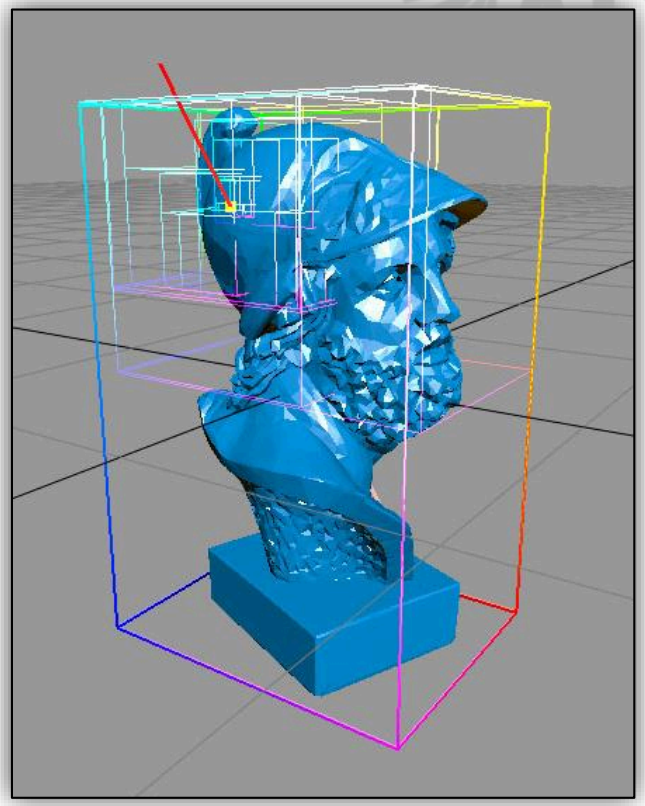
Computergrafik

Vorlesung im Wintersemester 2024/25
Kapitel 5: Räumliche Datenstrukturen

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

Inhalt: Räumliche Datenstrukturen

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

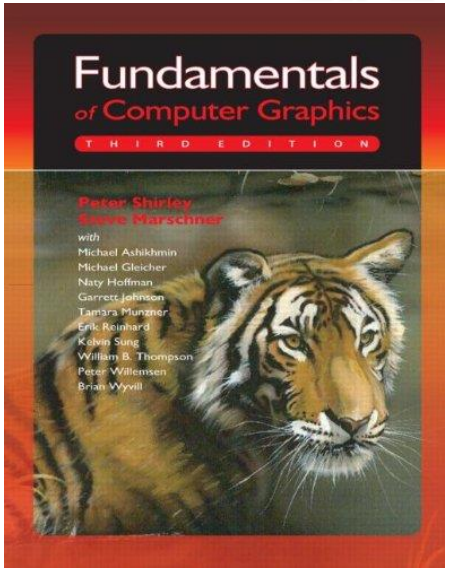


- GPU-Raytracing übernimmt Beschleunigung (fast) transparent – Wissen über diese Datenstrukturen ist trotzdem wichtig, z.B. für CPU-Raytracing, spezielle Operationen bei Monte Carlo-Verfahren, verwandte Anwendungen etc.

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

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)

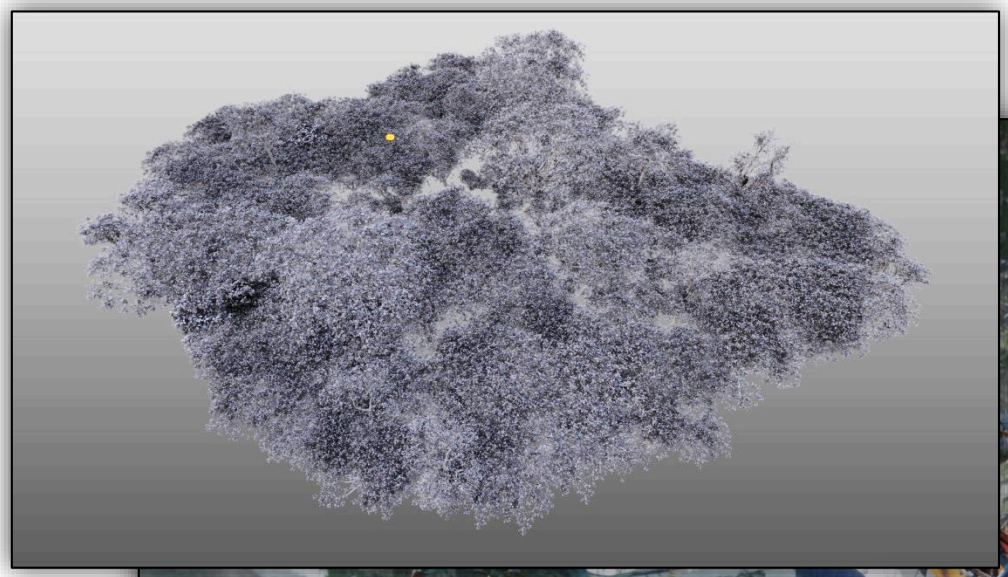


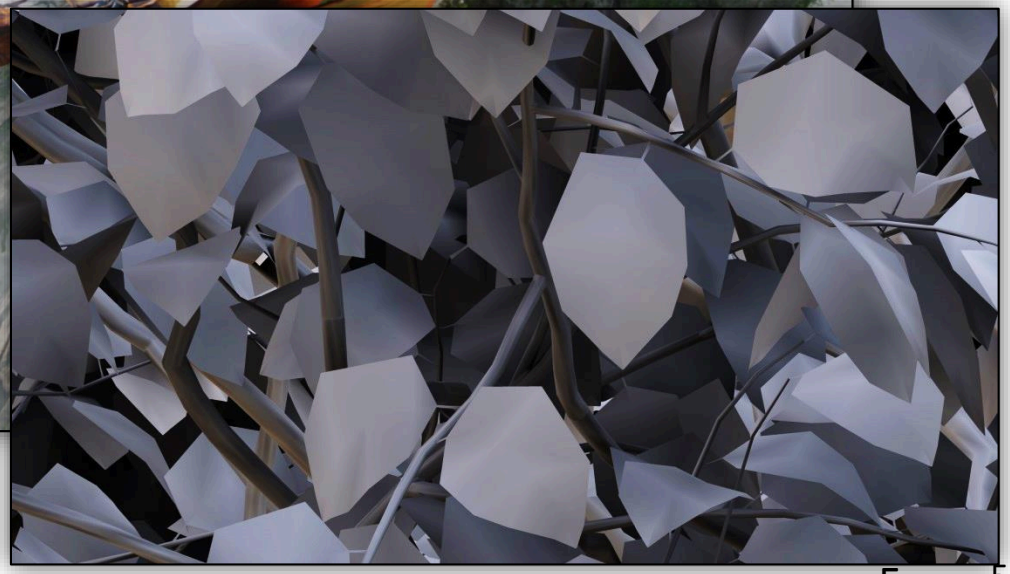
Raytracing Pseudocode

```
for ( y = 0; y < height; y++ ) {
  for ( x = 0; x < width; x++ ) {
    generateRay( ray, x, y );
    color = raytrace( ray, ... );
} }

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

  if ( !cast( ray, FLOAT_MAX ) ) return color; // optional: Environment Mapping
  // weitere(r) Sekundärstrahl(en)
  ...
}


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;
}
```

Raytracing: naiv viel zu viele Schnittberechnungen

Avatar (2009): geometrisches Detail / Anzahl Primitive





Bilder: Pantaleoni et al.

5

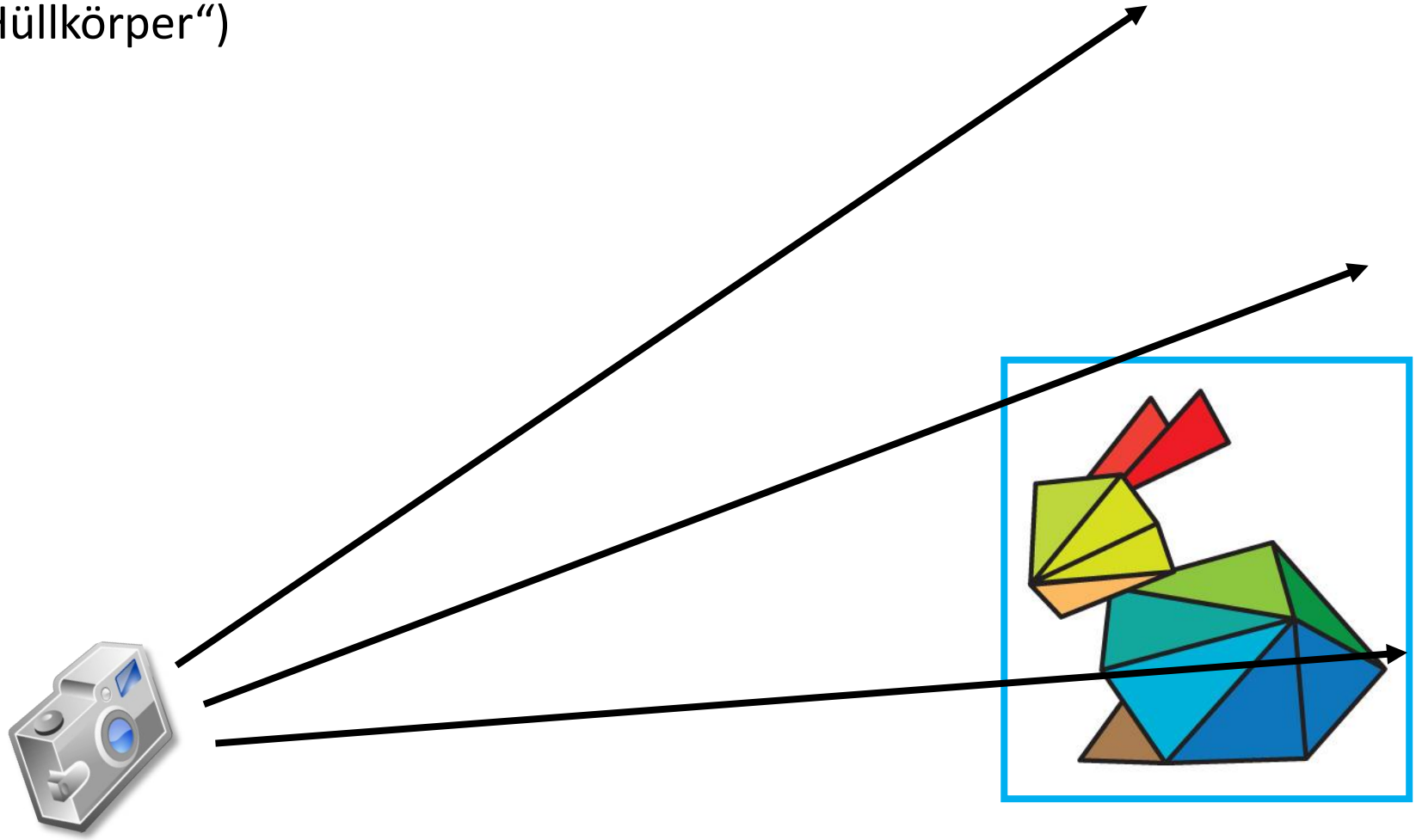
Optimierung des Raytracing

- der wesentliche Teilalgorithmus von Raytracing für die Laufzeit, so wie wir es bisher kennengelernt haben, ist:
  - finde Schnittpunkt des Strahls mit nahstem Primitiv der Szene
  - Schnitt mit allen Primitiven nimmt beinahe die gesamte Rechenzeit in Anspruch: Aufwand $O(n)$ bei n Primitiven
- Optimierungsansätze, z.B. optimierte Schnitttests lösen das Skalierungsproblem nicht
  - SIMD macht Sinn, z.B. teste 2×2 Primärstrahlen gleichzeitig auf Schnitt
  - Sekundärstrahlen und Pfadkonstruktion ist orthogonales Problem
- **Lösung: vermeide Schnittberechnungen**
  - frühzeitiges Ausschließen geometrischer Primitive, die nicht geschnitten werden
  - finde so potentiell geschnittene Objekte schneller
  - Unterteile den Raum oder die Menge der Primitive

Beschleunigung des Raycasting

Reduziere die Anzahl der Strahl-Primitiv-Schnitttests

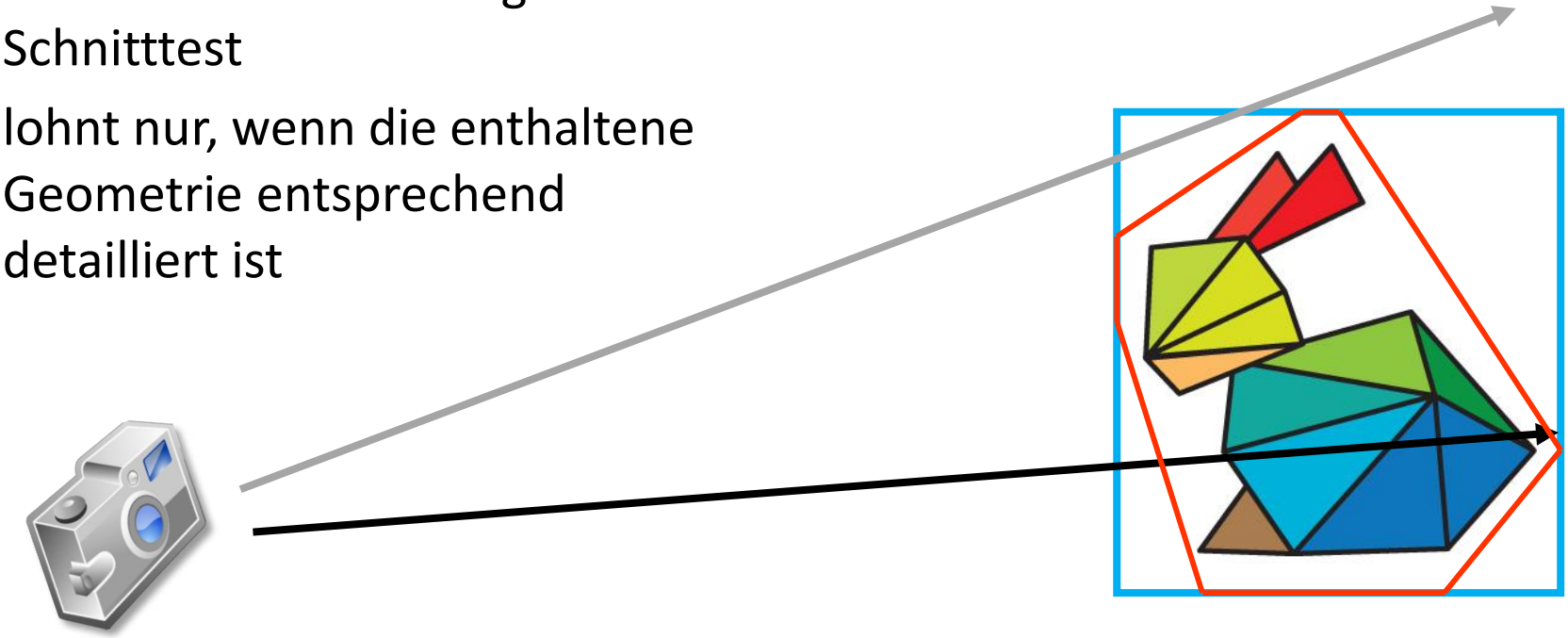
- schneller Ausschluss von Schnitttests mit Primitiven (Dreiecken) durch Schnitt mit einer (konservativ-großen) einschließenden Geometrie („Hüllkörper“)



