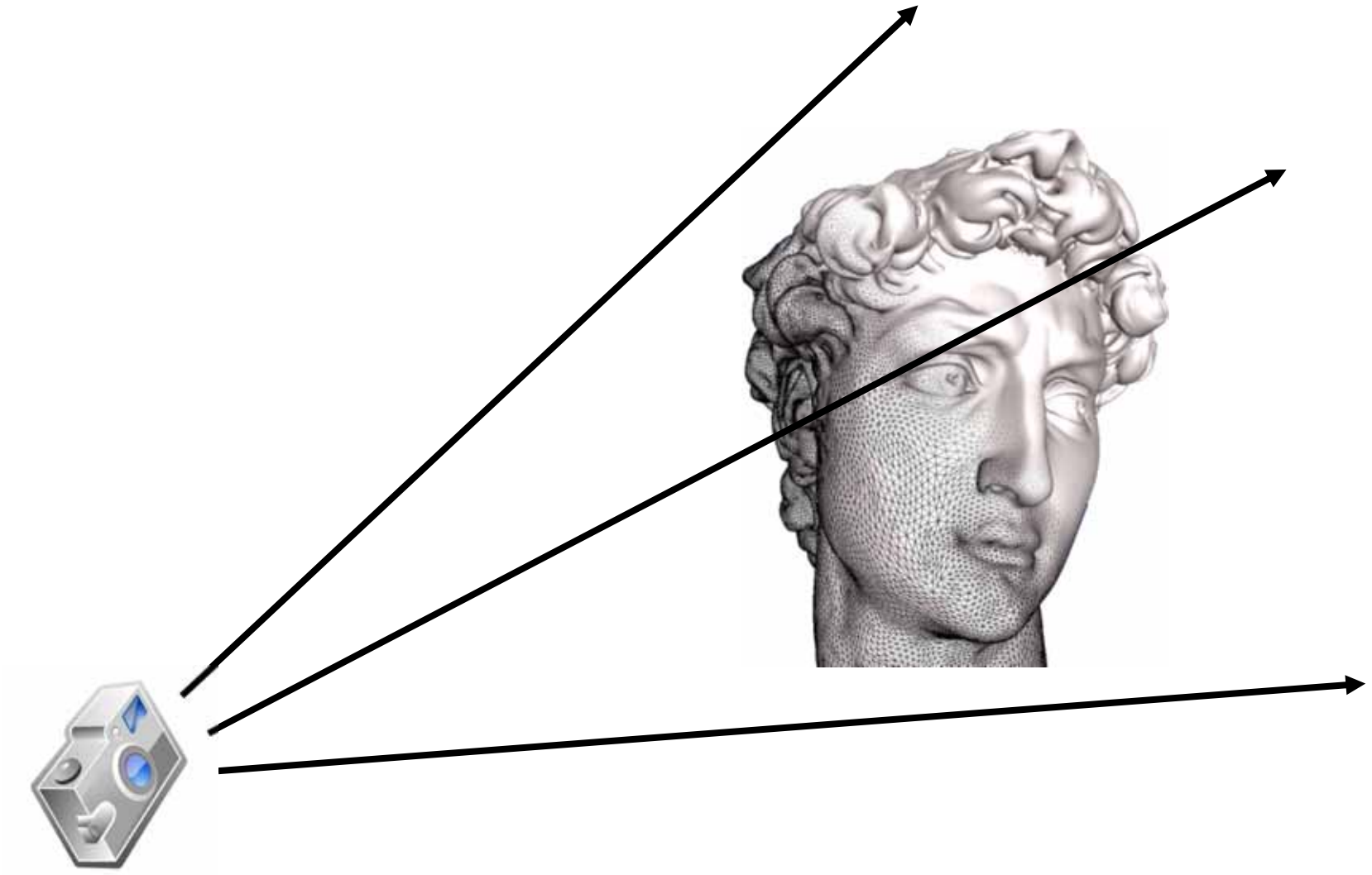
Optimierung: Weniger Schnittberechnungen

Raumunterteilung

- Idee/Ziel:
  - **finde potentiell geschnittene Objekte schneller**
  - **teste keine Objekte, die nicht in Frage kommen**
- Datenstrukturen zur **Raumunterteilung**
  - Bounding-Volume-Hierarchien (BVH)
  - Gitter, Hierarchien von Gittern
  - Oktalbäume (Octrees)
  - Binary-Space-Partition (BSP) Trees oder kD-Baum
  - Richtungsunterteilung
- wir gehen im Folgenden der Einfachheit halber davon aus, dass alle Objekte/Primitive in Weltkoordinaten platziert sind/sein können: eine Datenstruktur für alle Objekte

Beschleunigung des Ray Casting

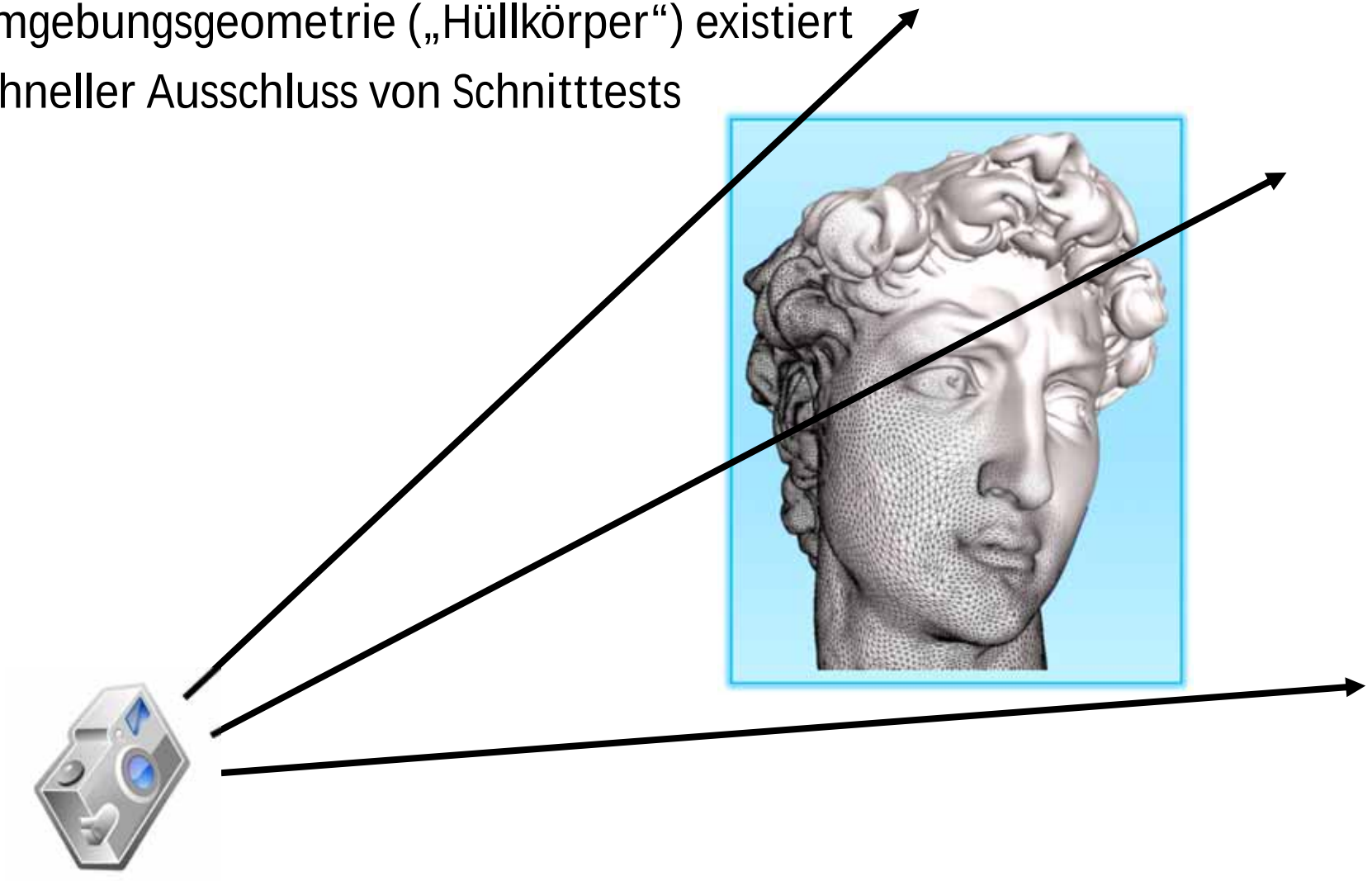
Reduziere die Anzahl der Strahl-Primitiv-Schnitttests



Beschleunigung des Ray Casting

Reduziere die Anzahl der Strahl-Primitiv-Schnitttests

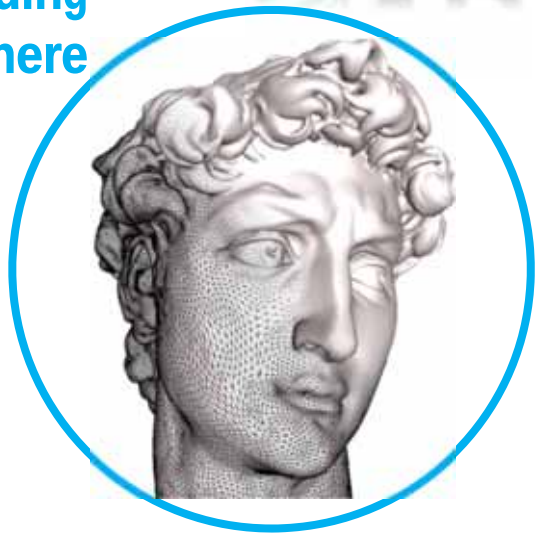
- Überprüfe zuerst, ob ein Schnitt mit einer (konservativ-großen) Umgebungsgeometrie („Hüllkörper“) existiert
- schneller Ausschluss von Schnitttests



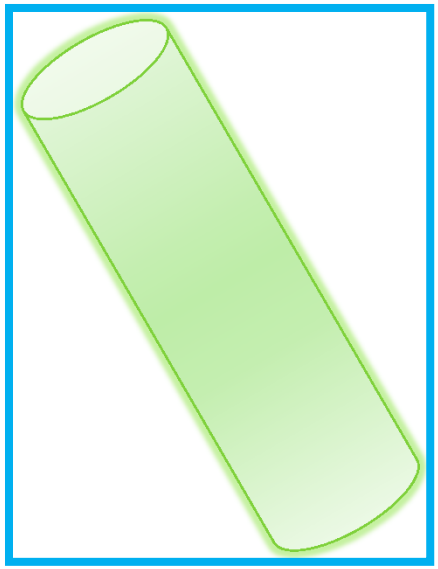
Konservative Hüllkörper

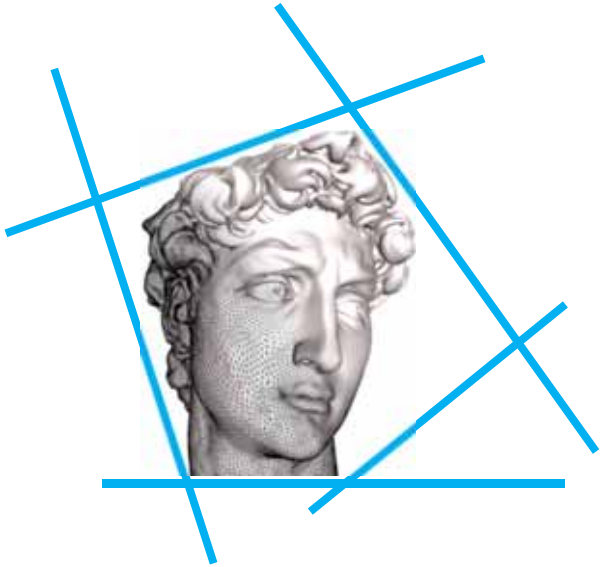
- Hüllkörper (engl. Bounding Volumes)
  - sollen möglichst enganliegend/klein sein, um wenige Falschpositive zu erzeugen
  - schnelle Schnittberechnung
- lohnt nur, wenn die enthaltene Geometrie entsprechend detailliert ist

Bounding Sphere

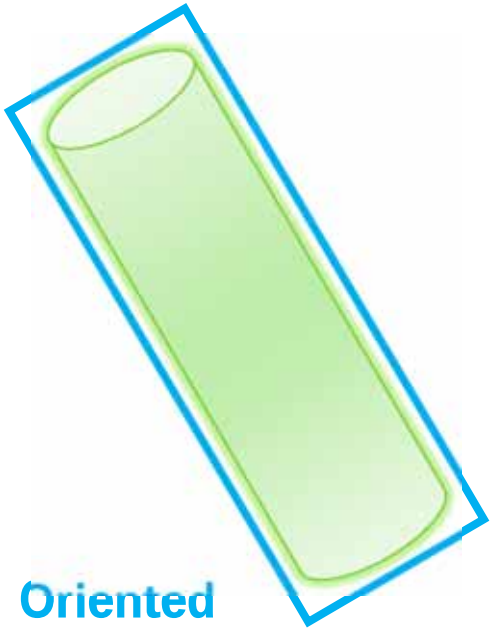


Axis-Aligned Bounding Box (AABB)





allgemeine konvexe Region (Kombination von begrenzenden Halbräumen und Spezialfall “Slabs”)

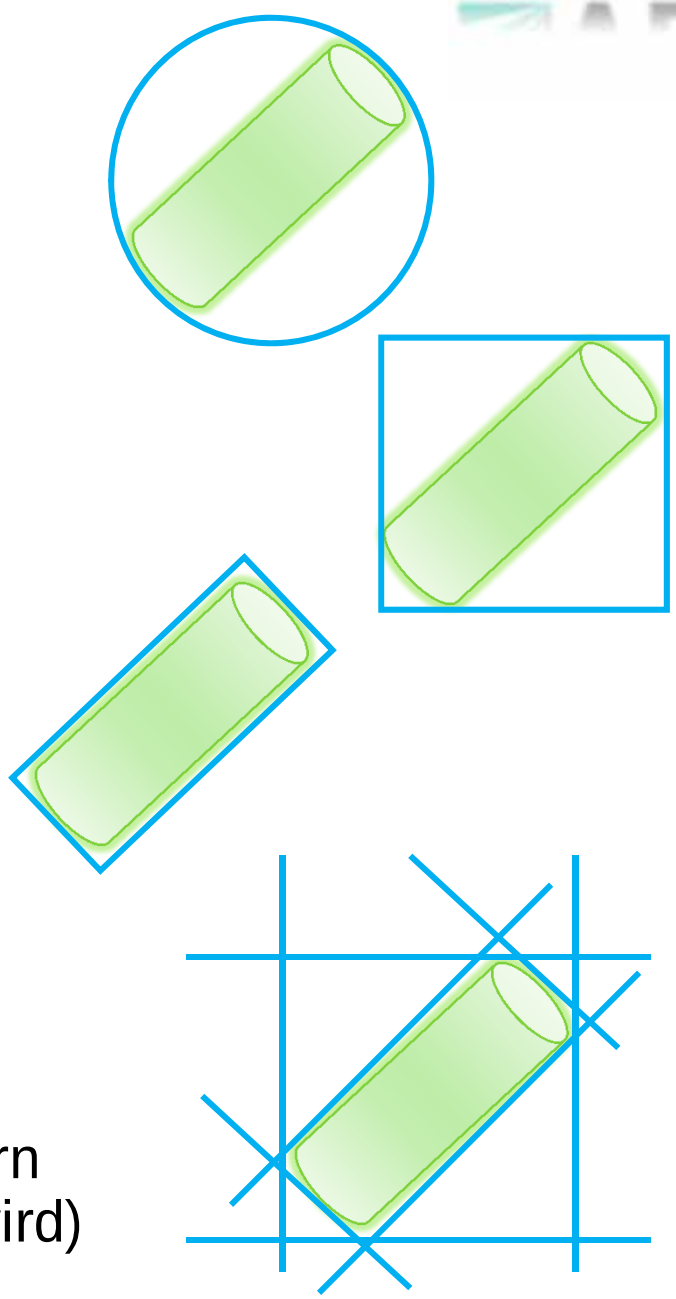


Oriented Bounding Box (OBB)

Optimierung: Bounding-Volumes

Hüllkörper, Bounding-Volumes

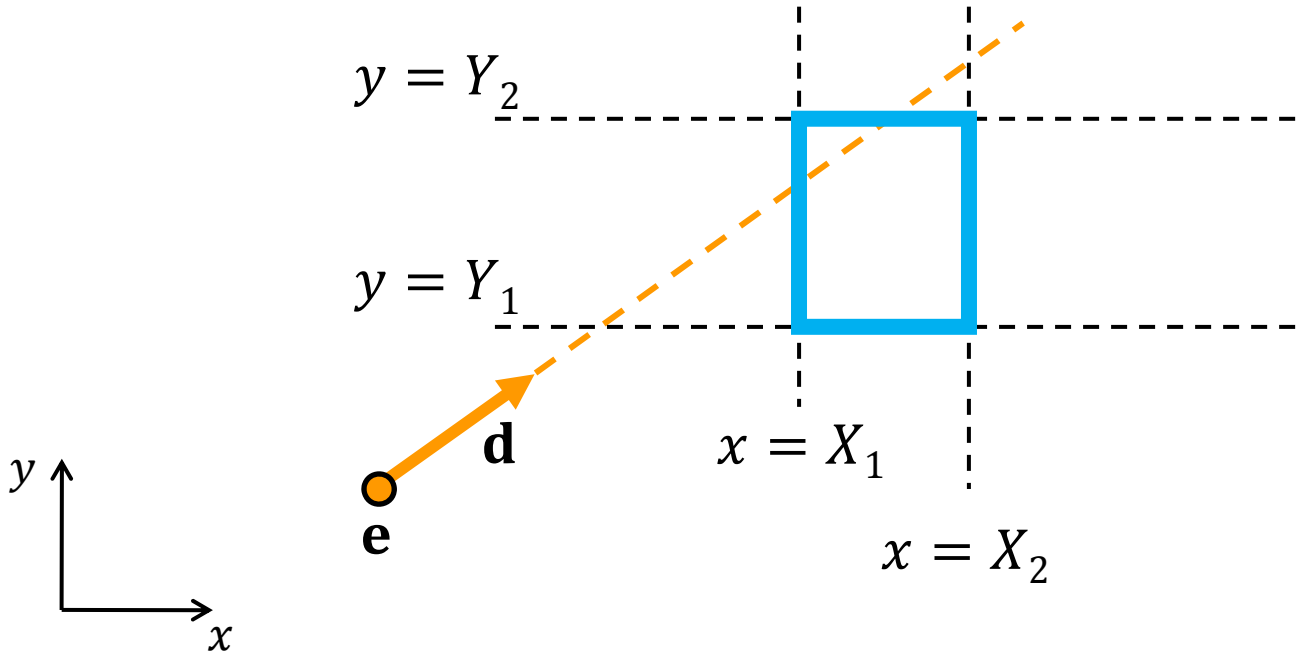
- Kugel
  - schneller Schnittalgorithmus
  - oft schlechte Effizienz, da zu groß
- Achsenparallele Box (AABB)
  - einfache Berechnung (min/max)
  - wichtigster Algorithmus
- Orientierte Bounding Box (OBB)
  - aufwändigere Berechnung (Hauptkomponentenanalyse der Menge von Eckpunkten)
- Slabs
  - Spezialfall: Schnitt von Paaren paralleler Halbebenen
  - gute Effizienz, schnelle Berechnung (sofern nicht die global beste Lösung gefordert wird)



Bounding Volume: AABB

Schnittberechnung Strahl-AABB

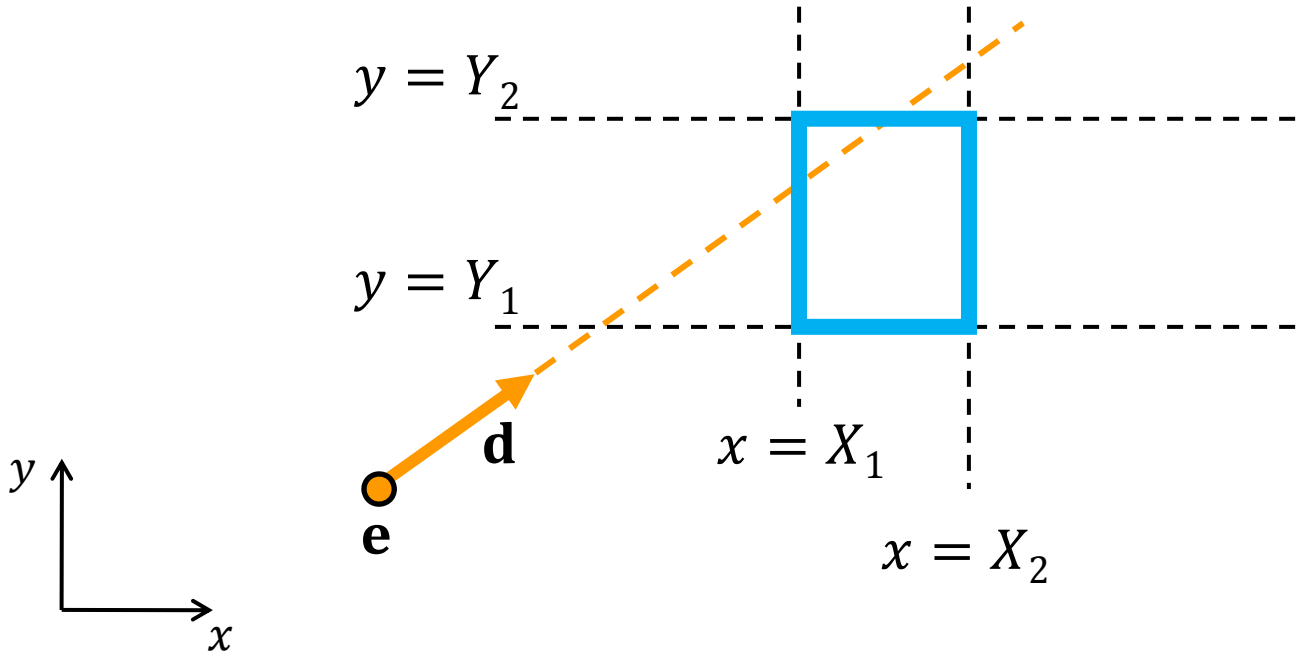
- AABB: $(X_1, Y_1, Z_1) \rightarrow (X_2, Y_2, Z_2)$
- Strahl: $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$
- naiver Ansatz:
  - 6 Ebenengleichungen: berechne alle Schnittpunkte
  - bestimme nahsten Schnittpunkt *innerhalb* der Box



Bounding Volume: AABB

Optimierungen

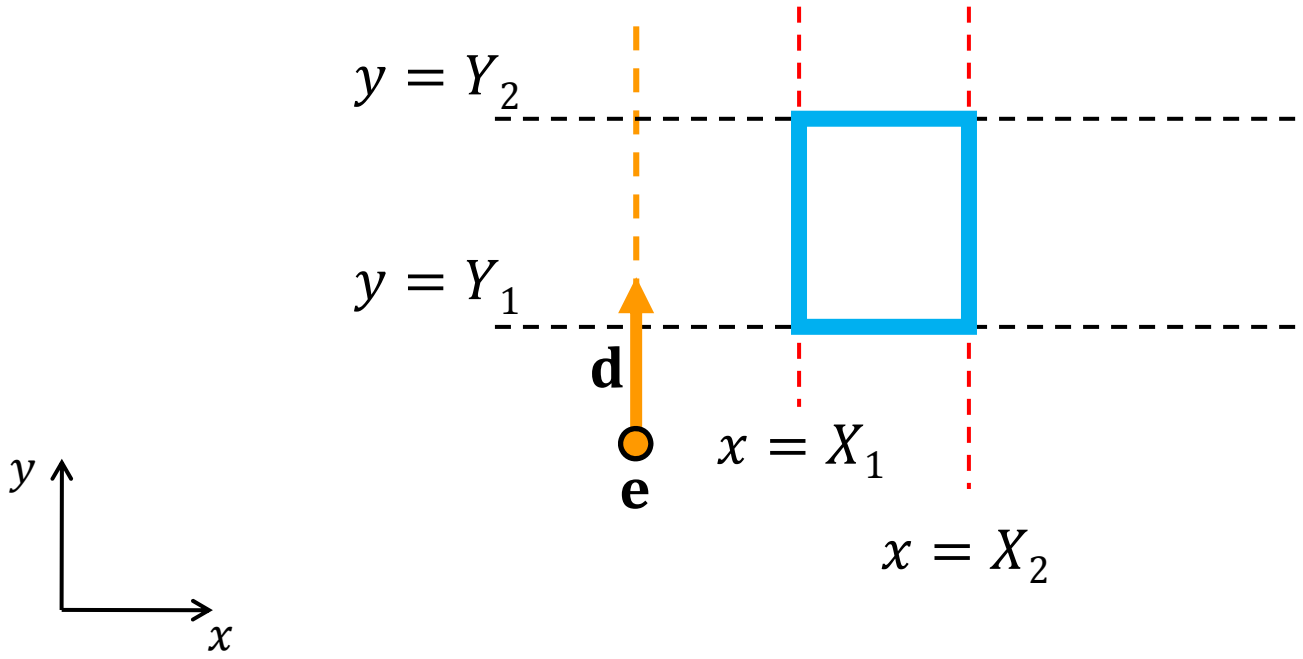
- der Schnitttest Strahl-AABB wird i.d.R. sehr oft benötigt
  → hier lohnt es sich zu optimieren!
- führe Berechnungen für jede Dimension jeweils zusammen aus
  - jeweils 2 Ebenen besitzen dieselbe Normale
  - Normalen sind achsenparallel: nur eine Komponente ungleich 0



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: Test, ob Strahl und Box parallel

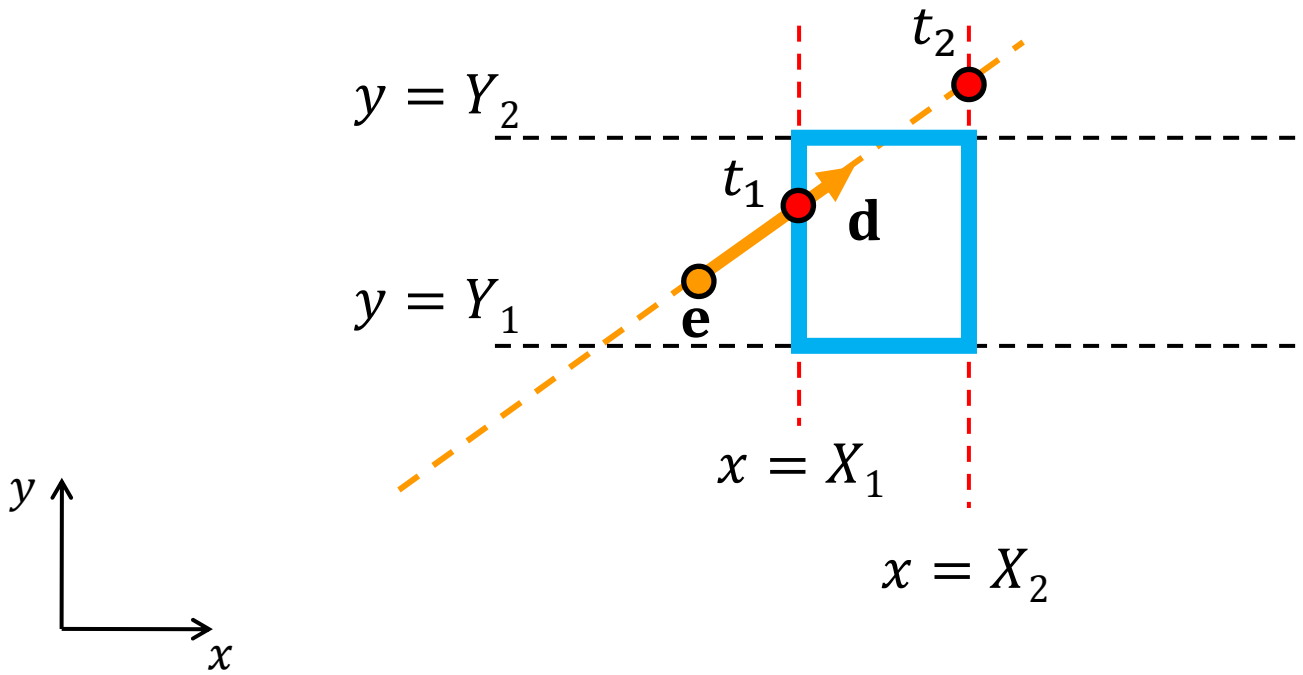
- wenn $d_x = 0$ (Strahl parallel zu yz-Ebene) $\wedge$ $(e_x < X_1 \vee e_x > X_2)$
  $\Rightarrow$ kein Schnitt
- analog für d_y und d_z



Bounding Volume: AABB

Schnittberechnung für jede Dimension

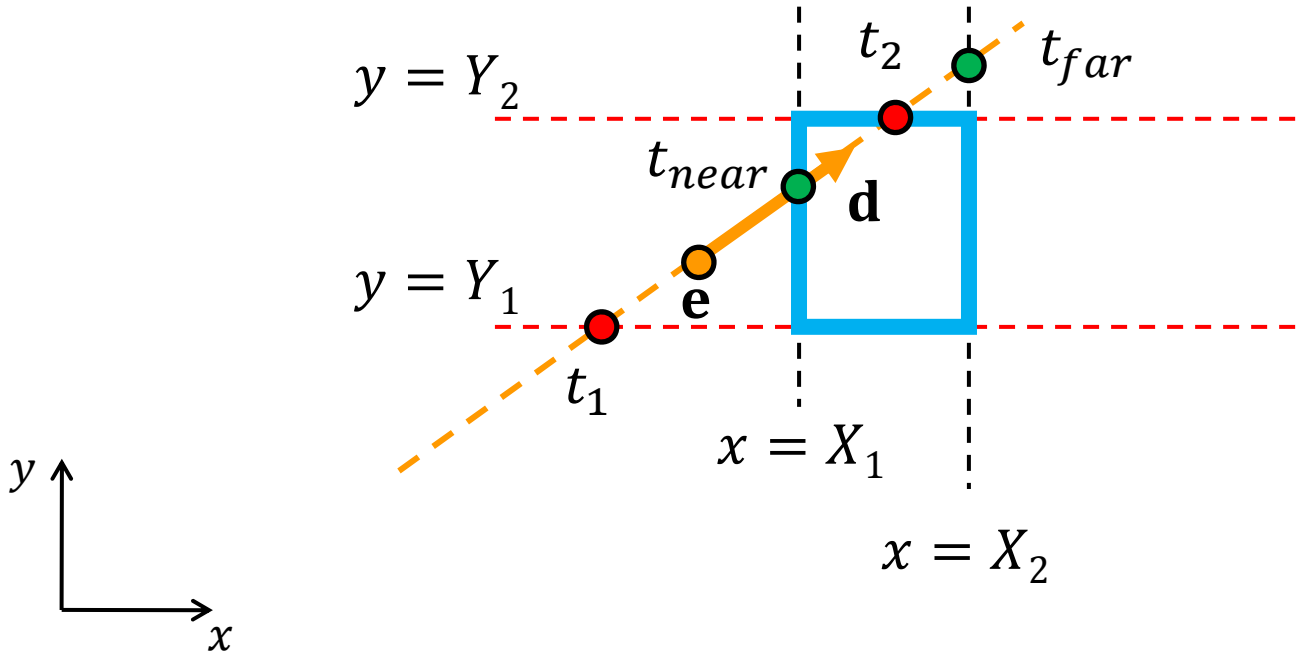
- berechne Entfernung zu den Ebenenschnitten t_1 und t_2
  - $t_1 = (X_1 - e_x)/d_x$
  - $t_2 = (X_2 - e_x)/d_x$
  - diese Berechnung ist analog zum Strahl-Ebenen Schnitt (beachte Normalen entlang der Achsen), allgemein: $t = \frac{d - \mathbf{e}\cdot\mathbf{n}}{\mathbf{d}\cdot\mathbf{n}}$



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: t_{near} und t_{far} bestimmen

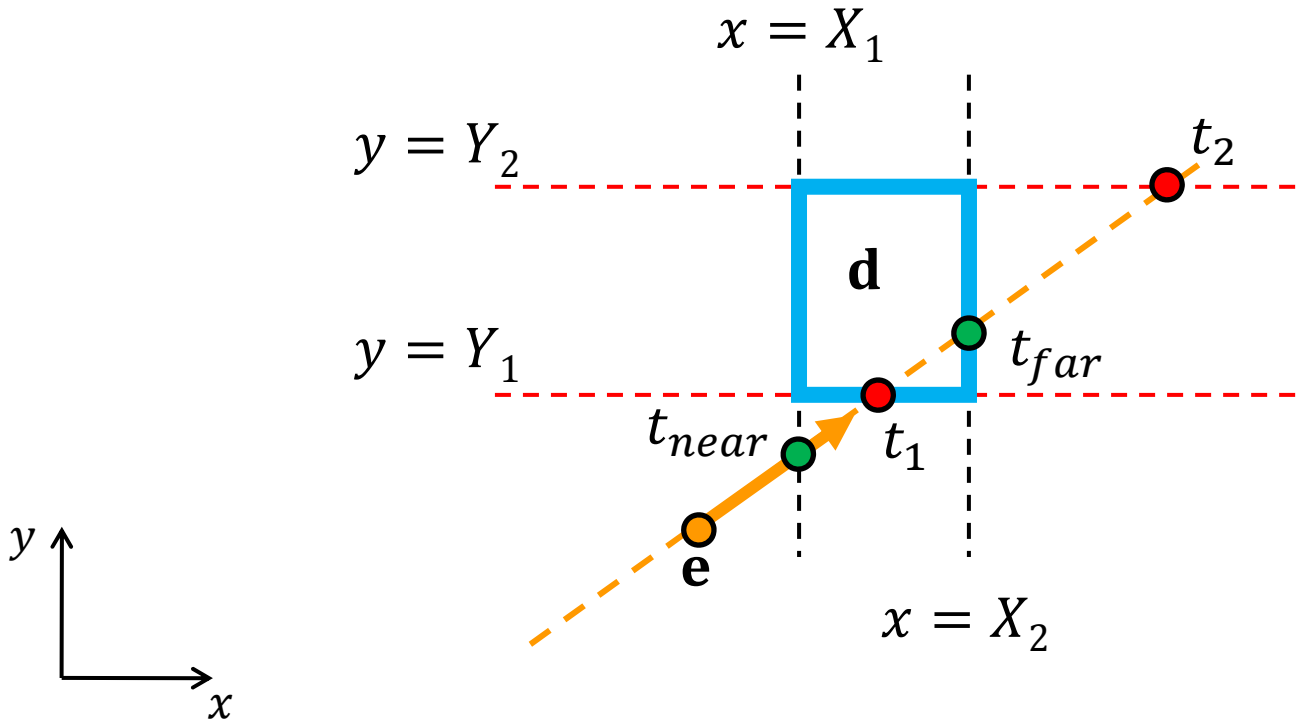
- Berechnung der Schnitte mit den weiteren Ebenen und Mitführen des nahsten und entferntesten Schnitts *mit der AABB*
  - Annahme $t_1 < t_2$
  - wenn $t_1 > t_{near}$, dann $t_{near} = t_1$ (hier nicht der Fall)
  - wenn $t_2 < t_{far}$, dann $t_{far} = t_2$



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: t_{near} und t_{far} bestimmen

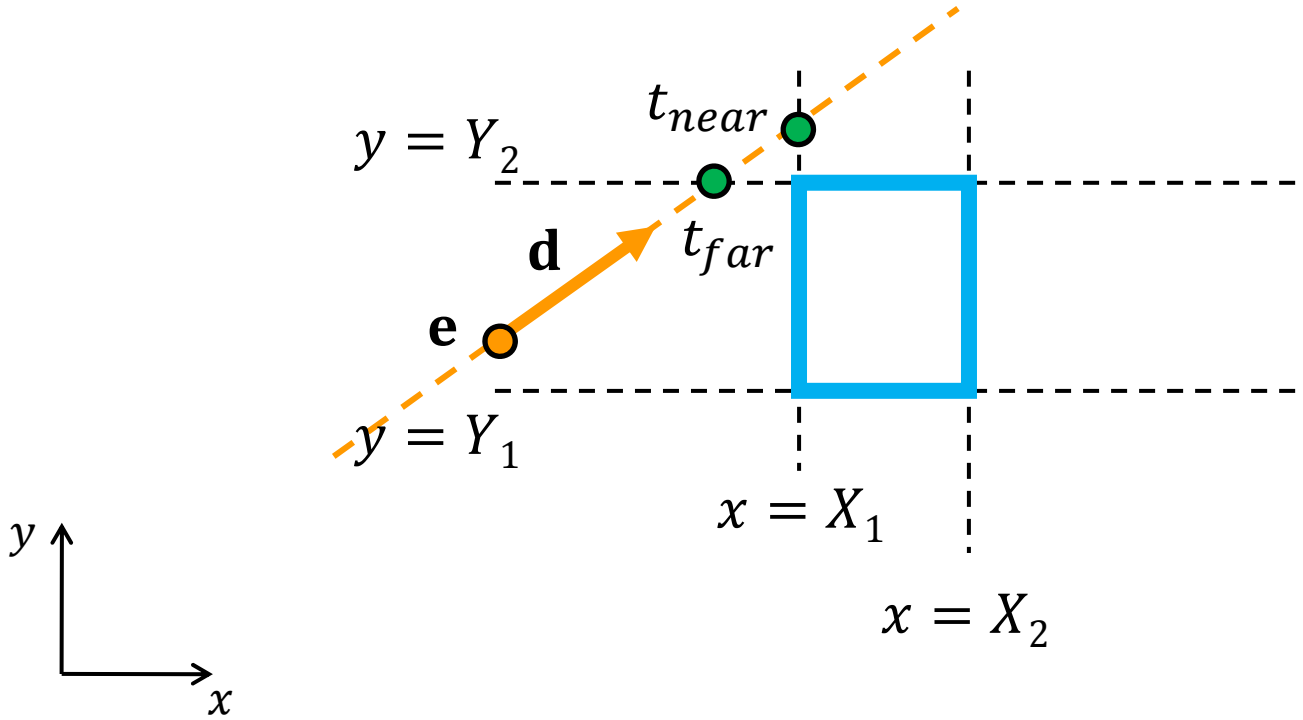
- Berechnung der Schnitte mit den weiteren Ebenen und Mitführen des nahsten und entferntesten Schnitts *mit der AABB*
  - wenn $t_1 > t_{near}$, dann $t_{near} = t_1$
  - wenn $t_2 < t_{far}$, dann $t_{far} = t_2$ (hier nicht der Fall)



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: existiert ein Schnittpunkt?

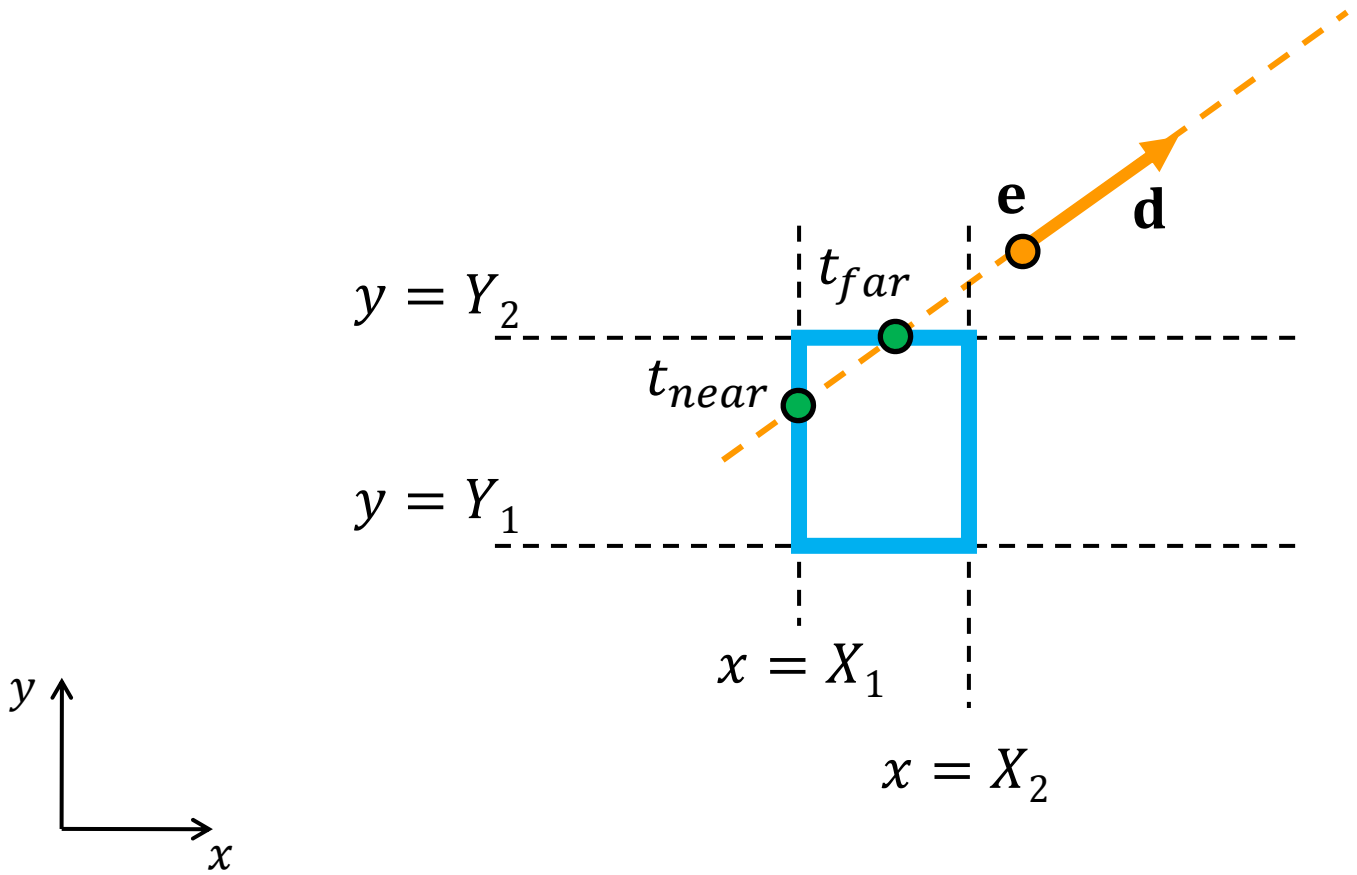
- wenn $t_{near} > t_{far}$ wird die Box nicht getroffen



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: Box hinter dem Strahlursprung?