Beschleunigung des Raycasting

Reduziere die Anzahl der Strahl-Primitiv-Schnitttests

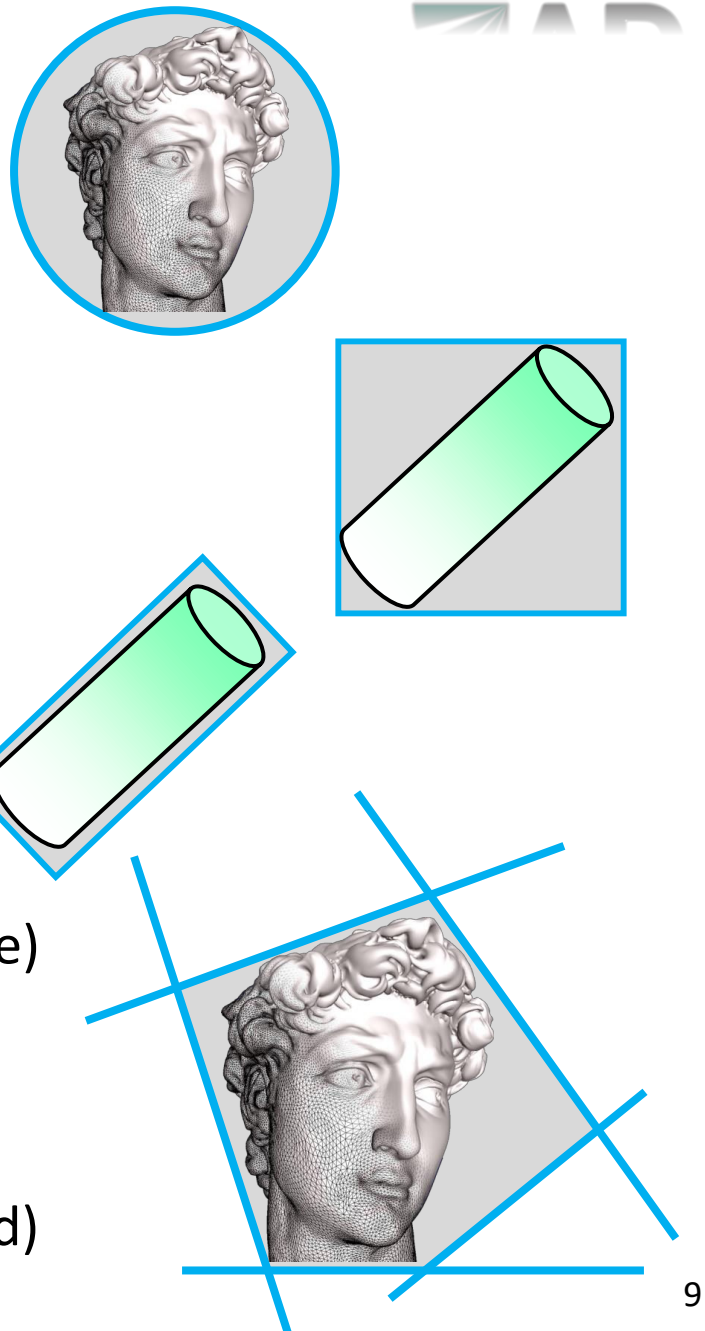
- schneller Ausschluss von Schnitttests mit Primitiven (Dreiecken)
- Hüllkörper (engl. Bounding Volume)
  - sollen möglichst enganliegend/klein sein, um wenige Falschpositive zu erzeugen
  - Trade-off zw. Effizienz und Speicher/Kosten sowohl bei Berechnung als auch beim Schnitttest
  - lohnt nur, wenn die enthaltene Geometrie entsprechend detailliert ist



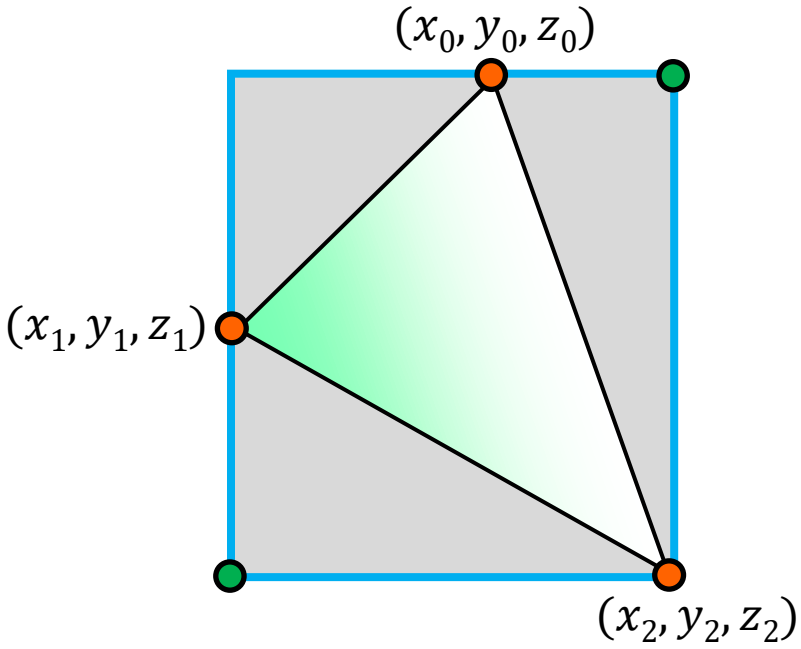
Optimierung: Bounding Volumes

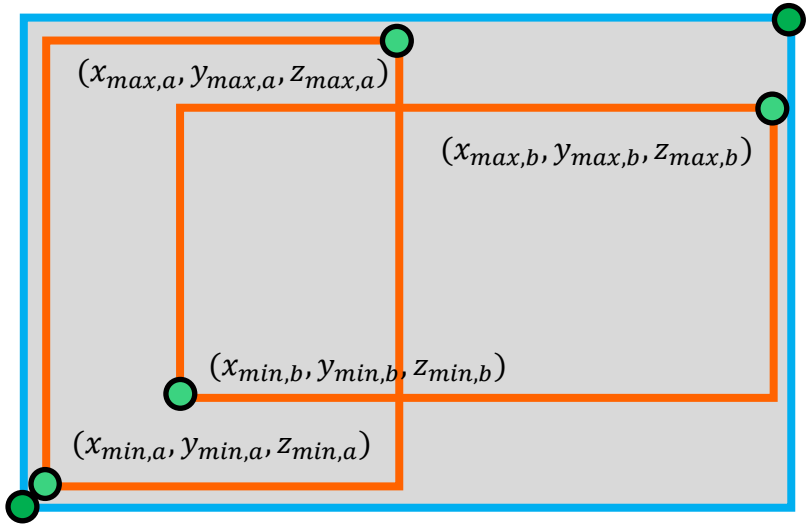
Hüllkörper, Bounding Volumes

- Kugel
  - schneller Schnittalgorithmus
  - oft schlechte Effizienz, da zu groß
- achsenparallele Box (AABB)
  - einfache Berechnung (min/max)
  - wichtigster Hüllkörper
- orientierte Bounding Box (OBB)
  - aufwändigere Berechnung (Hauptachsentransformation der Menge von Eckpunkten)
- allg. konvexe Region (begrenzende Halbräume)
  - Spezialfall „Slabs“: Schnitt von Paaren paralleler Halbebenen
  - gute Effizienz, schnelle Berechnung (sofern nicht die global beste Lösung gefordert wird)



Axis-Aligned Bounding Boxes



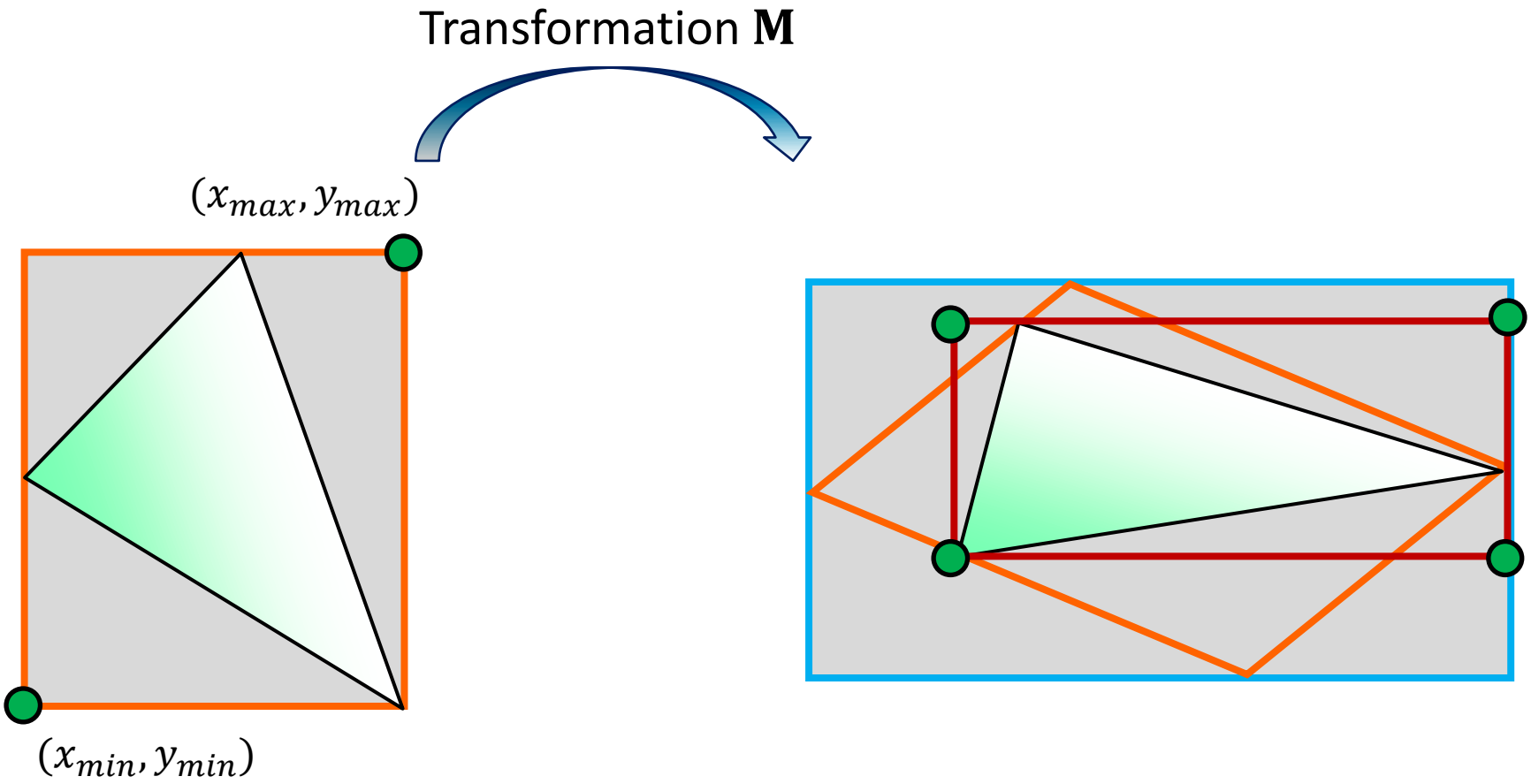


Minima/Maxima der Eckpunktkoord.,
mehrere Primitive: $\min$ bzw. $\max$
über jeweils alle x, y, z-Komponenten

Minima/Maxima der AABB-Ecken

AABBs transformierter Dreiecksnetze

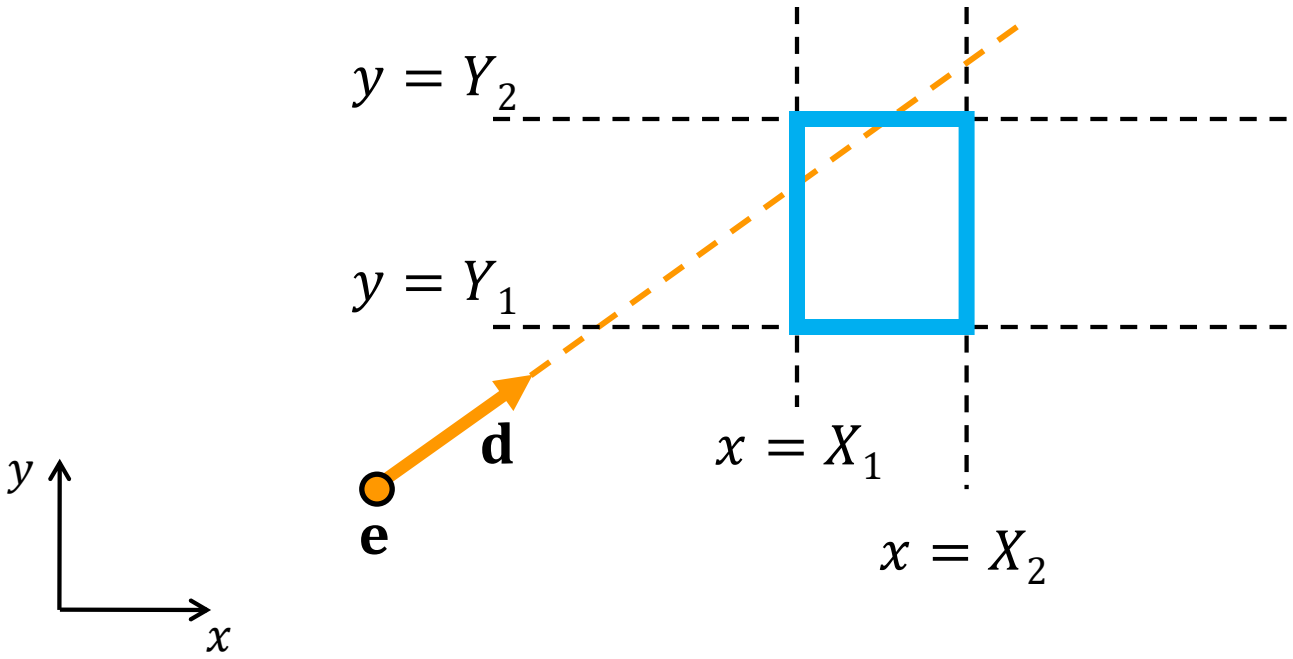
- kleinere AABB durch Transformation der Eckpunkte des Dreiecks(netzes) und Neubestimmung der Minima und Maxima
- erhöhte Kosten für Bestimmung der AABB, aber i.d.R. deutlich kleinerer Hüllkörper und somit weniger Falschpositive