- wenn $t_{far} < 0$ ist die Box hinter dem Strahl

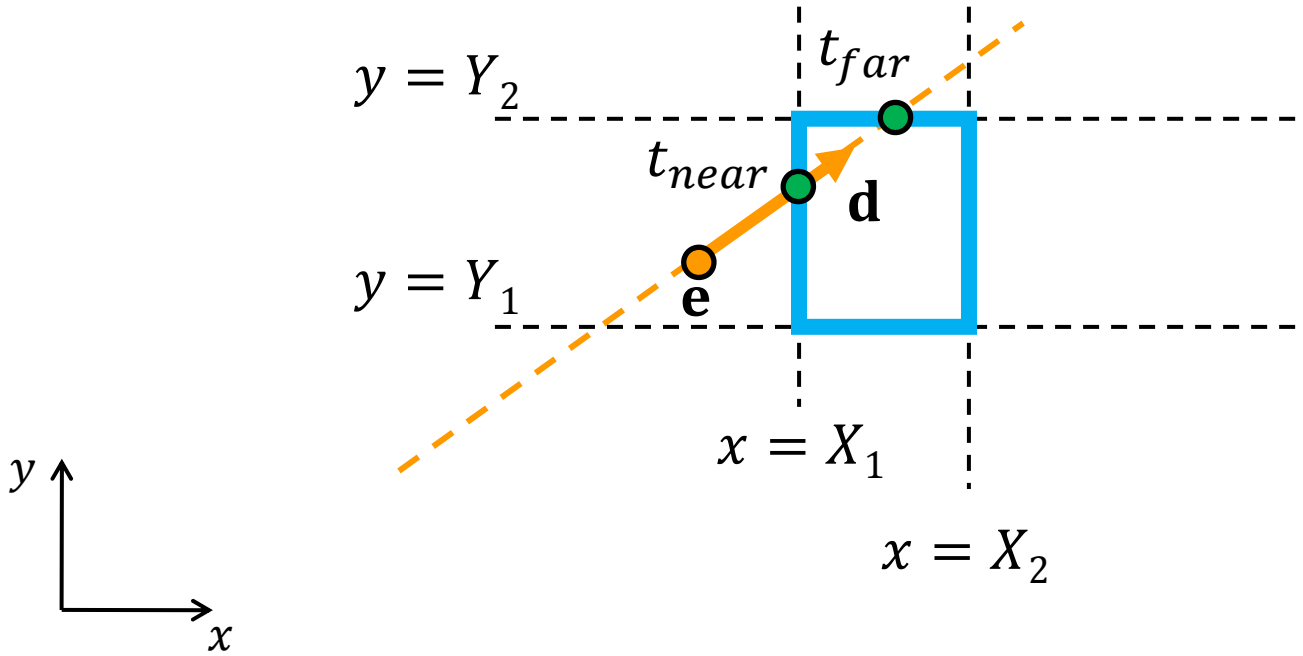


Bounding Volume: AABB

Richtigen Schnittpunkt zurückliefern

- wenn $t_{near} > 0 \rightarrow$ nahster Schnittpunkt bei t_{near}
- sonst: nahster Schnittpunkt bei t_{far} (Strahl beginnt innerhalb der AABB)

- wir sind zwar gerade nur daran interessiert zu testen, **ob** es einen Schnitt mit der AABB gibt, aber das Strahlsegment brauchen wir später noch!



Bounding Volume: AABB

Zusammenfassung Schnittberechnung Strahl-AABB

- für jede Dimension (hier für x gezeigt):
  wenn $d_x = 0$ (Strahl parallel zu yz-Ebene) $\wedge$ $(e_x < X_1 \vee e_x > X_2)$
  $\Rightarrow$ kein Schnitt

- für jede Dimension: berechne t_1 und t_2
  - $t_1 = (X_1 - e_x)/d_x$ $\qquad$ $t_2 = (X_2 - e_x)/d_x$
  - wenn $t_1 > t_2$ tausche t_1 und t_2 (benötigt wenn $d_x, d_y, d_z < 0$)
  - aktualisiere t_{near} und t_{far} (nahster und entferntester Schnitt bis zu diesem Zeitpunkt)
    - wenn $t_1 > t_{near}$, dann $t_{near} = t_1$
    - wenn $t_2 < t_{far}$, dann $t_{far} = t_2$

- wenn $t_{near} > t_{far}$ $\rightarrow$ Box nicht getroffen
- wenn $t_{far} < 0$ $\rightarrow$ Box hinter dem Strahl
- wenn $t_{near} > 0$ $\rightarrow$ nahster Schnitt bei t_{near}
- sonst $\rightarrow$ nahster Schnitt bei t_{far}

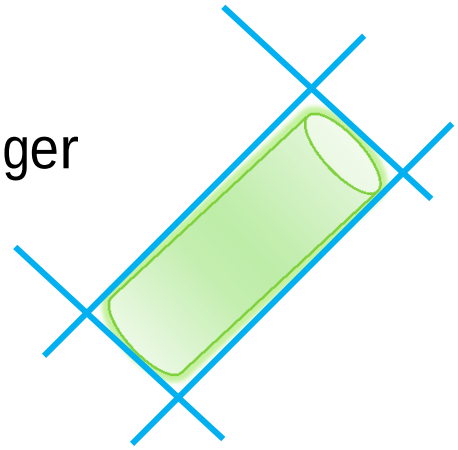
Bounding Volume: AABB

Einfache Optimierungen

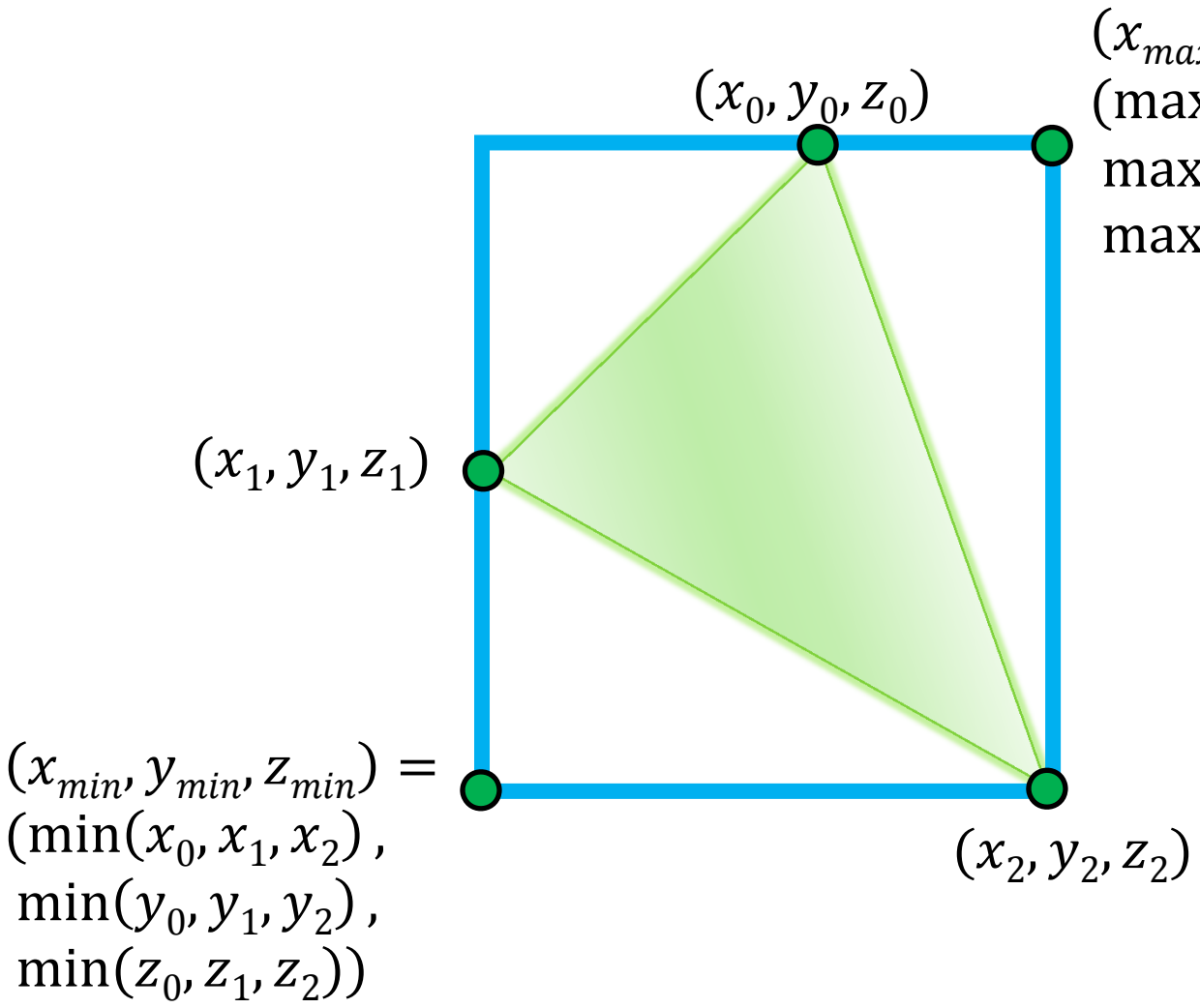
- $1/d_x$, $1/d_y$ und $1/d_z$ können einmal pro Strahl berechnet werden und für alle AABB-Tests verwendet werden

- Loop-Unrolling
  - Schleifen sind teuer
  - vermeide t_{near} und t_{far} Vergleich für die erste Dimension
  - SIMD: teste mehrere Strahlen/AABBs parallel

- Verallgemeinerung des Konzepts auf Slabs möglich
  - parallele Ebenenpaare
  - konvexe Form des Hüllkörpers
  - nur Berechnung von t_1 und t_2 ist etwas aufwändiger (wegen der beliebigen Orientierung der Ebenen)

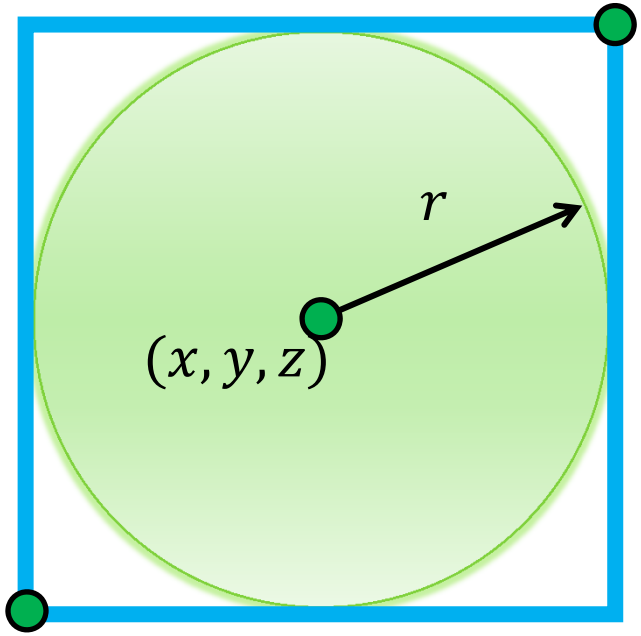


Bounding Box eines Dreiecks



Bounding Box einer Kugel

$(x_{max}, y_{max}, z_{max}) =$
$(x + r, y + r, z + r)$



$(x_{min}, y_{min}, z_{min}) =$
$(x - r, y - r, z - r)$

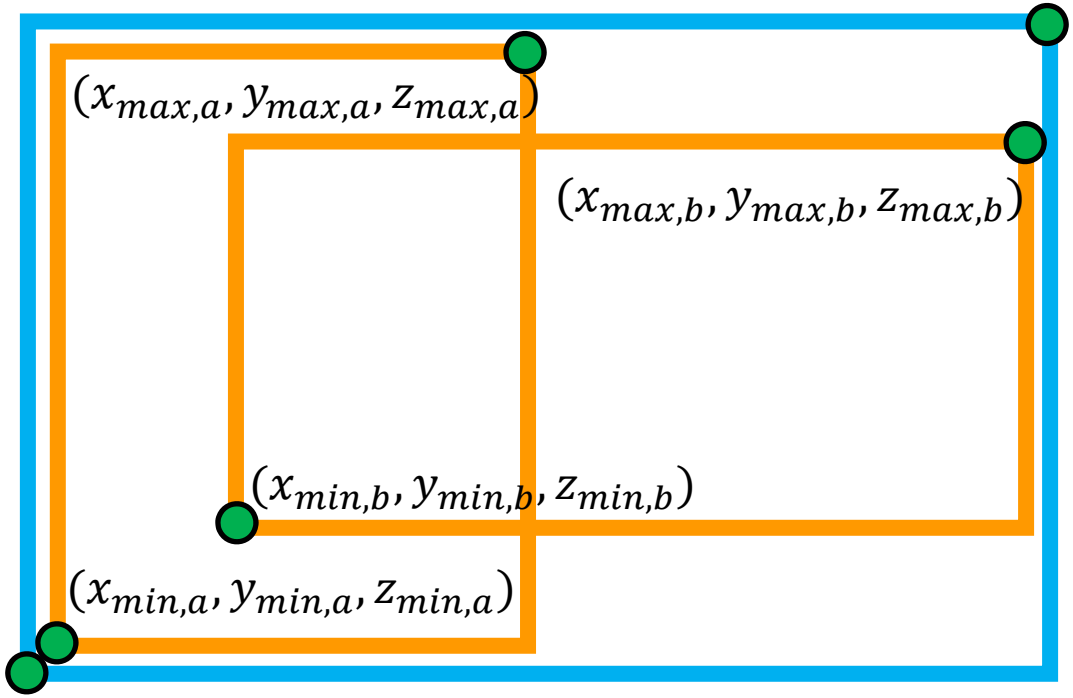
Bounding Box einer Ebene

- ...für eine Ebene die nicht senkrecht zu einer der Achsen ist

$(x_{max}, y_{max}, z_{max}) =$
$(+\infty, +\infty, +\infty)$

$(x_{min}, y_{min}, z_{min}) =$
$(-\infty, -\infty, -\infty)$

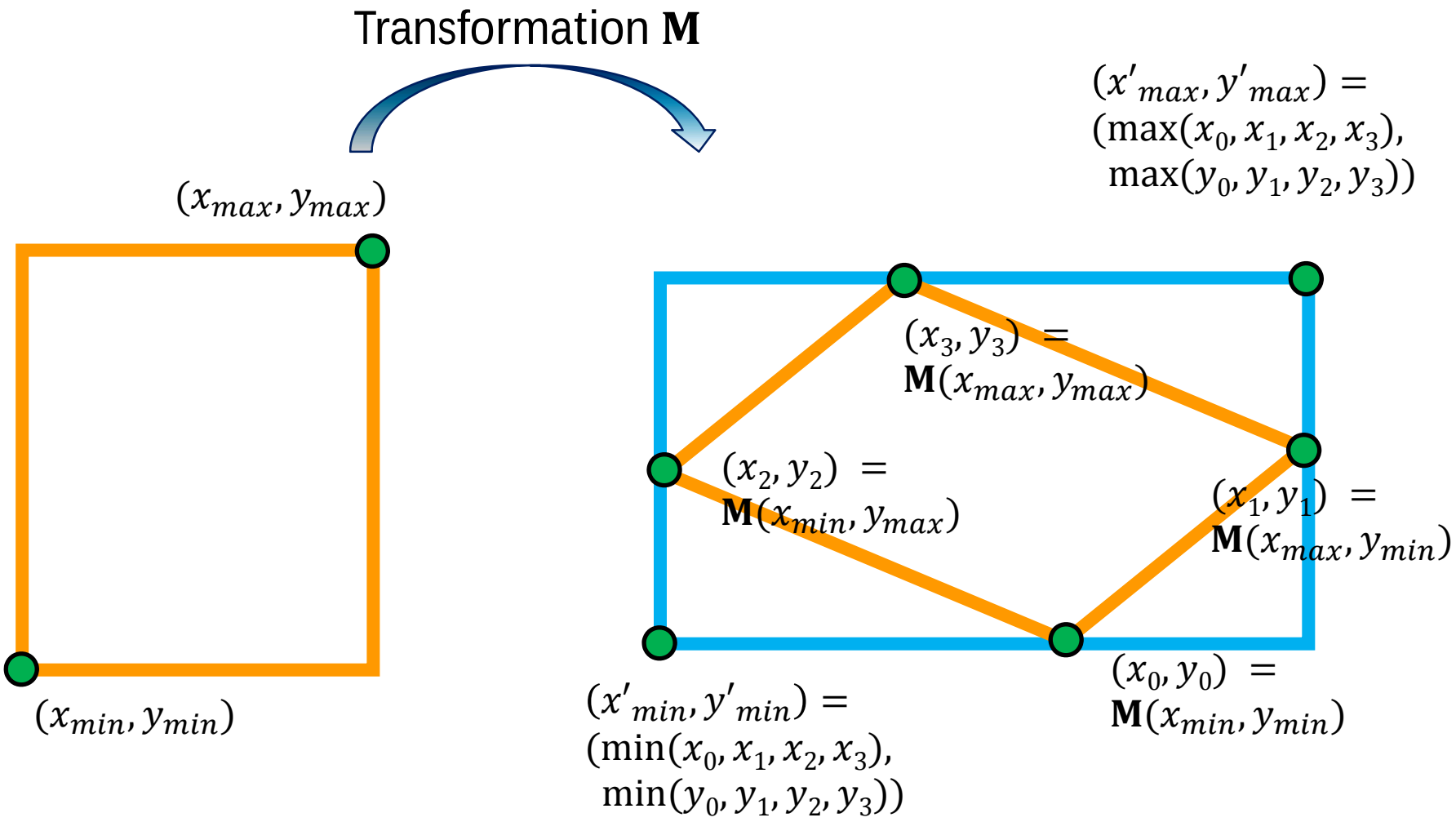
Bounding Box einer Gruppe von AABBs



$$(x_{max}, y_{max}, z_{max}) = (\max(x_{max,a}, x_{max,b}), \max(y_{max,a}, y_{max,b}), \max(z_{max,a}, z_{max,b}))$$

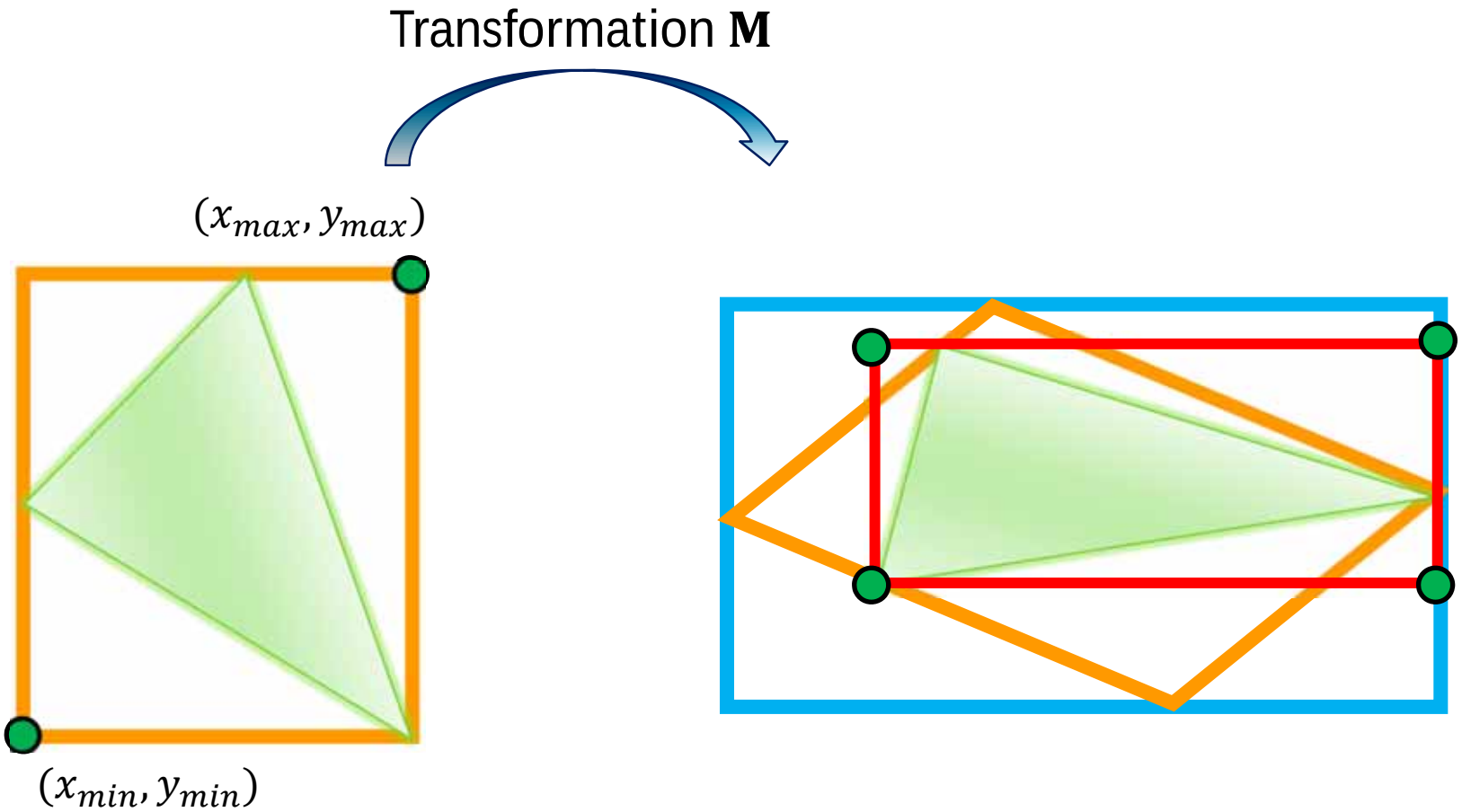
$$(x_{min}, y_{min}, z_{min}) = (\min(x_{min,a}, x_{min,b}), \min(y_{min,a}, y_{min,b}), \min(z_{min,a}, z_{min,b}))$$

Bounding Box einer transformierten AABB



Spezialfall: Dreieck

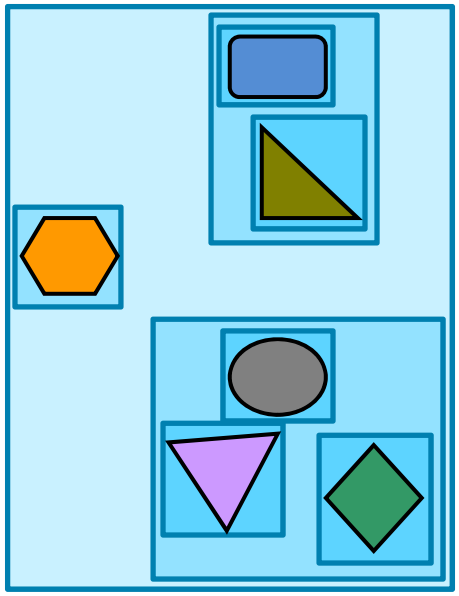
- kleinere AABB durch Transformation der Eckpunkte des Dreiecks(netzes) und Neubestimmung der Minima und Maxima
- erhöhte Kosten für Bestimmung der AABB, aber i.d.R. deutlich kleinerer Hüllkörper und somit weniger Falschpositive

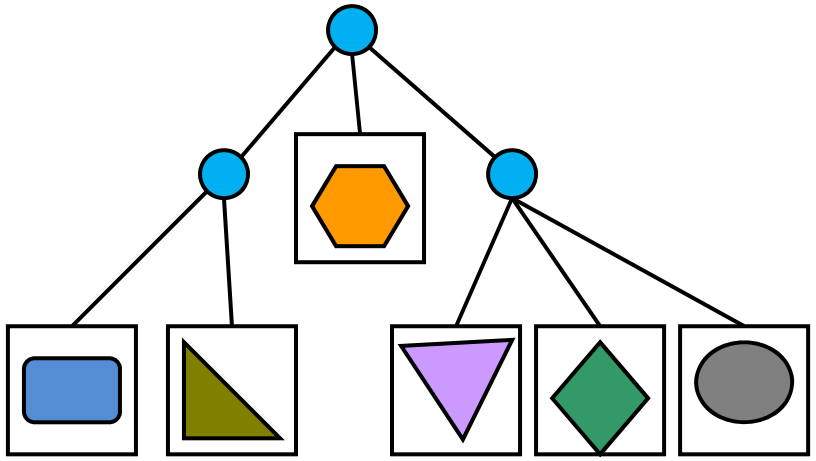


Optimierung: Bounding-Volume-Hierarchien

Grundidee

- Hierarchie von einschließenden Hüllkörpern
- automatisches Zusammenfassen von (Gruppen von) Objekten/Primitiven
- Vorteile:
  - Adaptivität der Raumunterteilung
  - schnellere Suche: Objektgruppen können ausgeschlossen werden
- Bsp. Hierarchie mit Hüllquadern

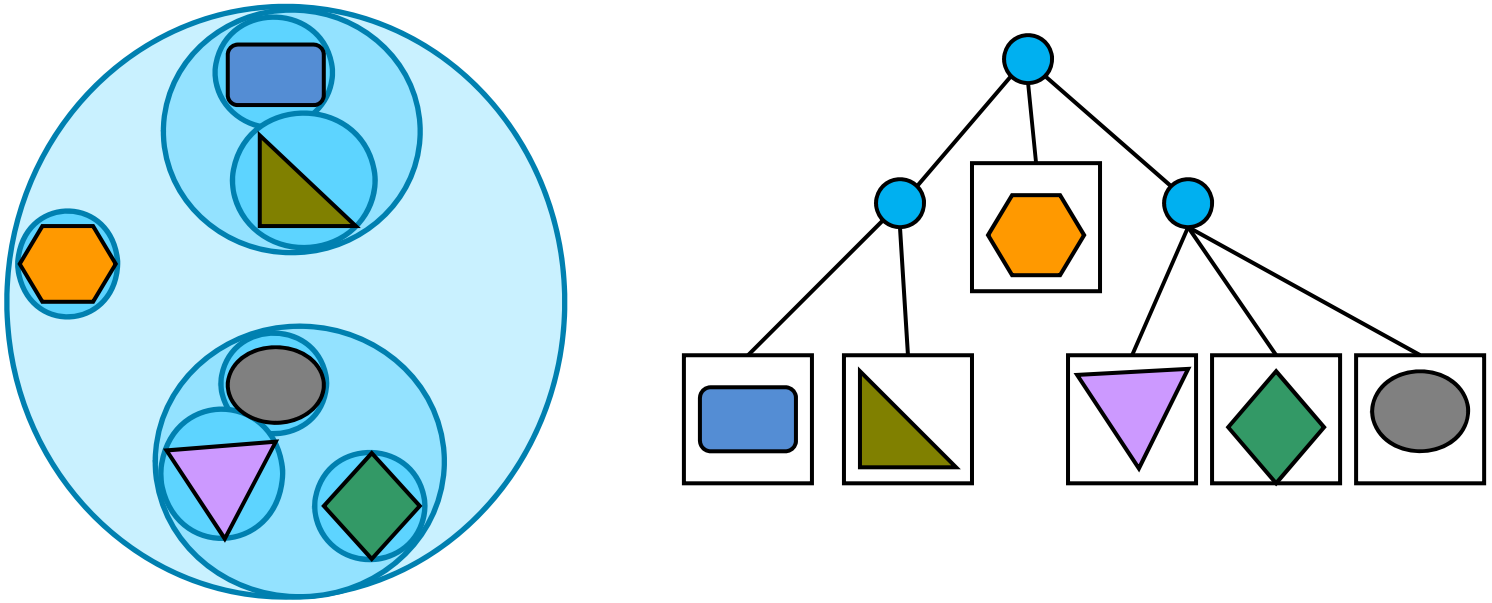




Optimierung: Bounding-Volume-Hierarchien

Grundidee

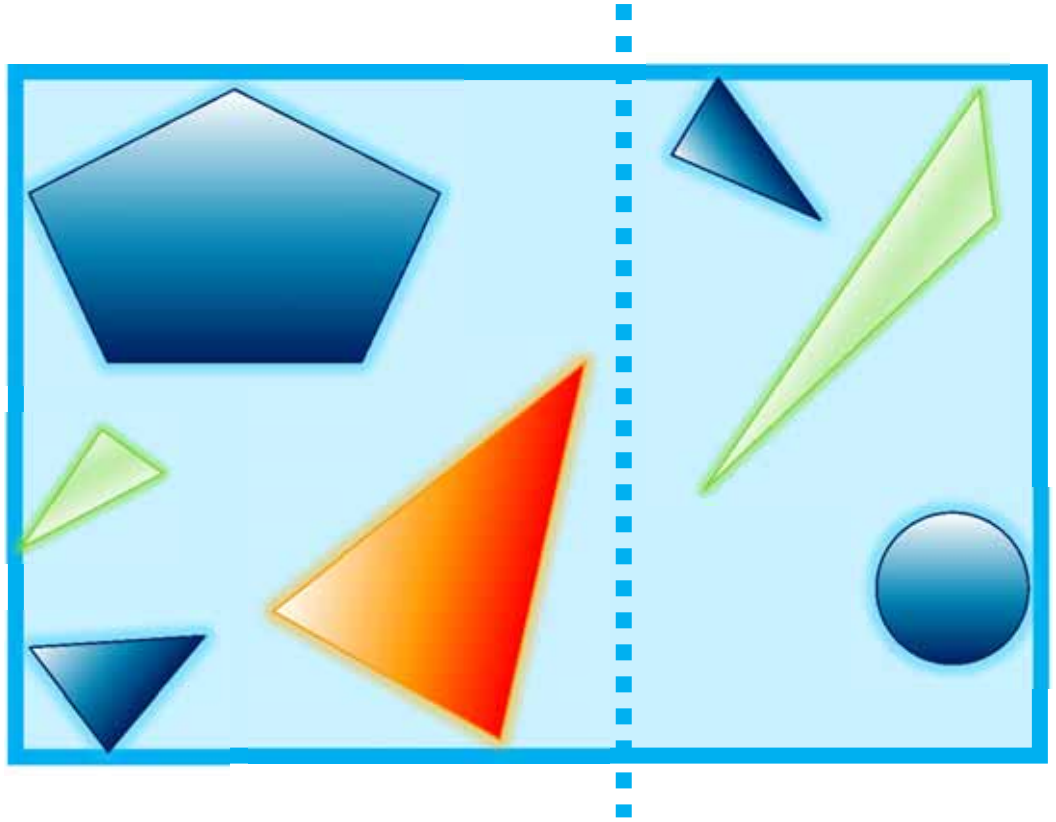
- Hierarchie von einschließenden Hüllkörpern
- automatisches Zusammenfassen von (Gruppen von) Objekten/Primitiven
- Vorteile:
  - Adaptivität der Raumunterteilung
  - schnellere Suche: Objektgruppen können ausgeschlossen werden
- Bsp. Hierarchie mit Hüllkugeln (im Folgenden verwenden wir nur AABBs)



Bounding Volume Hierarchie

Aufbau der Hierarchie

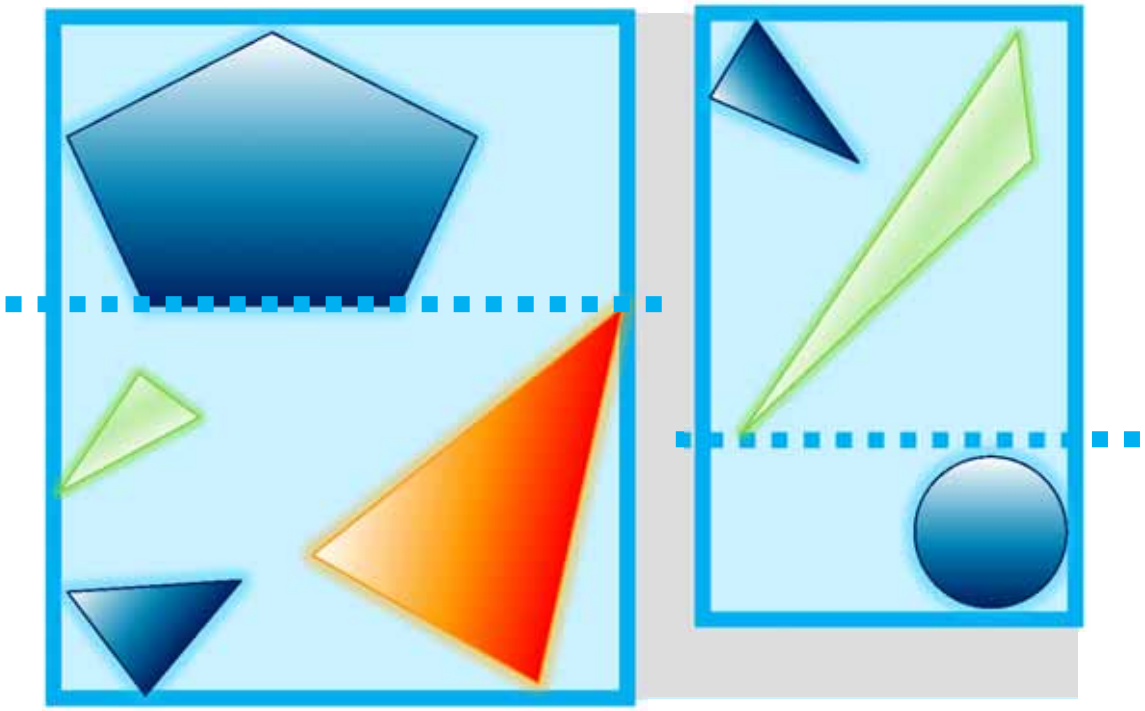
- bestimme die gemeinsame Bounding Box aller Objekte
- teile die Objekte in zwei Gruppen auf



Bounding Volume Hierarchie

Aufbau der Hierarchie

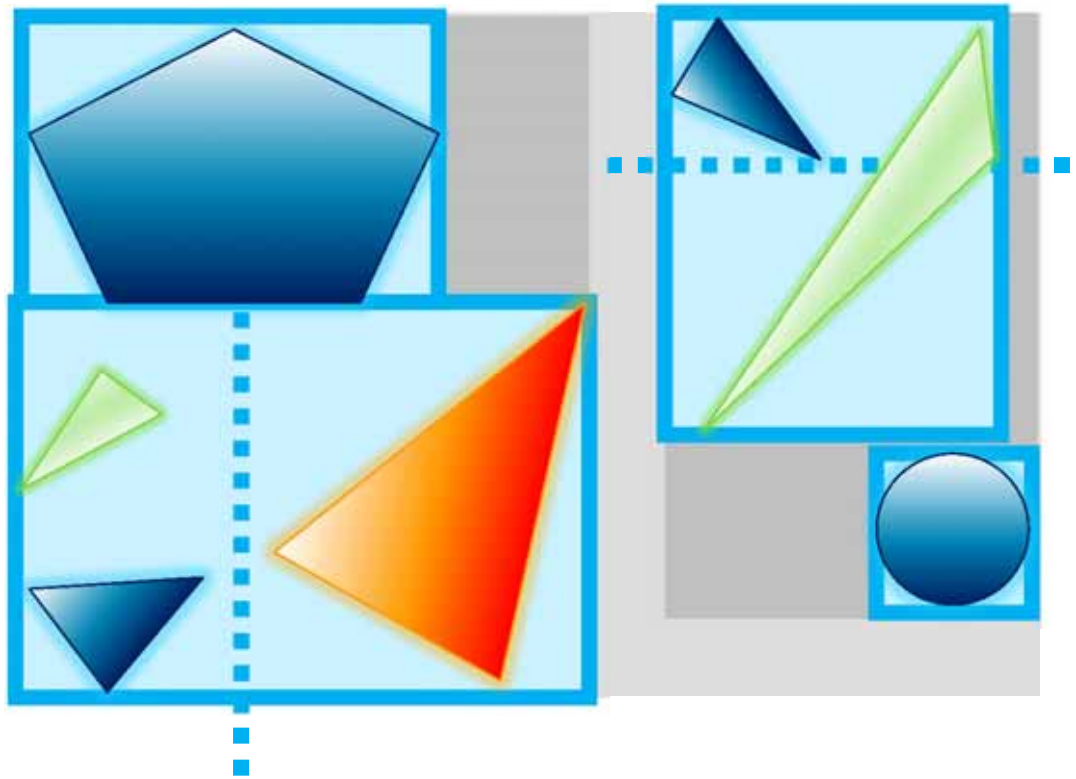
- bestimme die Bounding Box der Objekte jeder Gruppe
- teile die Objekte jeweils wieder in zwei Gruppen auf
- verfahre so rekursiv weiter



Bounding Volume Hierarchie

Aufbau der Hierarchie

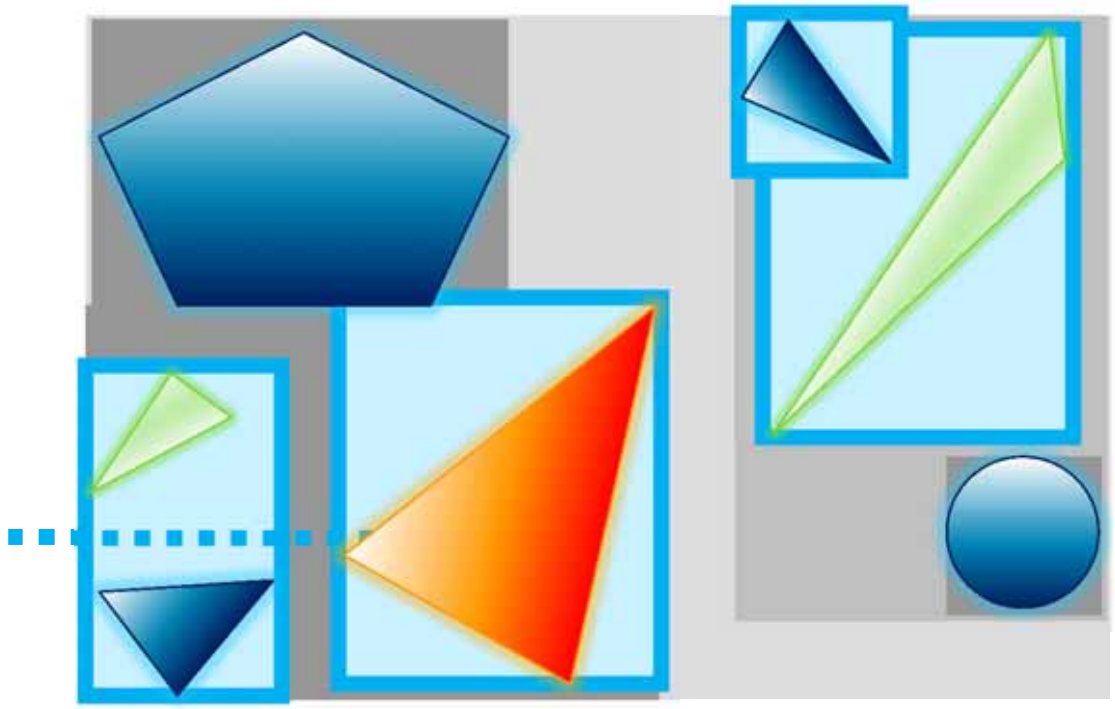
- bestimme die Bounding Box der Objekte
- teile die Objekte jeweils wieder in zwei Gruppen auf
- verfahre so rekursiv weiter