Bounding Volume: AABB

Schnittberechnung Strahl-AABB

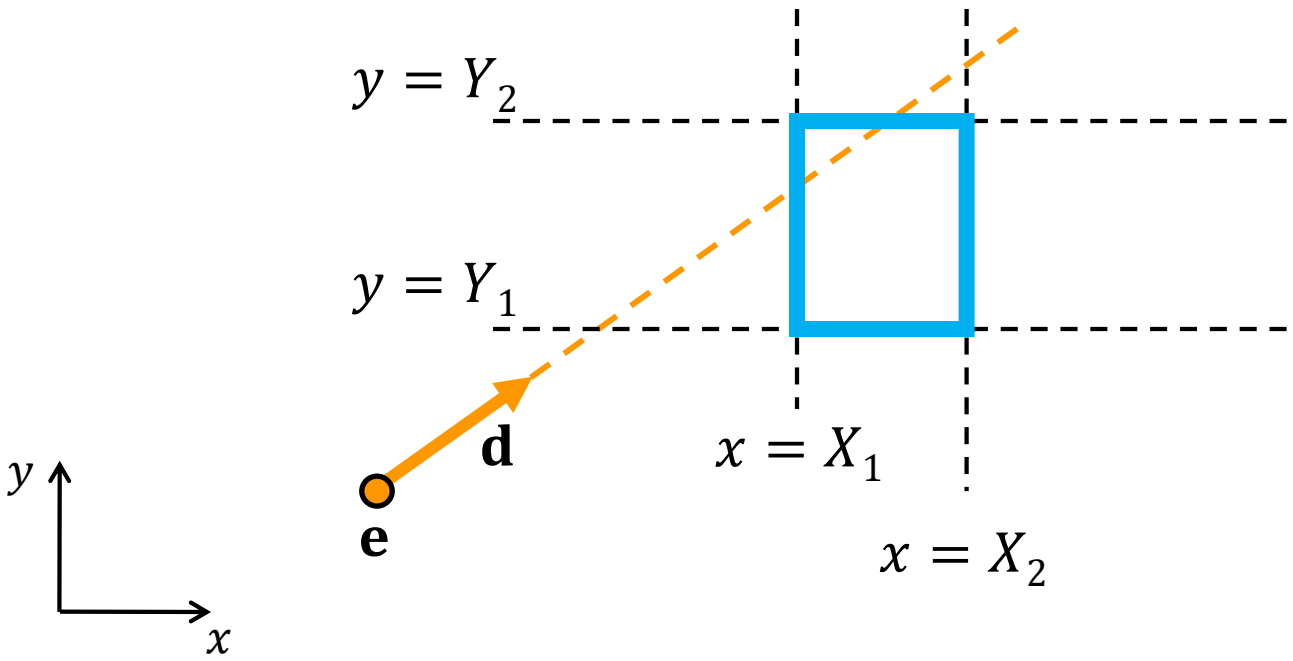
- AABB: $(X_1, Y_1, Z_1) \rightarrow (X_2, Y_2, Z_2)$
- Strahl: $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$
- wichtigster Hüllkörper – später sind wir nicht nur interessiert, **ob** es einen Schnitt mit der AABB gibt, sondern auch **wo** sie liegen



Bounding Volume: AABB

Schnittberechnung Strahl-AABB

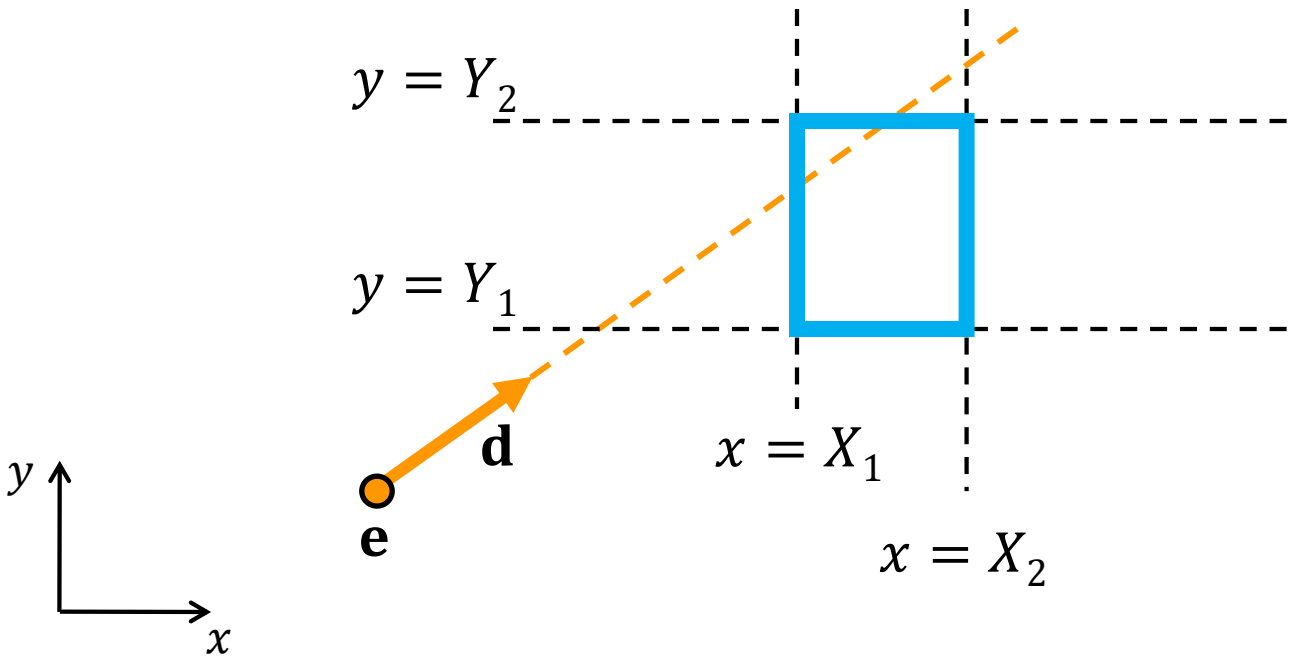
- AABB: $(X_1, Y_1, Z_1) \rightarrow (X_2, Y_2, Z_2)$
- Strahl: $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$
- naiver Ansatz:
  - 6 Ebenengleichungen: berechne alle Schnittpunkte
  - teste, ob einer der Schnittpunkte *auf* der Box liegt



Bounding Volume: AABB

Optimierungen

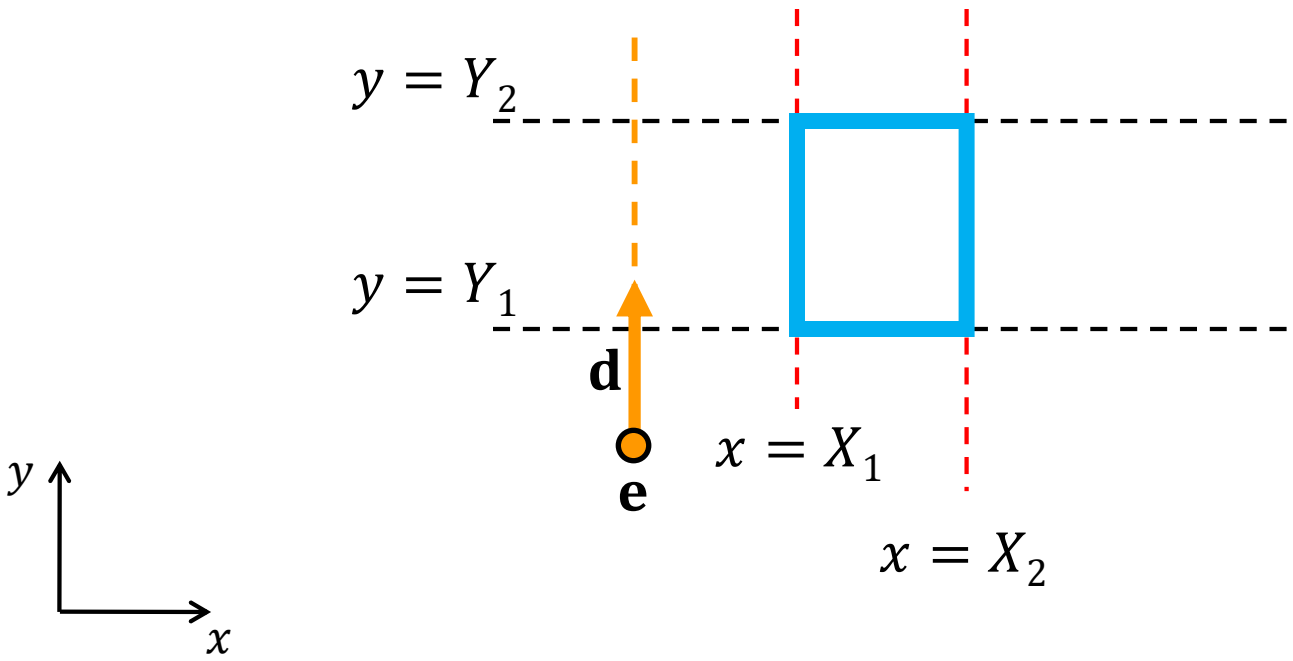
- der Schnitttest Strahl-AABB wird i.d.R. sehr oft benötigt
  → hier lohnt es sich zu optimieren!
- Beobachtung:
  - jeweils 2 Ebenen besitzen dieselbe Normale → führe Berechnungen für jede Dimension jeweils paarweise aus
  - Normalen sind achsenparallel: nur eine Komponente ungleich 0



Bounding Volume: AABB

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

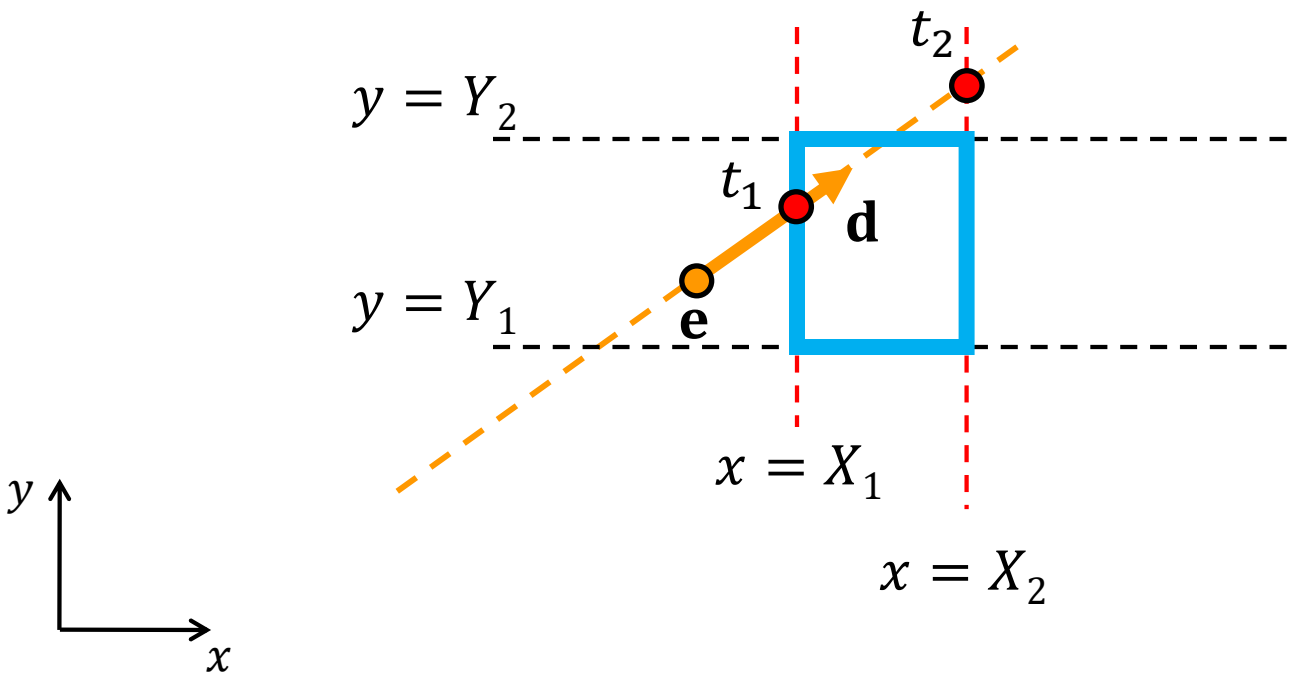
- vor den Ebenenschnitten zunächst Ausschlusstests…
- wenn $d_x = 0 \wedge (e_x < X_1 \vee e_x > X_2) \Rightarrow$ kein Schnitt
- analog für d_y und d_z



Bounding Volume: AABB

Schnittberechnung <u>für jede Dimension</u>

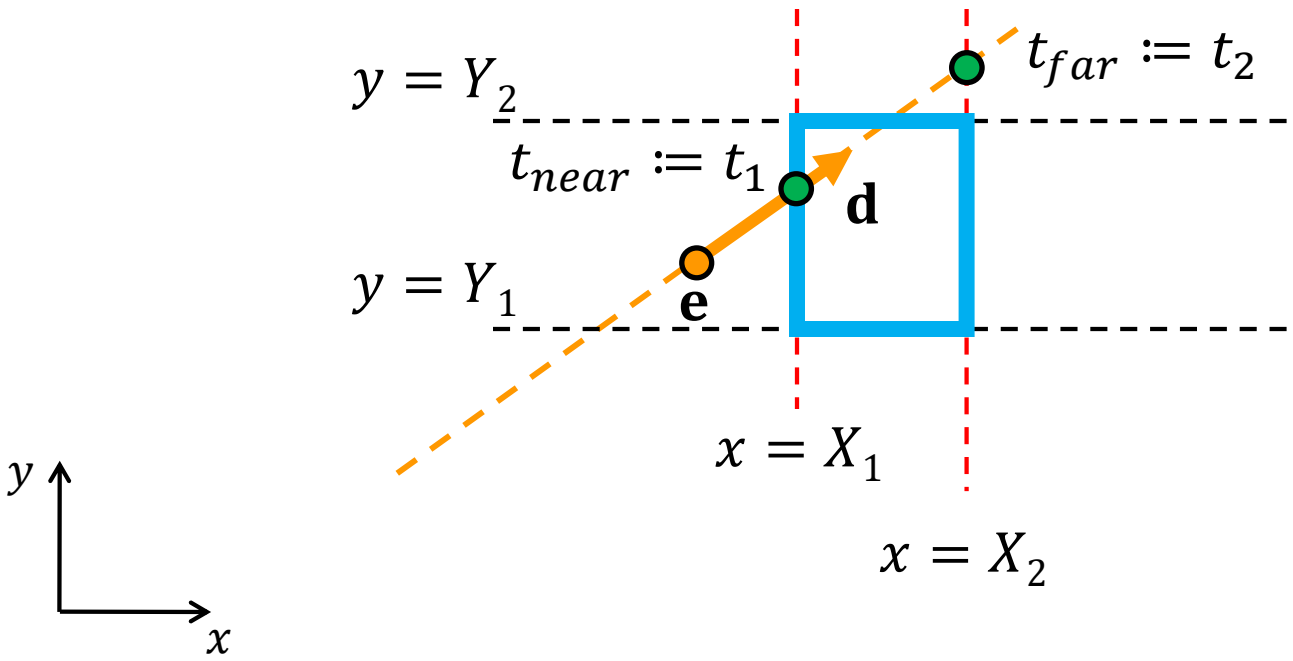
- berechne Entfernung zu den Ebenenschnitten t_1 und t_2
  - $t_1 = (X_1 - e_x)/d_x$
  - $t_2 = (X_2 - e_x)/d_x$
  - diese Berechnung ist Spezialfall des Strahl-Ebenen-Schnitt (Normalen entlang der Achsen), allgemein: $t = \frac{d - \mathbf{e}\cdot\mathbf{n}}{\mathbf{d}\cdot\mathbf{n}}$



Bounding Volume: AABB

Schnittberechnung für jede Dimension

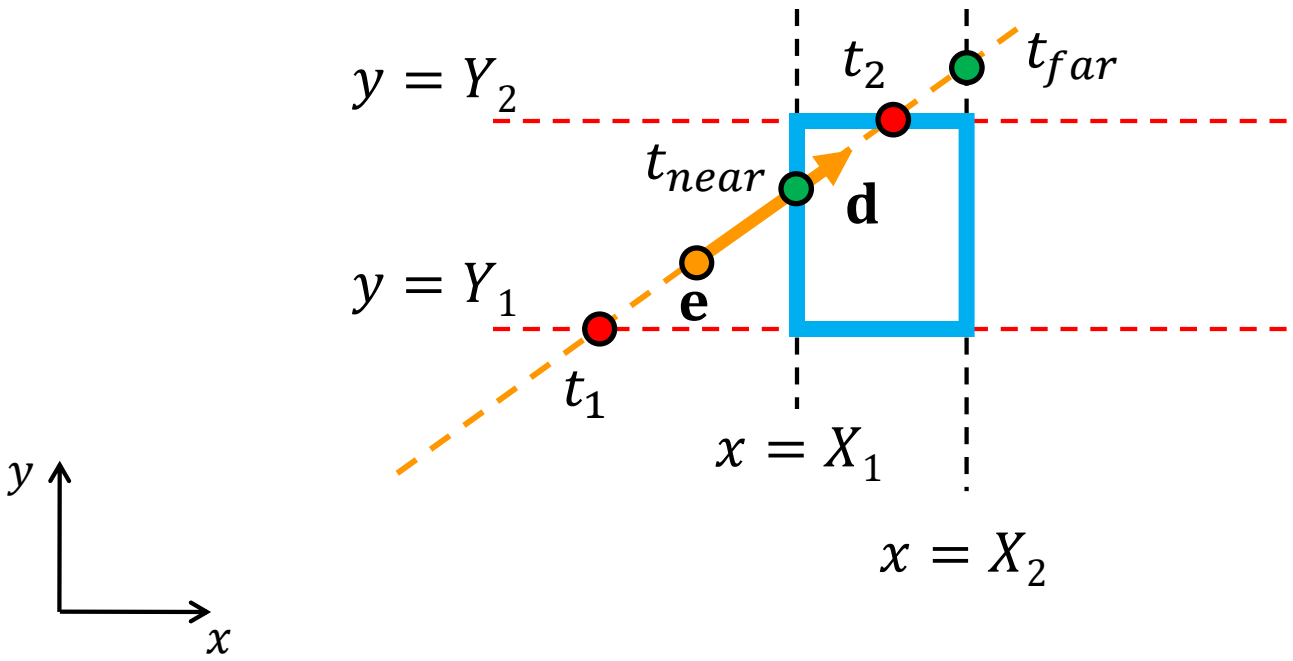
- berechne Entfernung zu den Ebenenschnitten t_1 und t_2
  - Annahme $t_1 \leq t_2$ (ansonsten werden die Werte getauscht)
  - $t_{near} \coloneqq t_1$ und $t_{far} \coloneqq t_2$
  - $[t_{near}; t_{far}]$ soll am Ende das Intervall des Strahls beschreiben, das mit der AABB überlappt



Bounding Volume: AABB

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

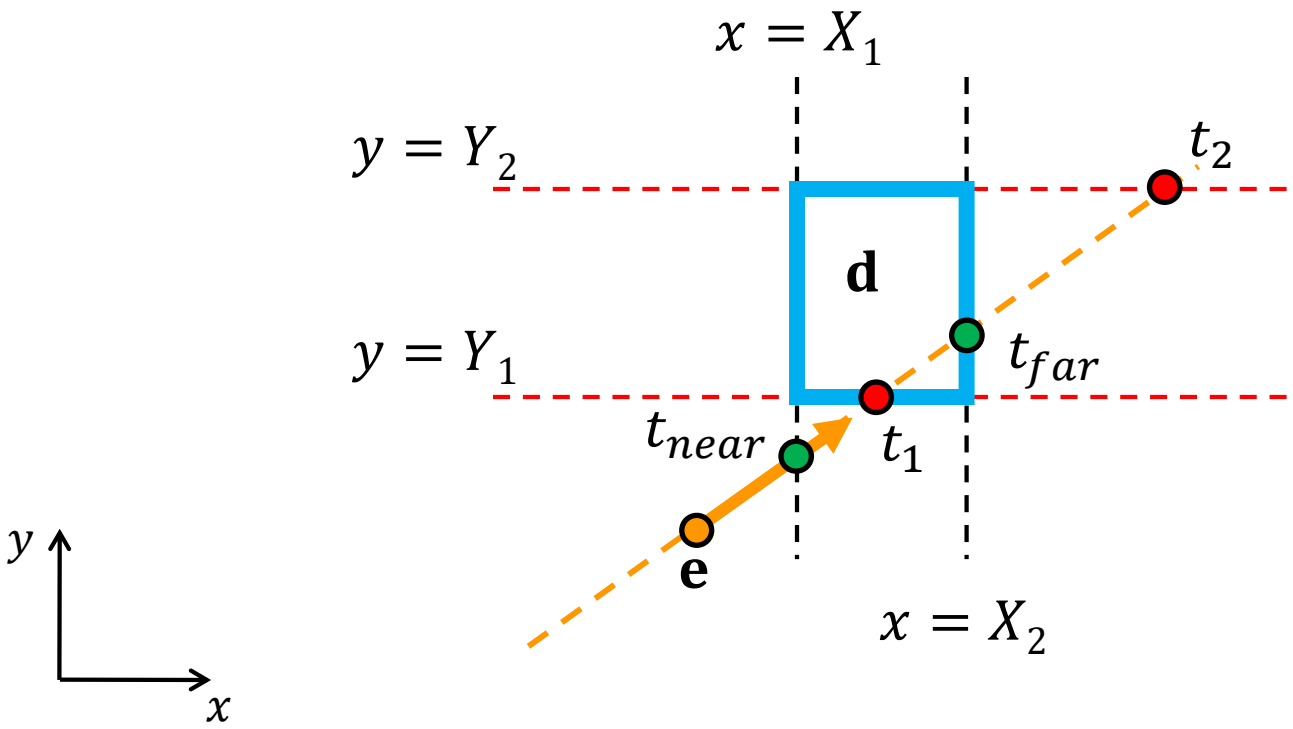
- Berechnung der Schnitte mit den weiteren Ebenenpaaren und Mitführen des nahsten und entferntesten Schnitts *mit der AABB*
  - Annahme $t_1 \leq t_2$
  - wenn $t_1 > t_{near}$, dann $t_{near} = t_1$ (hier nicht der Fall)
  - wenn $t_2 < t_{far}$, dann $t_{far} = t_2$



Bounding Volume: AABB

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

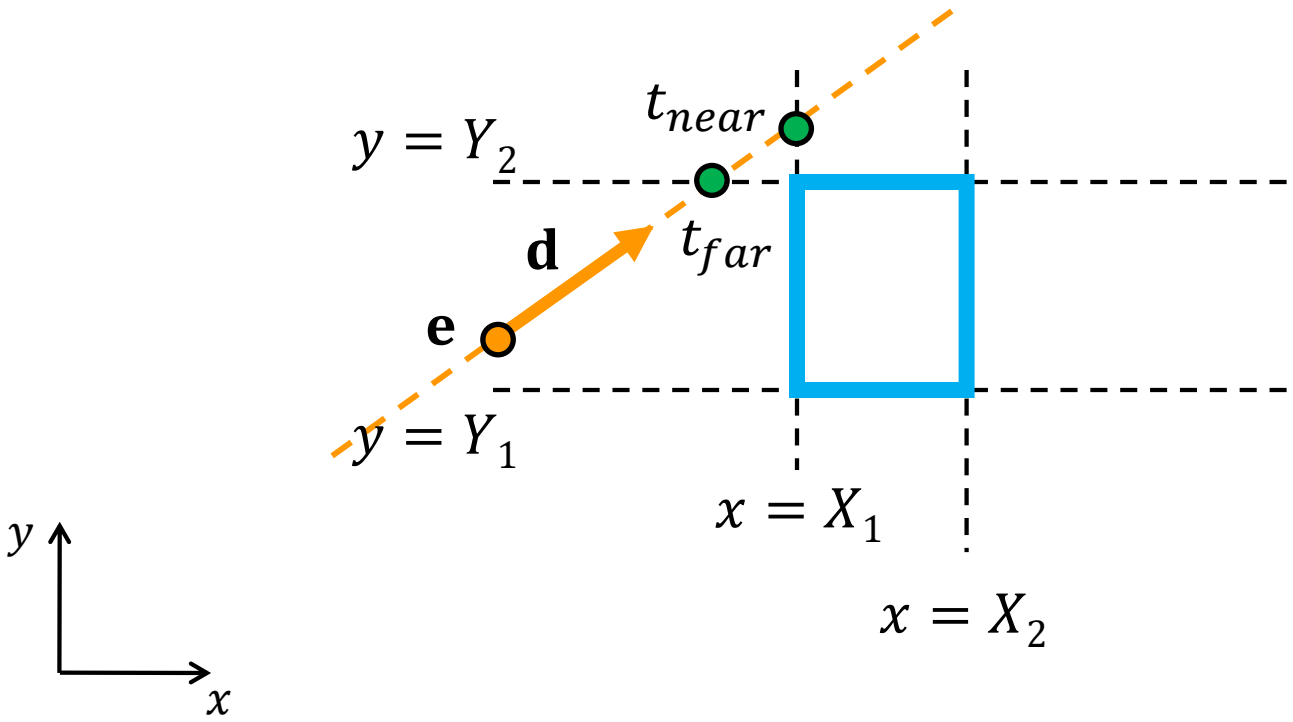
- Berechnung der Schnitte mit den weiteren Ebenenpaaren und Mitführen des nahsten und entferntesten Schnitts *mit der AABB*
  - wenn $t_1 > t_{near}$, dann $t_{near} = t_1$
  - wenn $t_2 < t_{far}$, dann $t_{far} = t_2$ (hier nicht der Fall)
- am Ende beschreibt $[t_{near}; t_{far}]$ den Überlapp des Strahls mit der AABB



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: existiert ein Schnittpunkt?

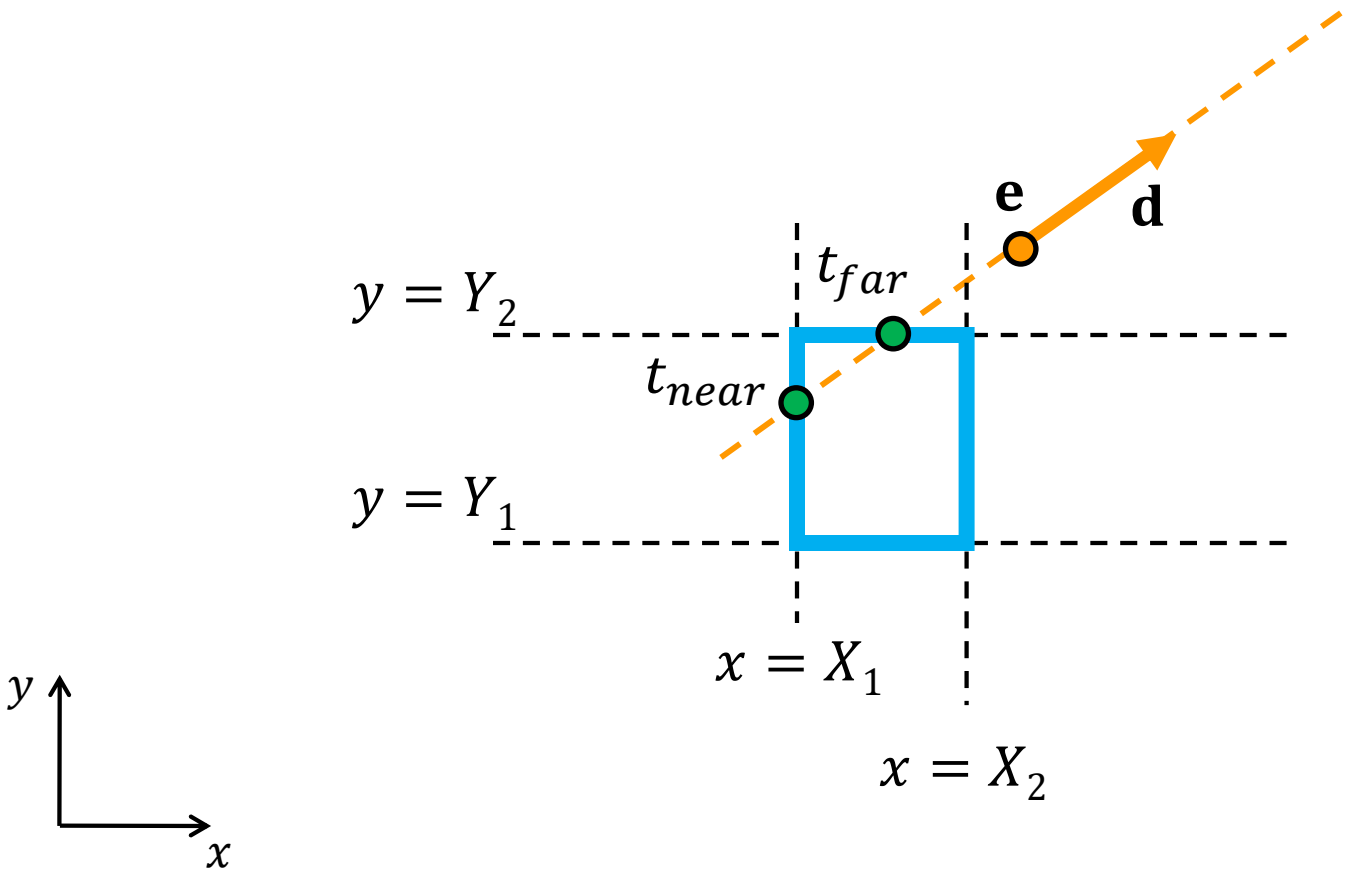
- am Ende beschreibt $[t_{near}; t_{far}]$ den Überlapp des Strahls mit der AABB, *außer in den folgenden Fällen…*
- wenn $t_{near} > t_{far}$ wird die Box nicht getroffen
- hier: der Strahl verlässt das Intervall $[Y_1; Y_2]$ bevor er $[X_1; X_2]$ erreicht



Bounding Volume: AABB

Schnittberechnung Strahl-AABB: Box hinter dem Strahlursprung?