Bounding Volume Hierarchie

Aufbau der Hierarchie

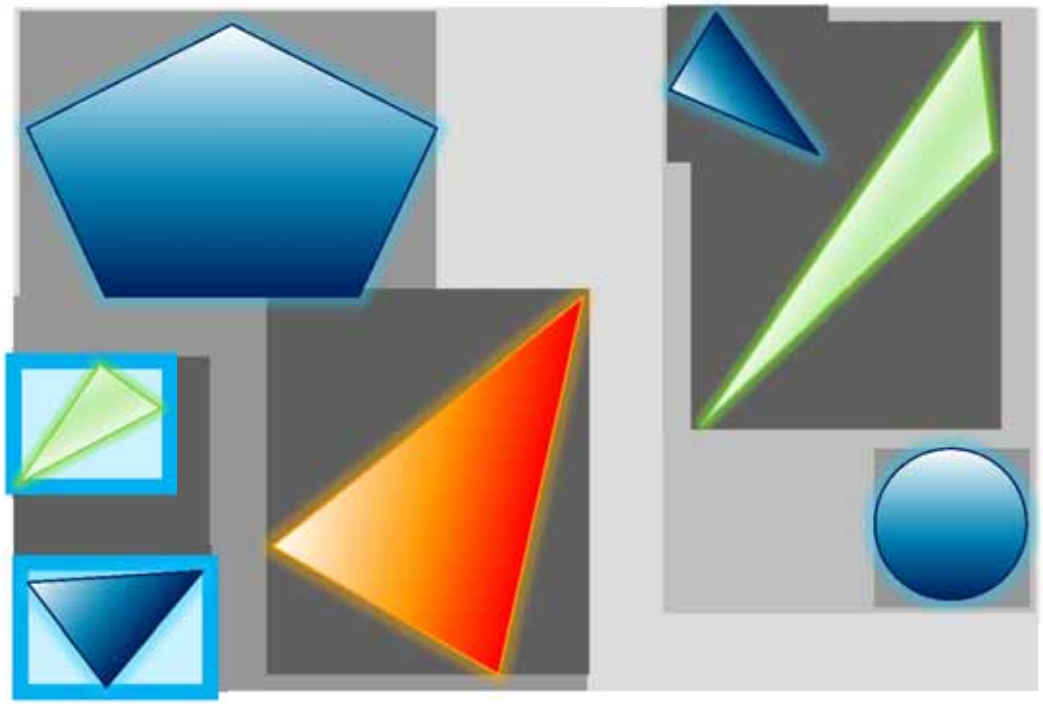
- bestimme die Bounding Box der Objekte
- teile die Objekte jeweils wieder in zwei Gruppen auf
- verfahre so rekursiv weiter



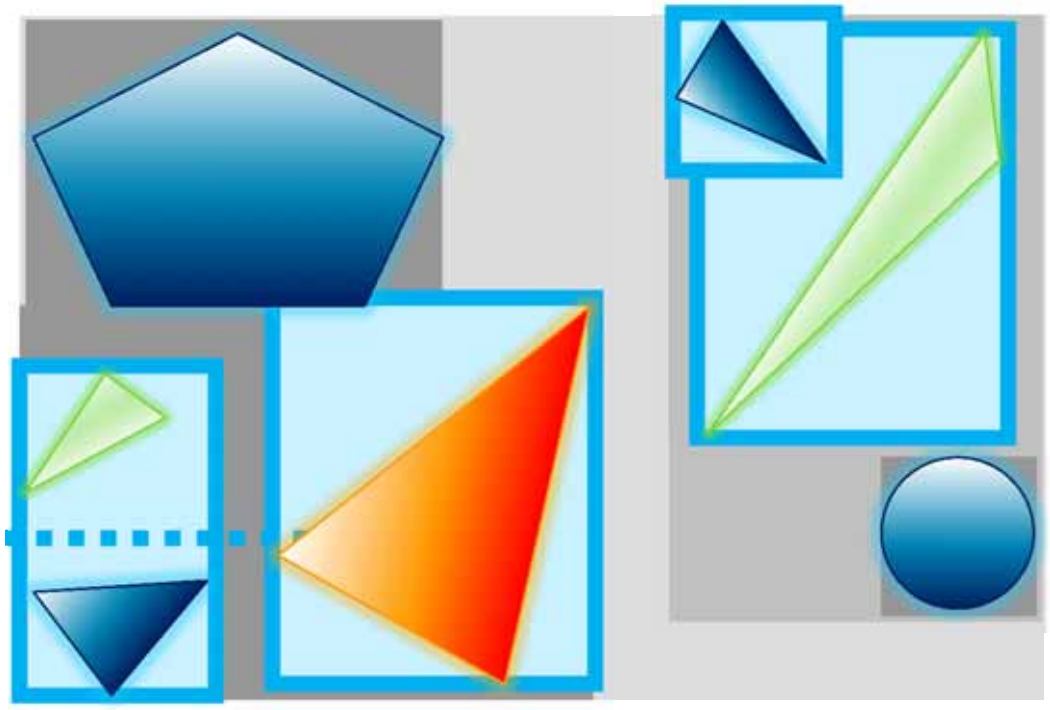
Bounding Volume Hierarchie

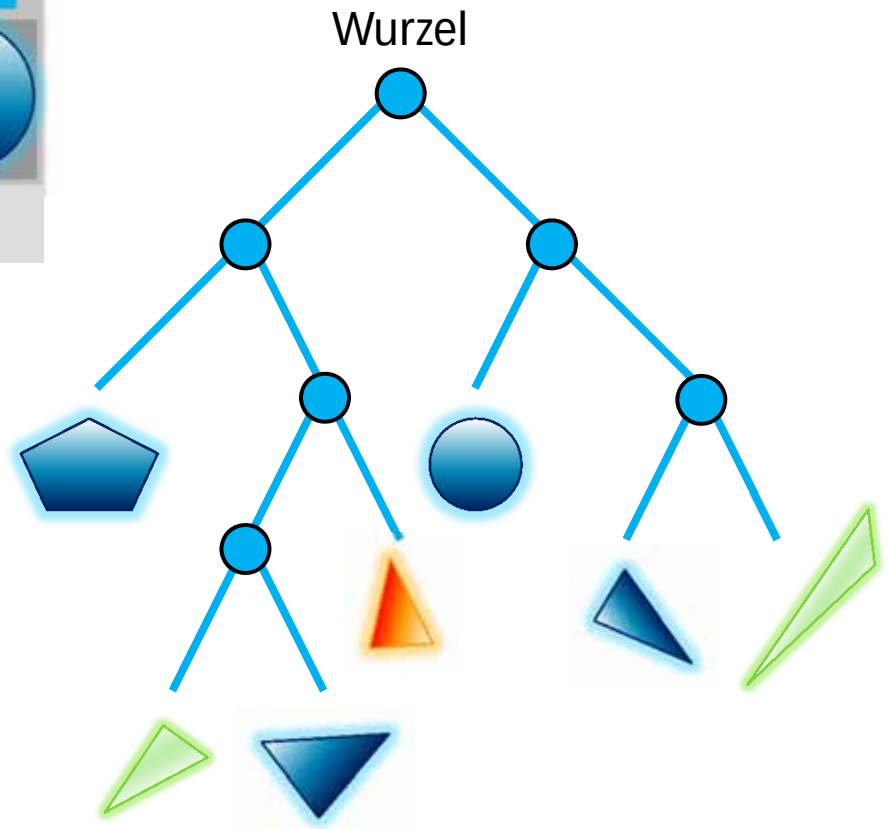
Aufbau der Hierarchie

- bestimme die Bounding Box der Objekte
- teile die Objekte in zwei Gruppen auf
- verfahre so rekursiv weiter (z.B. bis nur noch m Objekte in jeder Gruppe enthalten sind, hier $m = 1$)



Bounding Volume Hierarchie



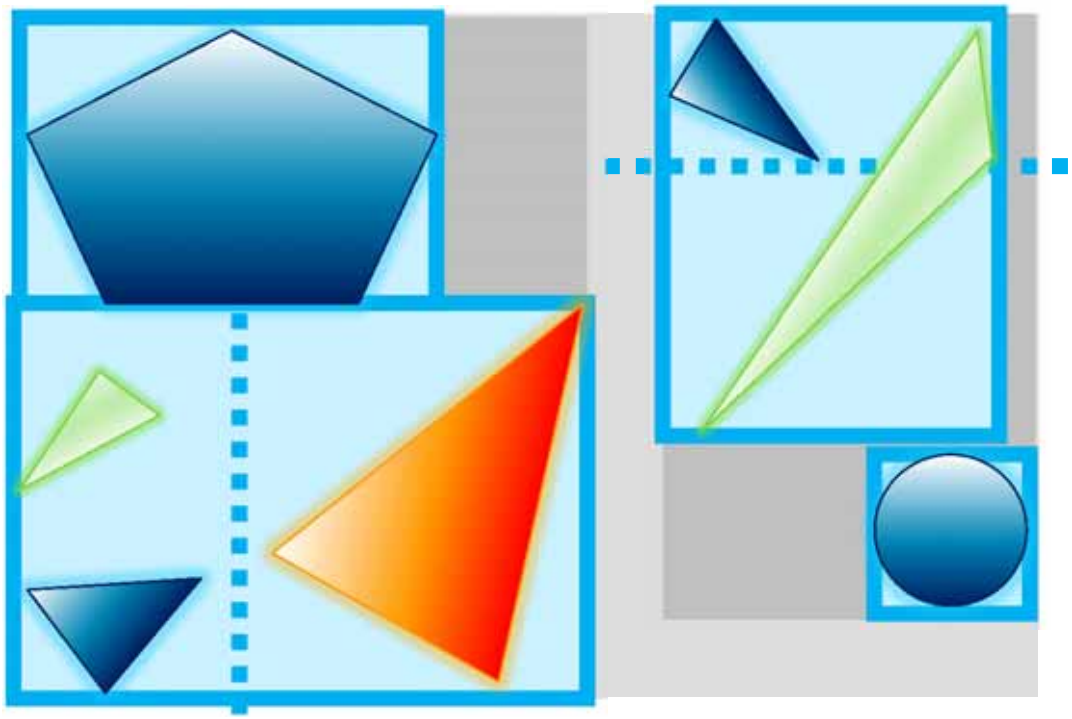


jeder Knoten speichert eine AABB
und Verweis auf weitere Kindknoten
bzw. auf Objekte/Primitive

Bounding Volume Hierarchie

Unterschiedliche Unterteilungskriterien (später mehr)

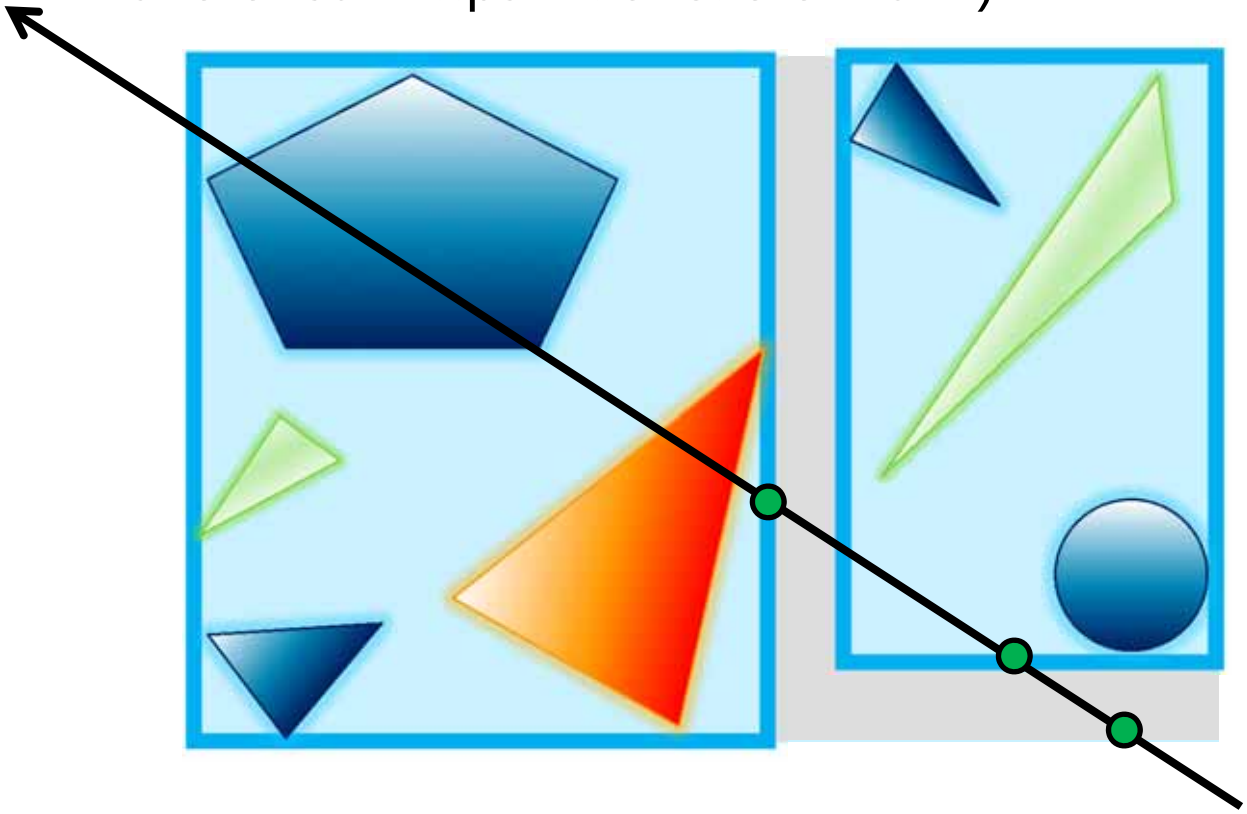
- unterteile in der Mitte senkrecht zur Achse der größten Ausdehnung
- unterteile einmal entlang der x-Achse, dann y-Achse, dann z-Achse, ...
- sortiere die Objekte (typ. entlang Achse der größten Ausdehnung) und unterteile so, dass in jeder Teilmenge etwa die gleiche Anzahl Objekte ist
- verwende Szenengraph bzw. Modellhierarchie
- minimiere Kostenfunktion (z.B. Surface Area Heuristics)

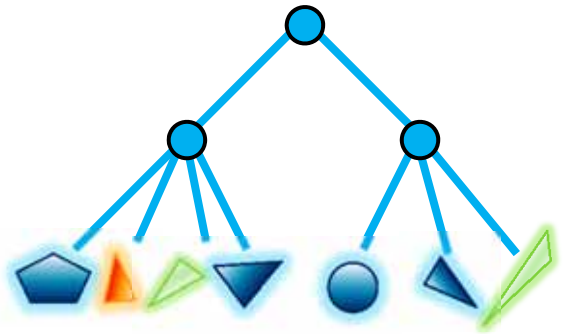


Schnittberechnung mit BVH

Beschleunigung der Schnittberechnung mit BVHs

- überprüfe auf Schnitt mit der Wurzel
- dann steige rekursiv ab (beginnend mit dem näheren Kindknoten)
- teste auf Schnitt mit Objekten im Kindknoten (steige evtl. weiter ab)
- prüfe anschließend noch die weiter entfernten Knoten (sofern ein näherer Schnittpunkt existieren kann)

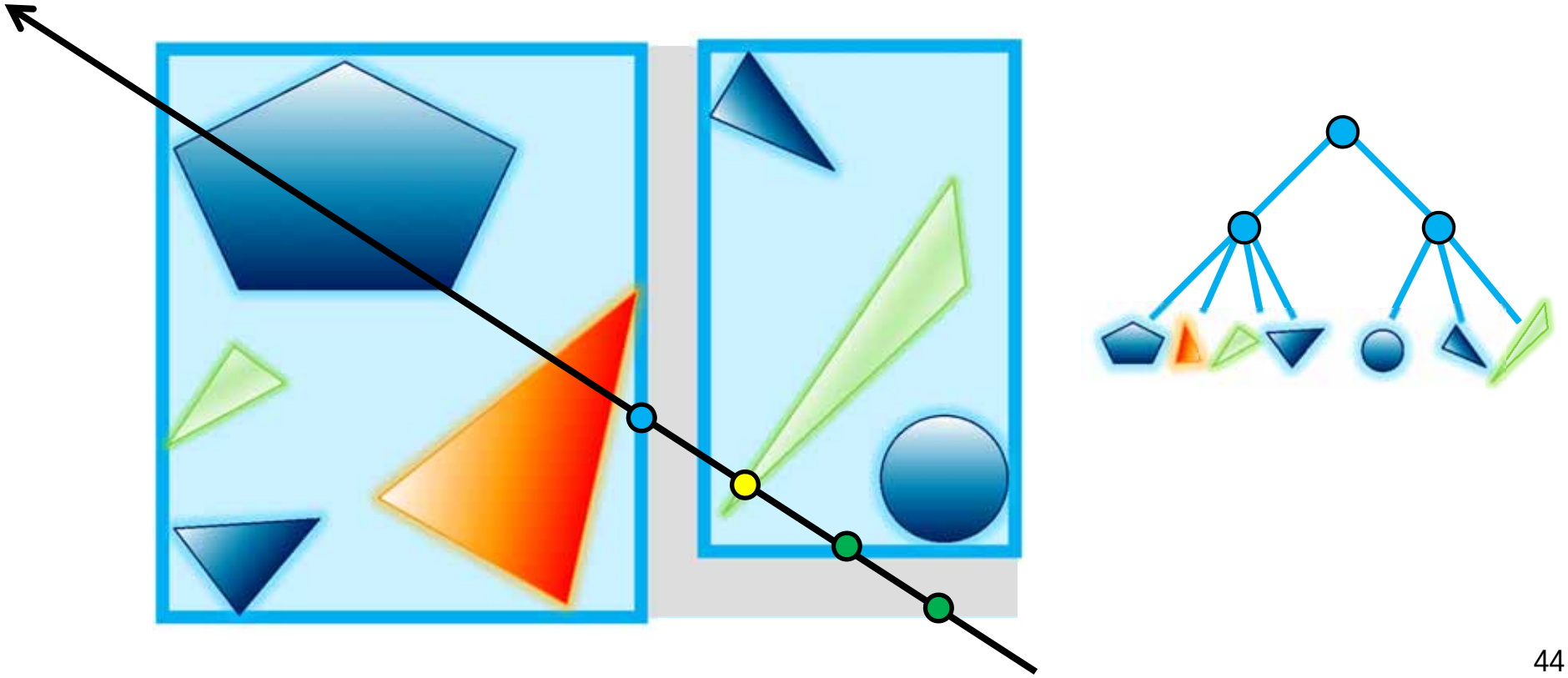




Schnittberechnung mit BVH

Beschleunigung der Schnittberechnung mit BVHs

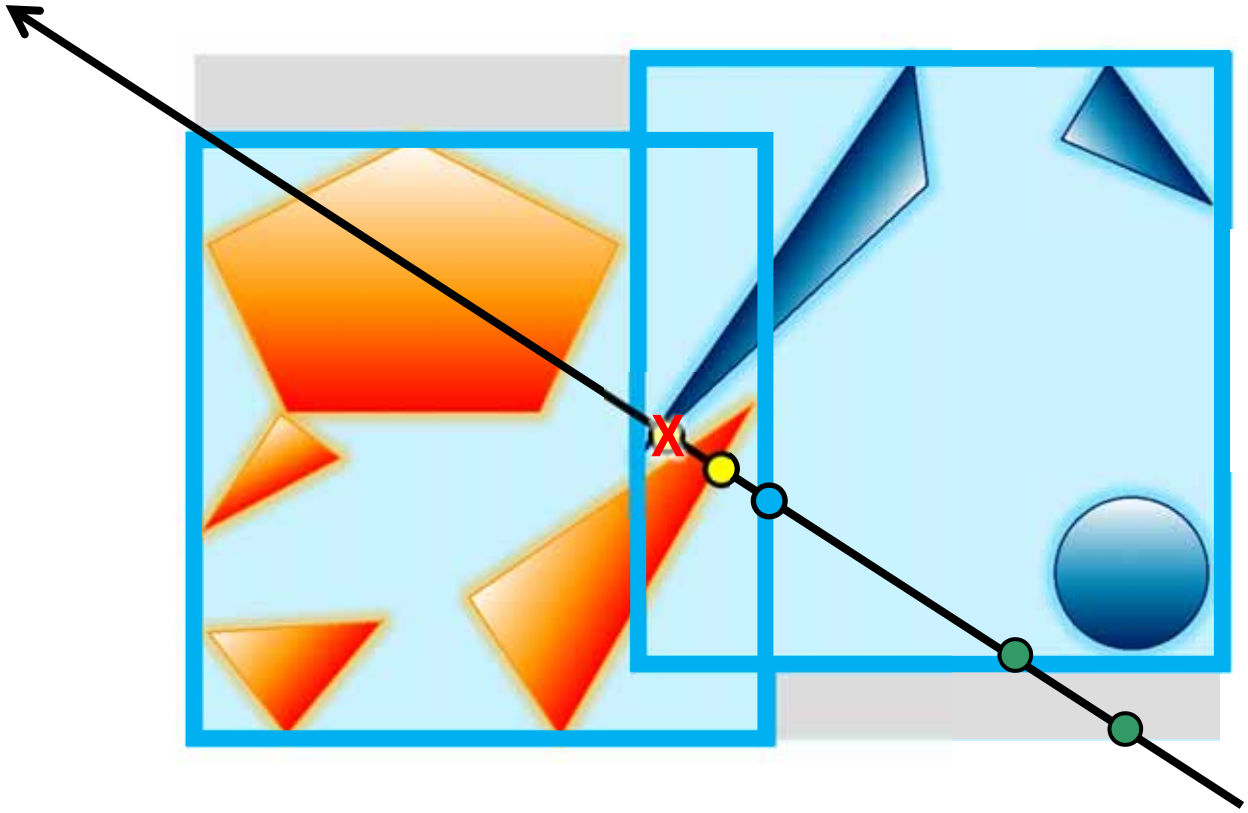
- teste auf Schnitt mit Objekten im Kindknoten (hier kein weiterer Abstieg, wir sind schon in einem Blattknoten)
- Schnittpunkt ● wurde berechnet
- der Schnittpunkt ist näher als die hintere AABB (Schnittpunkt ●)
  → keine weiteren Tests in diesem Ast des Baums notwendig

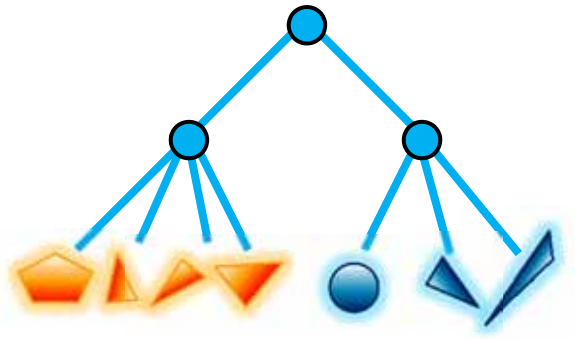


Schnittberechnung mit BVH

Beschleunigung der Schnittberechnung mit BVHs

- Vorsicht: liefere einen gefundenen Schnittpunkt nicht sofort zurück
- es könnte einen näheren in einem anderen Knoten geben
- Grund: Hüllkörper können sich überlappen





Bounding Volume Hierarchie

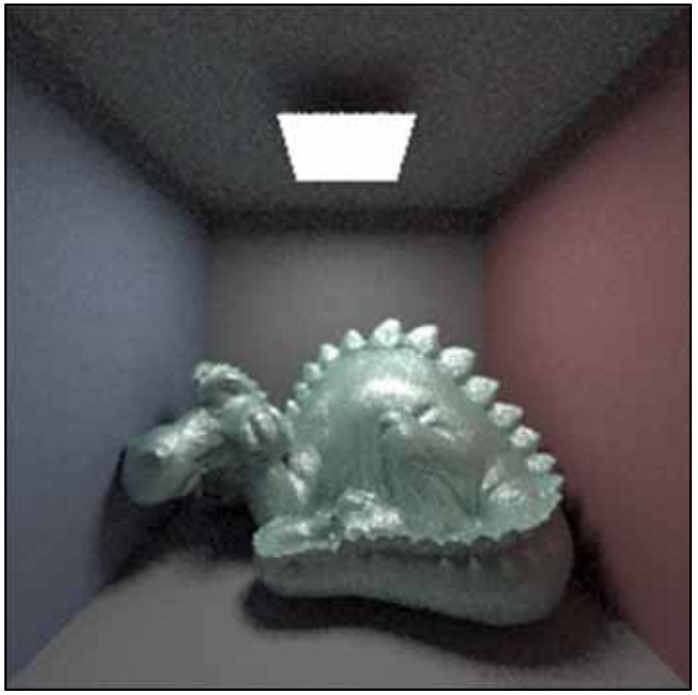
Fazit

- Vorteile
  - Konstruktion und Traversierung ist einfach
  - resultiert in einem Binärbaum (fixe, geringe Verzweigung)
  - Komplexität
    - im Mittel $O(\log n)$ Schnitttests für n Objekte/Primitive in der Szene (vgl. ohne BVH werden $O(n)$ Schnitttests benötigt)
    - Aufbau $O(n \log n)$ wenn in der Mitte der AABBs unterteilt wird
    - Aufbau $O(n \log^2 n)$, wenn die Objekte in einer AABB in zwei (etwa) gleich große Gruppen geteilt werden und eine $O(n \log n)$ Sortierung verwendet wird

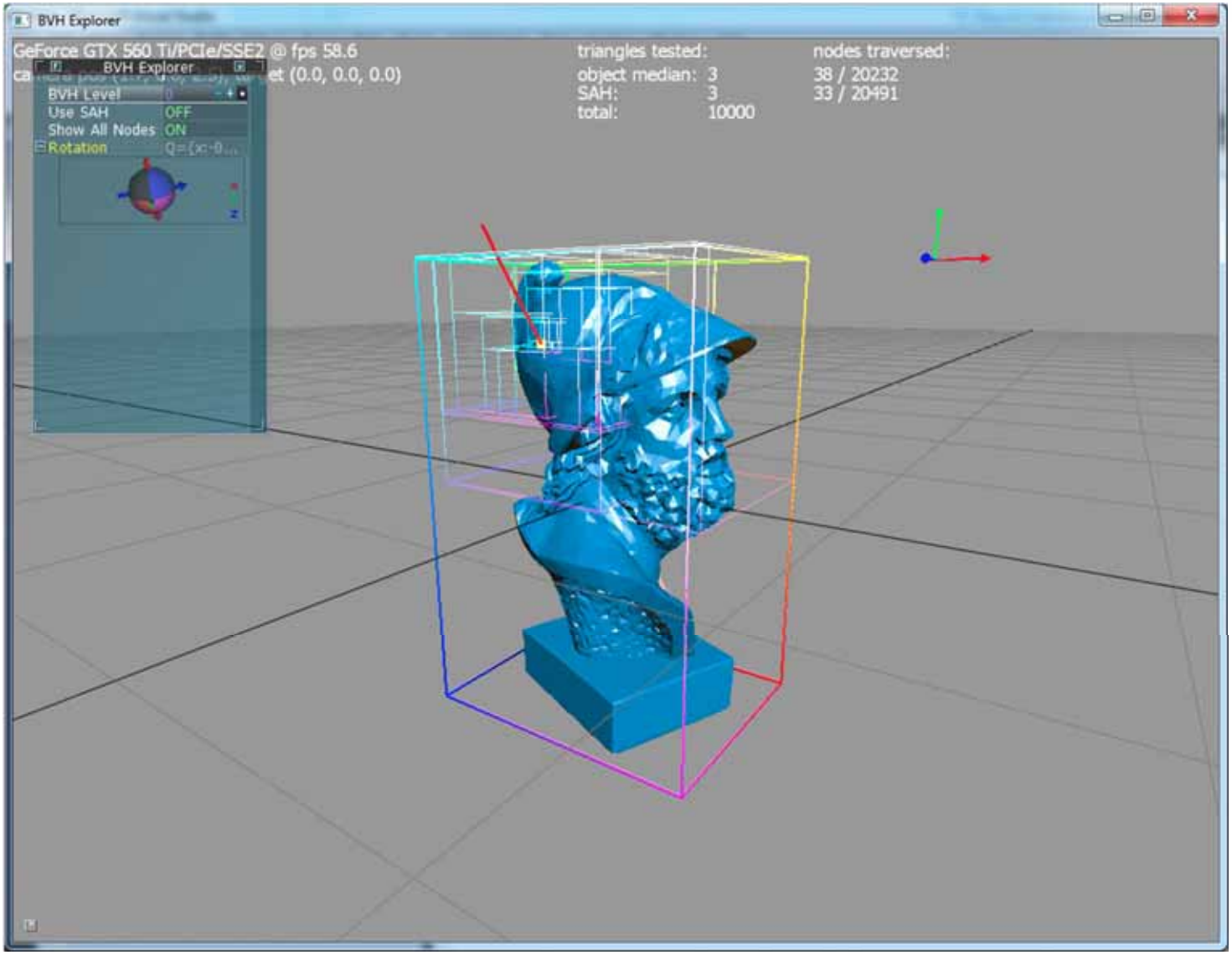
- Herausforderungen
  - Finden einer guten Unterteilung ist schwierig
    (weil nicht unbedingt klar ist, was eigentlich gut ist: wir wissen nicht vorab, welche Strahlen beim Ray Tracing verschossen werden)
  - ungeschickte Unterteilung kann zu schlechter Aufteilung führen

Performance von BVHs

- Path Tracing
  - 32x stochast. Supersampling, Auflösung 256^2 Pixel, 480000 Dreiecke
  - insgesamt ca. $6.3 \cdot 10^6$ Primär-, Sekundär- und Schattenstrahlen
- Brute Force (176462sec ≈ 49h):
  - ca. $3 \cdot 10^{12}$ Schnitttests
- Bounding Volume Hierarchie
  - Strategie:
    Unterteilung in 2 gleich große Gruppen
  - Aufbau 18.3sec, Rendering 59sec
  - 262143 AABB Knoten
  - $7.5 \cdot 10^8$ AABB Tests
  - $1.3 \cdot 10^8$ Schnitttests mit Dreiecken
- Vergleich
  - Brute Force: 3431× Schnitttests
  - BVH Beschleunigung: 2200×
    (BVH Aufbau berücksichtigt)



BVH Demo


BVH Explorer
GeForce GTX 560 Ti/PCIe/SSE2 @ fps 58.6
triangles tested:
nodes traversed:
object median: 3
38 / 20232
SAH: 3
33 / 20491
total: 10000
BVH Explorer
BVH Level
Use SAH OFF
Show All Nodes ON
Rotation

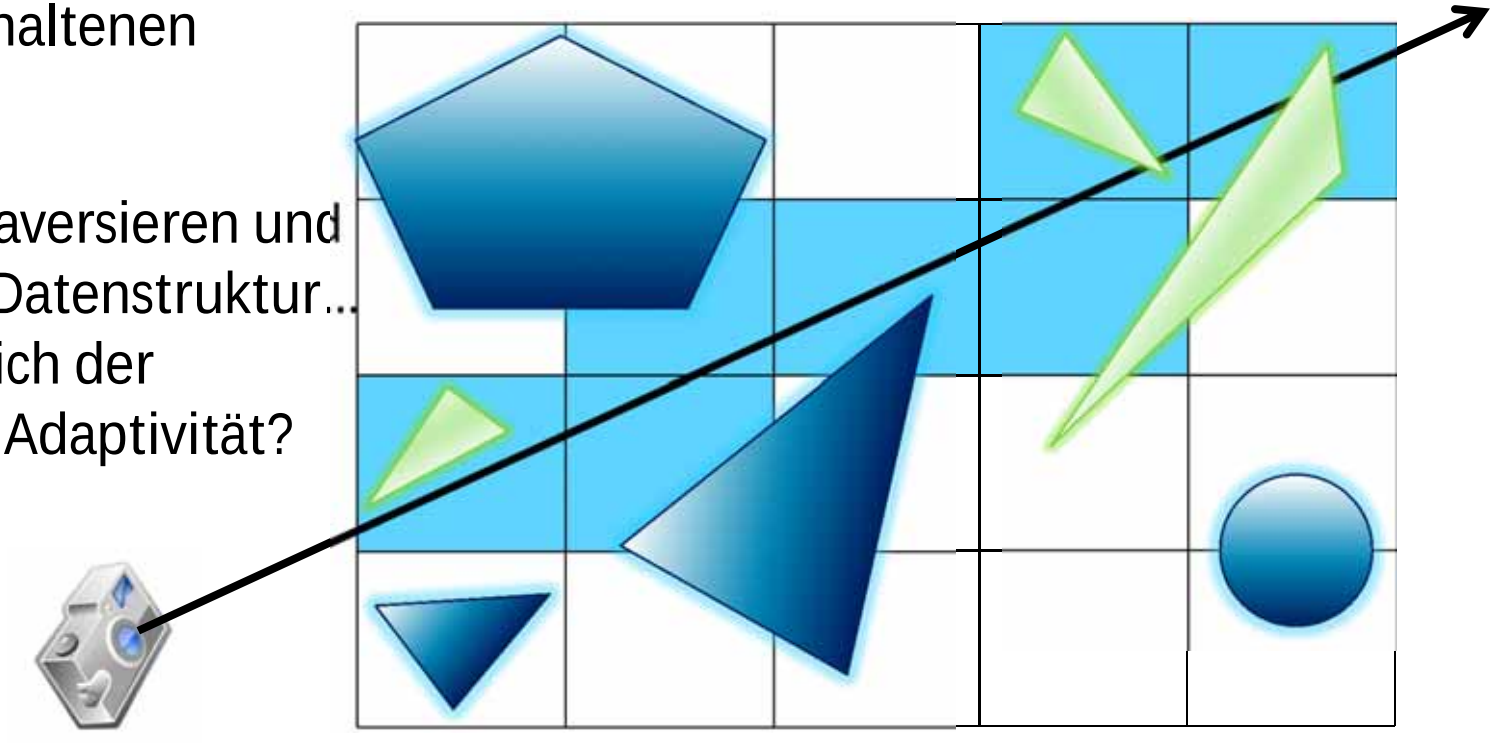
Inhalt: Räumliche Datenstrukturen

- Analyse der Kosten bei Ray Tracing
- Ansätze zur Beschleunigung von Ray Tracing
- Bildraumverfahren
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - **Bounding Volume Hierarchies**
  - **reguläre und adaptive Gitter**
  - **kD-Bäume** und BSP-Bäume

Raumunterteilung durch reguläre Gitter

Grundidee, Prinzip

- Unterteilung des Raumes in Zellen gleicher Größe und Form (daher die Bezeichnung „regulär“), ausgehend von der Bounding-Box der Szene
- Eintrag der Objekte in Zellen, die von ihnen geschnitten werden
- Traversierung der vom Strahl getroffenen Zellen und Schnittberechnung mit den enthaltenen Objekten
- einfaches Traversieren und Aufbau der Datenstruktur... aber lohnt sich der Verzicht auf Adaptivität?

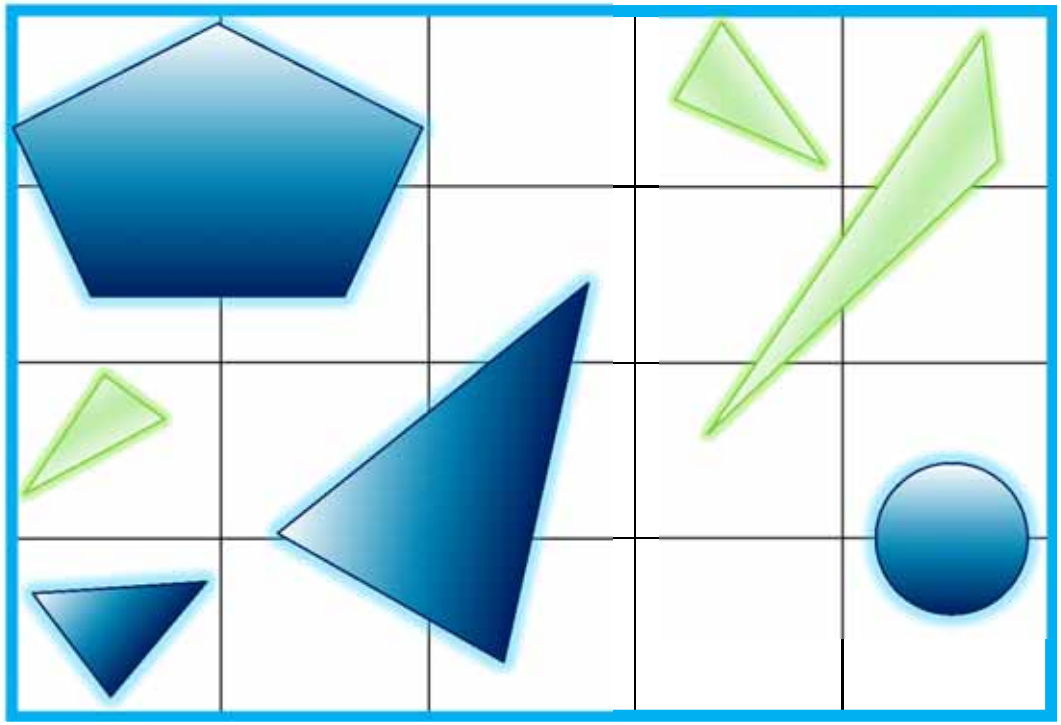


Raumunterteilung durch reguläre Gitter

Größe und Dimension des Gitters

- bestimme Bounding Box der gesamten Szene
- wähle Gitterauflösung n_x, n_y, n_z (n_x muss nicht gleich n_y sein etc.), d.h. die Zellen sind beliebige (aber immer gleich große) Quader

- hier: $n_x = 5, n_y = 4$

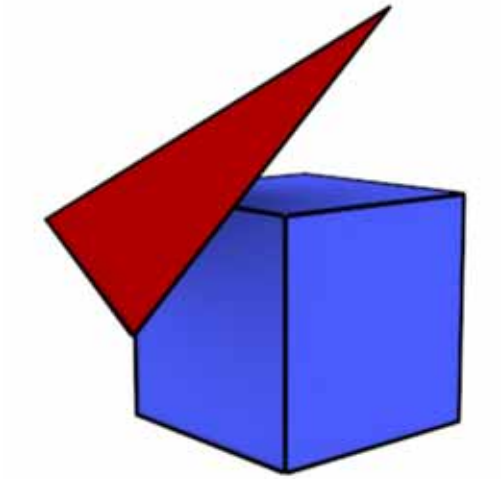


Bounding Box

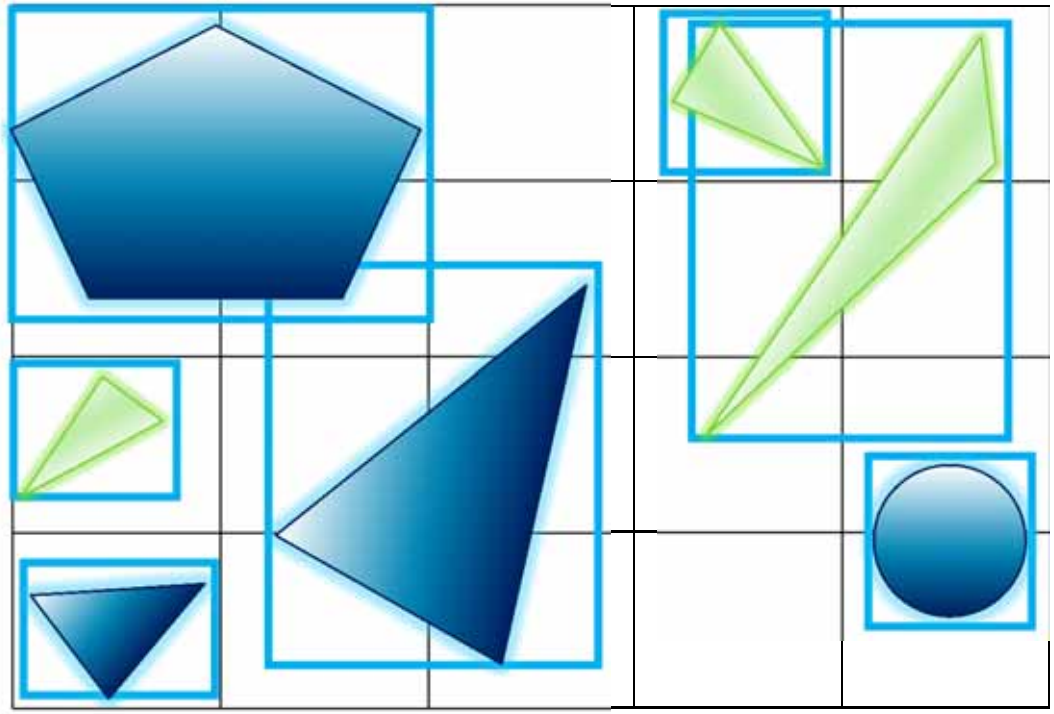
Raumunterteilung durch reguläre Gitter

Eintragen der Primitive in das Gitter

- bestimme die Gitterzellen, die ein Objekt schneidet
  - einfach und konservativ: verwende AABB des Objekts
  - AABB kann vor exaktem Test auch zur Vorauswahl der Zellen dienen
- überlappt ein Objekt mehrere Zellen, so wird es in jede Zelle eingetragen (verwende Zeiger)



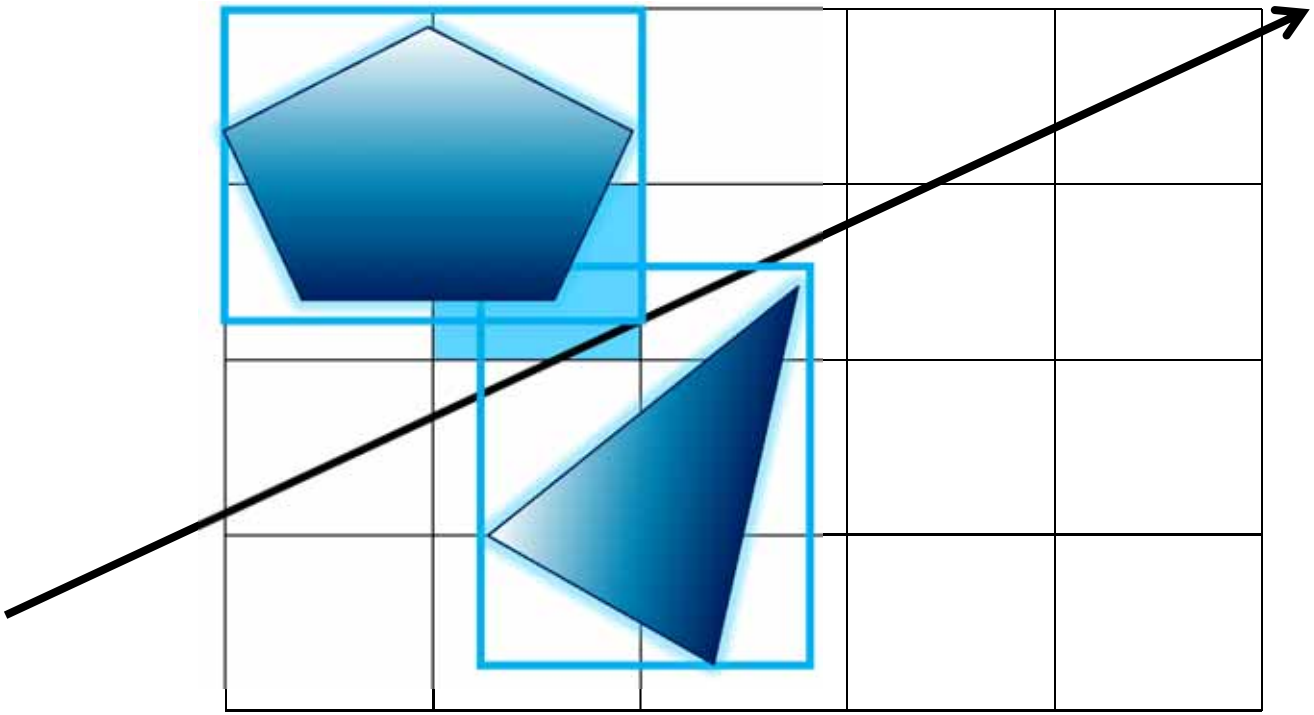
Schnitt Dreieck-Zelle?