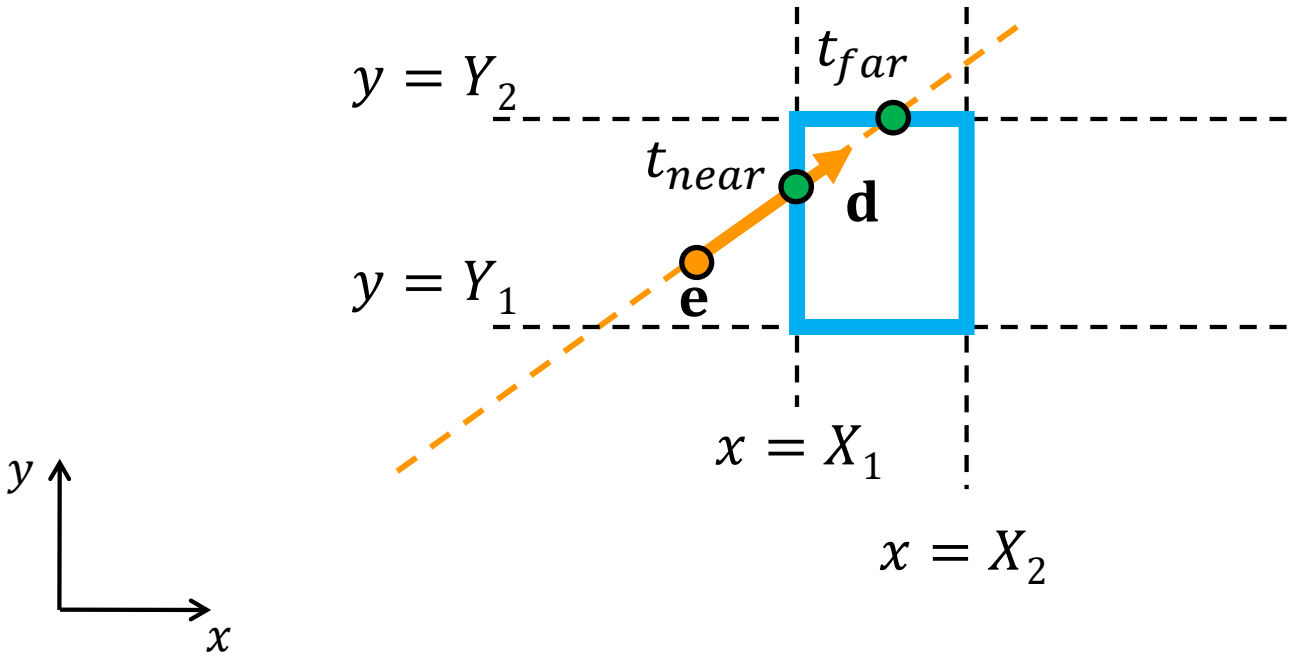
- wenn $t_{far} < 0$ ist die Box hinter dem Strahl



Bounding Volume: AABB

Richtigen Schnittpunkt zurückliefern (sofern benötigt)

- wenn $t_{near} > 0 \rightarrow$ nahster Schnittpunkt bei t_{near}
- sonst: Schnittpunkt bei t_{far} (Strahl beginnt innerhalb der AABB)
- t_{near} und t_{far} helfen uns also die Fälle elegant zu unterscheiden
- auch wenn wir gerade nur daran interessiert zu testen, **ob** es einen Schnitt mit der AABB gibt, brauchen wir das Strahlsegment später noch



Bounding Volume: AABB

Zusammenfassung Schnittberechnung Strahl-AABB

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

- für jede Dimension:
  - berechne $t_1 = (X_1 - e_x)/d_x$ und $t_2 = (X_2 - e_x)/d_x$
  - wenn $t_1 > t_2$ tausche t_1 und t_2 (benötigt wenn $d_x, d_y, d_z < 0$)
  - initialisiere bzw. 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 $t_{near} > t_{far}$ → Box nicht getroffen
- wenn $t_{far} < 0$ → Box hinter dem Strahl
- wenn $t_{near} > 0$ → nahster Schnitt bei t_{near}
- sonst → nahster Schnitt bei t_{far}

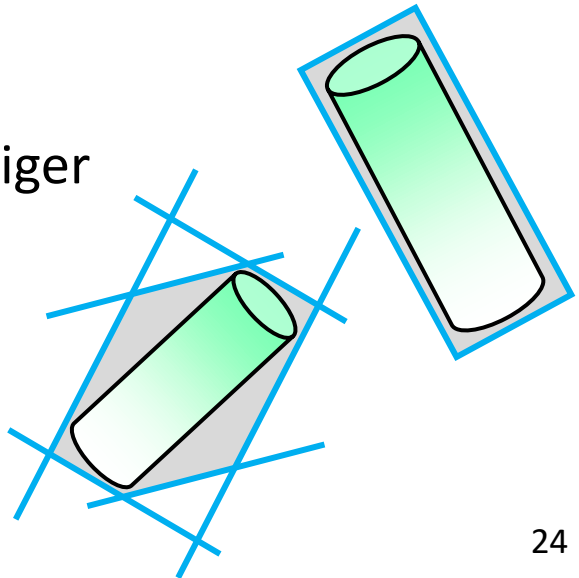
Bounding Volume: AABB

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**

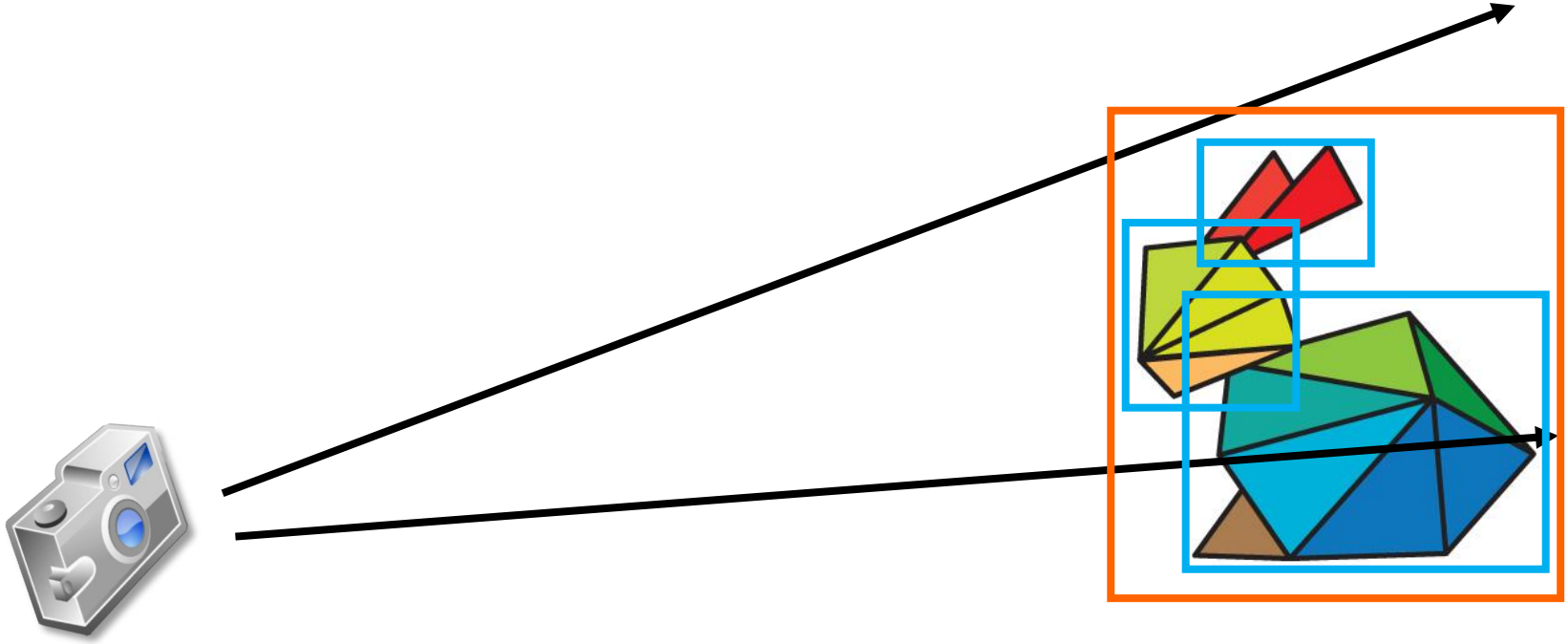
- Verallgemeinerung des Schnitttests auf OBBs und Slabs möglich
  - parallele Ebenenpaare
  - konvexe Form des Hüllkörpers
  - nur Berechnung von t_1 und t_2 ist etwas aufwändiger (wegen der beliebigen Orientierung der Ebenen)



Beschleunigung des Raycasting

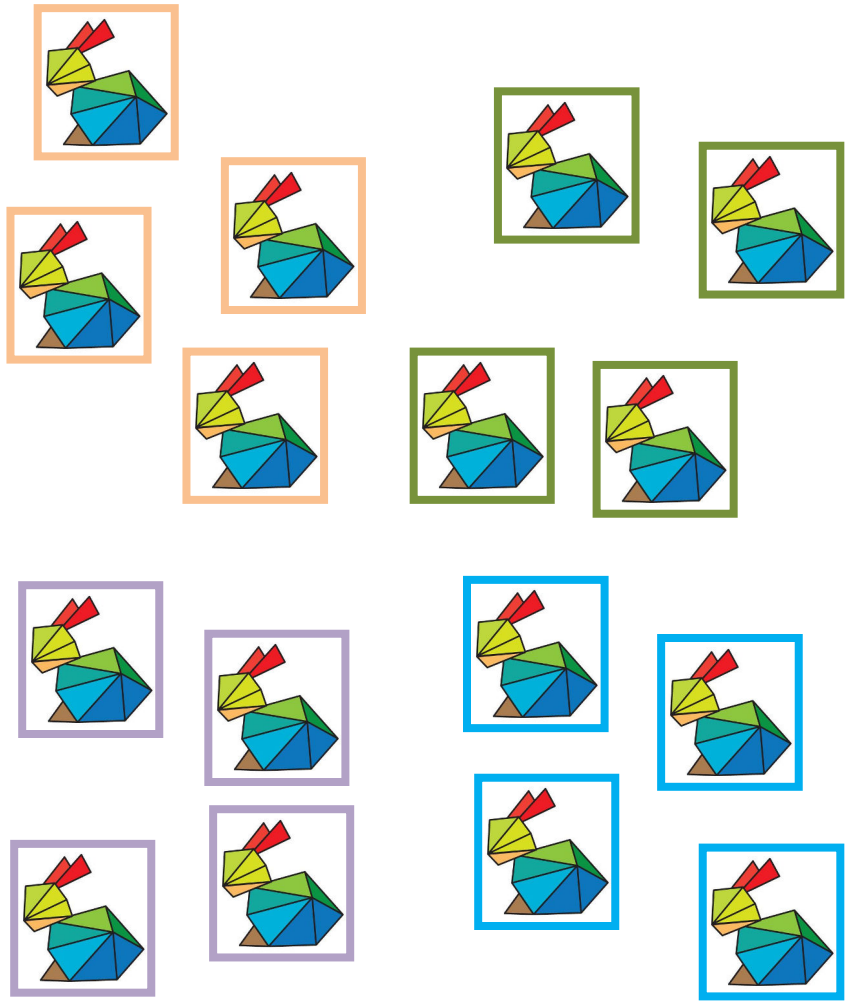
Reduziere die Anzahl der Strahl-Primitiv-Schnitttests

- schneller Ausschluss von Schnitttests mit Primitiven (Dreiecken)
- wie viel Geometrie und welche Teile davon soll ein einzelner Hüllkörper einschließen?
- wie geht man mit komplexen Szenen um?



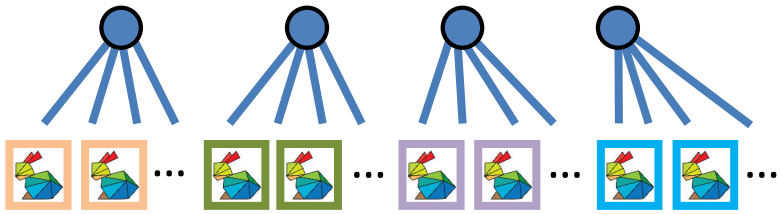
Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern



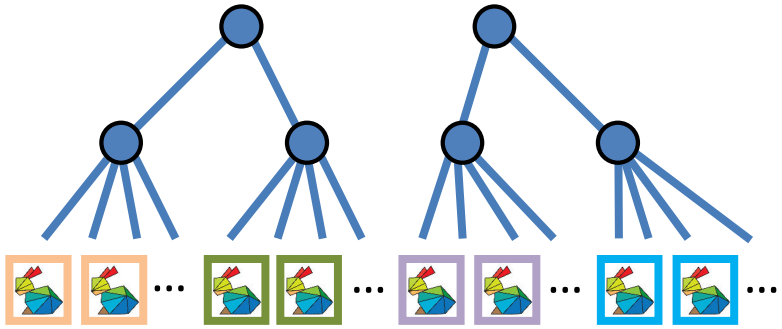
Beschleunigung des Raycasting

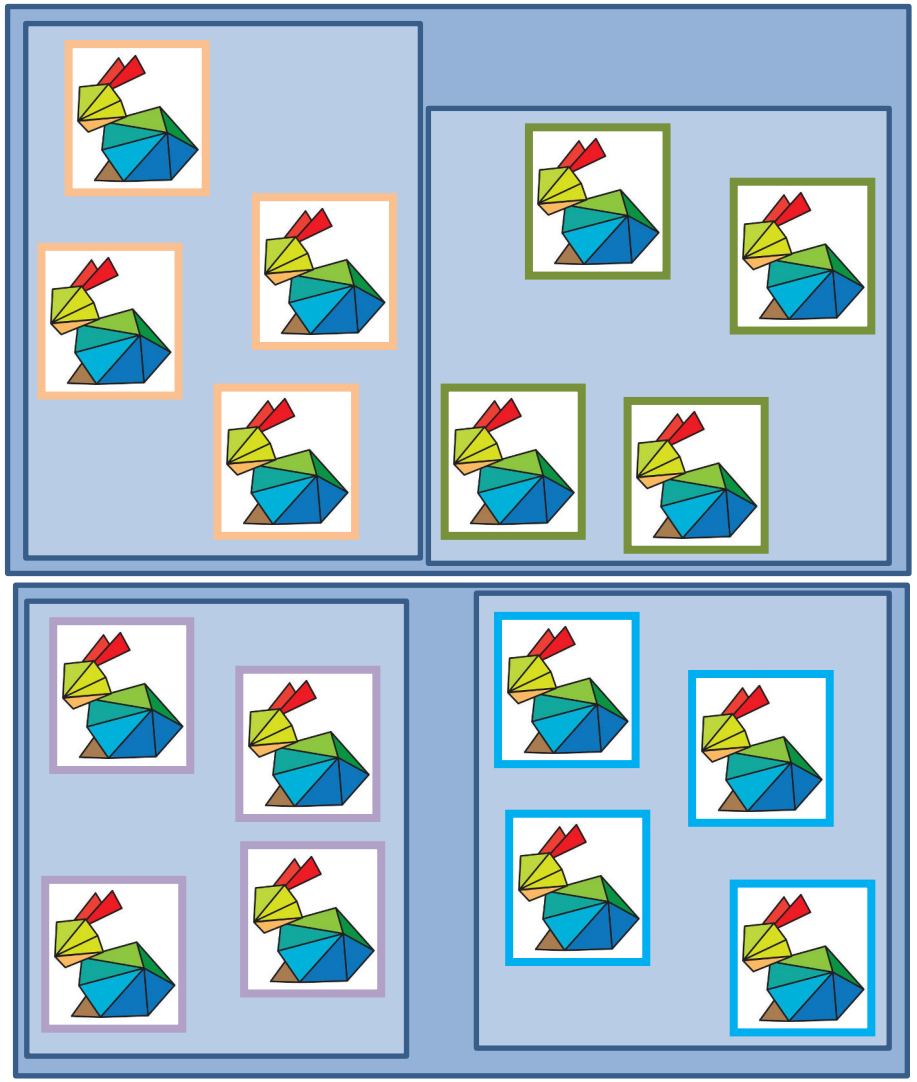
Grundidee: Hierarchie von einschließenden Hüllkörpern



Beschleunigung des Raycasting

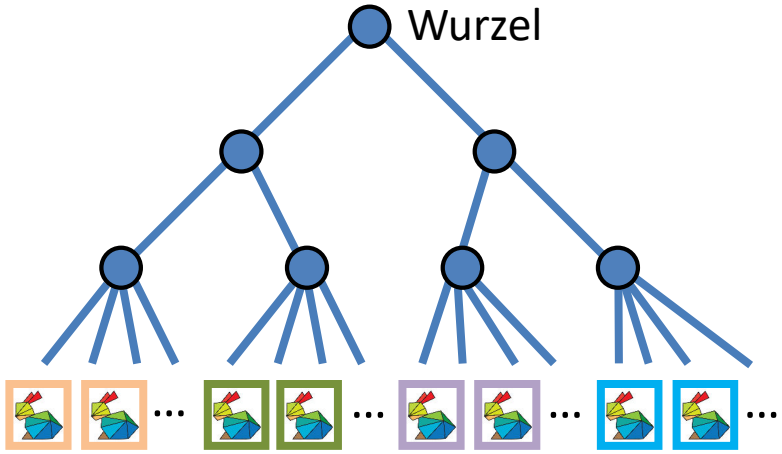
Grundidee: Hierarchie von einschließenden Hüllkörpern

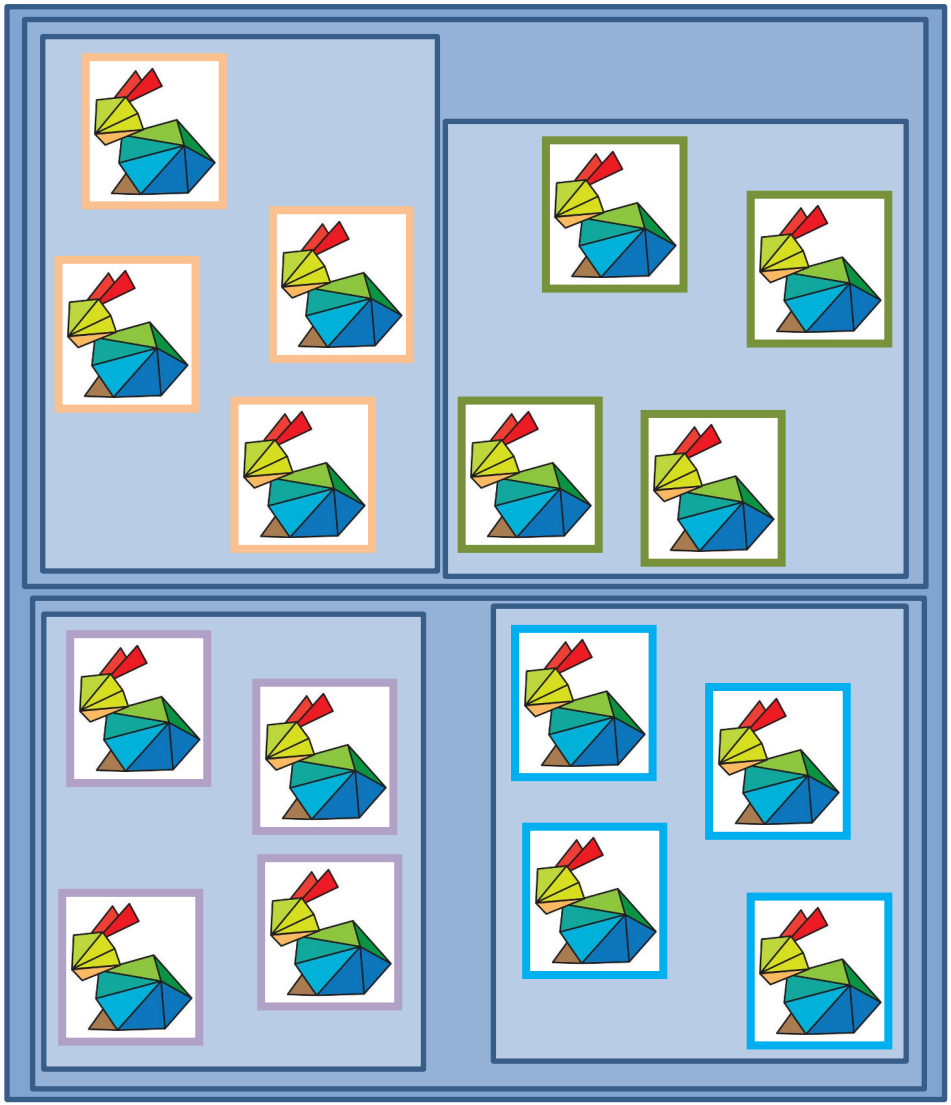




Beschleunigung des Raycasting

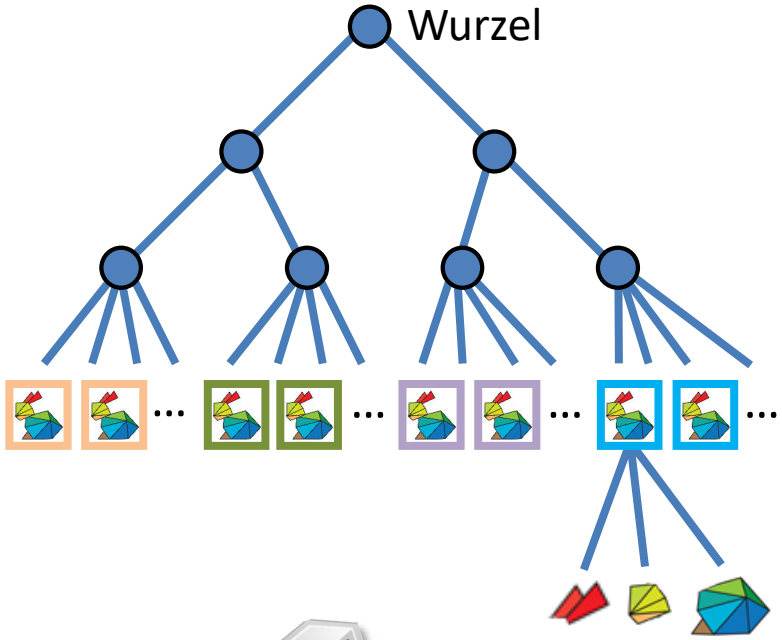
Grundidee: Hierarchie von einschließenden Hüllkörpern