Raumunterteilung durch reguläre Gitter

Schnittpunktberechnung

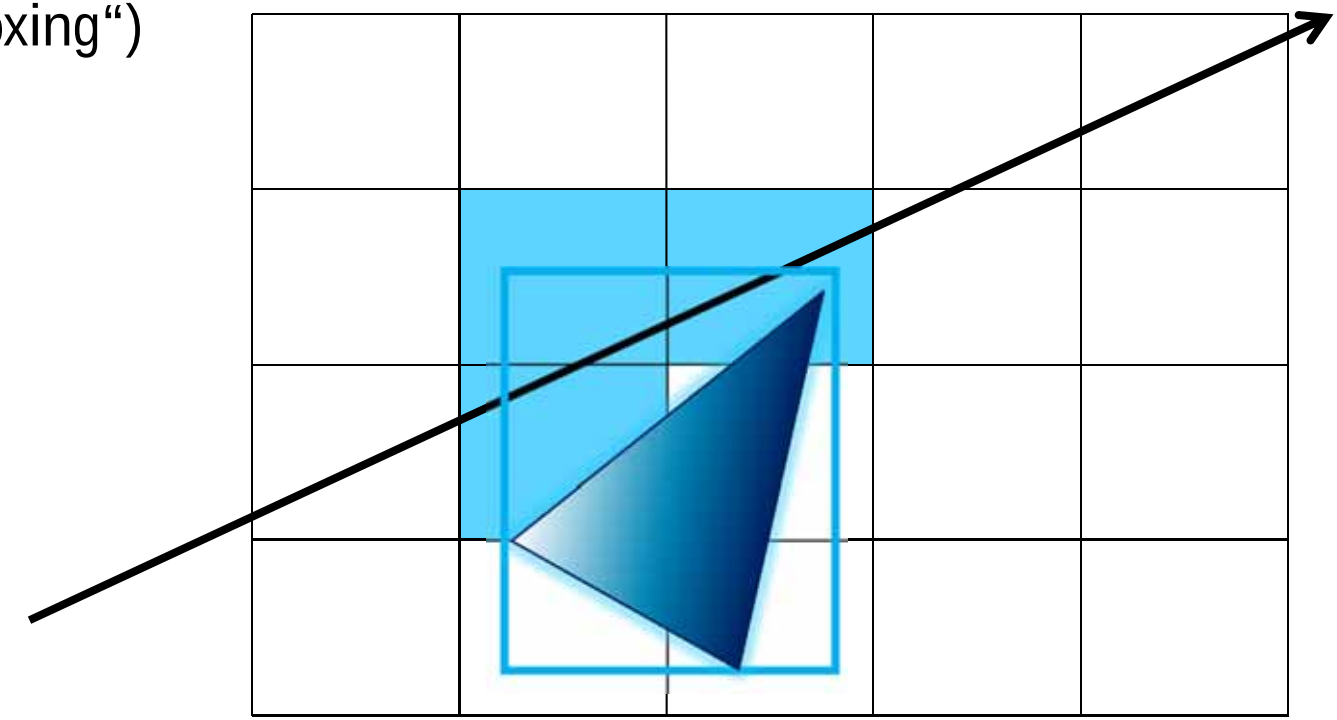
- teste für jede Zelle entlang des Strahls:
  gibt es **in dieser Zelle** Schnitte mit Objekten?
  - ja: gebe nahsten Schnittpunkt zurück
  - nein: weiter mit der Traversierung



Raumunterteilung durch reguläre Gitter

Vermeide mehrfache Schnitttests mit einem Objekt

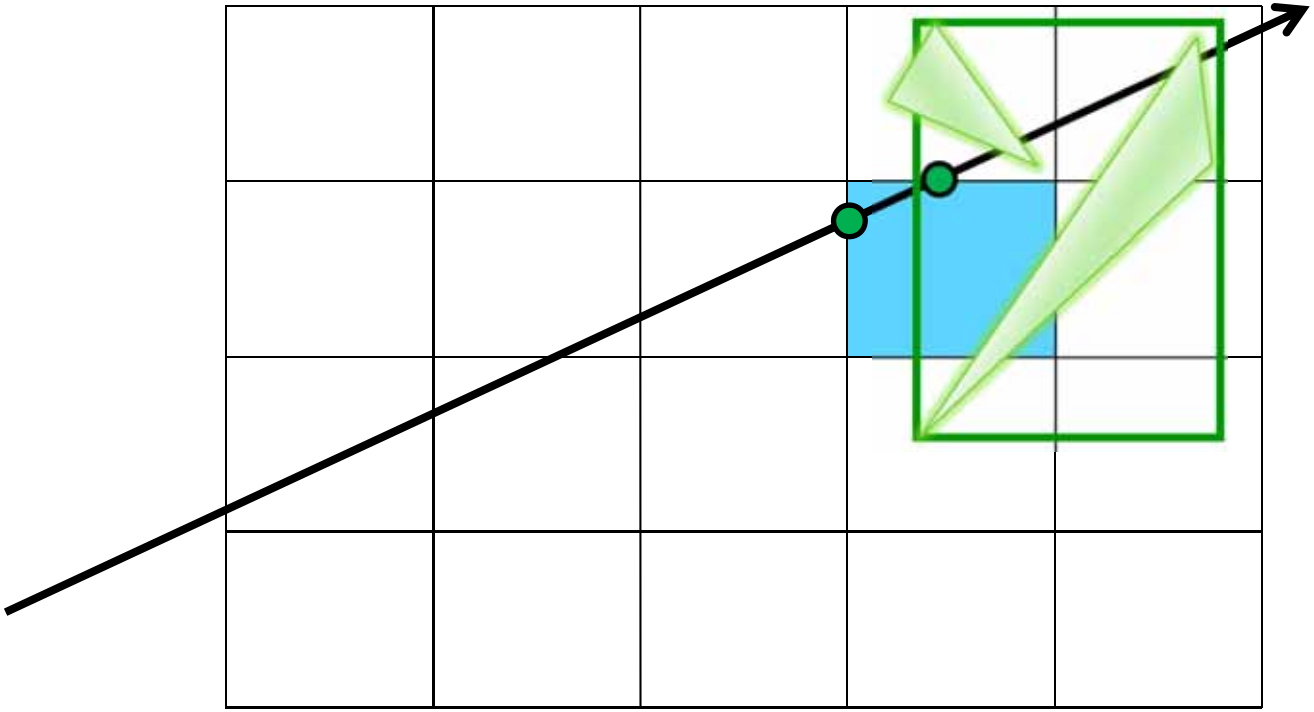
- führe den Schnitttest durch und markiere das Objekt als getestet
- Ziel: kein weiterer Schnitttest mit einem markierten Objekt
- wenn ein Schnittpunkt gefunden wurde (und er außerhalb der Zelle lag): speichere den Schnittpunkt, um ihn später nicht nochmal berechnen zu müssen („Mailboxing“)



Raumunterteilung durch reguläre Gitter

Ignoriere Schnitte außerhalb der aktuellen Zelle

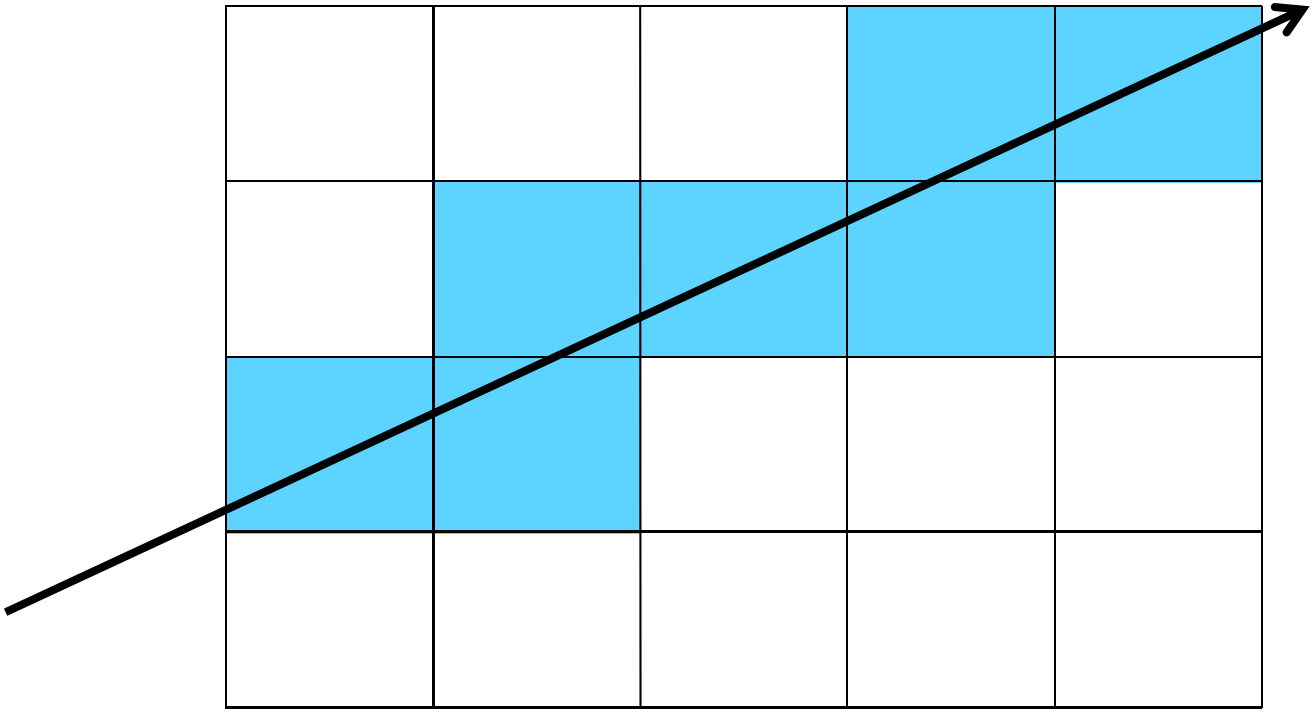
- verwerfe (oder speichere für später) Schnittpunkte außerhalb der aktuellen Zelle
- es könnte einen näheren Schnittpunkt mit einem anderen Objekt geben



Raumunterteilung durch reguläre Gitter

Traversierung des Gitters

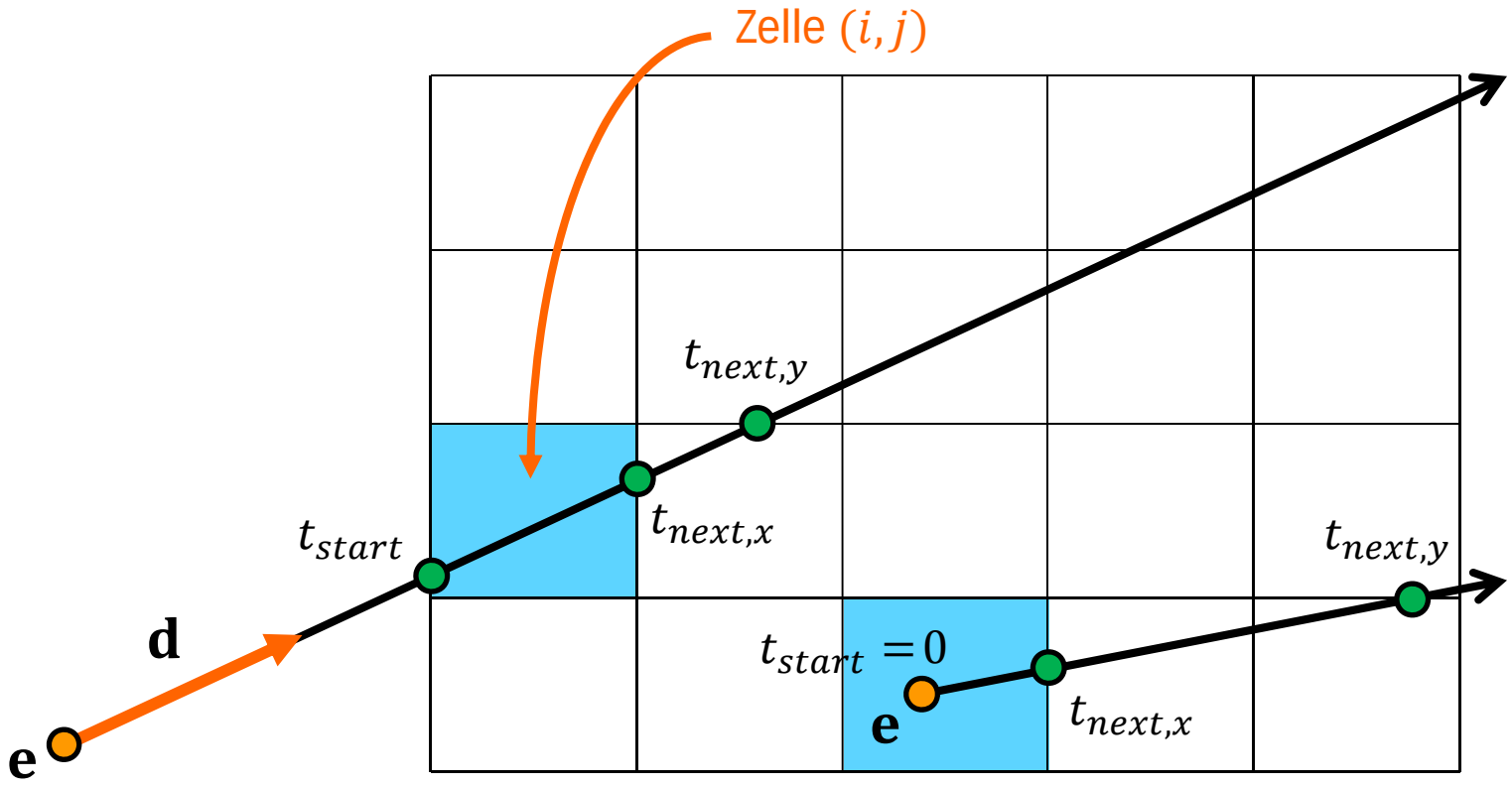
- wie findet man die Zellen die ein Strahl durchläuft? (nur Objekte/Primitive in diesen Zellen möchte man testen)



Raumunterteilung durch reguläre Gitter

Bestimmung der Start-Zelle

- schneide Strahl mit der Bounding Box der Szene
- aus den Schnittkoordinaten lässt sich der Zellenindex (i, j) und (zusammen mit $\mathbf{d}$) dann $t_{next,x}$ bzw. $t_{next,y}$ berechnen
- Strahlursprung kann auch in der Bounding Box der Szene liegen

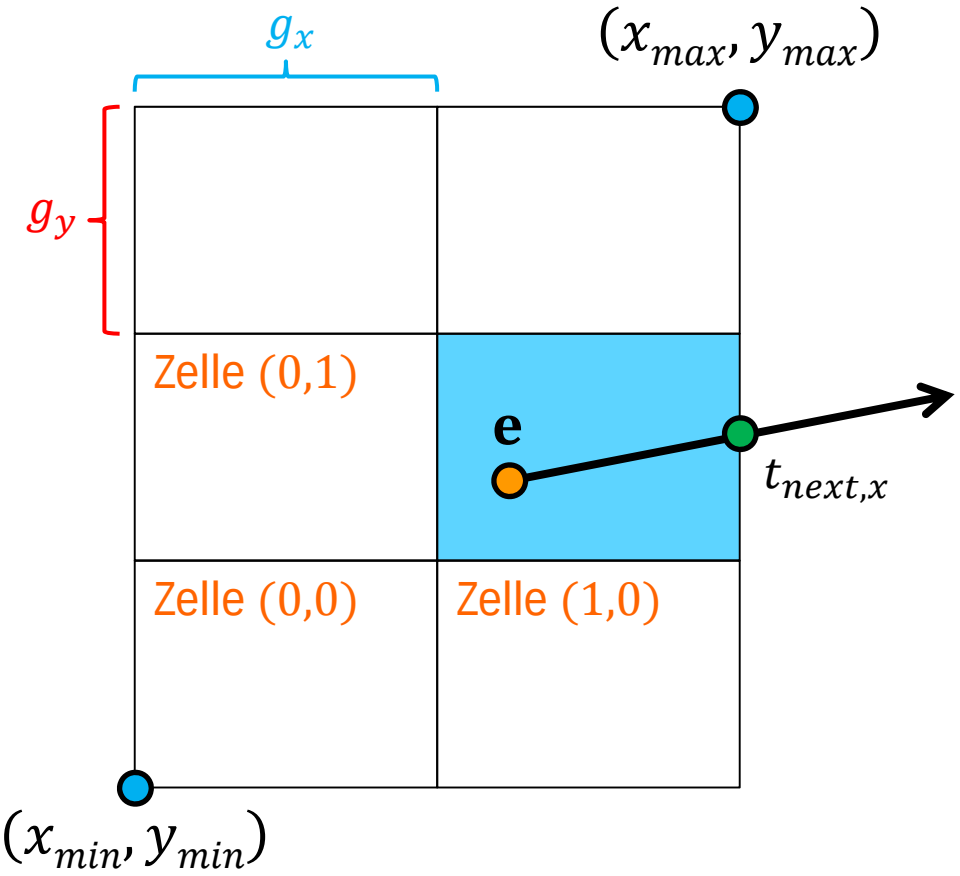


Raumunterteilung durch reguläre Gitter

Bestimmung der Start-Zelle

- Zellenindex (i, j) und $t_{next,x}$ bzw. $t_{next,y}$ aus Koordinaten berechnen
- $i = \lfloor (e_x - x_{min})/g_x \rfloor$ bzw. $j = \lfloor (e_y - y_{min})/g_y \rfloor$
- $e_x + t_{next,x} d_x = (i+1) g_x$ (für $d_x > 0$)

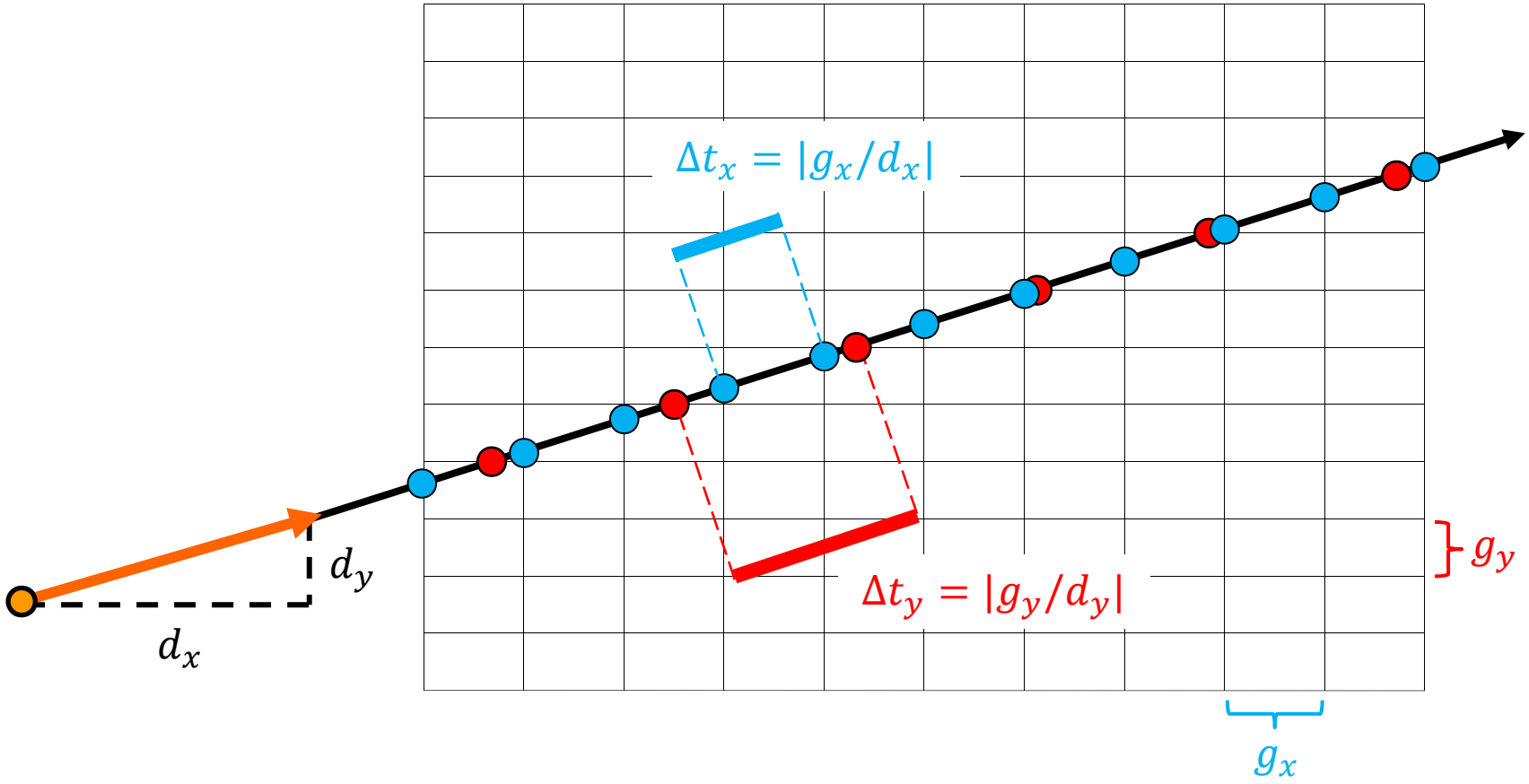
  $\Rightarrow\ t_{next,x} = \cdots$



Raumunterteilung durch reguläre Gitter

Sprung zur nächsten Zelle

- horizontale und vertikale Sprünge zur nächsten Spalte bzw. Zeile haben immer denselben Abstand

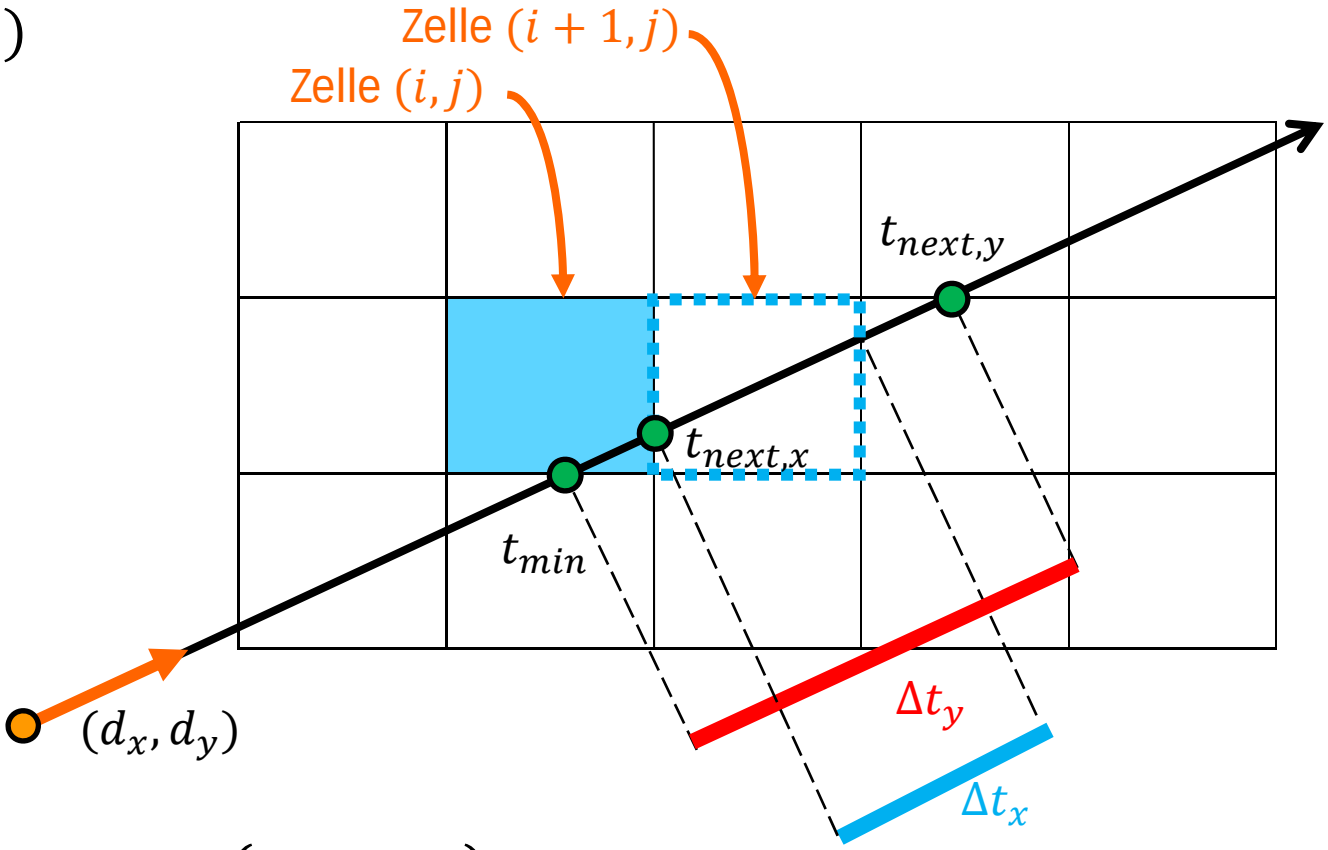


Raumunterteilung durch reguläre Gitter

Sprung zur nächsten Zelle

- Start: $t_{next,x}$ bzw. $t_{next,y}$ einmalig bestimmen
- ⚠ funktioniert wie folgt auch für $d_{x,y} < 0$
- Zelle auf (i, j) folgend (wir betrachten gerade die Stelle $t = t_{min}$):

if $(t_{next,x} < t_{next,y})$
  $i \mathrel{+}= \text{sign}_x$
  $t_{min} = t_{next,x}$
  $t_{next,x} \mathrel{+}= \Delta t_x$
else
  $j \mathrel{+}= \text{sign}_y$
  $t_{min} = t_{next,y}$
  $t_{next,y} \mathrel{+}= \Delta t_y$



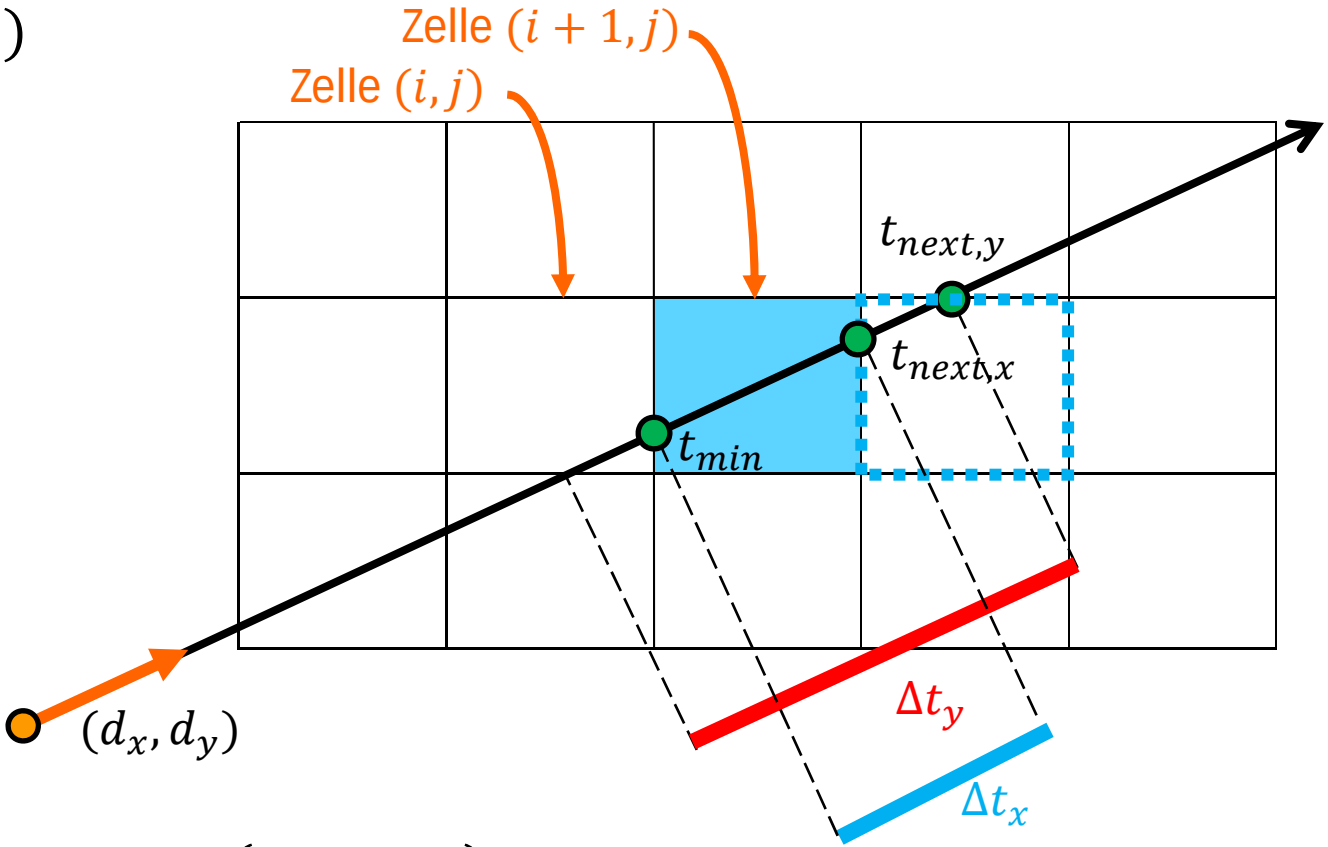
if $(d_{x,y} > 0)$ $\text{sign}_{x,y} = 1$ else $\text{sign}_{x,y} = -1$

Raumunterteilung durch reguläre Gitter

Sprung zur nächsten Zelle

- Start: $t_{next,x}$ bzw. $t_{next,y}$ einmalig bestimmen
- ⚠ funktioniert wie folgt auch für $d_{x,y} < 0$
- Zelle auf (i, j) folgend (wir betrachten gerade die Stelle $t = t_{min}$):

if $(t_{next,x} < t_{next,y})$
  $i \mathrel{+}= \text{sign}_x$
  $t_{min} = t_{next,x}$
  $t_{next,x} \mathrel{+}= \Delta t_x$
else
  $j \mathrel{+}= \text{sign}_y$
  $t_{min} = t_{next,y}$
  $t_{next,y} \mathrel{+}= \Delta t_y$



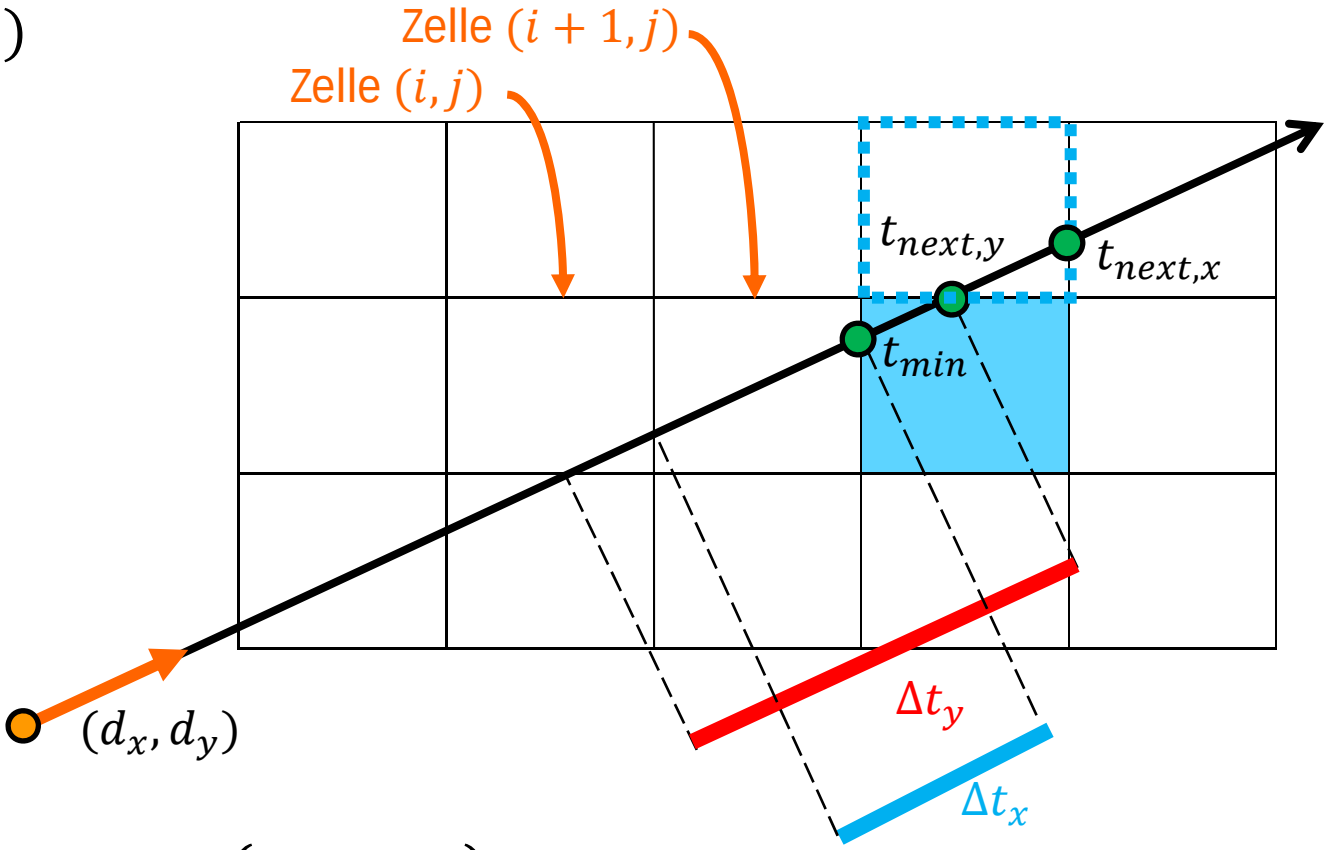
if $(d_{x,y} > 0)$ $\text{sign}_{x,y} = 1$ else $\text{sign}_{x,y} = -1$

Raumunterteilung durch reguläre Gitter

Sprung zur nächsten Zelle

- Start: $t_{next,x}$ bzw. $t_{next,y}$ einmalig bestimmen
- ⚠ funktioniert wie folgt auch für $d_{x,y} < 0$
- Zelle auf (i, j) folgend (wir betrachten gerade die Stelle $t = t_{min}$):

if $(t_{next,x} < t_{next,y})$
  $i \mathrel{+}= \text{sign}_x$
  $t_{min} = t_{next,x}$
  $t_{next,x} \mathrel{+}= \Delta t_x$
else
  $j \mathrel{+}= \text{sign}_y$
  $t_{min} = t_{next,y}$
  $t_{next,y} \mathrel{+}= \Delta t_y$



if $(d_{x,y} > 0)$ $\text{sign}_{x,y} = 1$ else $\text{sign}_{x,y} = -1$

Raumunterteilung durch reguläre Gitter

Zusammenfassung

- bestimme AABB der Szene und erzeuge Gitter
- trage Objekte/Primitive in das Gitter ein
- für jeden Strahl $\mathbf{r}(t)$
  - bestimme Startzelle $z(i,j,k)$, t_{start}, $t_{next,x}$, $t_{next,y}$ und $t_{next,z}$ (in 3D)
    - berechne $sign_x$, $sign_y$ und $sign_z$ aus d_x, d_y, d_z
- solange $0 \leq i < n_x \wedge 0 \leq j < n_y \wedge 0 \leq k < n_z$
  - für jedes Primitiv P in $z(i,j,k)$
    - teste auf Schnitt von $\mathbf{r}(t)$ und P
  - wenn Schnitt(e) in aktueller Zelle gefunden, dann gebe nahsten zurück
  - sonst: gehe weiter zu nächster Zelle
  - optional: Mailboxing

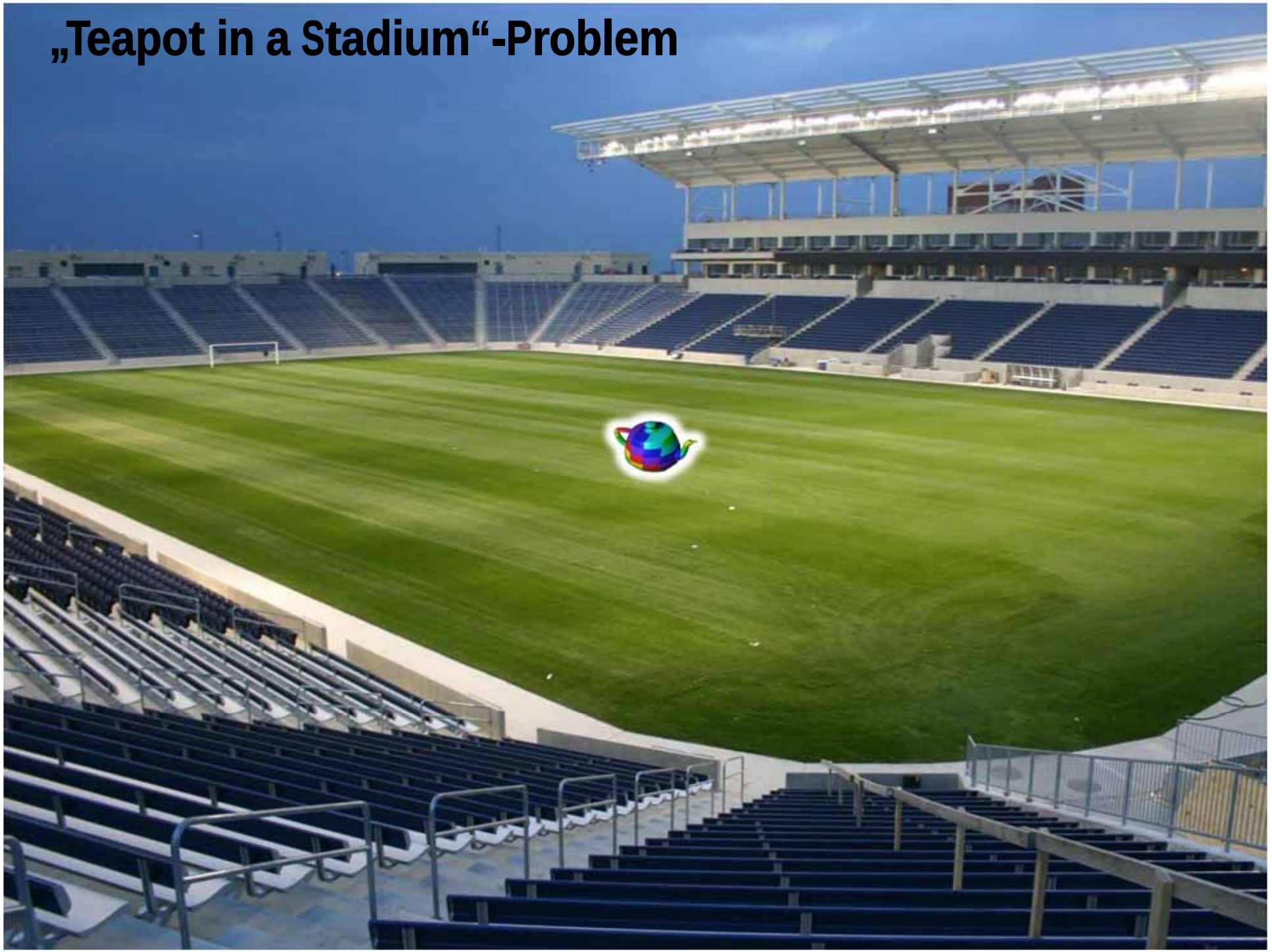
Raumunterteilung durch reguläre Gitter

Fazit

- Vorteile
  - einfach zu konstruieren
  - einfach zu traversieren

- Nachteile
  - in der Regel nur wenige Zellen belegt
  - u.U. viel Geometrie in wenigen Zellen konzentriert

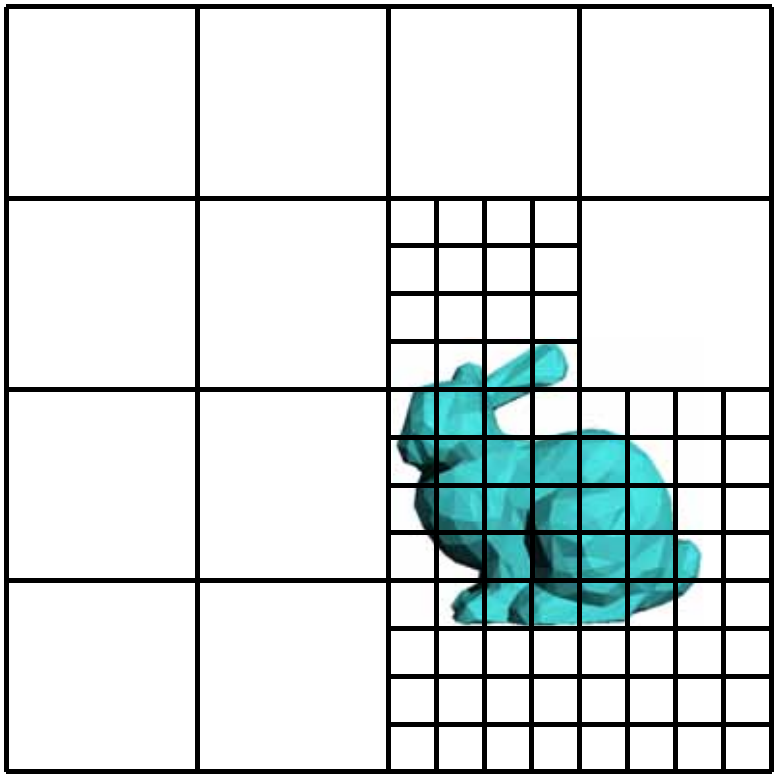
„Teapot in a Stadium“-Problem



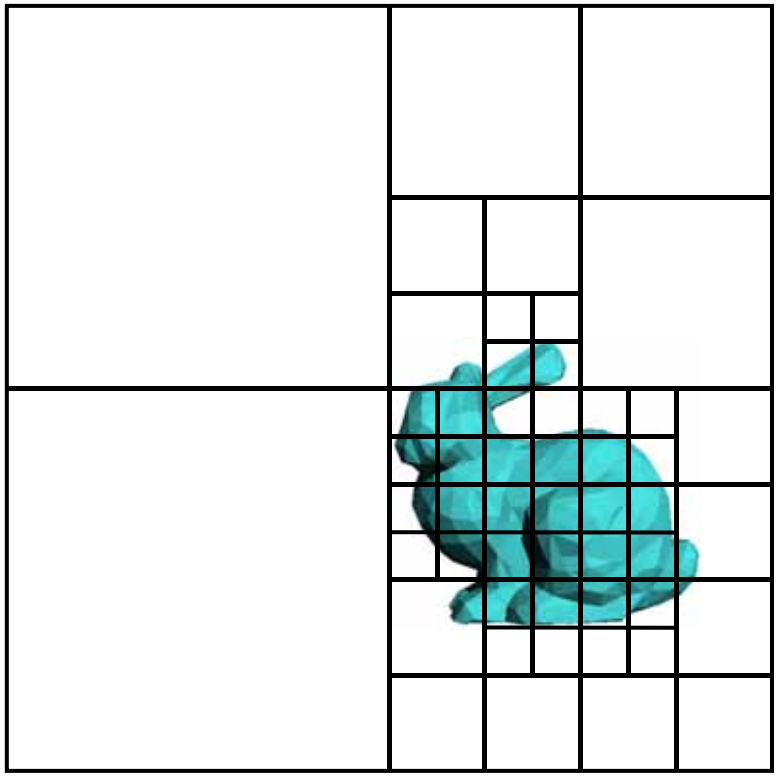
Raumunterteilung durch adaptive Gitter

Grundidee

- rekursive Unterteilung einer Zelle bis
  - sie nur noch maximal eine vorgegebene Anzahl Primitive enthält, oder
  - eine maximale Anzahl Unterteilungen durchgeführt wurde



Verschachtelte Gitter (Nested Grids)

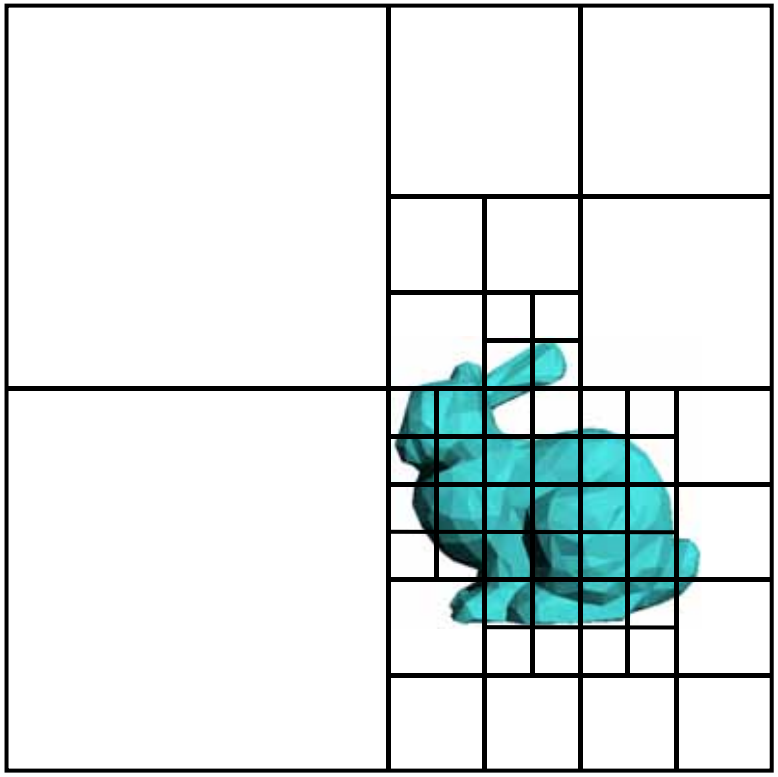


Oktalbaum (Octree), bzw. hier in 2D ein Quadtree (jeder Knoten hat 4 Kinder)

Oktalbaum/Octree

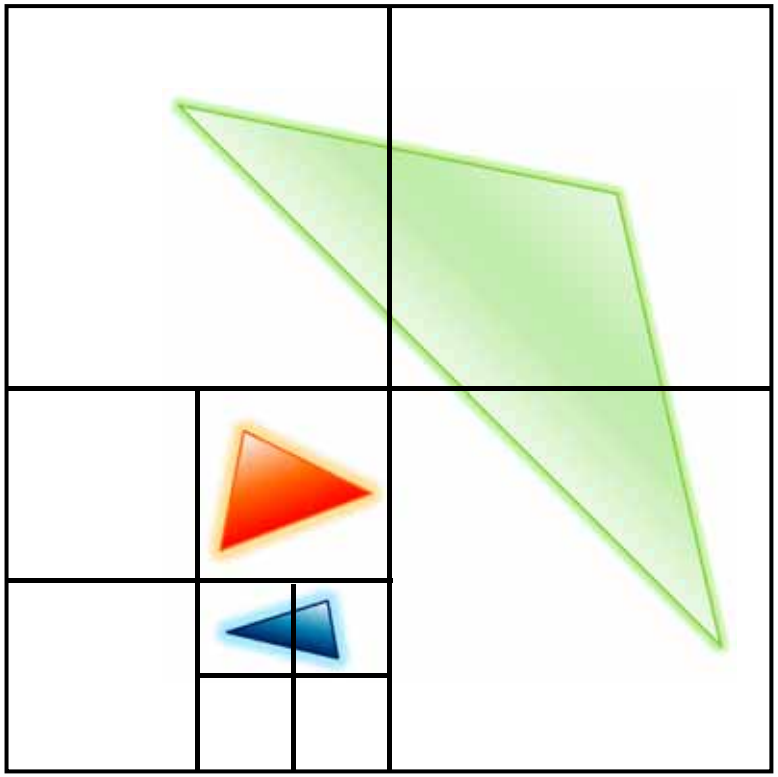
Hierarchische Raumunterteilung mit Octrees

- beginne mit der Bounding-Box für die ganze Szene
- rekursive 1-zu-8 Unterteilung (daher „Oktalbaum“, je 1-zu-2 in jeder Richtung) der Zellen, falls noch zu viele Primitive in einer Zelle sind
  - adaptive Unterteilung erlaubt große Schritte im leeren Raum
  - feine Unterteilung dort (und nur dort) wo Geometrie ist
- Traversierung ist relativ teuer durch häufige Auf- und Abbewegung in der Hierarchie

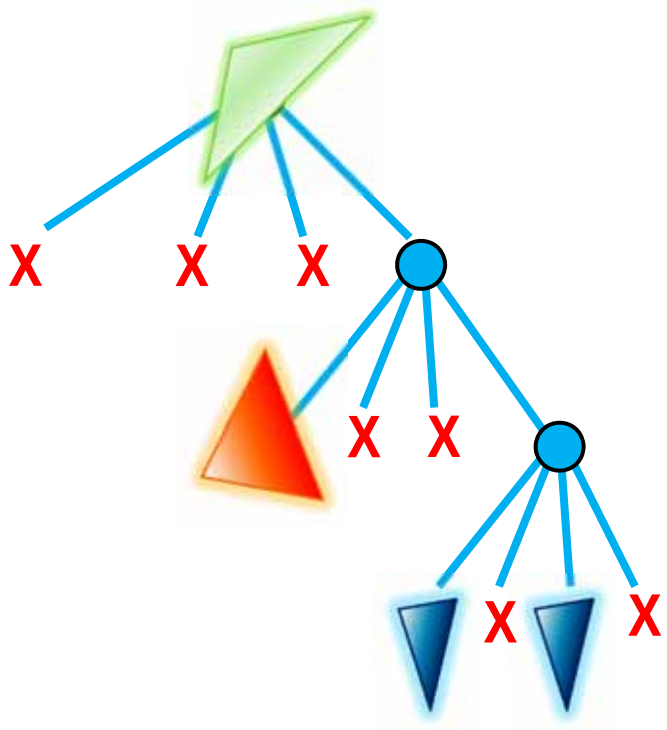


Oktalbaum/Octree

- Verweise auf Primitive werden in inneren Knoten gespeichert (grün) *oder* in den Blättern des Baums (orange, blau), dann u.U. mehrfach
- grünes Objekt könnte auch in den 3 leeren Kindknoten der Wurzel sein



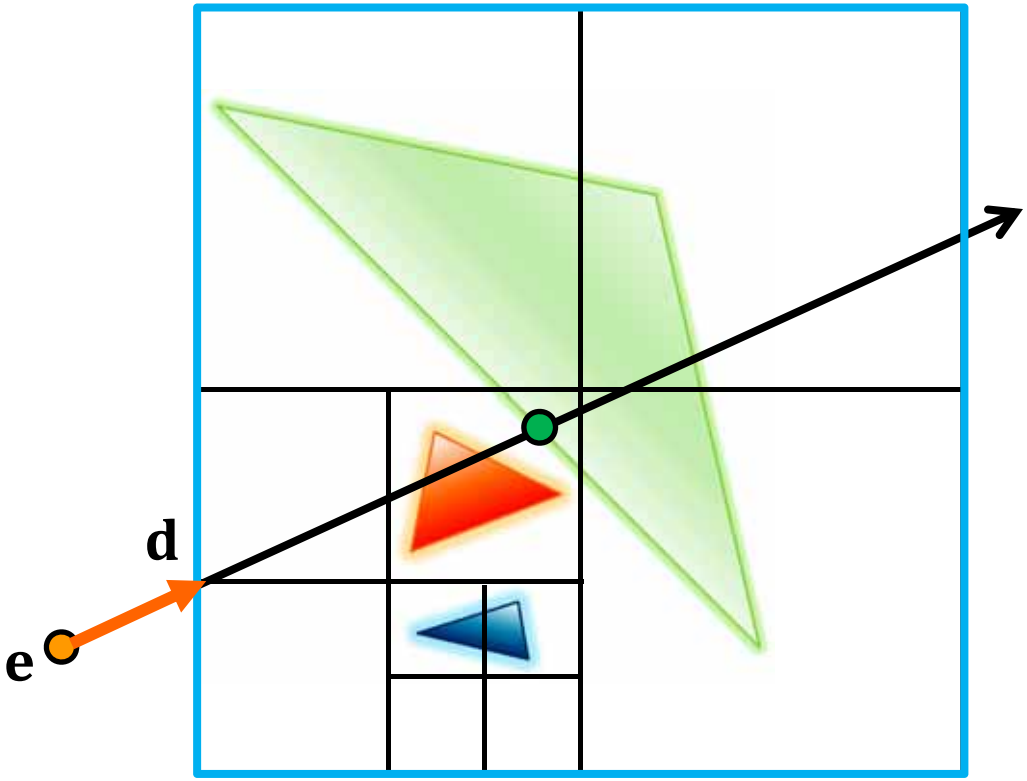
Oktalbaum (Octree), hier in 2D dargestellt als Quadtree



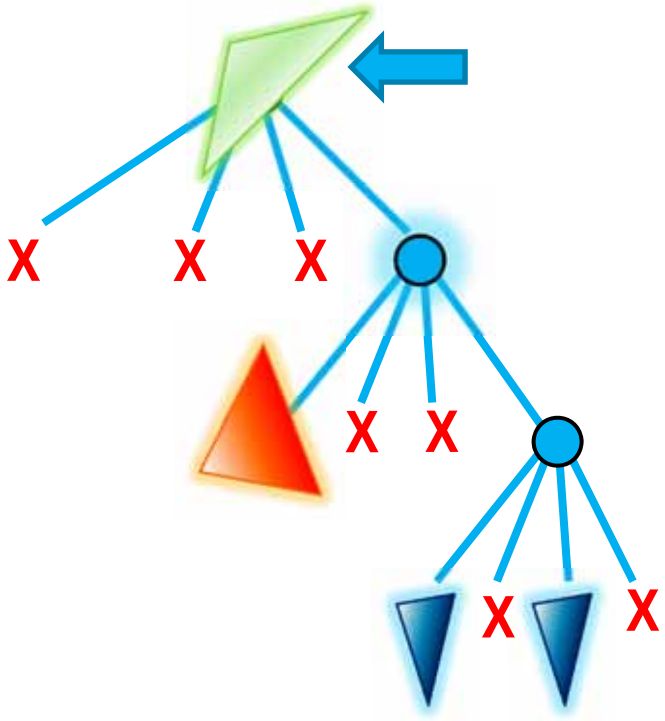
Oktalbaum/Octree

Traversierung

- Test gegen AABB der Wurzel (= AABB der gesamten Szene)
  - wenn Schnitt mit AABB → Test mit Objekten gespeichert in der Wurzel (Achtung: auch dieser Schnitt ● muss nicht der nahste sein weil der Wurzelknoten mit anderen überlappt!)



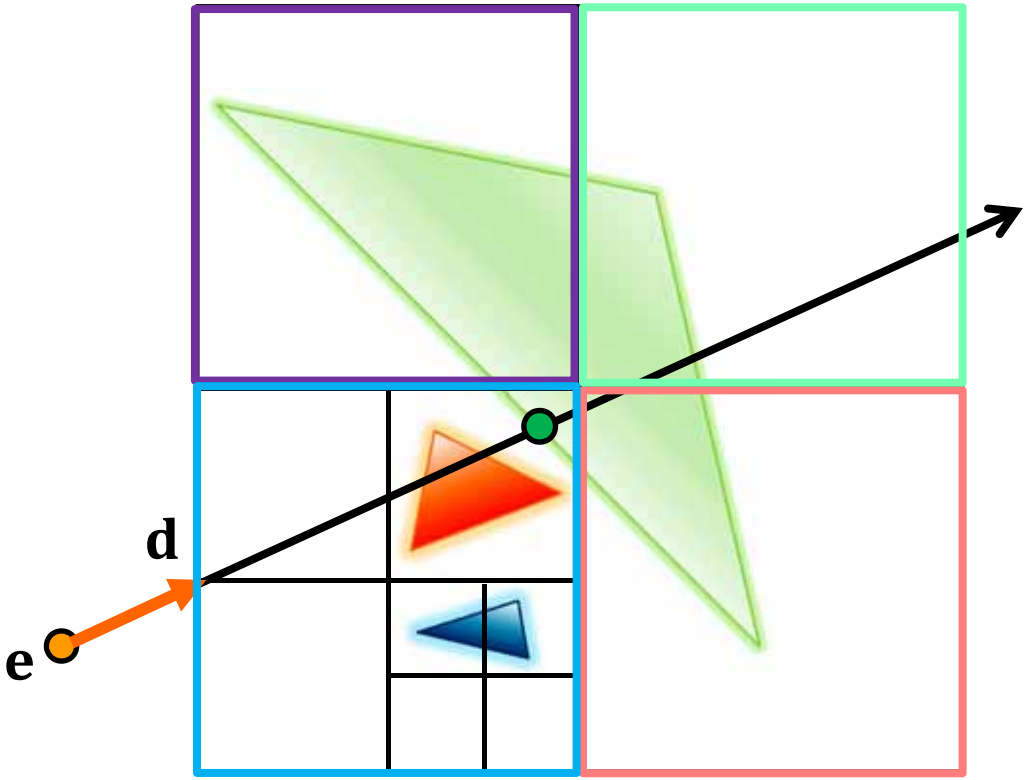
Quadtree



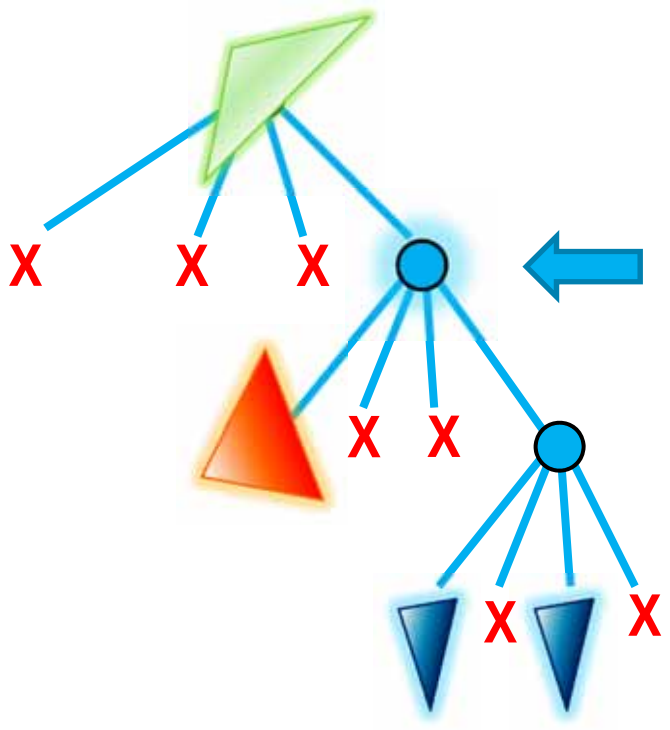
Oktalbaum/Octree

Traversierung

- überprüfe alle Kindknoten
  - ob sie näher als der gefundene Schnittpunkt sind
  - nur ☐ ist hier näher (abgesehen davon: die anderen 3 sind leer)



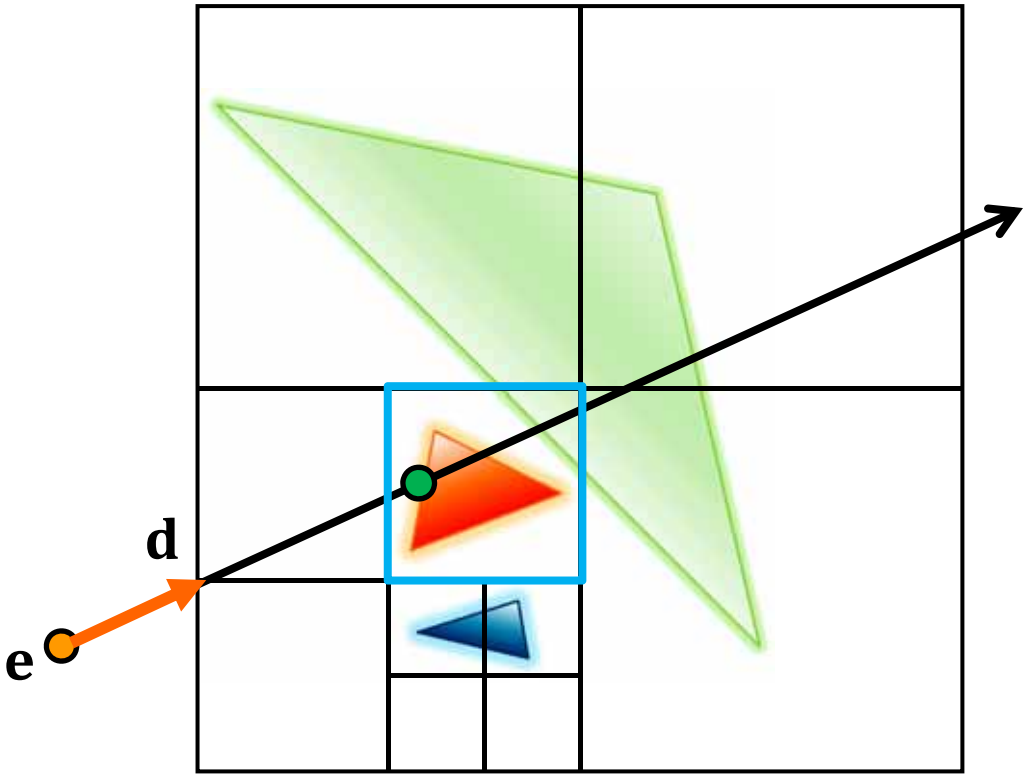
Quadtree



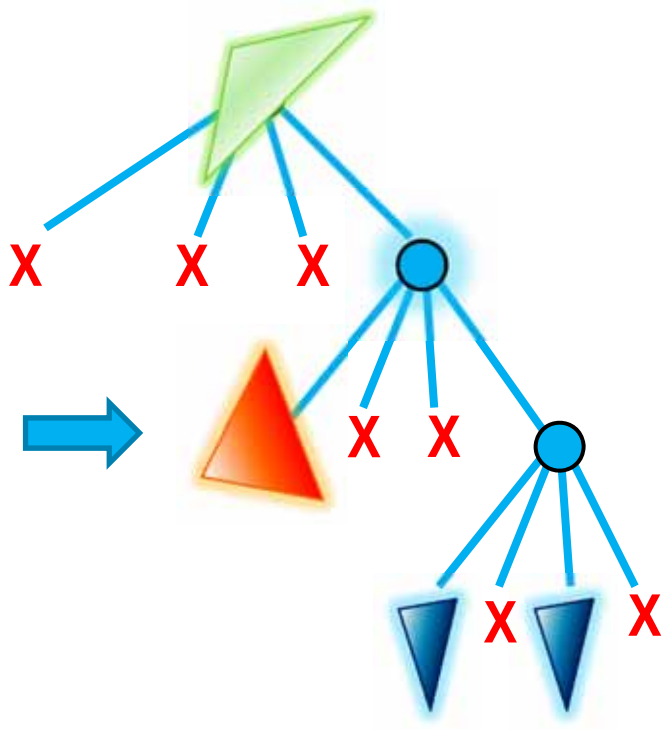
Oktalbaum/Octree

Traversierung

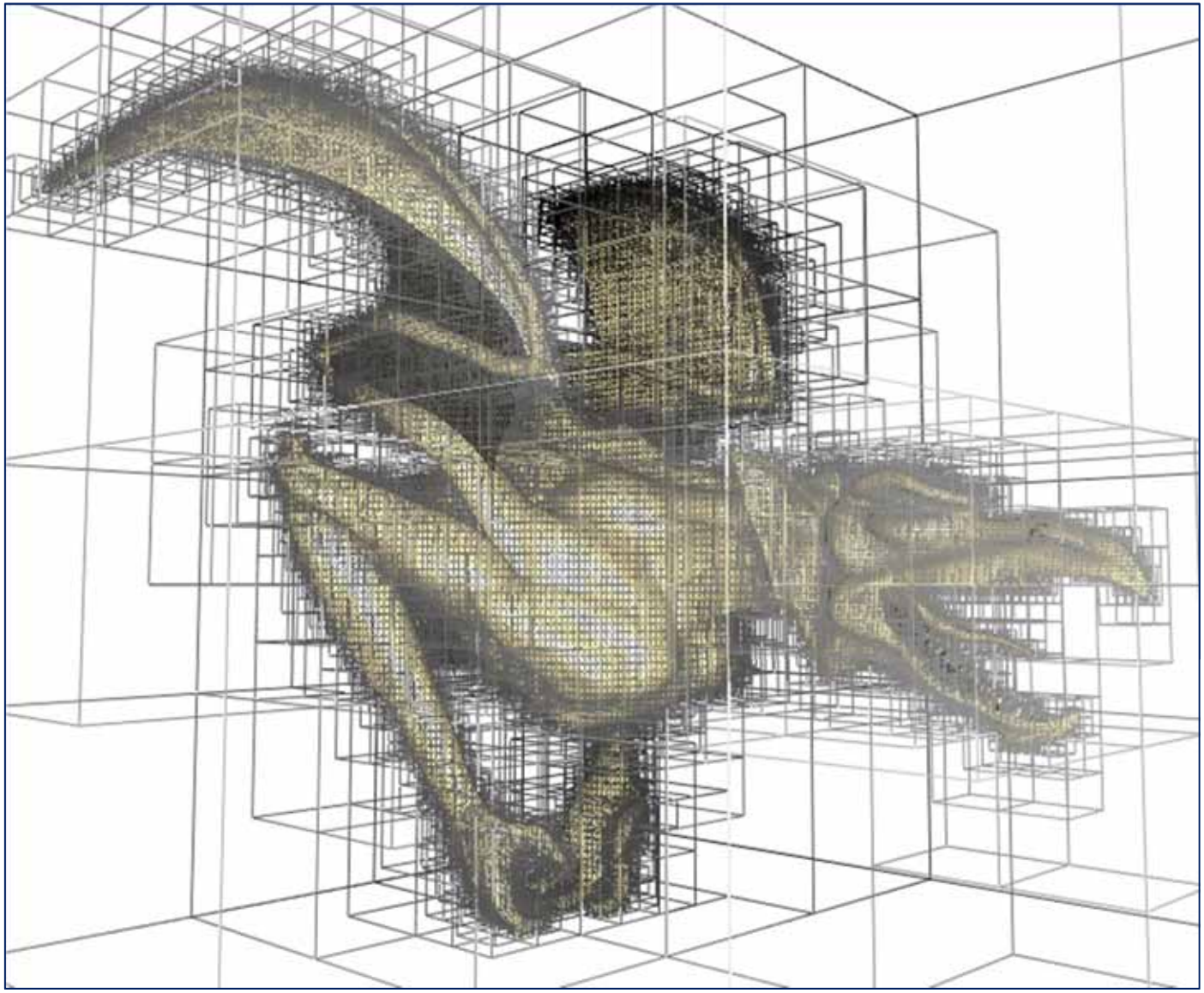
- Teste wiederum alle näheren, nicht-leeren Kindknoten
  - nur einer wird geschnitten und ist nicht leer → neuer Schnittpunkt
  - hier sind wir fertig – hätten wir nichts geschnitten müssten wir u. U. wieder in der Hierarchie aufsteigen und weitersuchen



Quadtree



Oktalbaum/Octree

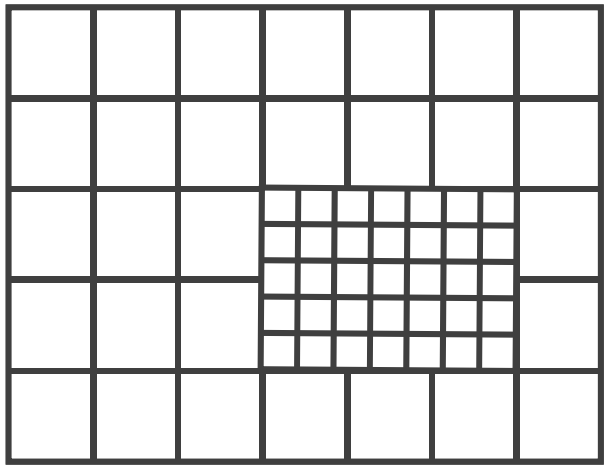


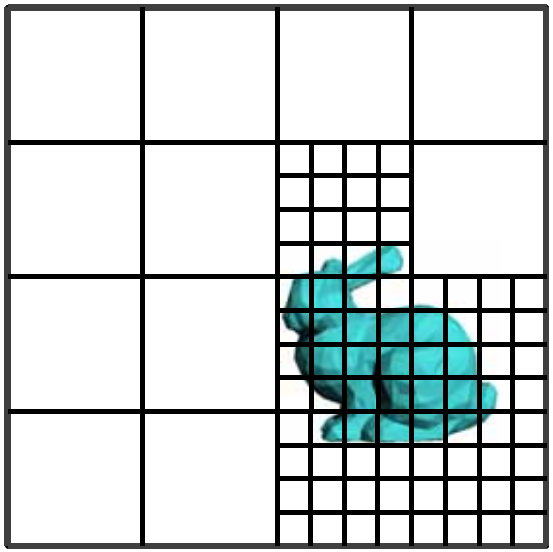
[Lefebvre 2004]

Verschachtelte Gitter

- Kombination der Ideen aus regulären Gittern und Octrees
  - nutze effiziente Traversierung von regulären Gittern
  - nutze Adaptivität einer hierarchischen Repräsentation

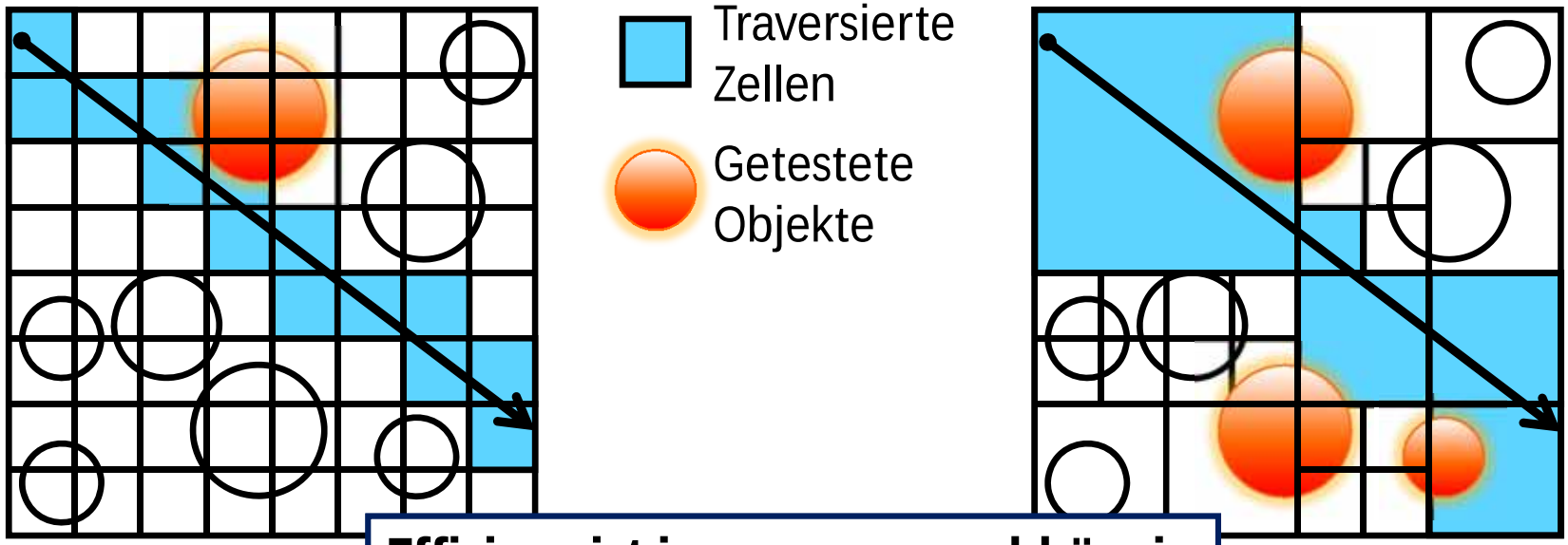
- Aufbau
  - grobes Gitter für Bounding-Box der Szene
  - Bounding-Box von Objekt/Primitiv-Gruppen mit dicht gepackten Objekten werden durch neues Gitter ersetzt
  - Problem: was sind gute Gruppen?





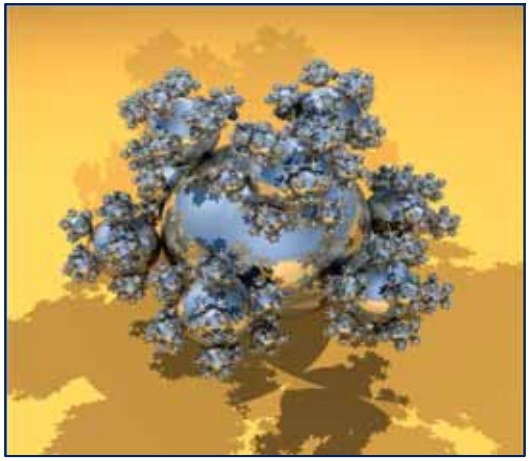
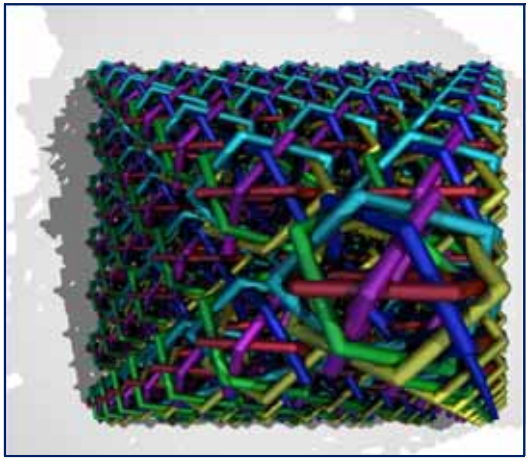
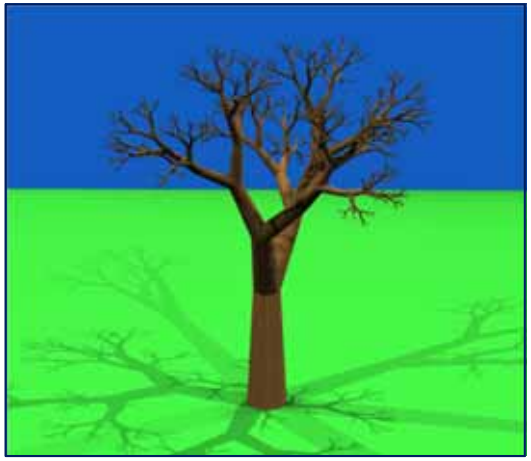
Vergleich Gitter vs. Octree

- Gitter
  - einfacher Algorithmus
  - schlecht bei ungleich verteilter Geometrie: kein effizientes Überspringen von leerem Raum
  - adaptiv durch verschachtelte Gitter
  - ähnlicher Aufwand für ähnliche Strahlen
- Octree
  - Adaptivität mit feiner Granularität
  - häufige Vertikalbewegung (auf und ab in der Hierarchie)
  - Traversierungskosten für zwei ähnliche Strahlen können stärker variieren (schwierig bei guter Ausnutzung paralleler Hardware)



Effizienz ist immer szenenabhängig

Vergleich

Zeit		Kugeln	Ringe	Baum
Uniformes Gitter	grob	244	129	1517
	fein	38	83	781
Hierarchisches Gitter		34	116	34

Vlastimil Havran, Best Efficiency Scheme Project

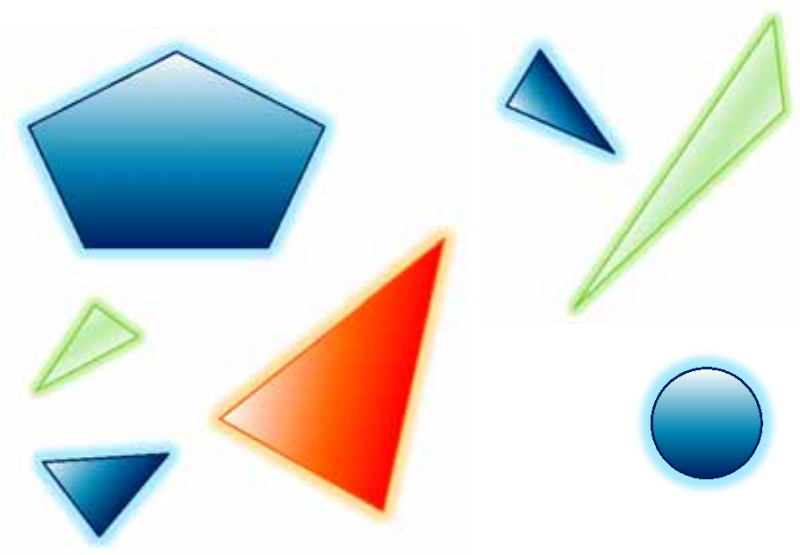
Inhalt: Räumliche Datenstrukturen

- Analyse der Kosten bei Ray Tracing
- Ansätze zur Beschleunigung von Ray Tracing
- Bildraumverfahren
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - **Bounding Volume Hierarchies**
  - **reguläre und adaptive Gitter**
  - **kD-Bäume und BSP-Bäume**

BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

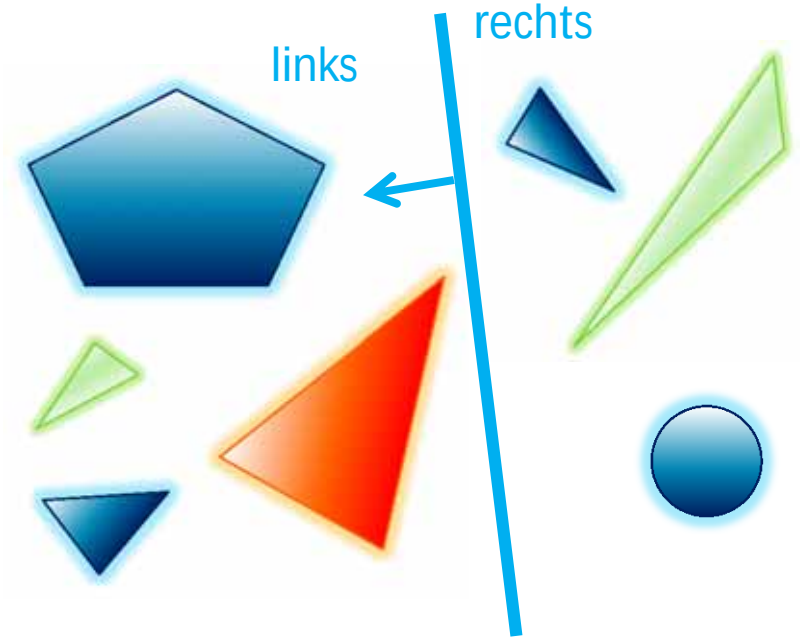
- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur

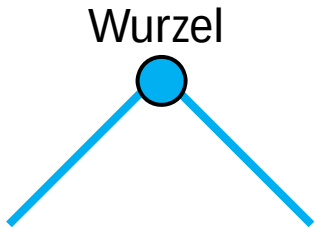


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur

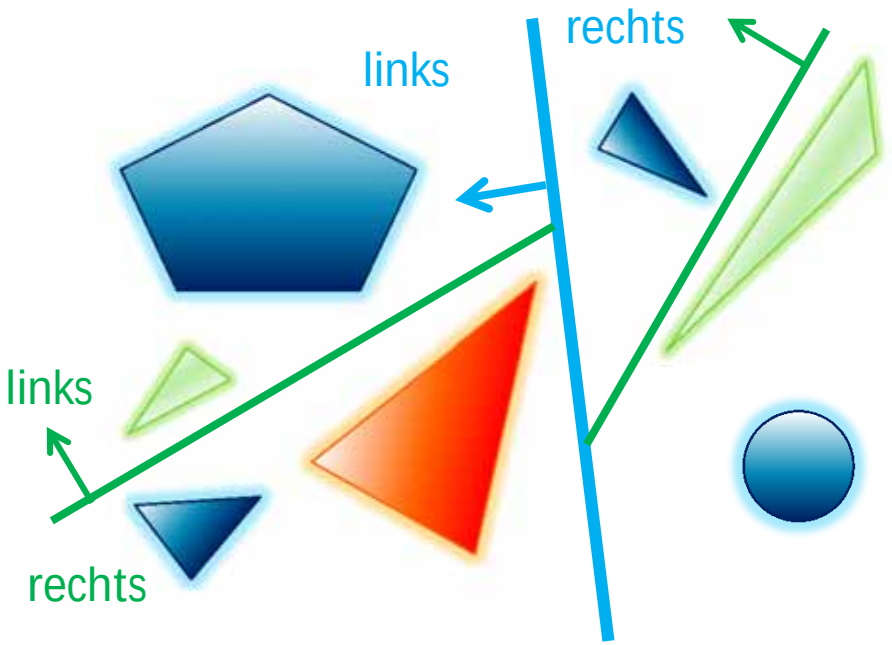


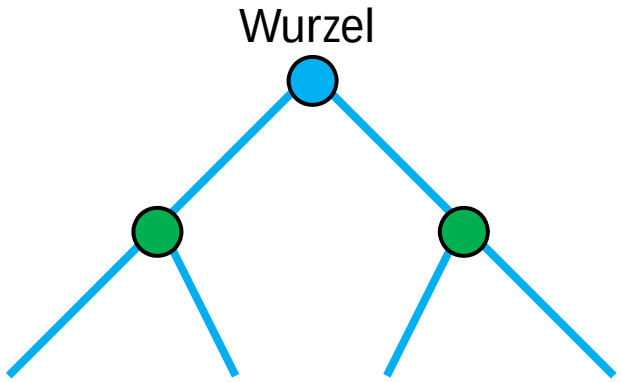


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur

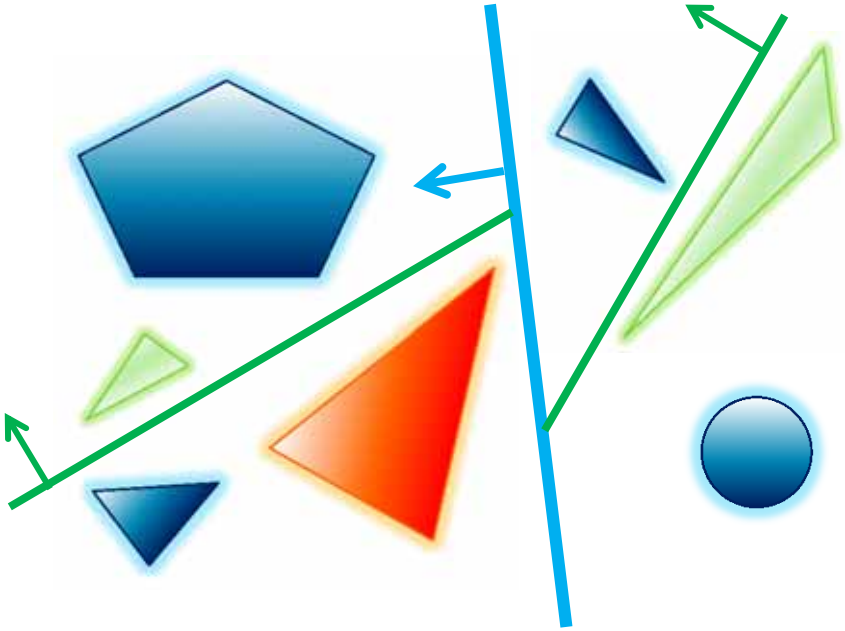


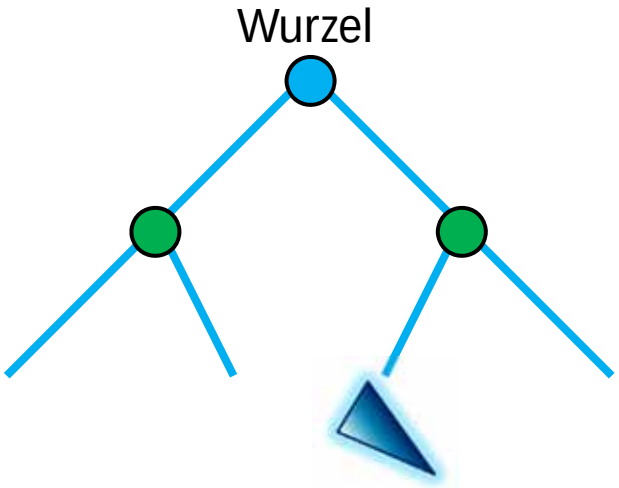


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur

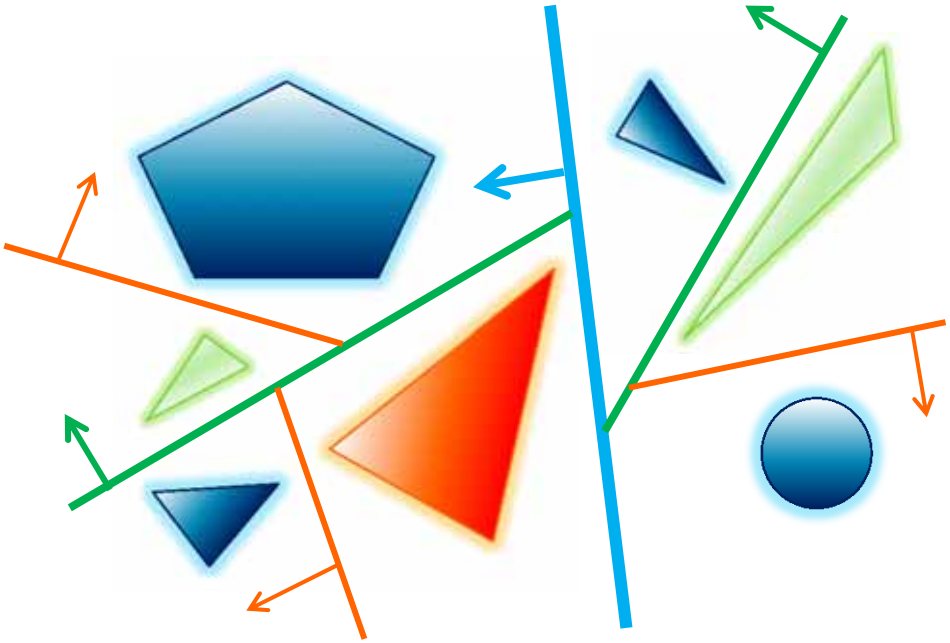


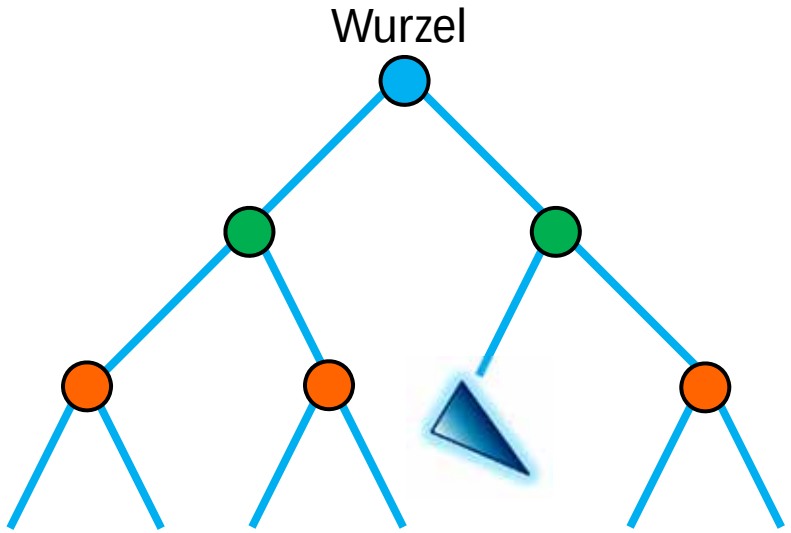


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur

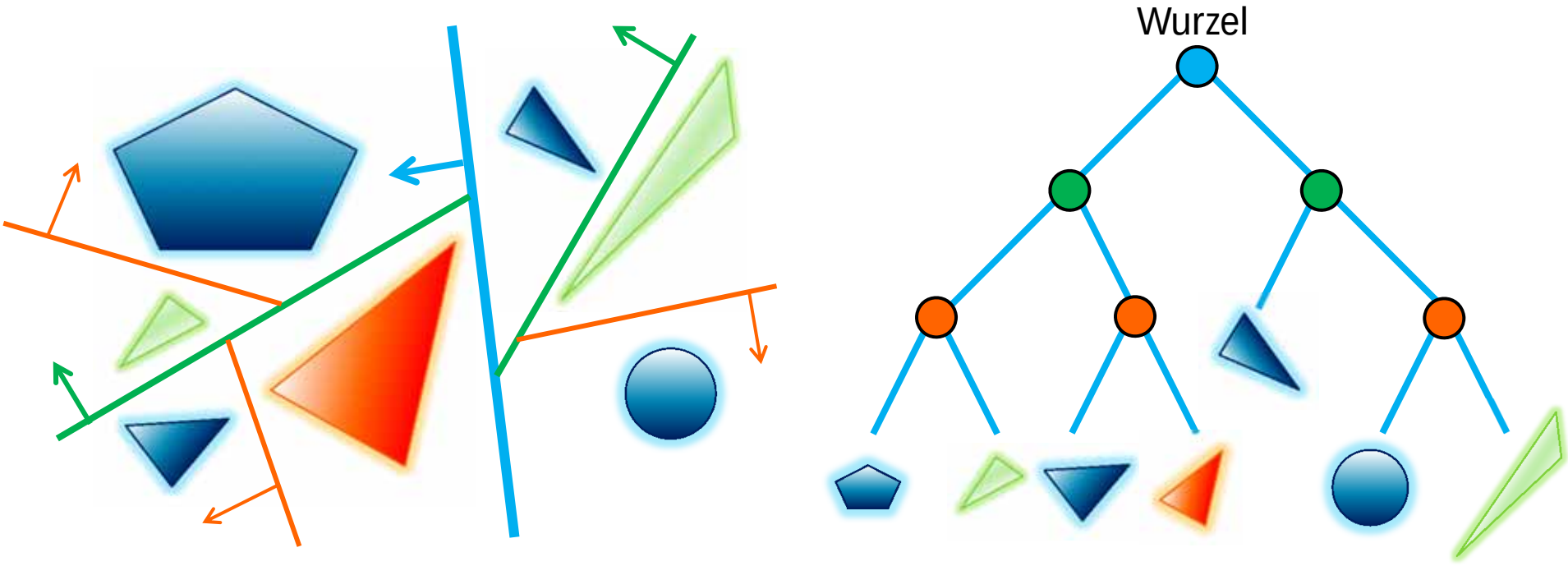




BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um den Raum rekursiv zu unterteilen
  → ergibt Binärbaumstruktur
- Unterschied: BSP-Bäume verwenden beliebig orientierte Ebenen, kD-Bäume nur Ebenen die senkrecht zur x-, y- oder z-Achse sind



BSP-Baum und kD-Baum

Konstruktion

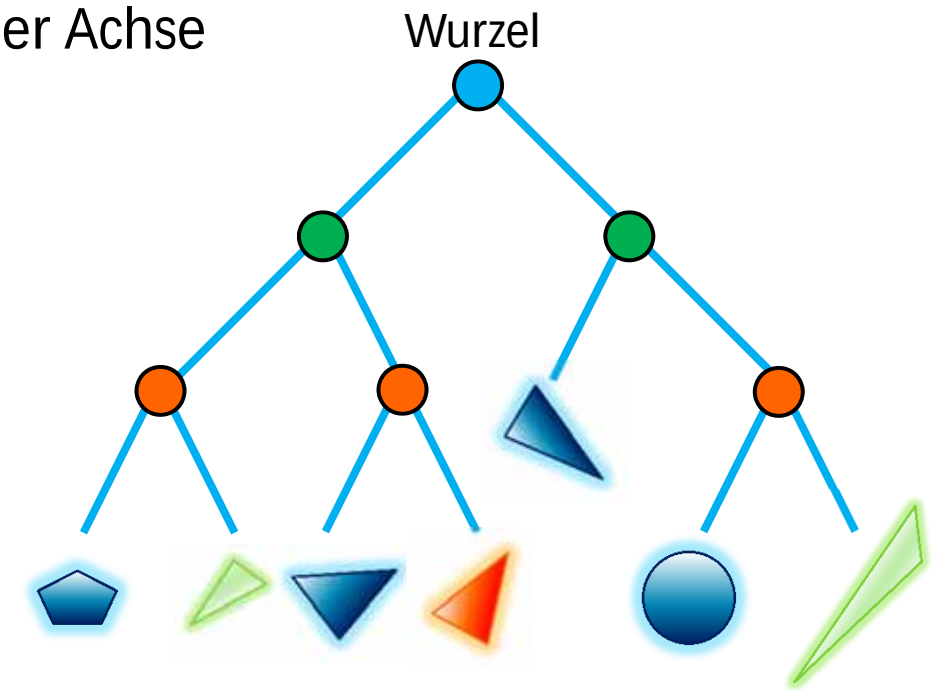
- Konstruktion findet rekursiv, ähnlich wie BVH, statt
  - initialisiere Wurzelknoten: enthält alle Objekte/Primitive der Szene
  - unterteile den Wurzelknoten rekursiv…
    - …bis ein Knoten nur noch eine vorgegebene maximale Anzahl Primitive enthält
    - …oder eine maximale Rekursionstiefe erreicht ist
      (um die heuristische Natur zu verdeutlichen: ein Erfahrungswert besagt Tiefe $= 8 + 1.3\log(n)$ für n Primitive)
  - teile die Primitive auf und bilde einen „linken“ und einen „rechten“ Kindknoten

BSP-Baum und kD-Baum

Eigenschaften (im Vergleich mit BVH)

- echte Raumunterteilung: Knoten überlappen sich nicht (vgl. BVH)
- gute Anpassung an Geometrie möglich (v.a. bei BSP-Bäumen)
- die Blattknoten speichern die Primitive
- innere Knoten speichern die Split-Ebenen
  - BSP-Baum: Normale der Ebene und Abstand zum Ursprung
  - kD-Baum: die zur Ebene senkrechte Achse und die Position entlang dieser Achse
  - Zeiger auf die Kindknoten

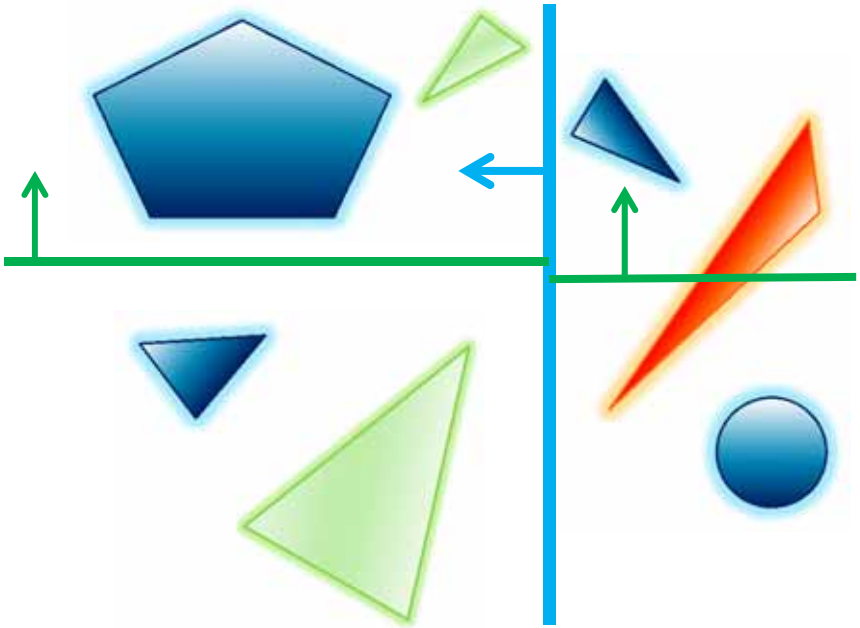
- kD-Bäume sind einfacher zu konstruieren (keine Ebenen mit freier Orientierung) und werden daher häufiger eingesetzt

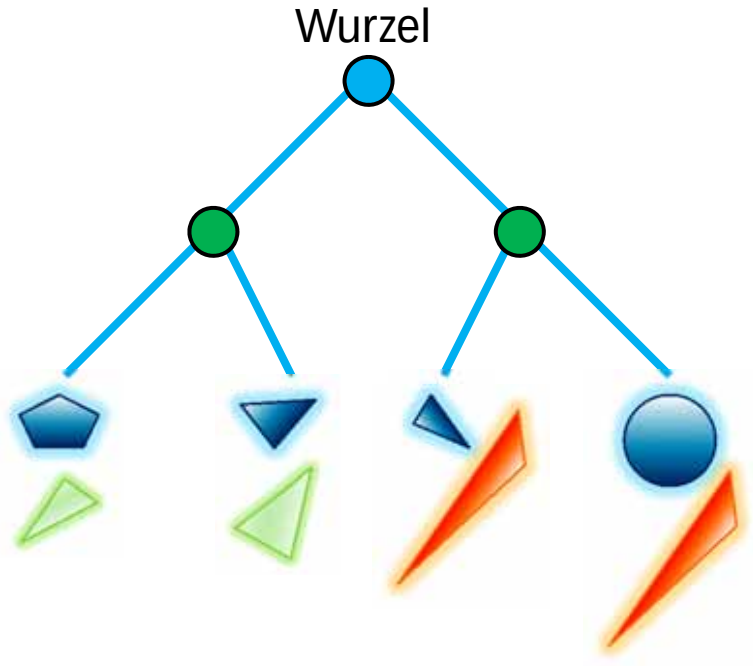


kD-Bäume

Eigenschaften

- Split-Ebenen senkrecht zur x-, y- oder z-Achse
- Primitive die eine Split-Ebene schneiden
  - werden meist in beide Kindknoten eingefügt:
    ein Primitiv kann also in mehreren Knoten vorkommen

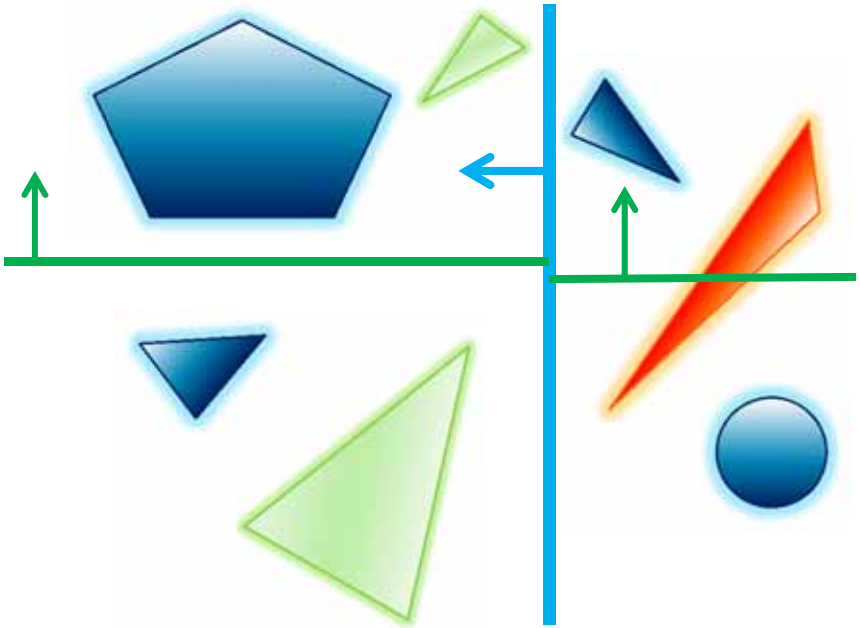


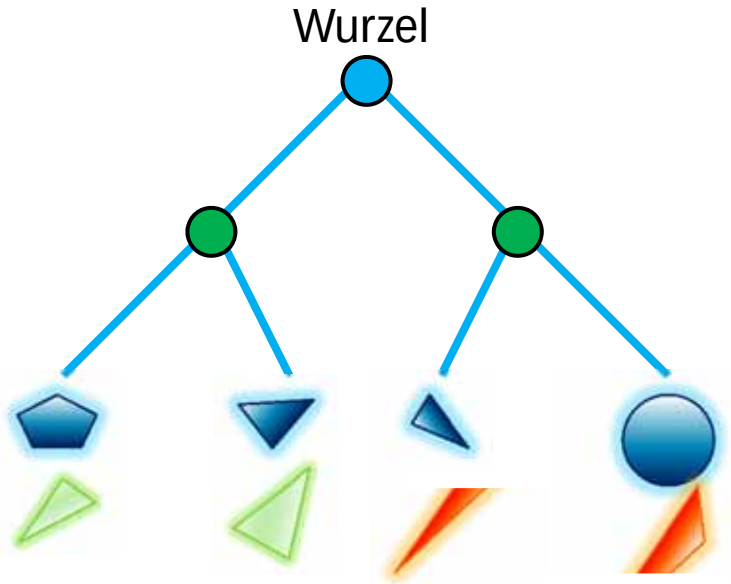


kD-Bäume

Eigenschaften

- Primitive die eine Split-Ebene schneiden
  - werden meist in beide Kindknoten eingefügt:
    ein Primitiv kann also in mehreren Knoten vorkommen
  - optional: zerschneide das Primitiv
    (in der Regel macht man das aber **nicht**!)





kD-Baum und BVH Konstruktion

Aufteilung (Split) eines Knotens für kD-Baum- und BVH-Konstruktion

- Bestimmung der „optimalen“ Split-Ebene
  - räumliches Mittel (spatial median):
    teile Knoten in der Mitte entlang der Achse der größten Ausdehnung **oder**
    teile erst entlang der x-, dann y-, dann z-, dann wieder x-Achse, usw.
    - Aufbau $O(n \log n)$
  - Objektmittel (object median):
    teile Knoten, so dass linker und rechter Kindknoten gleich viele Primitive enthalten (typischerweise entlang größter Ausdehnung)
    - Aufbau $O(n \log^2 n)$, wenn eine $O(n \log n)$ Sortierung verwendet wird
  - Kostenfunktion (Surface Area Heuristic)
    - Ziel: im Mittel sollen zufällige Strahlen, die den betrachteten Knoten schneiden, den gleichen Aufwand verursachen, egal welcher der Kindknoten weiter traversiert wird
    - resultiert in einem – **bei der Traversierung** – balancierten Baum
    - Aufbau $O(n \log^2 n)$ möglich
      http://www.cgg.cvut.cz/members/havran/ARTICLES/ingo06rtKdtree.pdf
- Anm. die Wahl einer guten Split-Ebene ist bei kD-Bäumen schon schwierig – bei BSP-Bäumen ist der „Suchraum“ noch viel größer

Konstruktion „guter“ kD-Bäume und BVH

- wie unterteilt man Knoten am besten?
  - so das Ray Tracing schnell geht ☺
  - Idee: stelle eine Kostenfunktion auf und minimiere

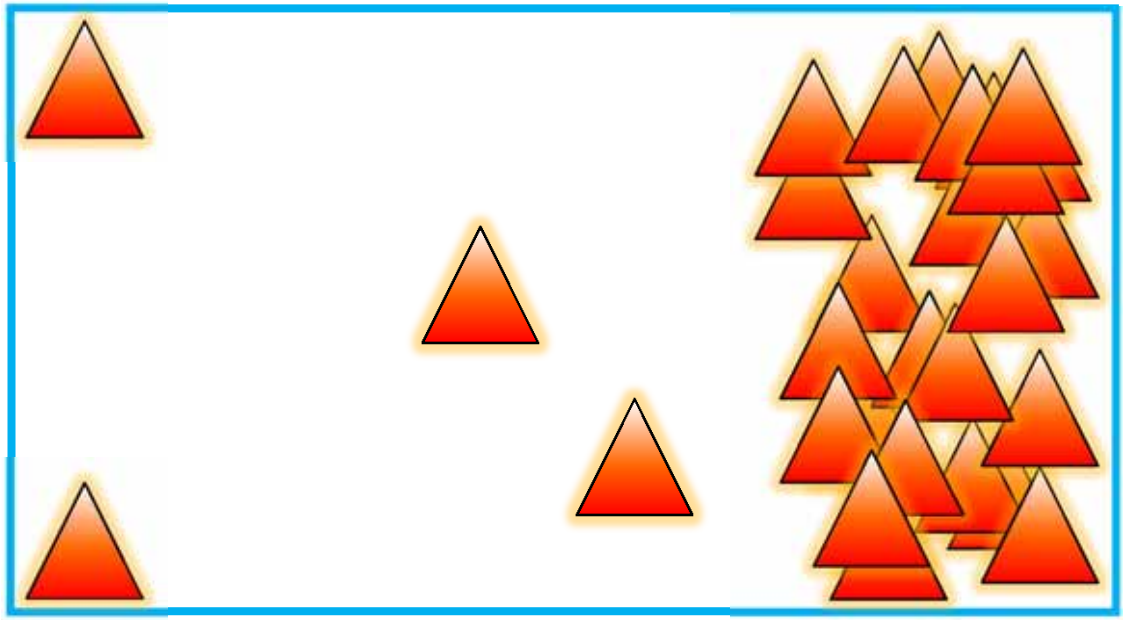
- Kosten für das Traversieren eines unterteilten Knotens

$$C = C_T + P(\text{treffe } L)C(L) + P(\text{treffe } R)C(R)$$

  - wichtig: Unterteilung nur dann, wenn die Kosten dadurch geringer werden, als den Schnitt mit allen Objekten des Knoten zu berechnen
  - C_T: Kosten für das Traversieren des (neuen) inneren Knotens (= Entscheidung, ob mit linkem oder rechtem Kind fortgefahren wird)
  - $P(\text{treffe } L), P(\text{treffe } R)$: Wahrscheinlichkeit, dass der Strahl den linken bzw. rechten Kindknoten trifft
  - $C(L), C(R)$: Kosten für das Traversieren und Schnitttests des linken und rechten Kindknotens

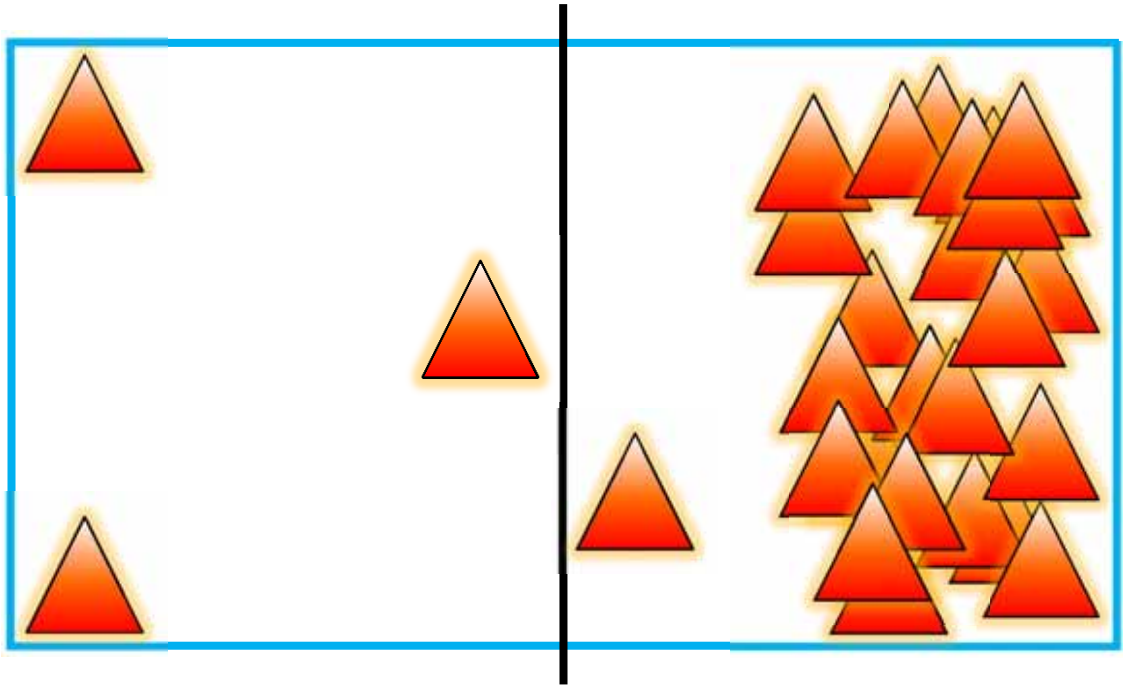
Konstruktion „guter“ kD-Bäume und BVH

- Beispiel
  - SAH ist für kD- und BSP-Bäume, sowie für BVH einsetzbar
  - wir betrachten während der Konstruktion Bounding Boxes von Objekten eines Knotens auch wenn wir kD- und BSP-Bäume erzeugen



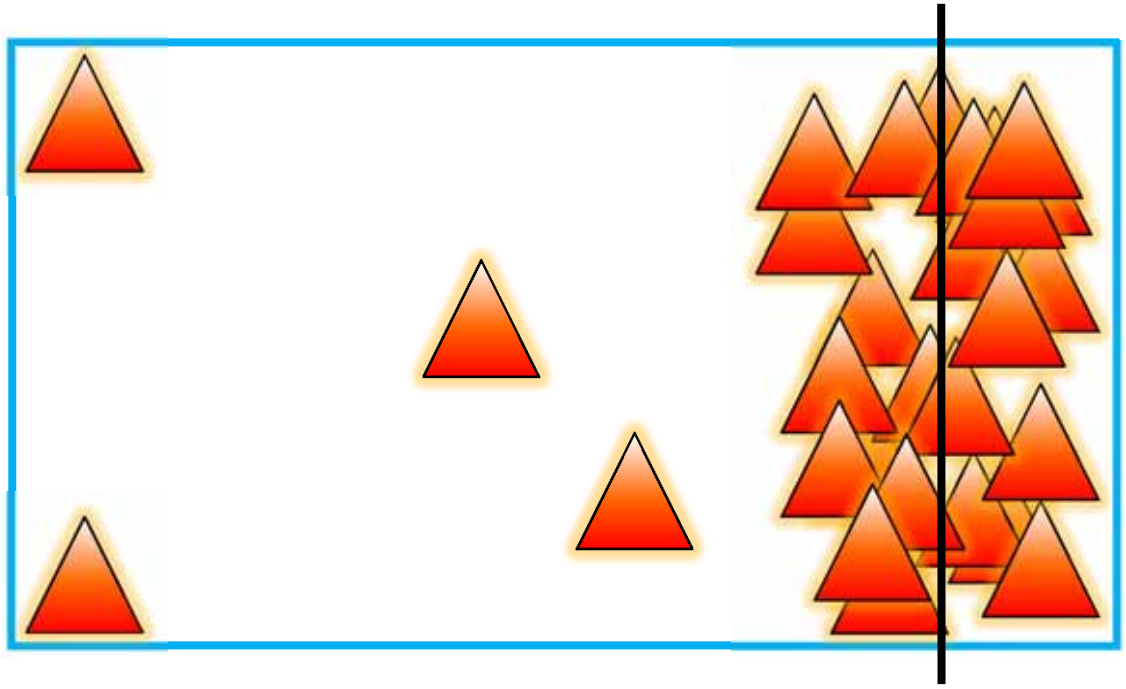
Konstruktion „guter“ kD-Bäume und BVH

- Beispiel: Unterteilung in der Mitte
  - Motivation: gleiche Wahrscheinlichkeiten für das Treffen des linken bzw. rechten Kindknotens aus
  - aber: keine Berücksichtigung von $C(L)$ bzw. $C(R)$



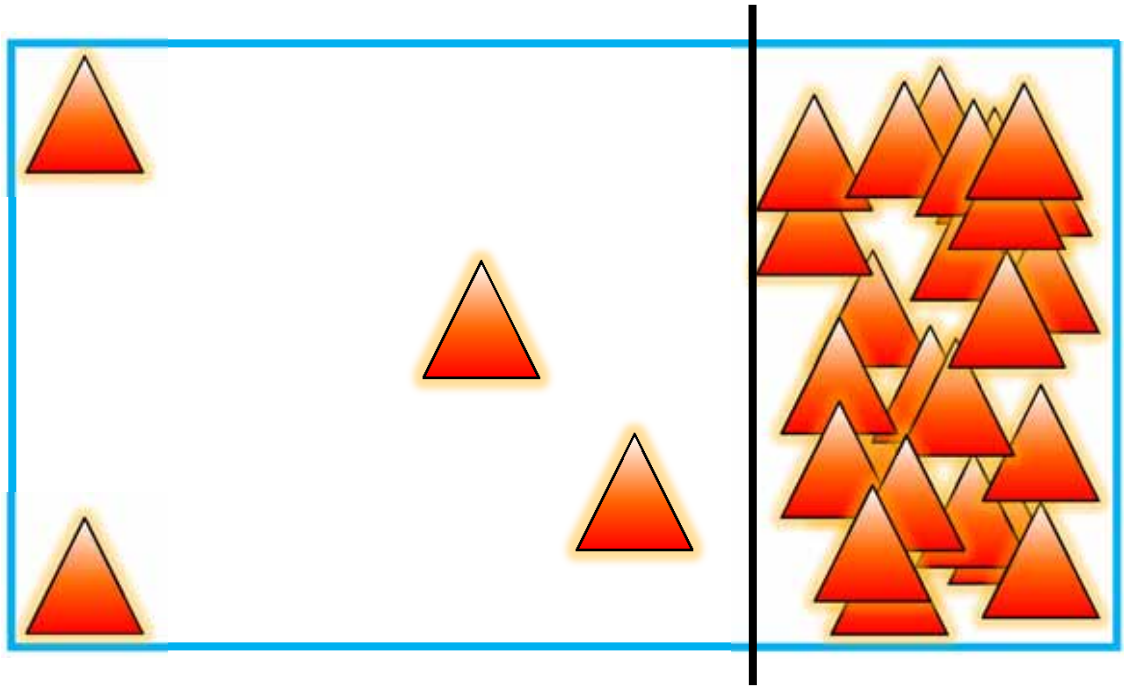
Konstruktion „guter“ kD-Bäume und BVH

- Beispiel: Unterteile nach Objekt-Median
  - Motivation: gleiche $C(L)$ und $C(R)$ aus, d.h. in etwa genauso viele Dreiecke in jedem Kindknoten, die geschnitten werden müssen
  - aber: keine Berücksichtigung der Wahrscheinlichkeiten für das Treffen des linken bzw. rechten Kindknotens



Konstruktion „guter“ kD-Bäume und BVH

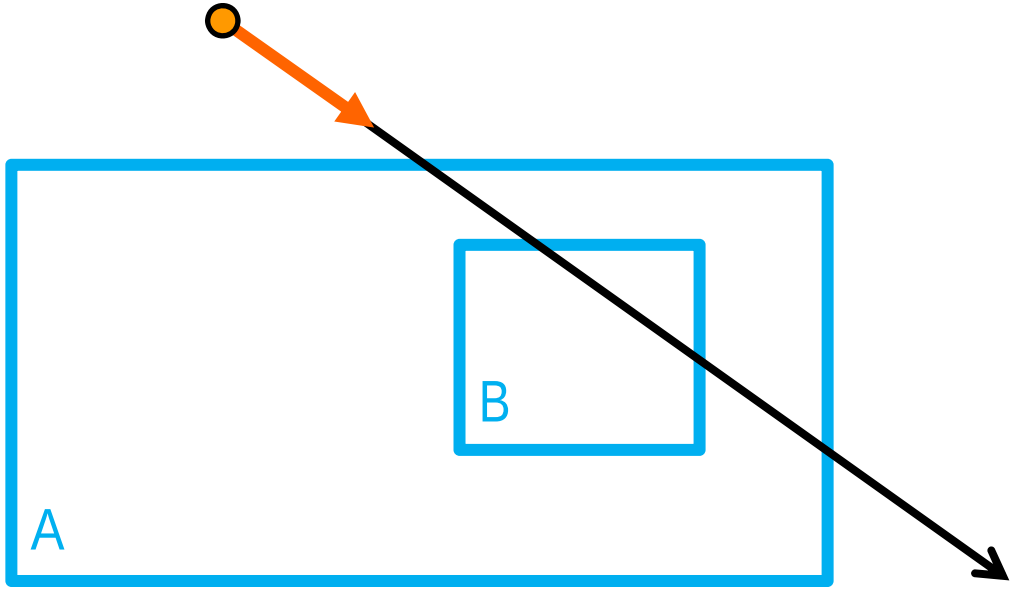
- Beispiel: kostenoptimierte Unterteilung
  - Isolation von komplexen Szenenteilen
  - Erzeuge große dünnbesetzte/leere Bereiche



Konstruktion „guter“ kD-Bäume und BVH

Eine Konsequenz aus Croftons Theorem (gilt allg. für konvexe Objekte)

- geg. eine Box B, komplett enthalten in einer Box A
- die Wahrscheinlichkeit, dass ein zufälliger Strahl, der A schneidet, auch B schneidet ist: $SA(B)/SA(A)$
- $SA(.)$ ist die Oberfläche einer Box



Surface Area Heuristics (SAH)

Kostenfunktion

- Kostenfunktion für das Unterteilen eines kD-Baum/BVH Knotens p

$$C = C_T + \frac{SA(B_l)}{SA(B_p)}|P_l| + \frac{SA(B_r)}{SA(B_p)}|P_r|$$

  - B_l, B_r und B_p sind die Bounding Boxes der Primitive im linken und rechten Kindknoten, bzw. des betrachteten Knotens p
  - $|P_l|$ und $|P_r|$ Anzahl der Primitive im linken und rechten Kindknoten
    - zur Abschätzung der Kosten für die Kindknoten
  - C_T sind die Kosten der Traversierung eines kD-Baum/BVH Knotens relativ zu den Kosten eines Strahl-Primitiv-Schnitttests
    - durch den konstanten Overhead C_T lohnt sich Unterteilung bei kleineren Knoten irgendwann nicht mehr
- Ziel: finde die Unterteilung, die die Kostenfunktion minimiert

Surface Area Heuristics (SAH)

Optimierung/Bestimmung der Unterteilung

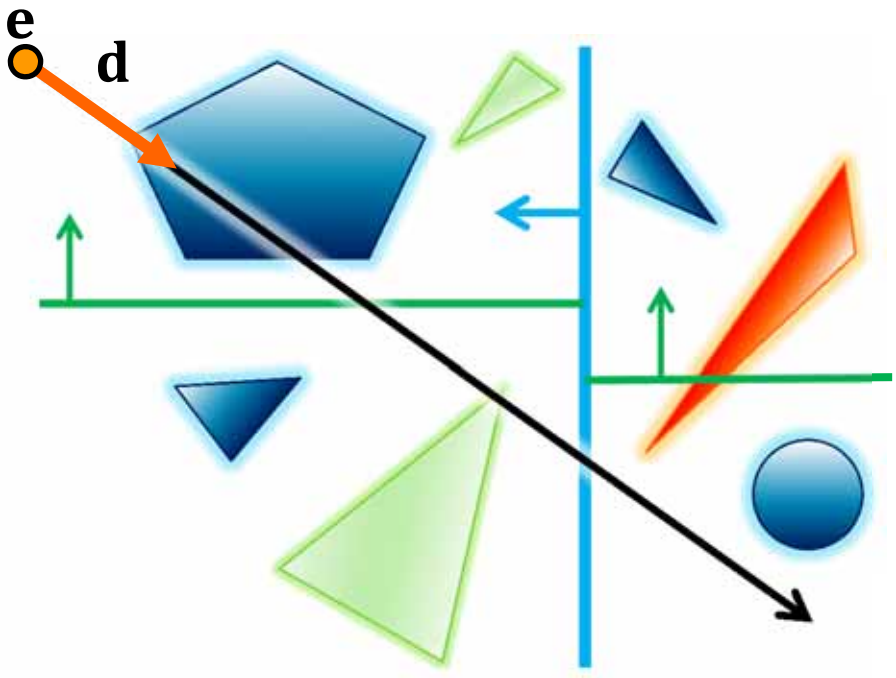
- aufwändig, da der Suchraum groß sein kann
- daher meist approximative Konstruktion
  - erzeuge einige Unterteilungskandidaten
    (Ebenen zur Unterteilung entlang der x-, y- und z-Achse)
    - natürlich innerhalb der Bounding Box B_p
  - berechne Kostenfunktion für jeden Kandidaten
    (nicht vernachlässigbare Kosten!)
  - wähle Kandidaten mit den niedrigsten Kosten

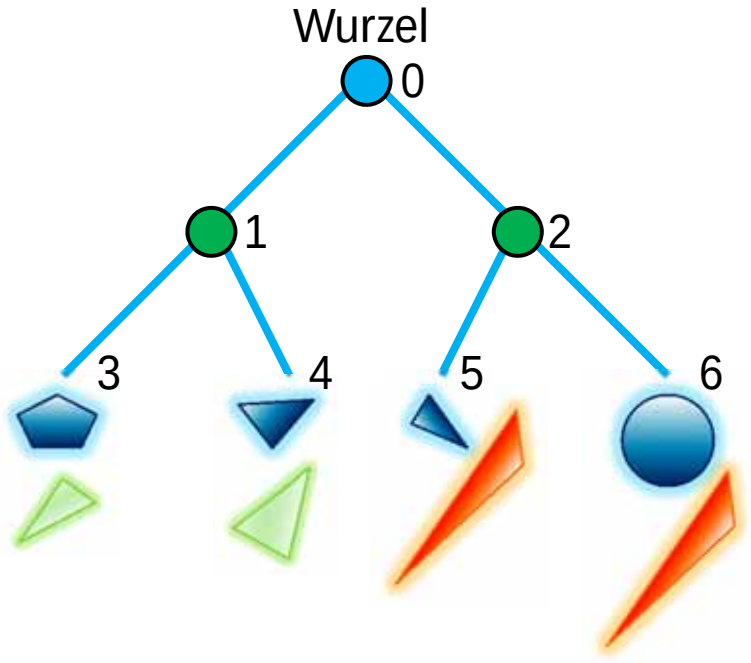
- gut konstruierte kD-/BSP-Bäume bzw. BVHs können um ein Vielfaches schneller sein, als schlecht konstruierte

BSP-Baum und kD-Baum

Traversierung

- Strahl $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$, mit $t_{min} \leq t \leq t_{max}$ (t_{min}, t_{max} z.B. aus Schnitt mit AABB der ganzen Szene, oder $t_{min} = 0, t_{max} = \infty$)
- Ziel: Traversierung der Knoten „von vorne nach hinten“
- verwende Stack, um die Knoten für die Bearbeitung zu halten
- zu Beginn enthält der Stack den Wurzelknoten (0)



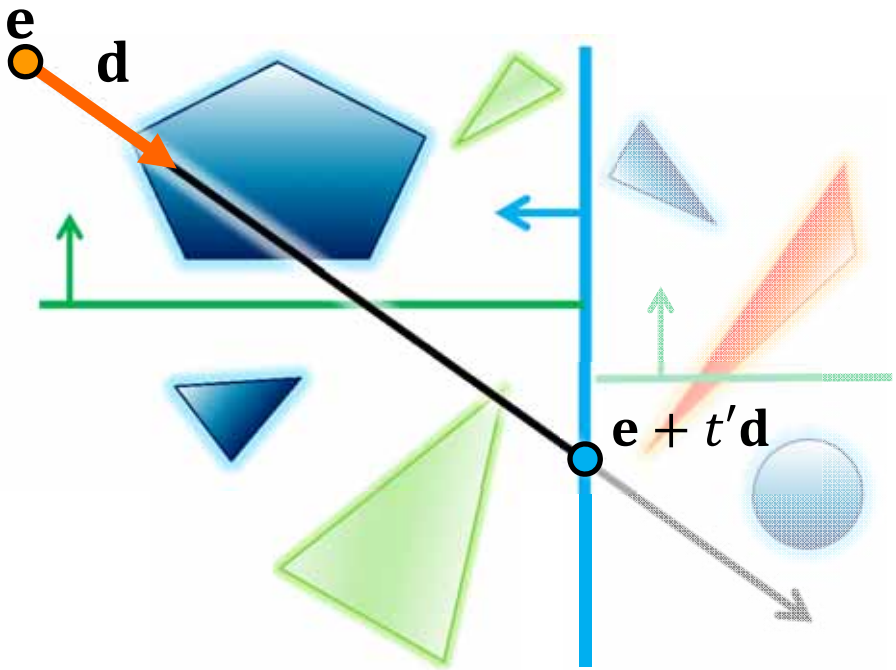


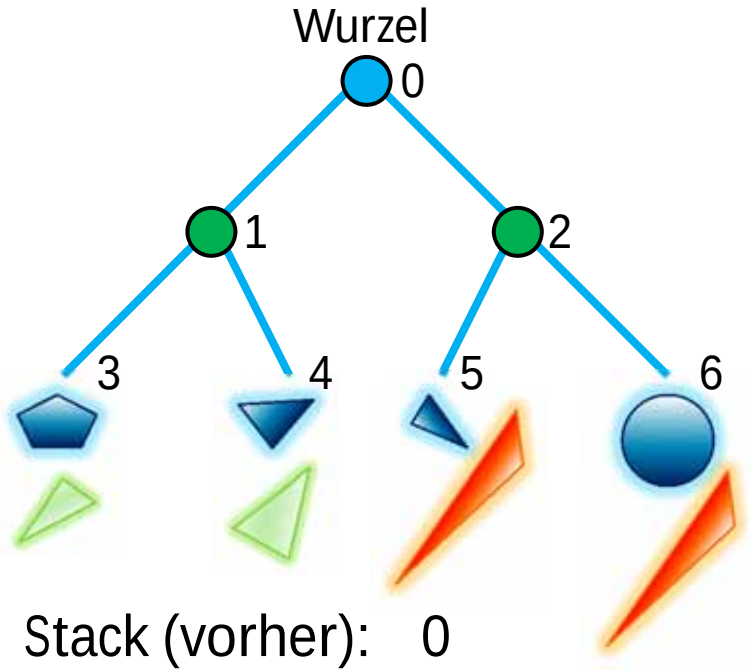
Stack: 0

BSP-Baum und kD-Baum

Traversierung

- Strahl $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$, mit $0 \leq t \leq t_{max}$
- schneide Strahl mit der Split-Ebene (entnehme ersten Knoten vom Stack)
  - der Schnitt bei liegt t' mit $0 \leq t' \leq t_{max}$
  - fahre mit beiden Kindern rekursiv fort
  - zuerst der Kindknoten, der $\mathbf{e}$ enthält





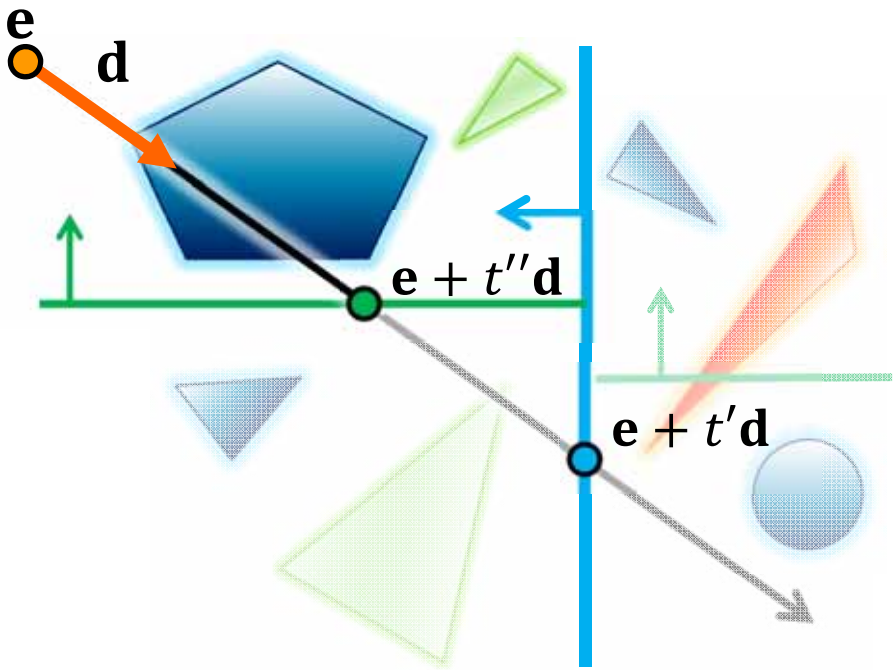
Stack (vorher): 0

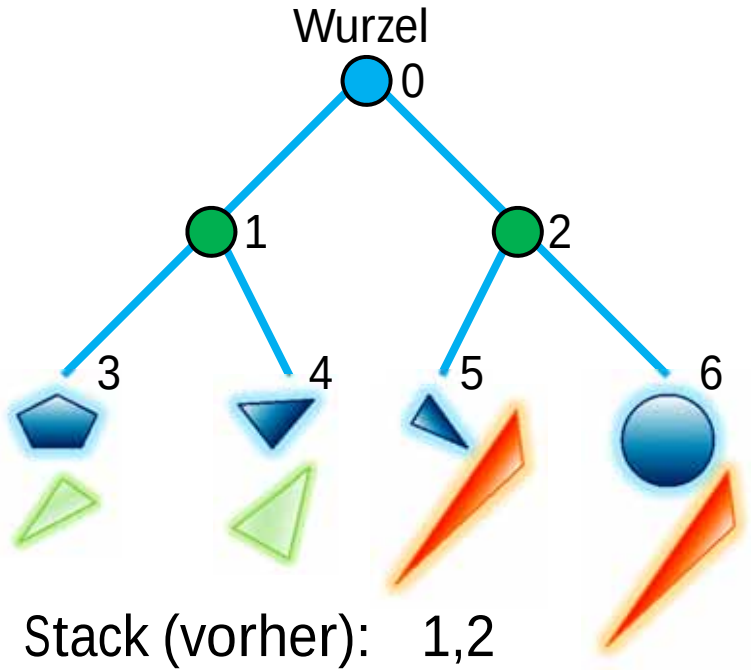
Stack (nachher): 1,2

BSP-Baum und kD-Baum

Traversierung

- schneide Strahl mit der Split-Ebene (entnehme Knoten 1 vom Stack)
  - liegt der Schnitt bei t'' mit $0 \leq t'' \leq t'$ (das muss nicht so sein)
    - dann fahre mit beiden Kindern rekursiv fort
    - zuerst der Kindknoten, der $\mathbf{e}$ enthält
  - (der andere Fall kommt gleich)





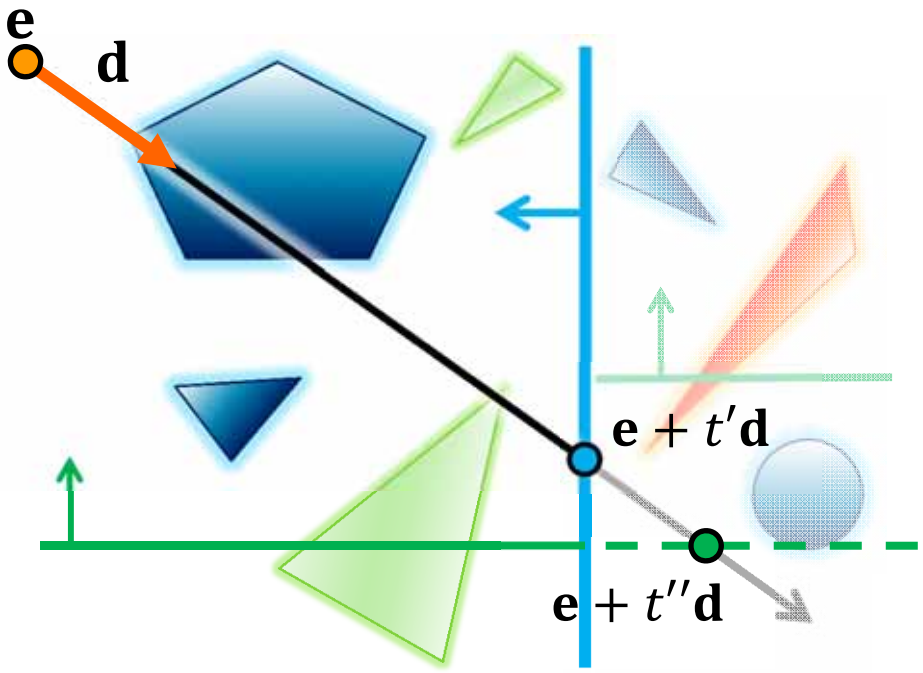
Stack (vorher): 1,2

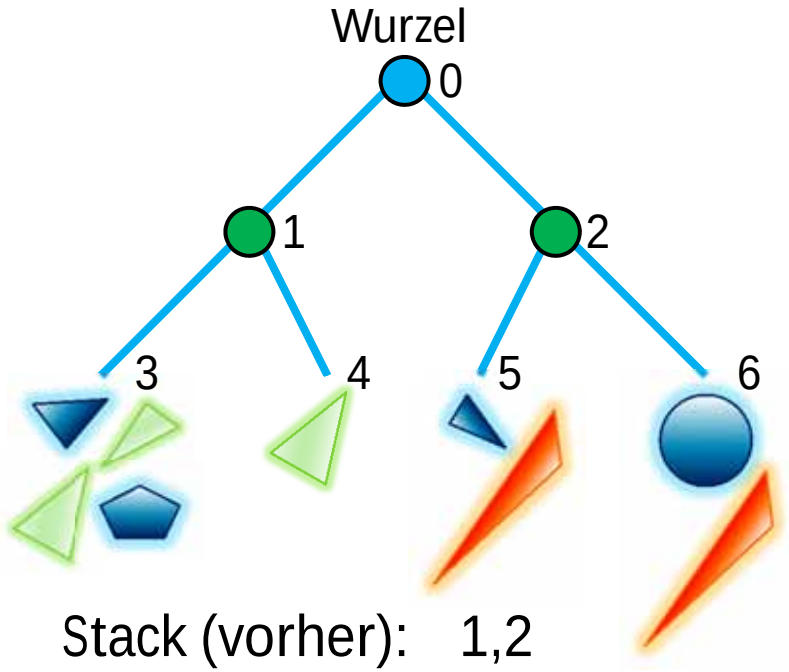
Stack (nachher): 3,4,2

BSP-Baum und kD-Baum

Traversierung: „der andere Fall“

- **Achtung: nur zur Illustration, hat nichts mit dem Beispiel zu tun!**
- schneide Strahl mit der Split-Ebene (entnehme Knoten 1 vom Stack)
  - hier liegt der Schnitt bei t'' mit $t'' > t'$
  - nur der Kindknoten, der $\mathbf{e}$ enthält wird traversiert (hier Knoten 3)
  - hier: bevor Knoten 4 erreicht wird, verlassen wir den Teilbaum





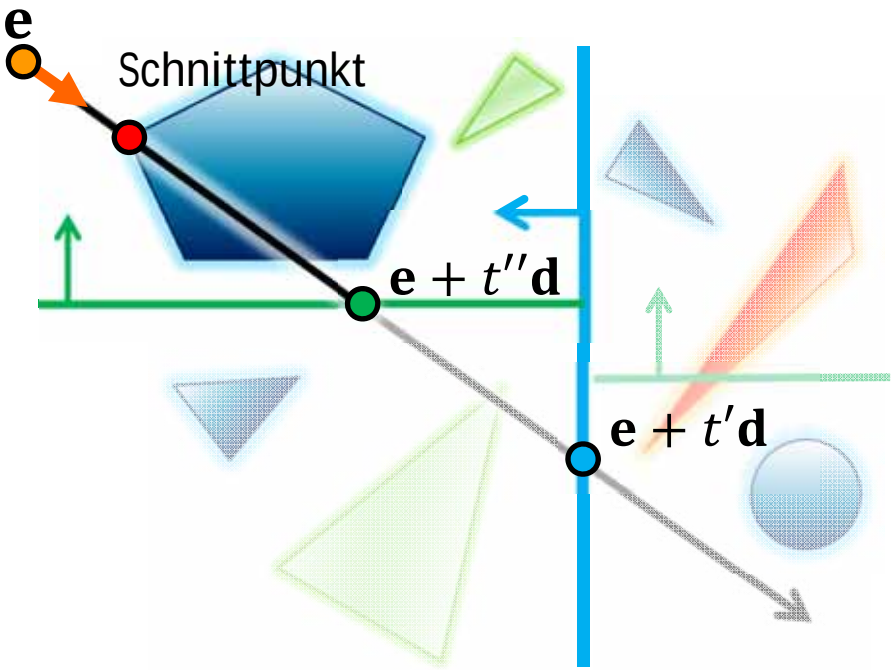
Stack (vorher): 1,2

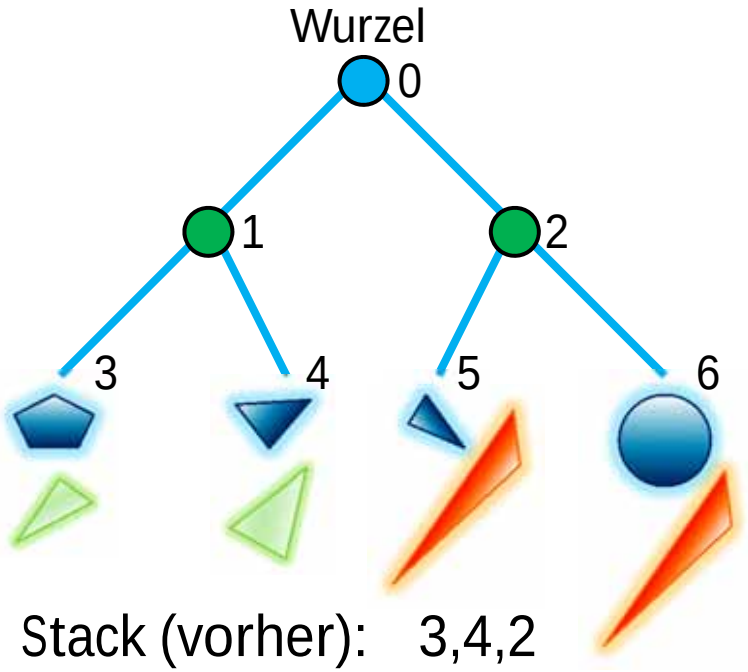
Stack (nachher): 3,2

BSP-Baum und kD-Baum

Traversierung

- wenn Blattknoten erreicht, dann teste Schnittpunkte mit Primitiven
- gebe Schnittpunkt zurück, wenn innerhalb $[0, t'')$
  (bzw. $[t'', t')$, wenn Primitive im hinteren Halbraum getestet wurden)
- Vorteil „echter“ Raumunterteilung: kein möglicher Schnitt weiter hinten
- in diesem Beispiel sind wir jetzt fertig!



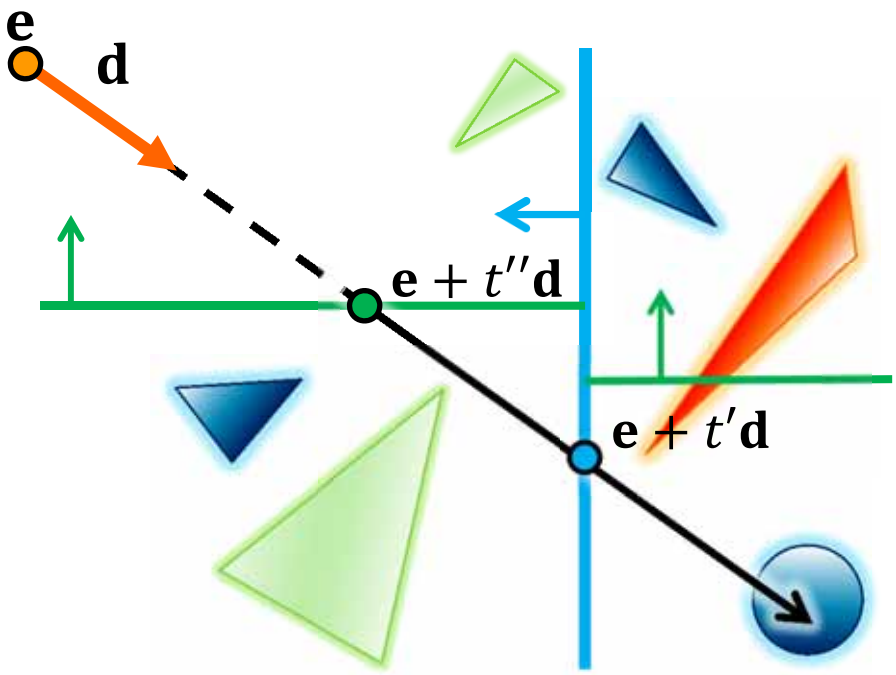


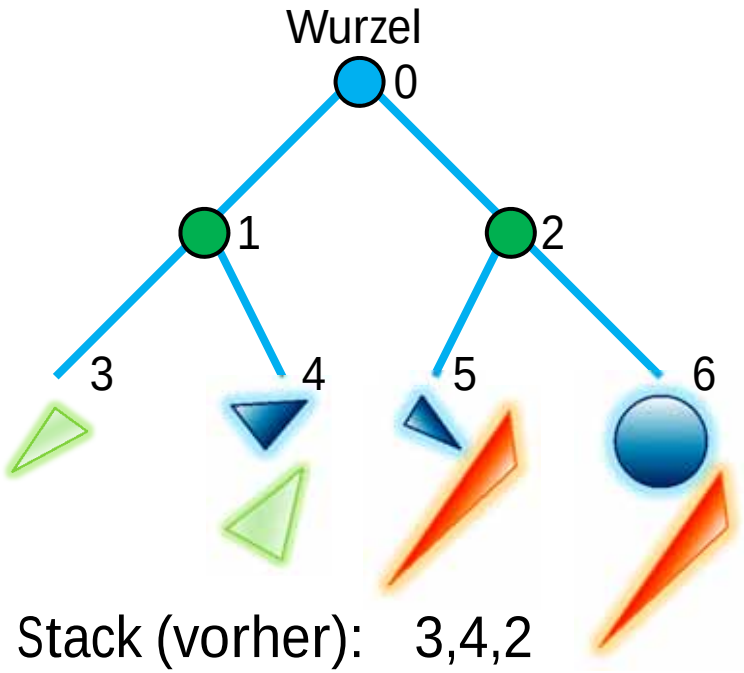
Stack (vorher): 3,4,2
Stack (nachher): 4,2

BSP-Baum und kD-Baum

Traversierung

- keine Schnittpunkte im Knoten 3 gefunden
- fahre mit nächstem Knoten vom Stack fort → Knoten 4
- auch dort gibt es keine Schnittpunkte für $t'' \leq t \leq t'$





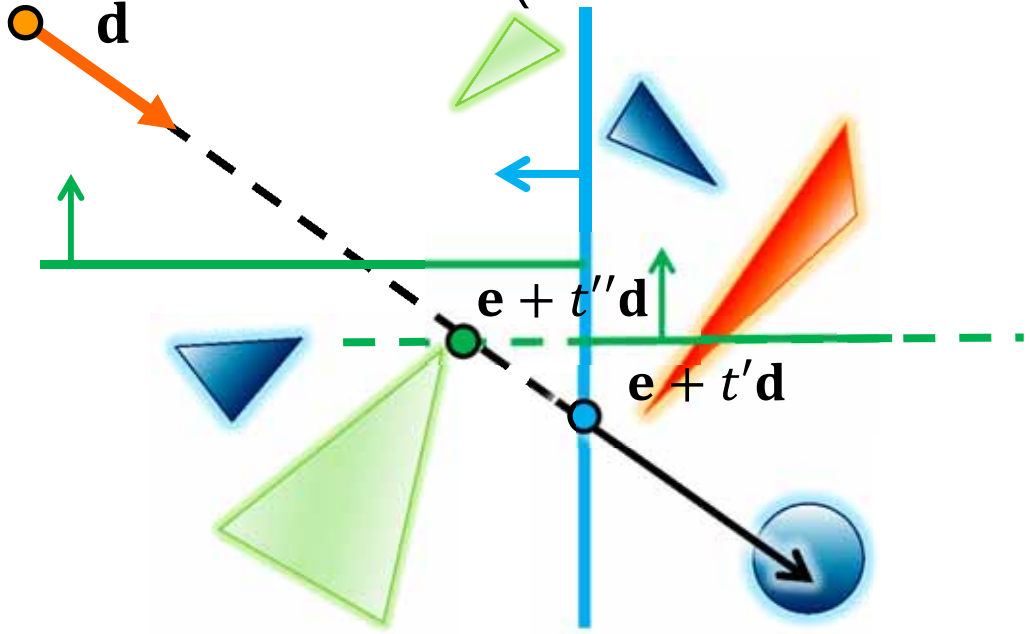
Stack (vorher): 3,4,2

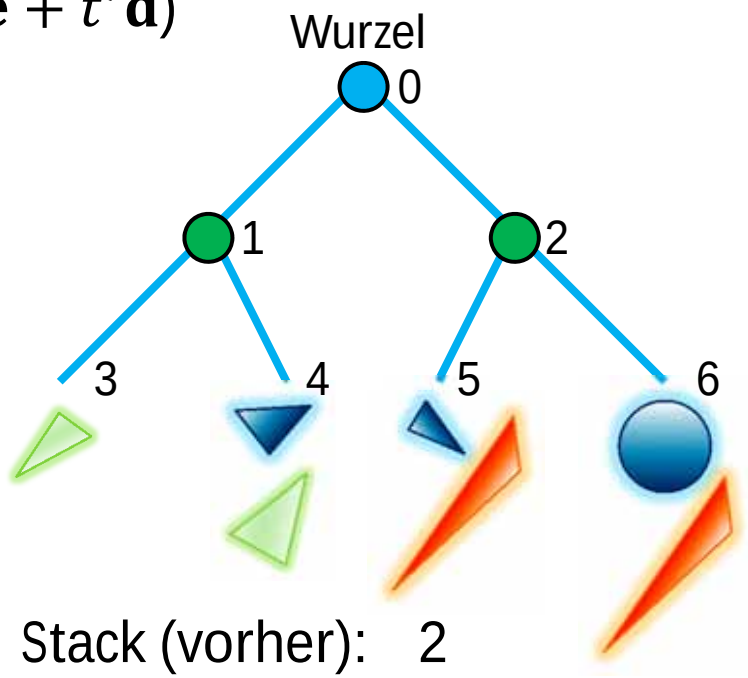
Stack (nachher): 2

BSP-Baum und kD-Baum

Traversierung

- fahre mit nächstem Knoten vom Stack fort → Knoten 2
- schneide Strahl mit der Split-Ebene (Knoten 2)
  - liefert den Schnitt bei t''
    - wenn $t'' \in [t', t_{max})$, dann wären wir in Knoten 5 (nicht der Fall)
    - wenn $t'' \notin [t', t_{max})$, dann fahre mit dem Kind, das den Startpunkt enthält fort (hier: Knoten 6 enthält $\mathbf{e} + t'\mathbf{d}$)





Stack (vorher): 2

Stack (nachher): 6

BSP-Baum und kD-Baum

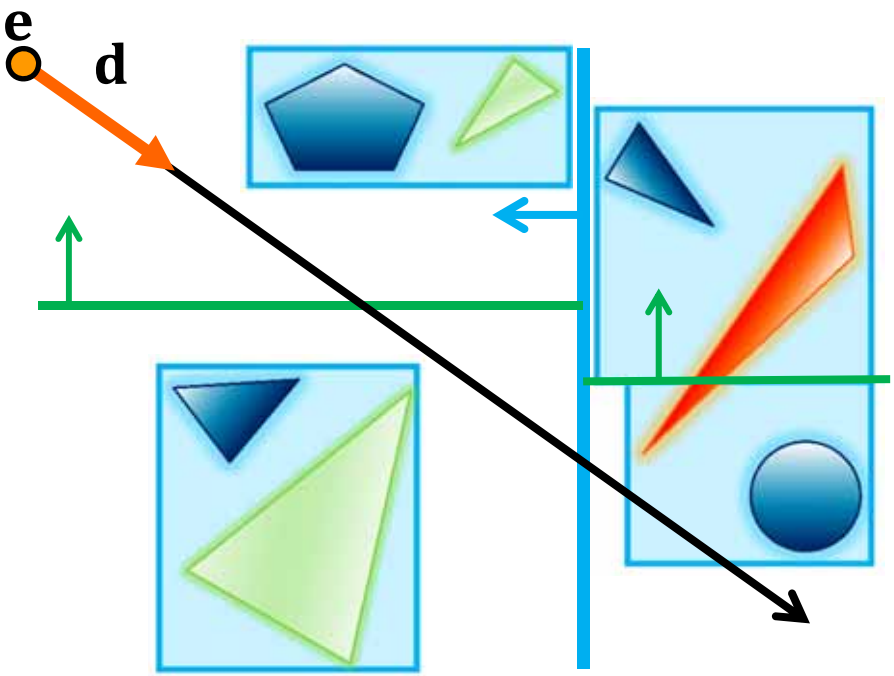
Traversierung

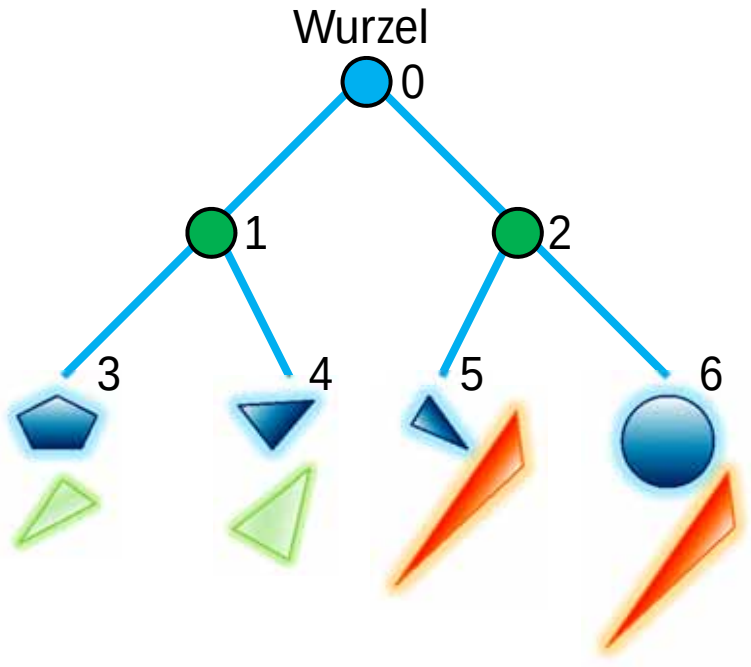
- Strahl $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$, mit $0 \leq t \leq t_{max}$
- schneide Strahl mit der Split-Ebene (beginnend beim Wurzelknoten)
  - Schnitt mit der Split-Ebene bei t'
    - wenn t' auf dem Strahlsegment liegt:
      traversiere beide Kinder rekursiv, zuerst das Kind, das $\mathbf{e}$ enthält
    - liegt t' nicht auf dem Strahlsegment:
      dann fahre nur mit dem Kind, das $\mathbf{e}$ enthält fort
  - Blattknoten erreicht:
    - teste Schnittpunkte mit Primitiven
    - gebe Schnittpunkt zurück, wenn innerhalb des aktuellen Strahlsegments
  - Reihenfolge der Traversierung der Knoten: von vorne nach hinten!
- Komplexität
  - $O(\log n)$ Schnitttests für n Objekte/Primitive in der Szene

BSP-Baum und kD-Baum

Kombination mit AABBs

- i.d.R. speichert man für die Primitive eines Knotens zusätzlich eine AABB
- hilfreich, wenn Primitive nur einen kleinen Teil des Raums einnehmen
- für Schnitttest pro Knoten bedeutet das: teste nur auf Schnitt mit Primitiven, wenn die AABB geschnitten wird
- Isolation dünnbesetzter Bereiche ist daher wünschenswert



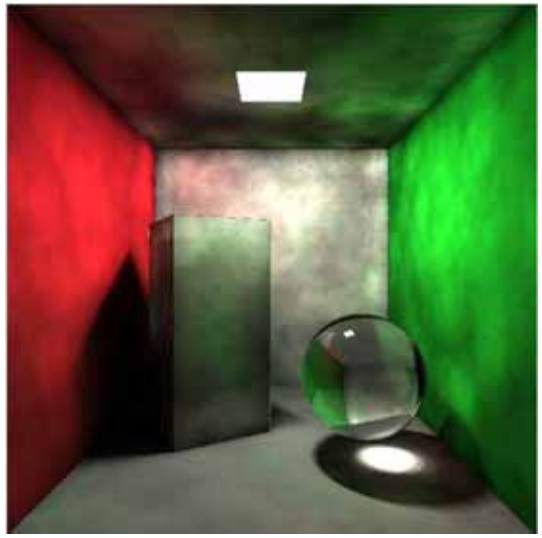


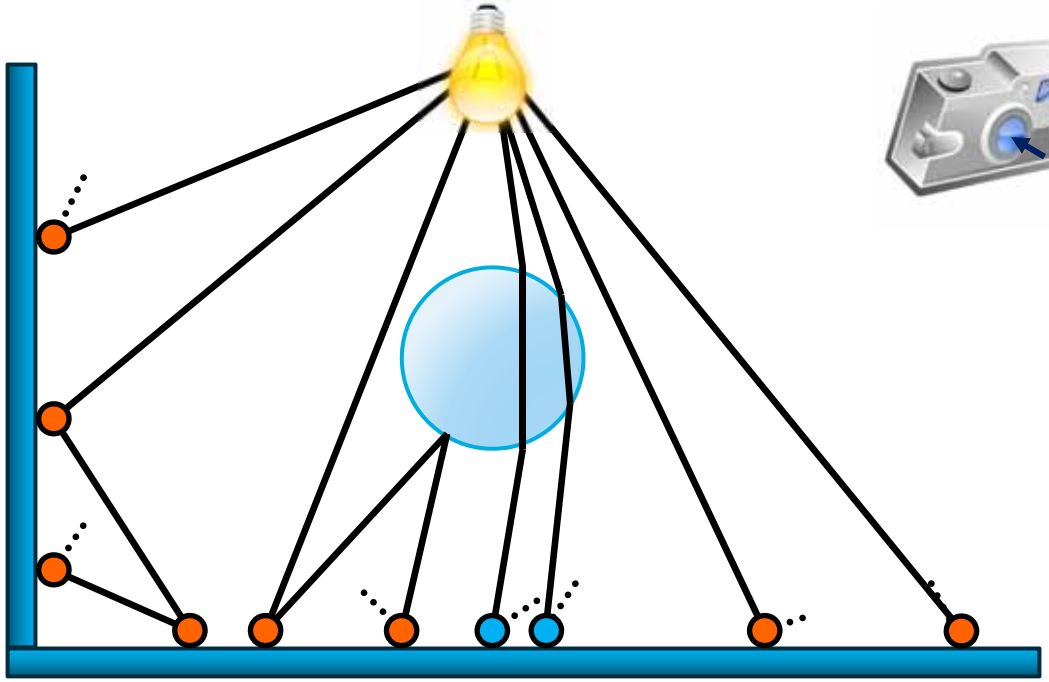
Anwendung: kD-Bäume und Photon Mapping

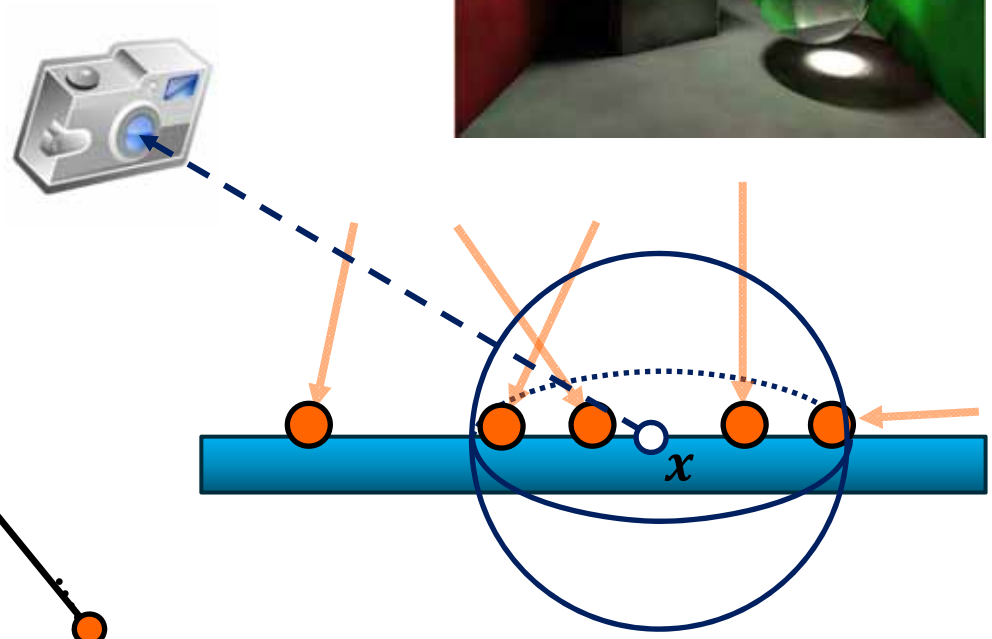
Exkurs: Photon Mapping

Grundidee

- sende Photonen (Energiepakete) von den Lichtquellen aus
- verfolge und speichere Photonen, wenn sie eine diffuse Fläche treffen
- Helligkeit $\propto$ Photonenenergie pro Fläche
  - Suche die n-nächsten Photonen oder alle Photonen in einer Umgebung mit Radius r



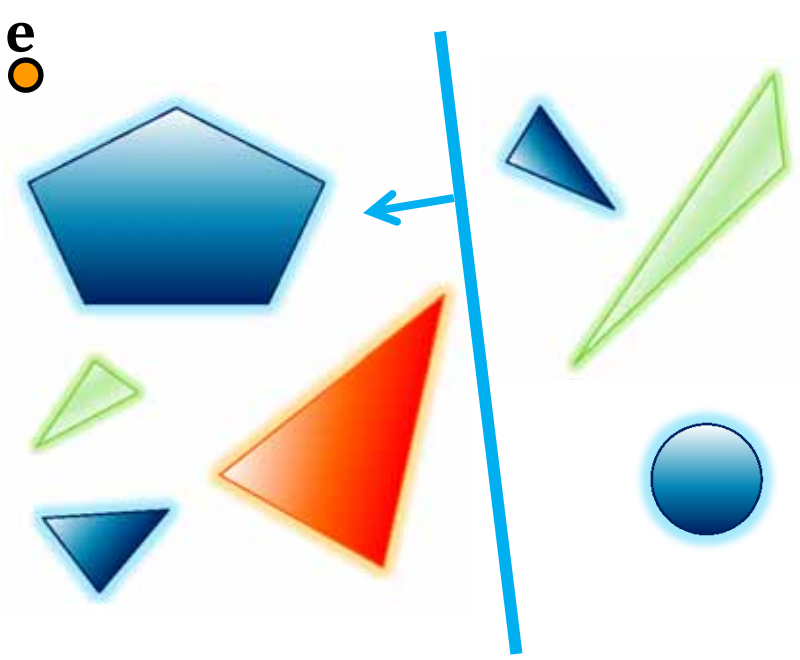


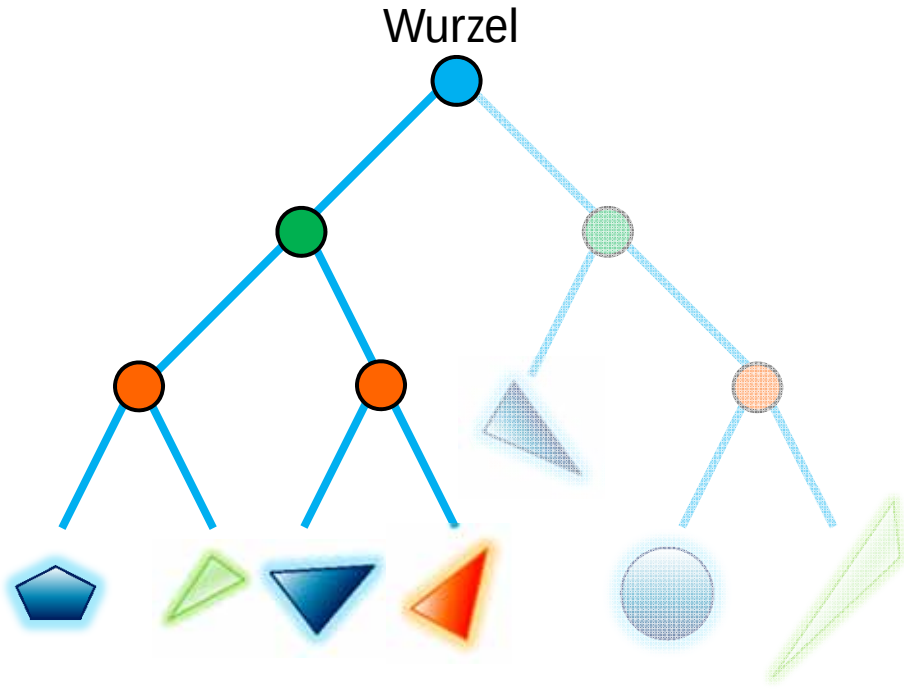


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

- **e** liegt im positiven Halbraum der Split-Ebene der Wurzel
  - d.h. alle Objekte im negativen Halbraum sind „weiter hinten“

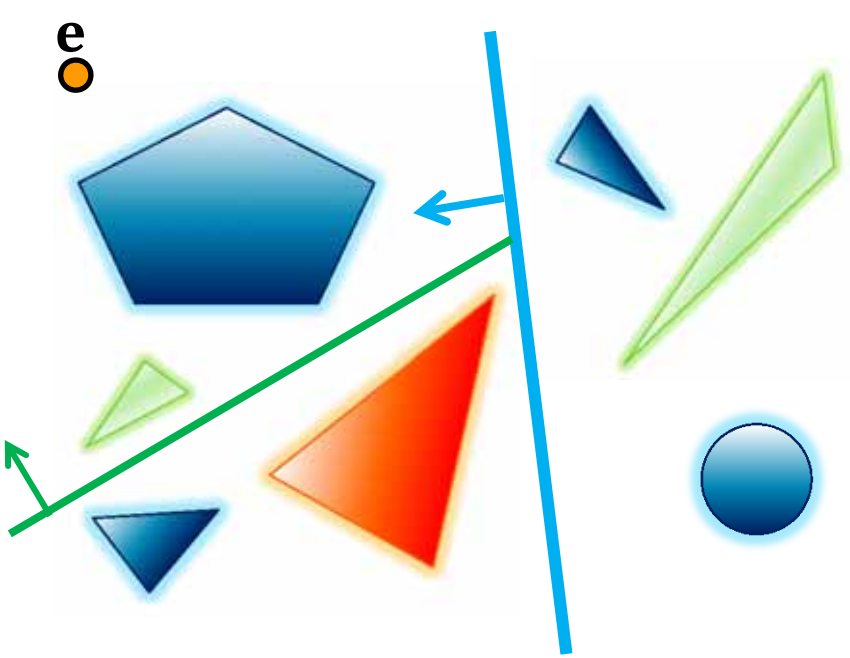


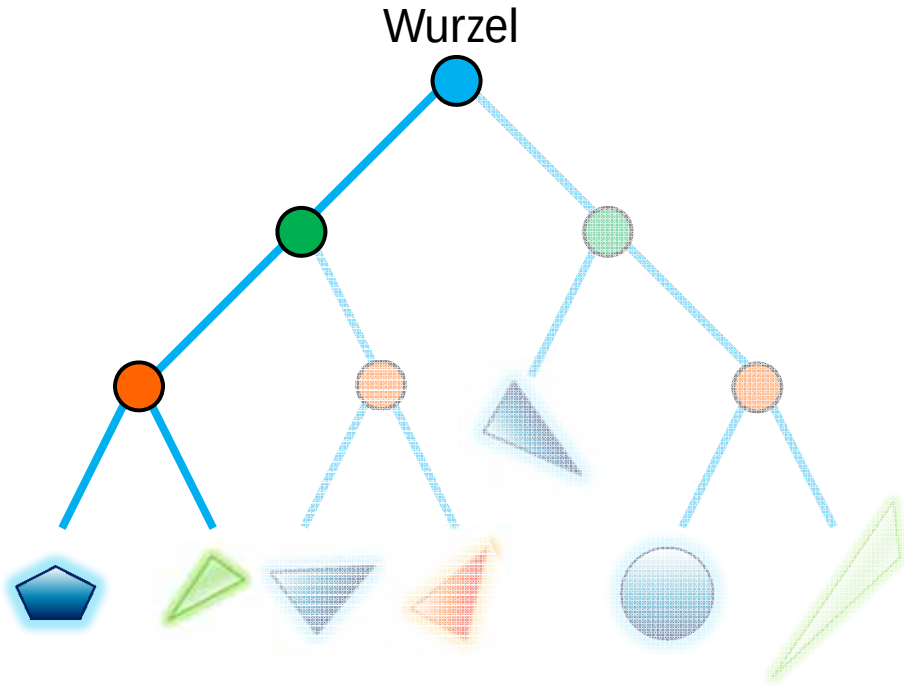


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

- **e** liegt im positiven Halbraum der Split-Ebene des Knotens
  - d.h. alle Objekte im negativen Halbraum sind „weiter hinten“



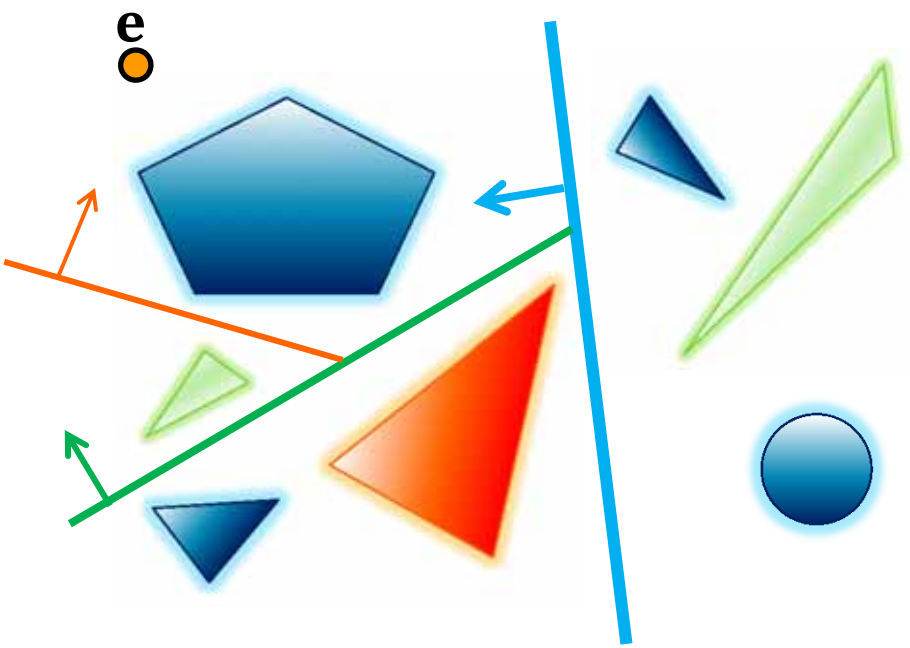


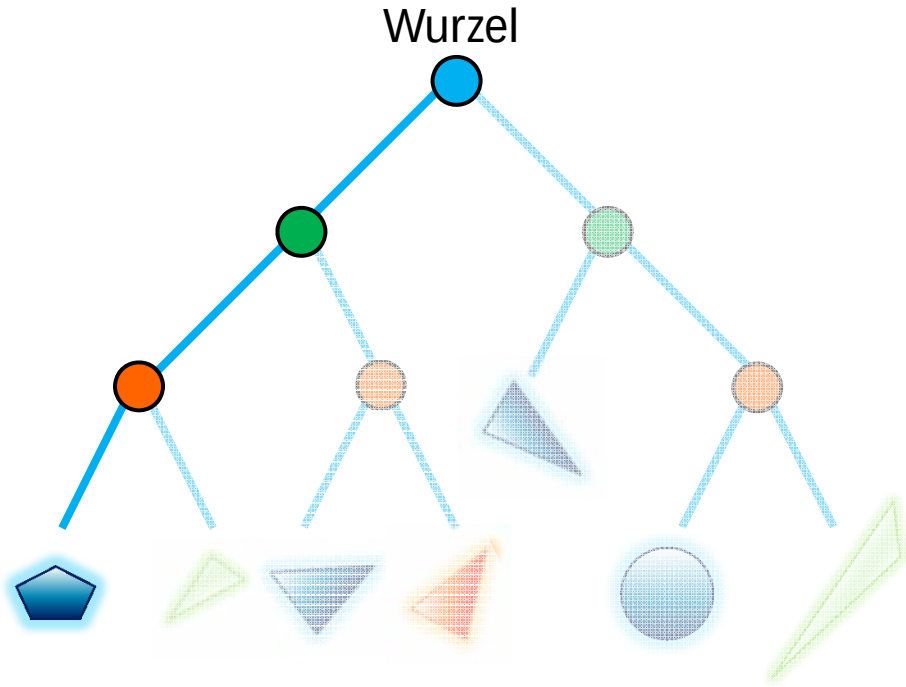
Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

- **e** liegt im positiven Halbraum der Split-Ebene des Knotens
  - d.h. alle Objekte im negativen Halbraum sind „weiter hinten“

dieses Prozedere findet den Unterraum in dem sich **e** befindet, aber nicht immer das nahste Objekt!

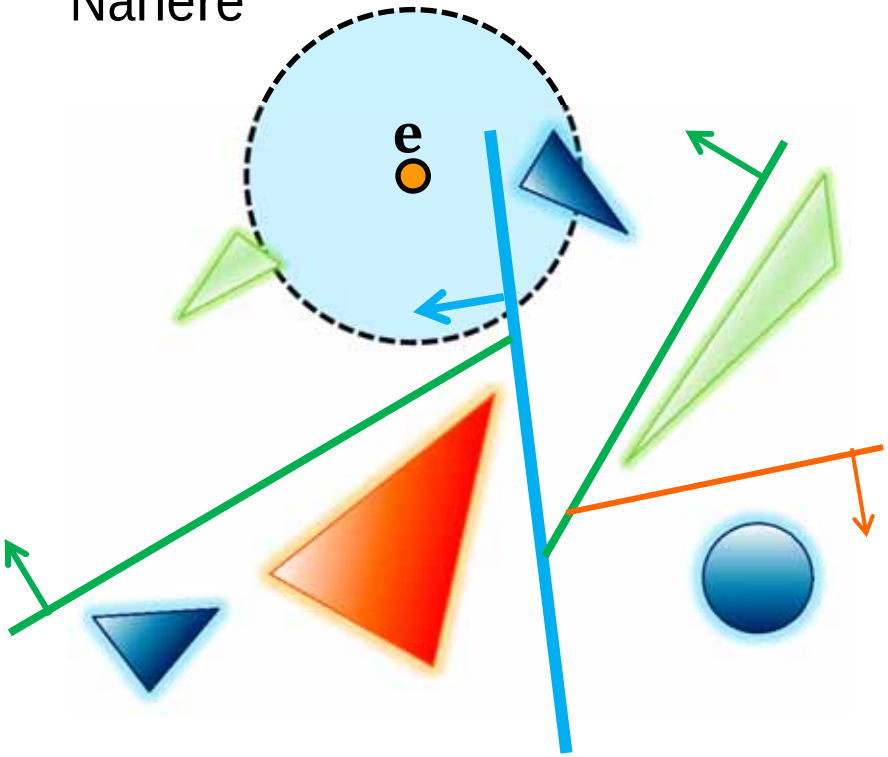


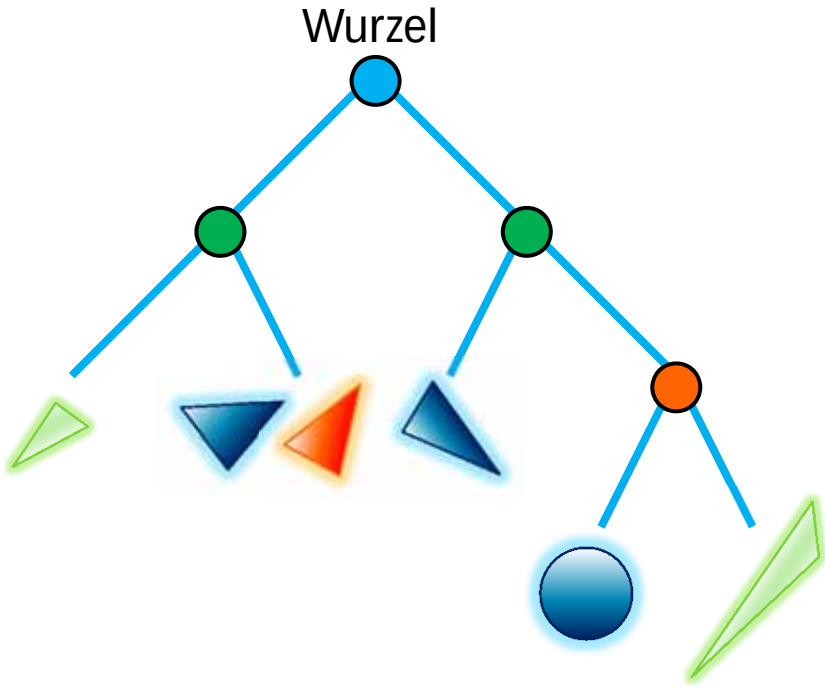


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt **e**

- das Objekt  im selben Unterraum wie **e** ist nicht das Nahste
- das gesuchte Objekt  kann in jedem Unterraum liegen, der näher an **e** liegt, als das bisher gefundene Primitiv
- speichere Abstand zum jeweils nahsten gefundenen Objekt und suche Nähere

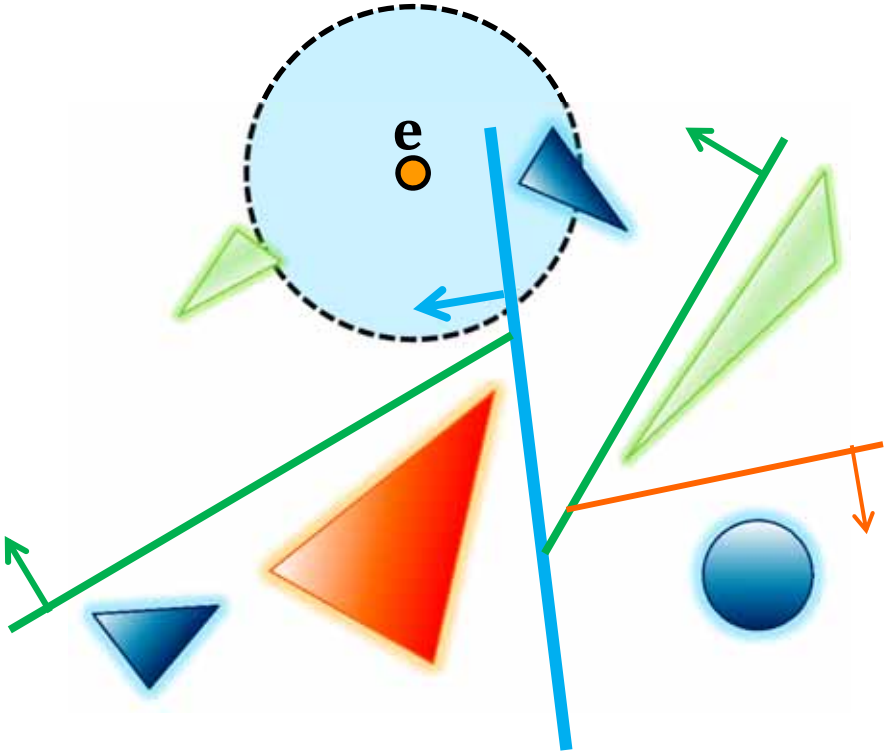


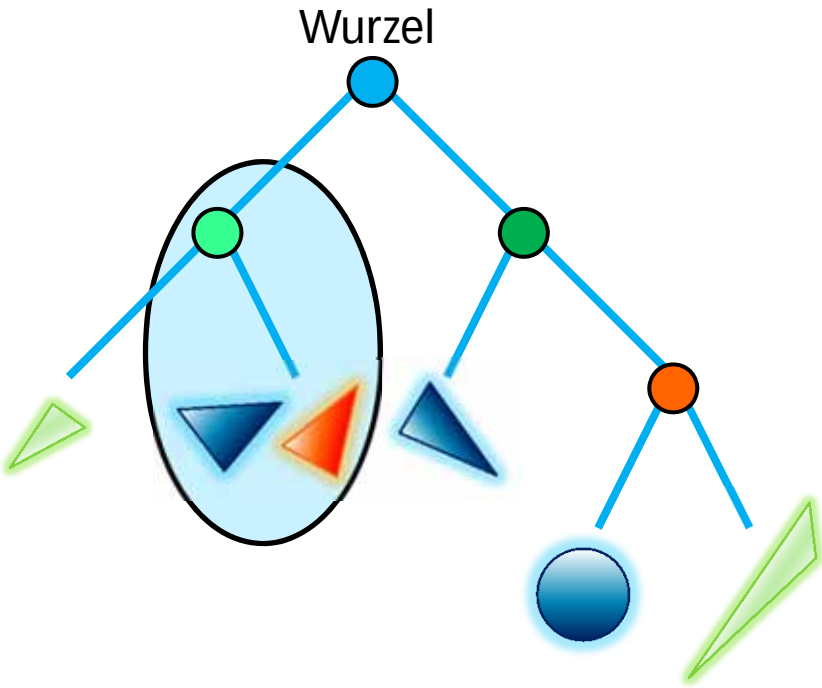


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

- das gesuchte Objekt kann in jedem Unterraum liegen, der näher an **e** liegt, als das bisher gefundene Primitiv
- gehe zum Elternknoten ● und prüfe ob der andere Kindknoten in Frage kommt (hier nicht der Fall)

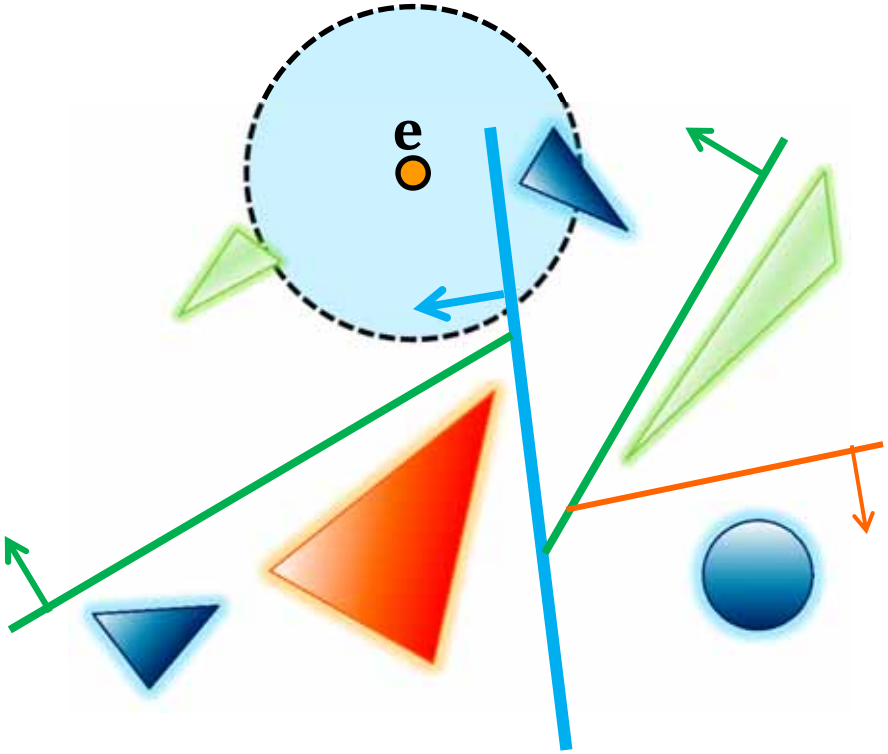


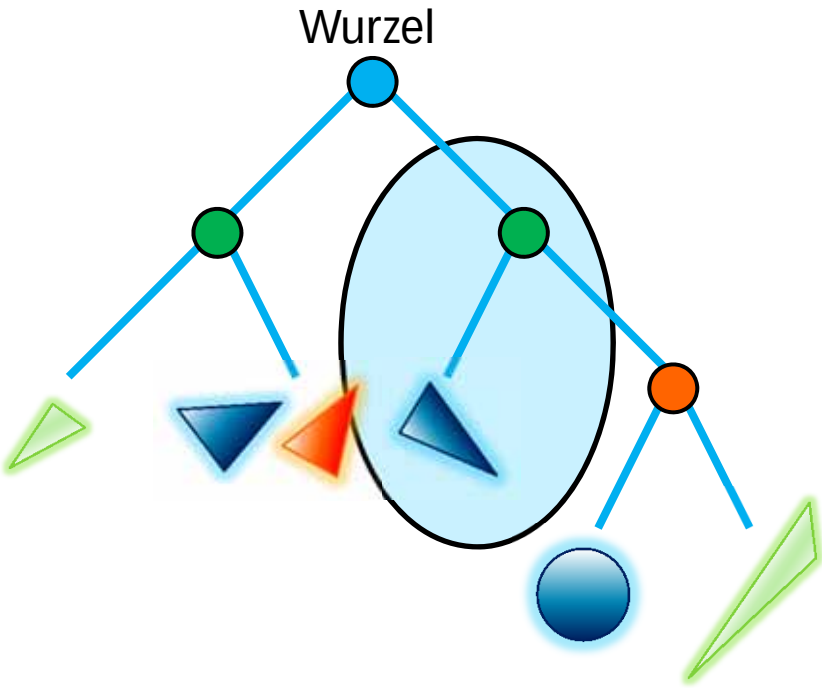


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt $\mathbf{e}$

- gehe im Baum weiter nach oben und prüfe den anderen Kindknoten
- steige in diesem Teilbaum durch Tiefensuche ab (depth-first traversal): jeweils den Ast des näheren Teilbaums zuerst
- im Mittel $O(\log n)$, worst case $O(n)$ (n Objekte/Primitive)

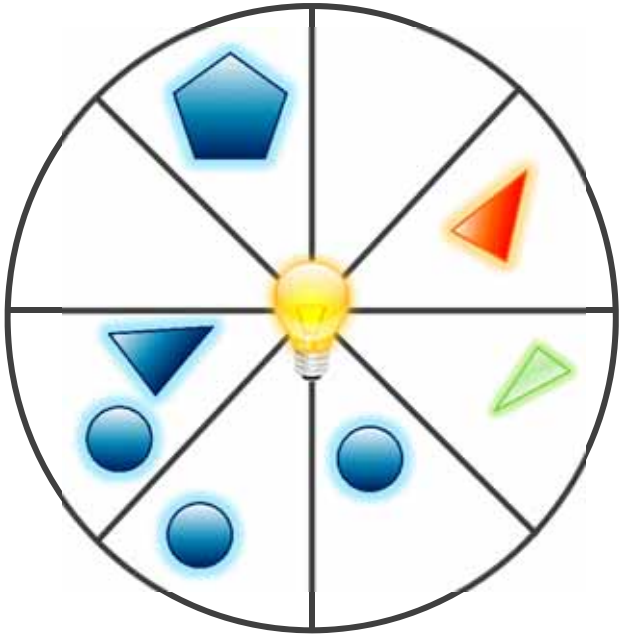




Richtungsunterteilung

Idee: Unterteile Objekte nach Richtung, in der sie liegen

- Sortierung für einige diskrete Richtungen
  - optional: jeweils nach Entfernung
- Bsp. Beschleunigung des Schattentests bei Punktlichtquellen
- werden sehr selten eingesetzt

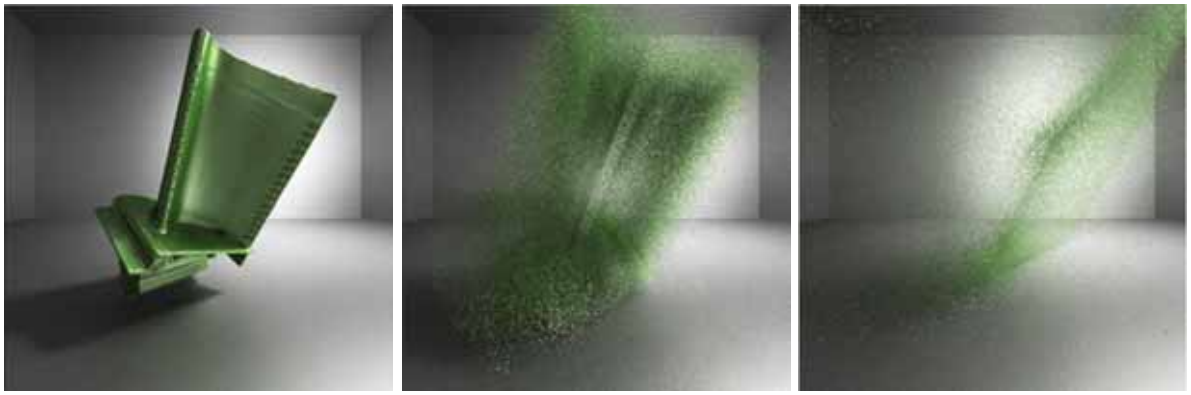


Modernes Ray-Tracing

- SIMD Architekturen und kleine Strahlenbündel (typ. 4-16 Strahlen)
  - kohärente Strahlenpakete: Gruppen von Strahlen mit ähnlichem Ursprung und Richtung (z.B. Primärstrahlen, Schattenstrahlen etc.)
  - können oft sehr schnell verarbeitet werden:
    oft ähnliche Wege durch die (hierarchische) Beschleunigungsstruktur
    - Bsp. der Schnitttest Strahl-AABB kann mit SIMD Befehlen einfach auf einen Test 4-Strahlen-AABB erweitert werden
  - inkohärente Pakete können Parallelität der HW nicht (gut) nutzen
  - gleiche Berechnungen (oft Dreieck als einziger Primitivtyp)
- Geschwindigkeit (siehe auch http://embree.github.io/data/embree-siggraph-2013-final.pdf und https://research.nvidia.com/sites/default/files/publications/nvr-2012-02.pdf):
  - in der Größenordnung von 30-100 Mio. kohärenten Strahlen pro Sekunde auf einer Quadcore CPU (Grafikhardware 200-400 Mio.)
  - bei komplexen Szenen (hunderttausende Dreiecke) mit hierarchischen Datenstrukturen (typ. BVH oder kD-Bäume)
  - paralleler Aufbau der Datenstrukturen ist aktuelles Forschungsthema!

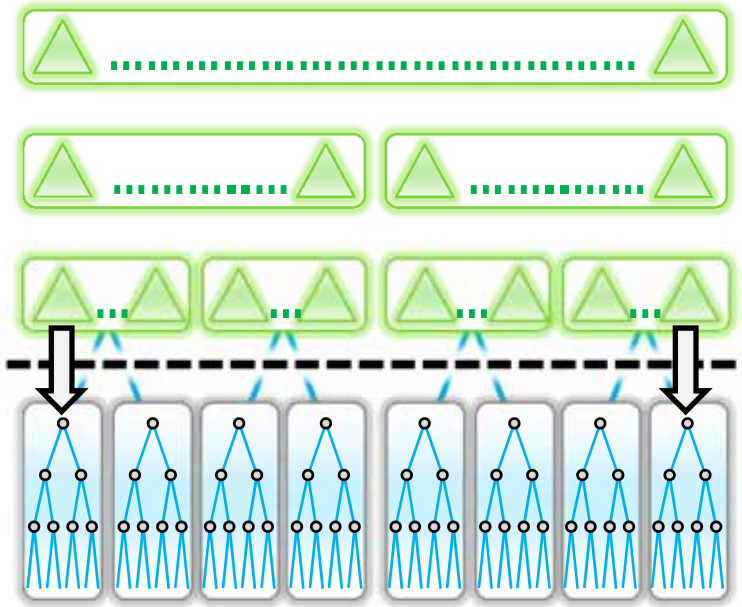
Parallele Konstruktion räumlicher Datenstrukturen

- Neuberechnung und inkrementelle Updates in dynamischen Szenen



Bilder:
Pantaleoni und Luebke

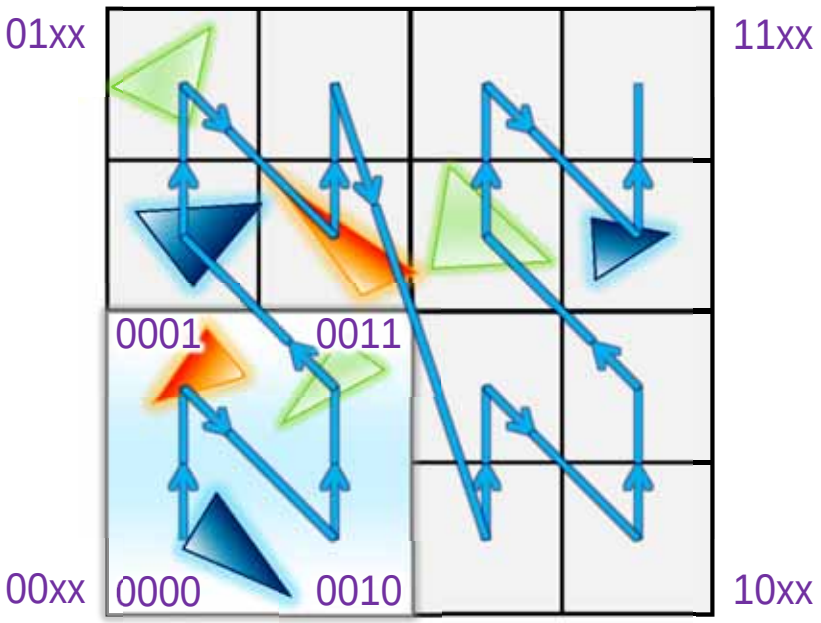
- State of the Art: wenige Millisekunden für 1 Mio. Dreiecke auf Grafik-Hardware, 20ms auf CPUs (gute Qualität)
  - Geschwindigkeit beim Aufbau und Qualität der Hierarchie sind widersprüchliche Kriterien
  - ein weiteres Problem ist der temporäre Speicherbedarf bei der Konstruktion

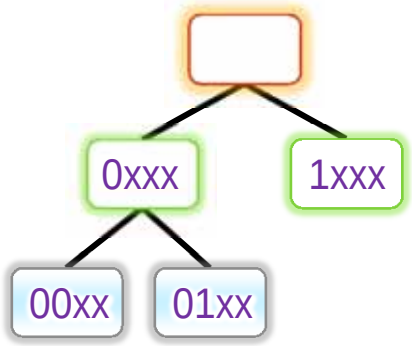


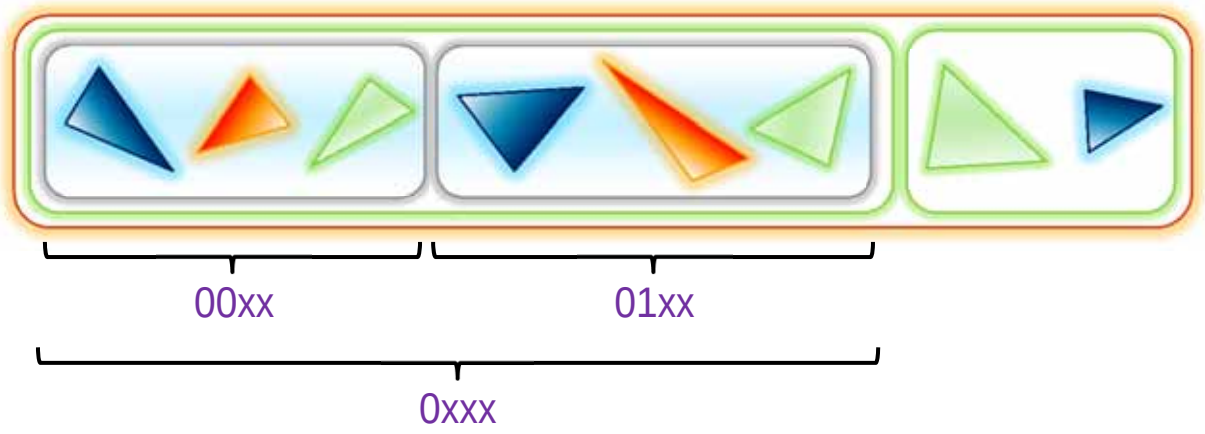
Parallele Konstruktion räumlicher Datenstrukturen

Grundidee: Vorsortierung durch Quantisierung und Morton Codes

- quantisiere Primitivmittelpunkte
- Zuweisung Morton Codes zu Zellen
- sortiere Primitive nach Zellcodes
- ergibt implizite Baumstruktur





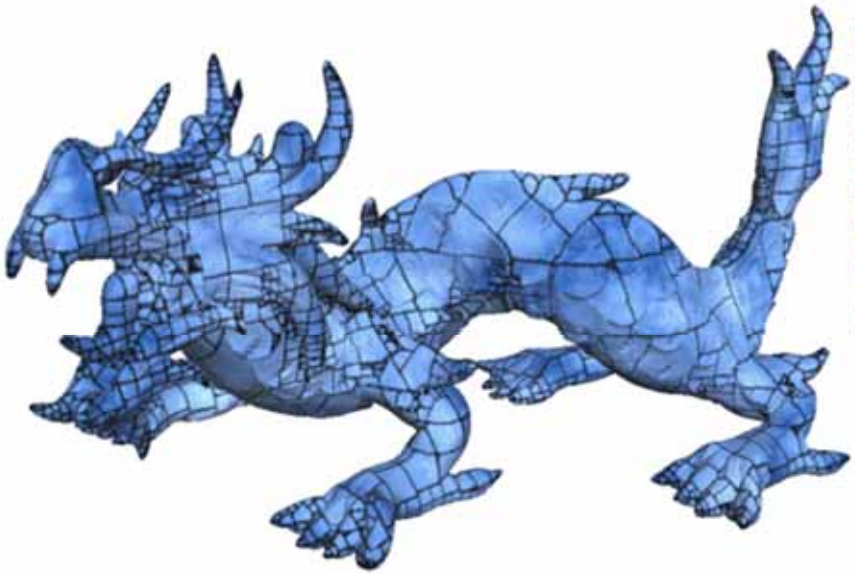


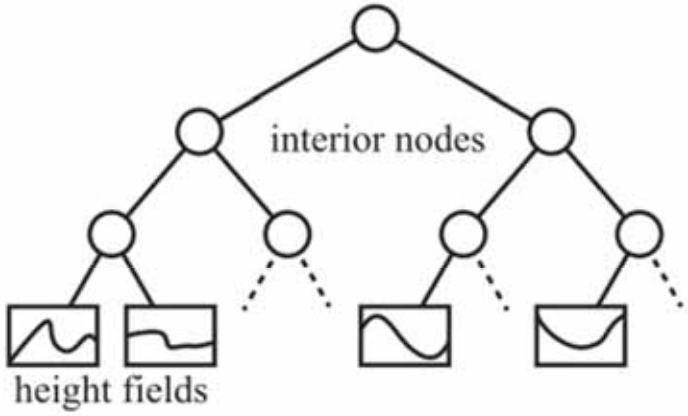
Fazit (Achtung: Erfahrungswerte!)

- Kombination verschiedener Beschleunigungstechniken
  - oft SIMD-Optimierung zusammen mit BVH oder kD-Baum
    - BVH mit AABBs: schnelle Erzeugung, gute Hierarchie
    - kD-Baum: etwas aufwändiger zu konstruieren, aber typischerweise etwas besser als BVH, kombiniert mit AABBs pro Knoten
    - BSP-Baum: aufwändig zu konstruieren, oft nur ein bisschen besser als ein gut konstruierter kD-Baum
    - Gitter: eher selten, aber Aufbau auf (Grafik-)Hardware einfacher
    - hierarchische Gitter und Octrees: eher eingesetzt für Simulationen
- weitere Verwendung (nicht nur) in der Computergrafik
  - Bounding-Volumes und BVH: Culling in der Echtzeit-Grafik
  - kD-/BSP-Bäume: Sortierung, Suche, ...
  - Octrees: Speicherung von räumlichen Daten, z.B. 3D-Texturen
  - Kollisionserkennung (auch in der Robotik)

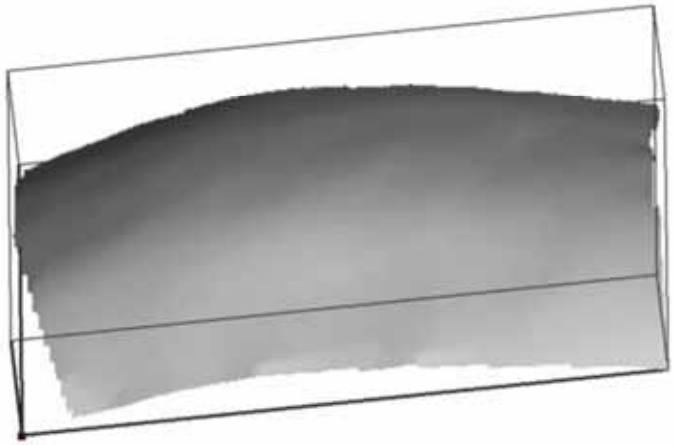
Aktuelle Forschungsthemen

- Rasterized Bounding Volume Hierarchies



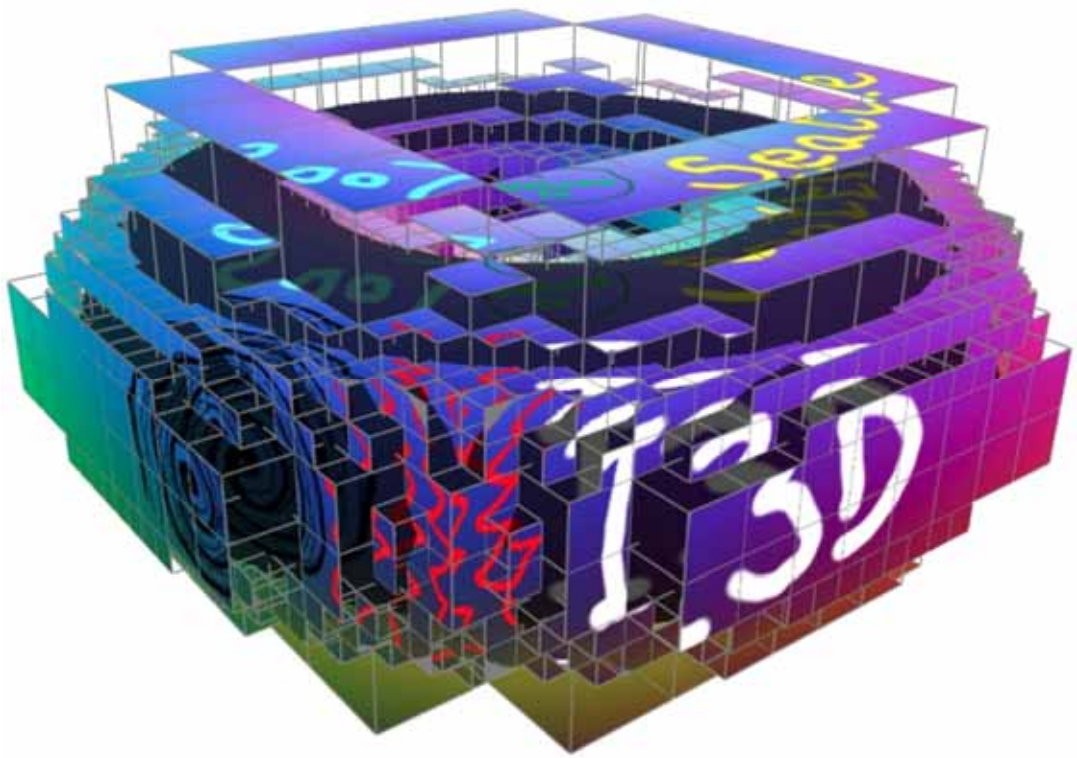


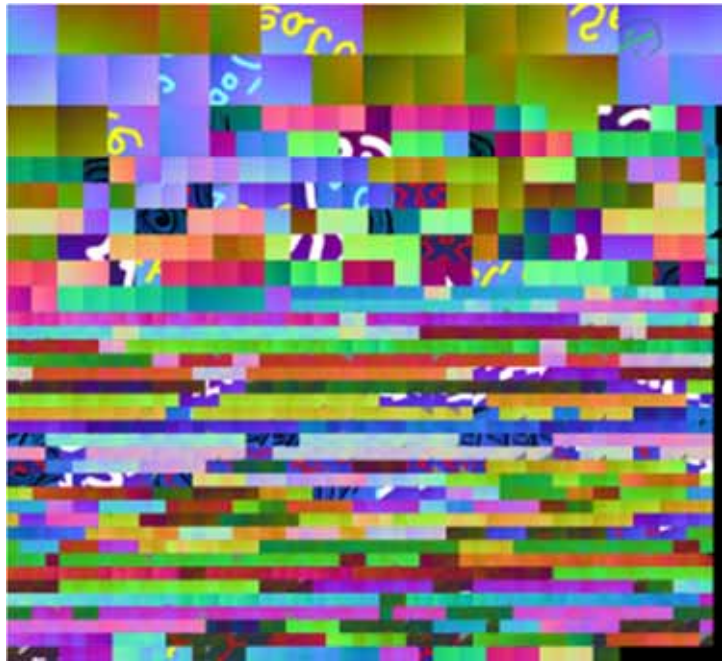
- weitere Forschungsziele:
  - schnellerer Aufbau
  - schnellere Traversierung
  - weniger Speicher, ...



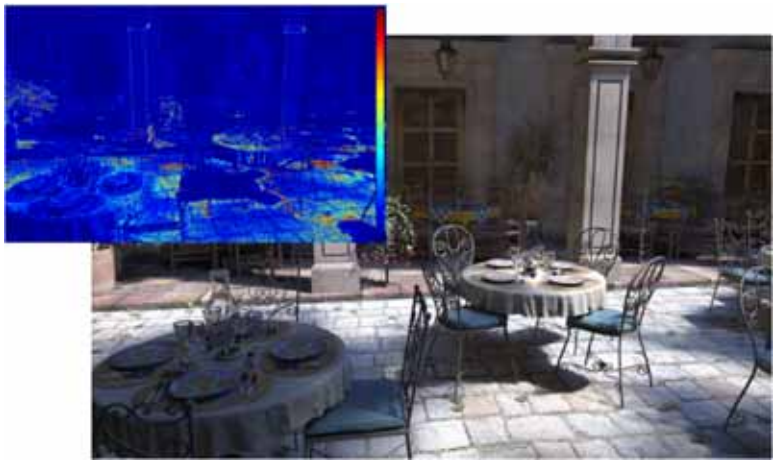
Forschungsthemen

- kompakte Speicherung von und schneller Zugriff auf Texturen



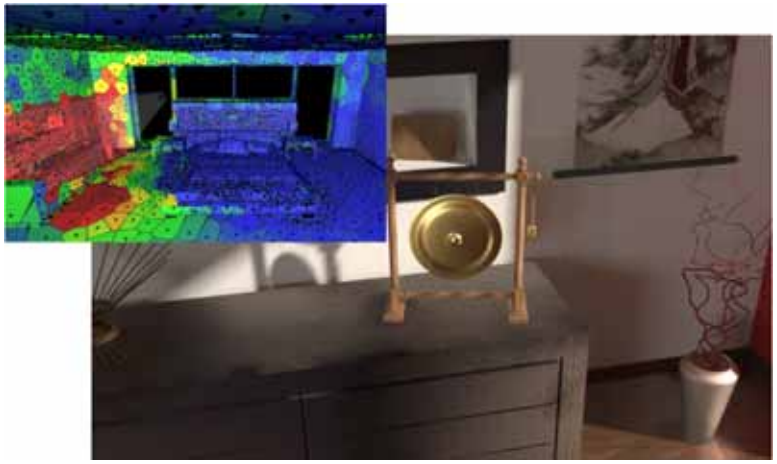


Caching Visibility and On-Surface / In-Volume Signals

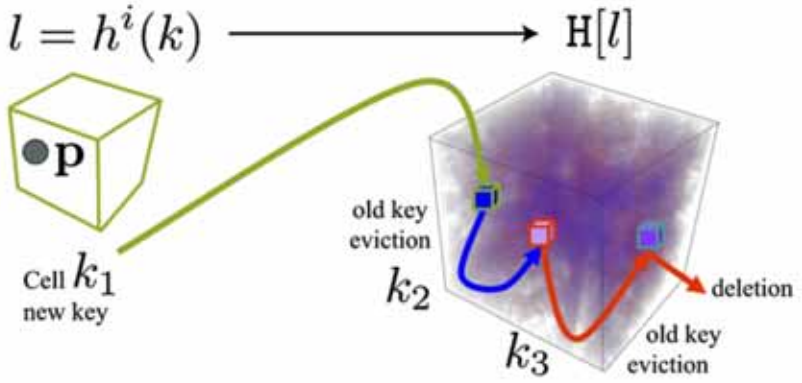


Eurographics 2013

Eurographics 2012



2x VMV 2013



Shape Modeling International 2012, Computers & Graphics