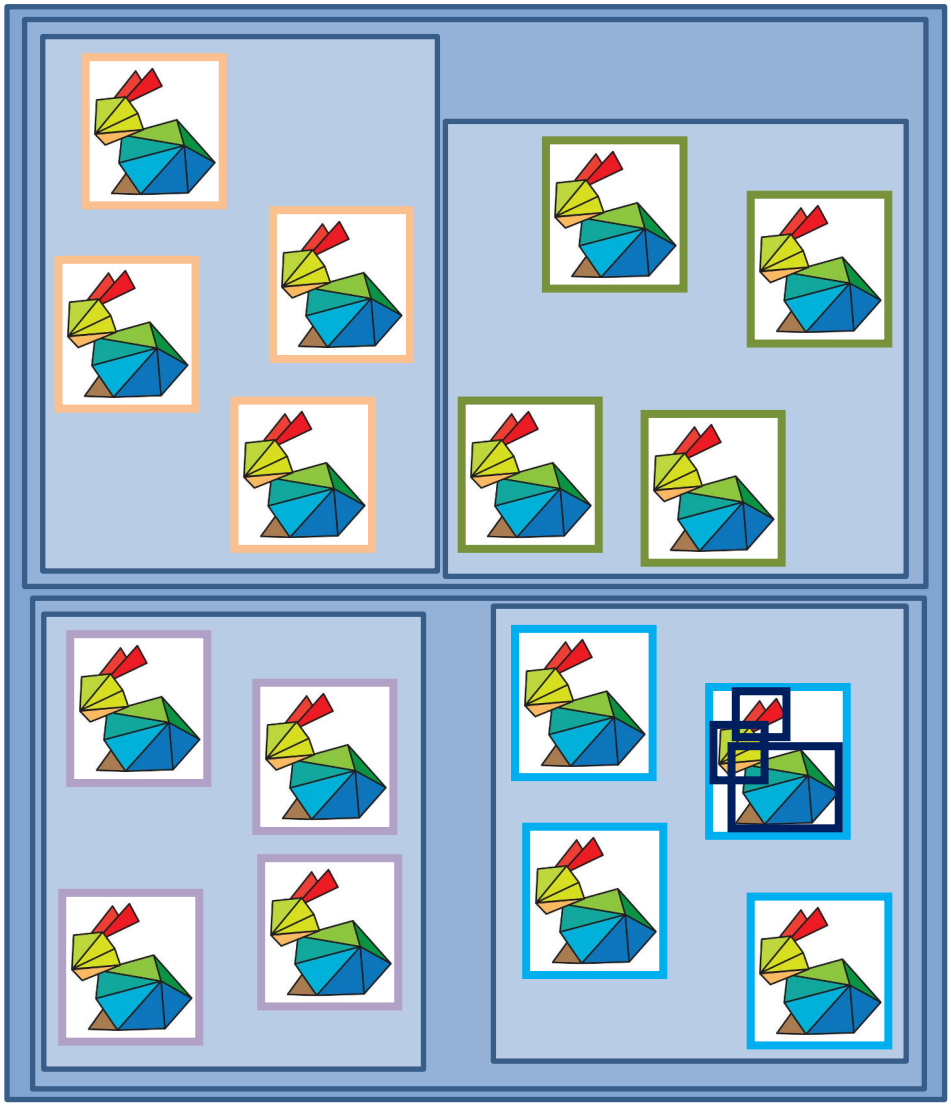




Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

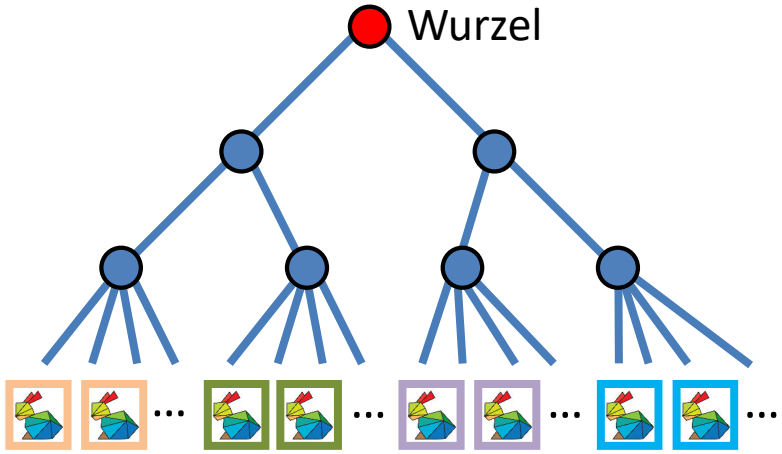


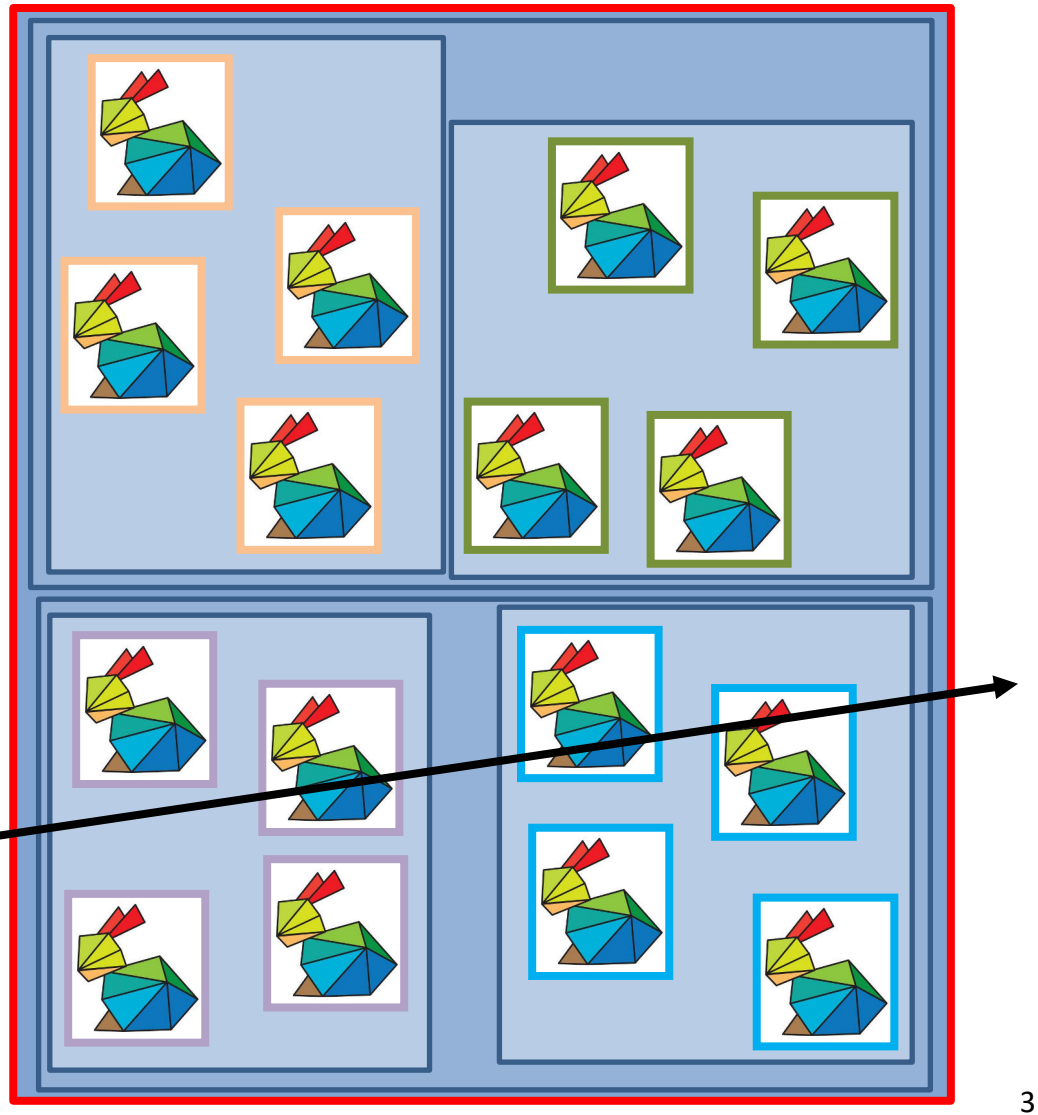


Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

- schnelle Suche:
  ganze Gruppen können
  ausgeschlossen werden

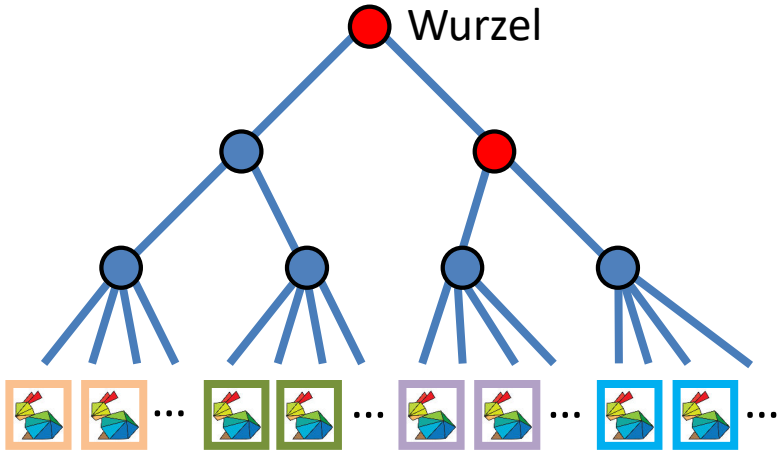


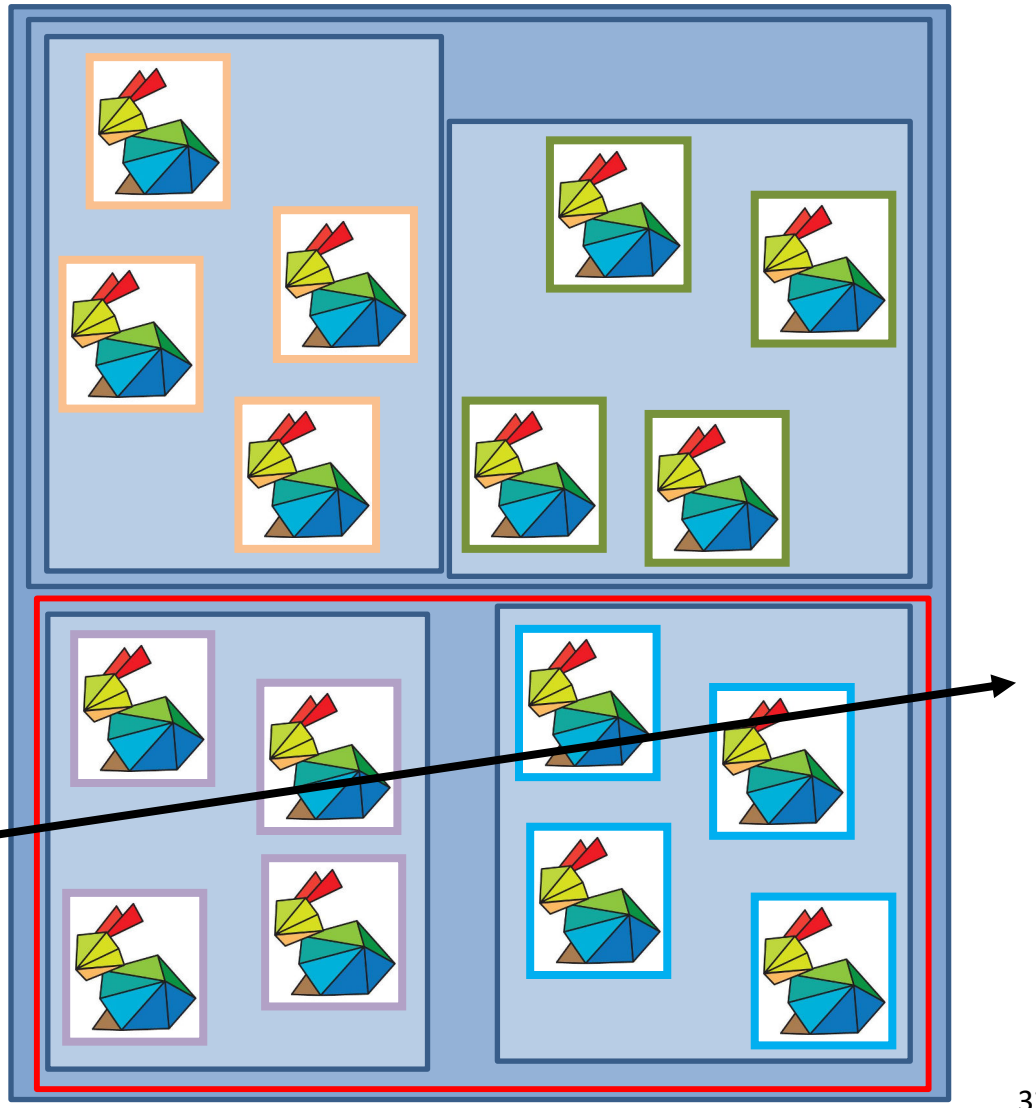


Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

- schnelle Suche:
  ganze Gruppen können
  ausgeschlossen werden

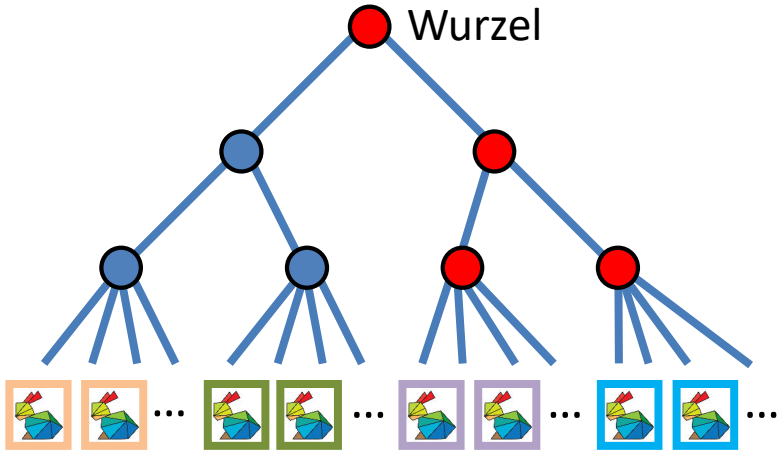


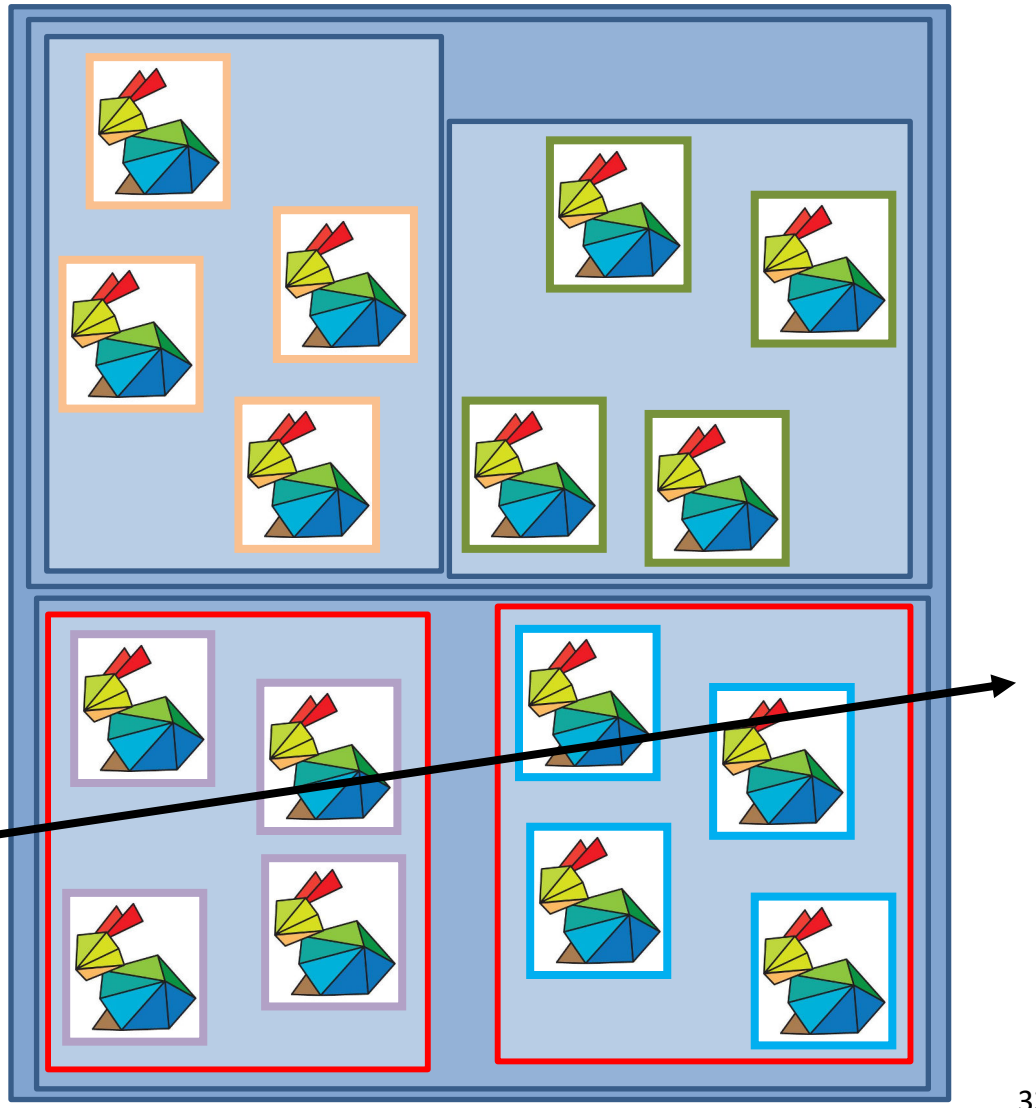


Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

- schnelle Suche:
  ganze Gruppen können
  ausgeschlossen werden

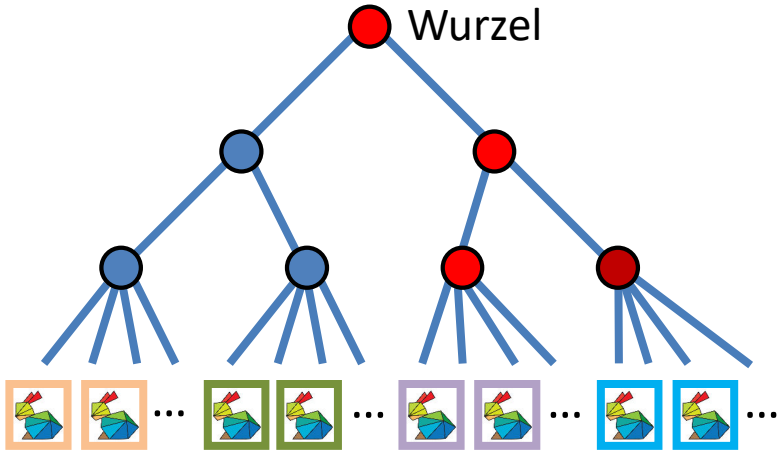


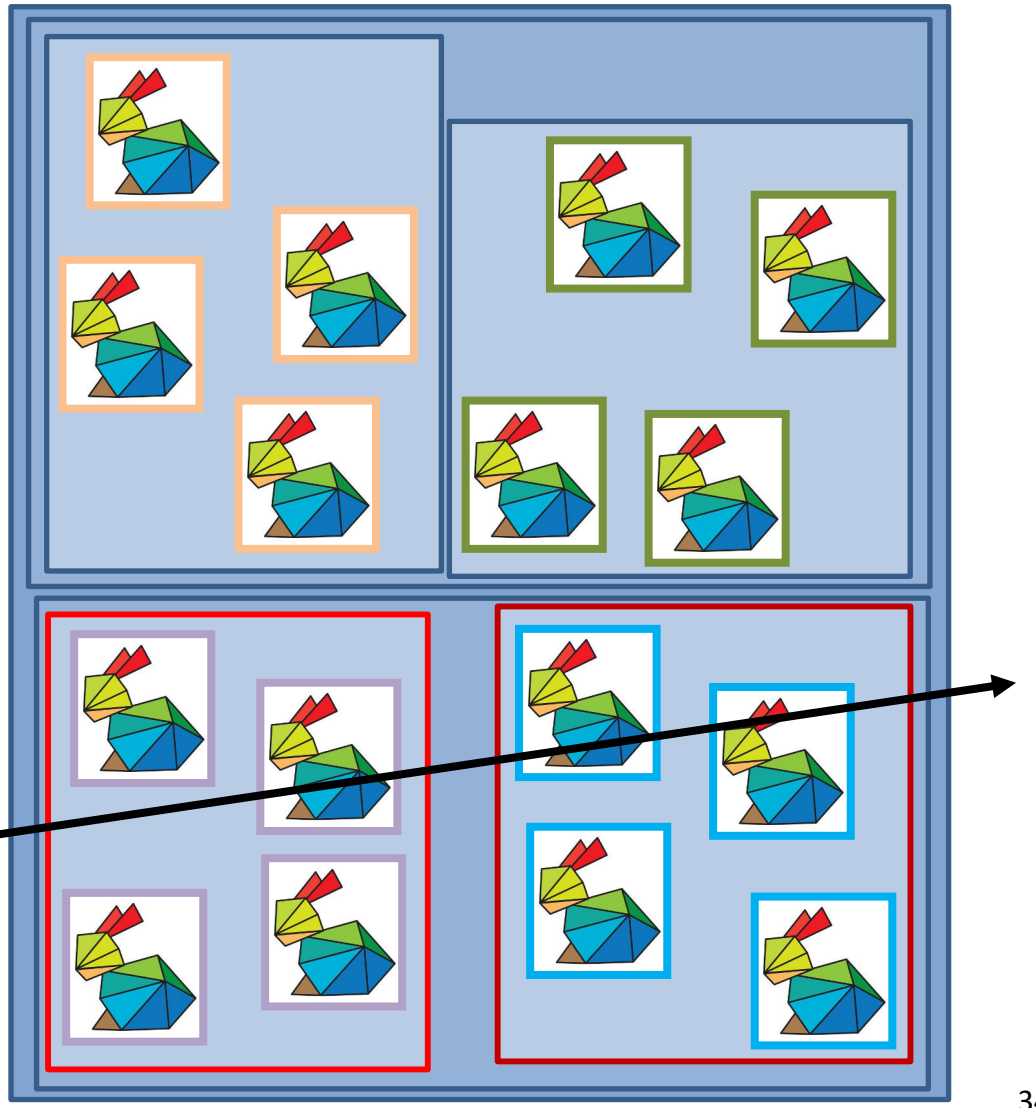


Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

- schnelle Suche:
  ganze Gruppen können
  ausgeschlossen werden

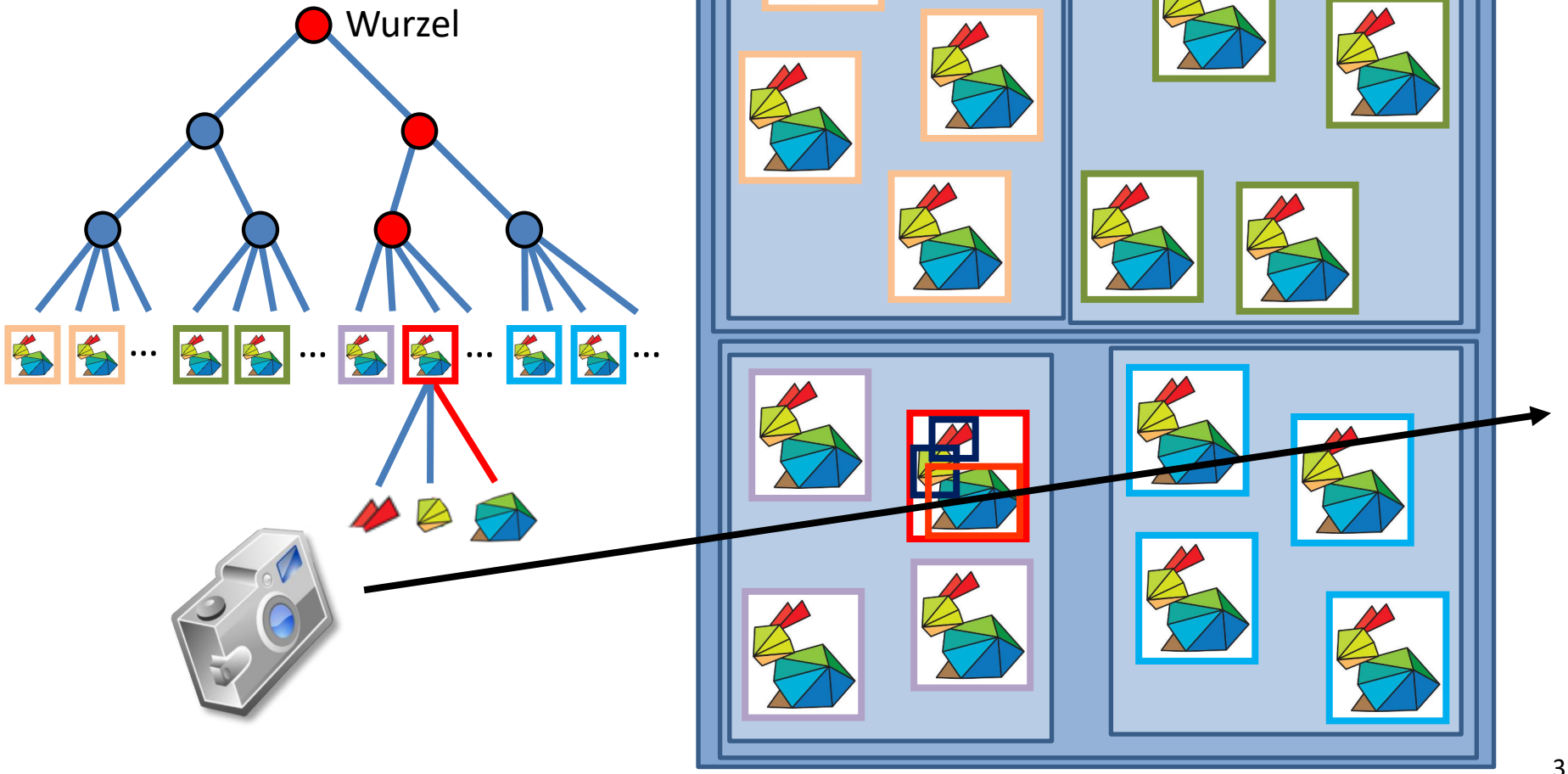




Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

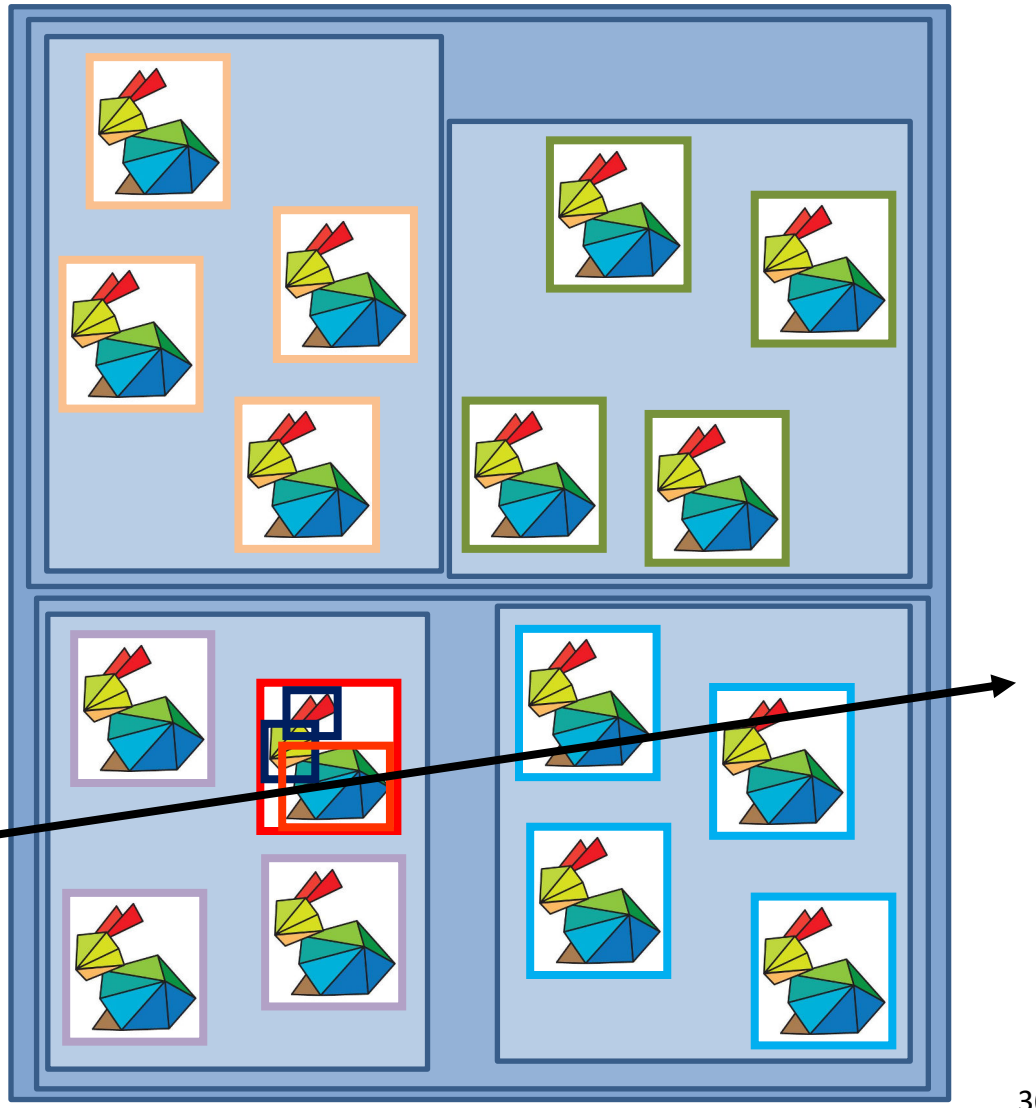
- schnelle Suche:
  ganze Gruppen können
  ausgeschlossen werden



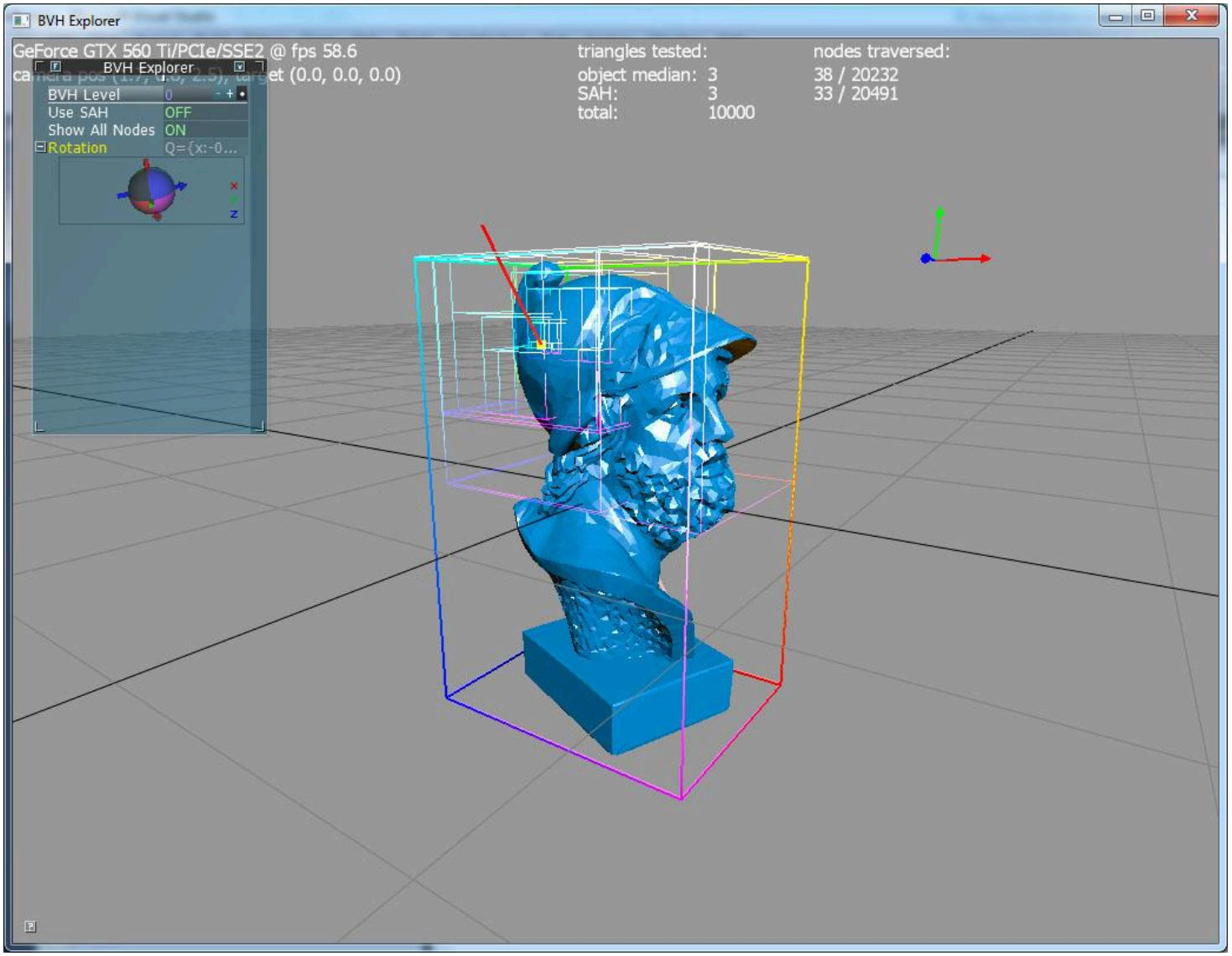
Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

- schnelle Suche:
  ganze Gruppen können ausgeschlossen werden
- … durch automatisches Zusammenfassen von (Gruppen von) Primitiven, oder rekursives Unterteilen der Primitive in Gruppen
- Tiefensuche im Baum,
  Kosten für Strahlschuss:
  $O(n) \rightarrow O(\log n)$



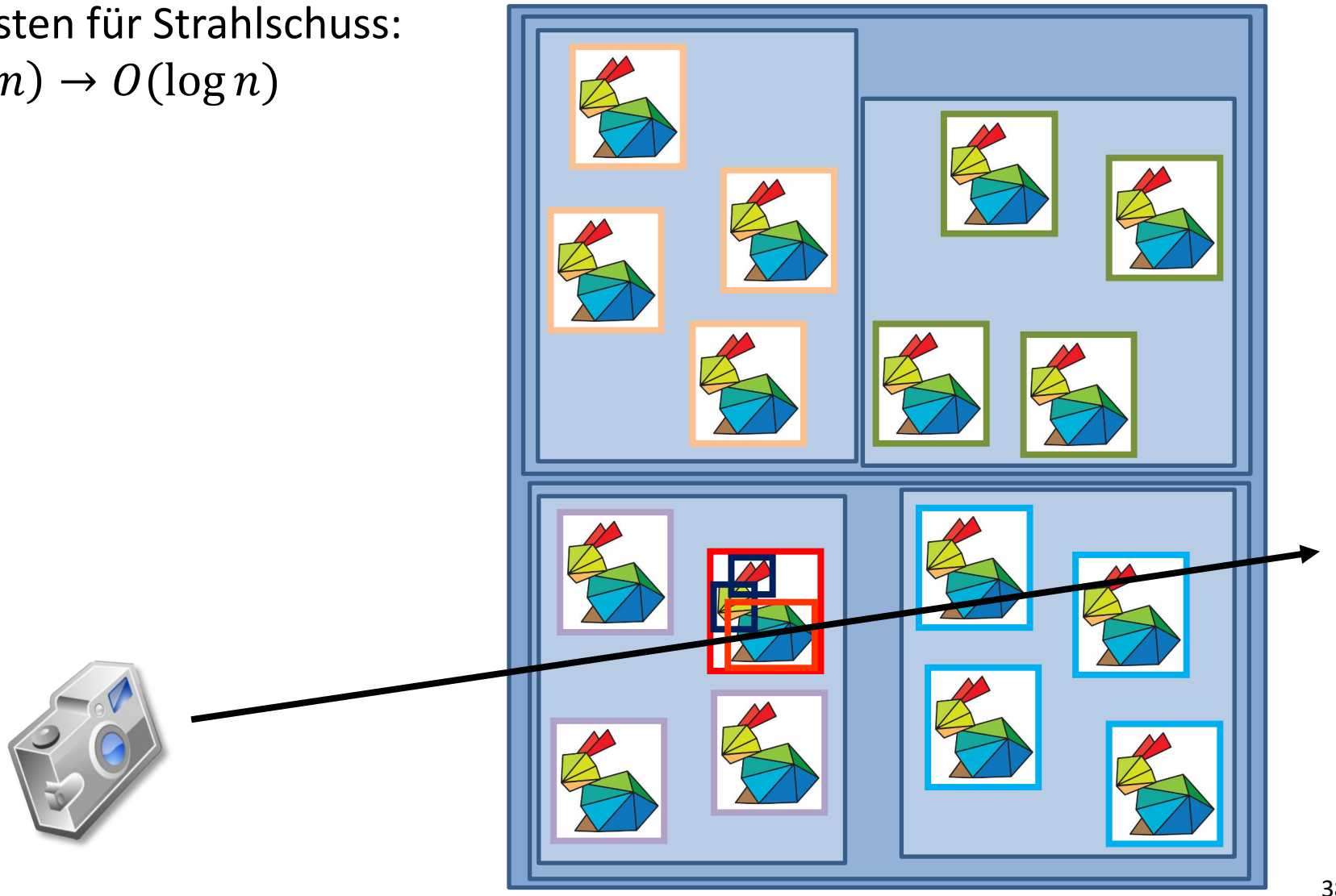
Hüllkörper-Hierarchie (engl. bounding volume hierachy, BVH)



Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

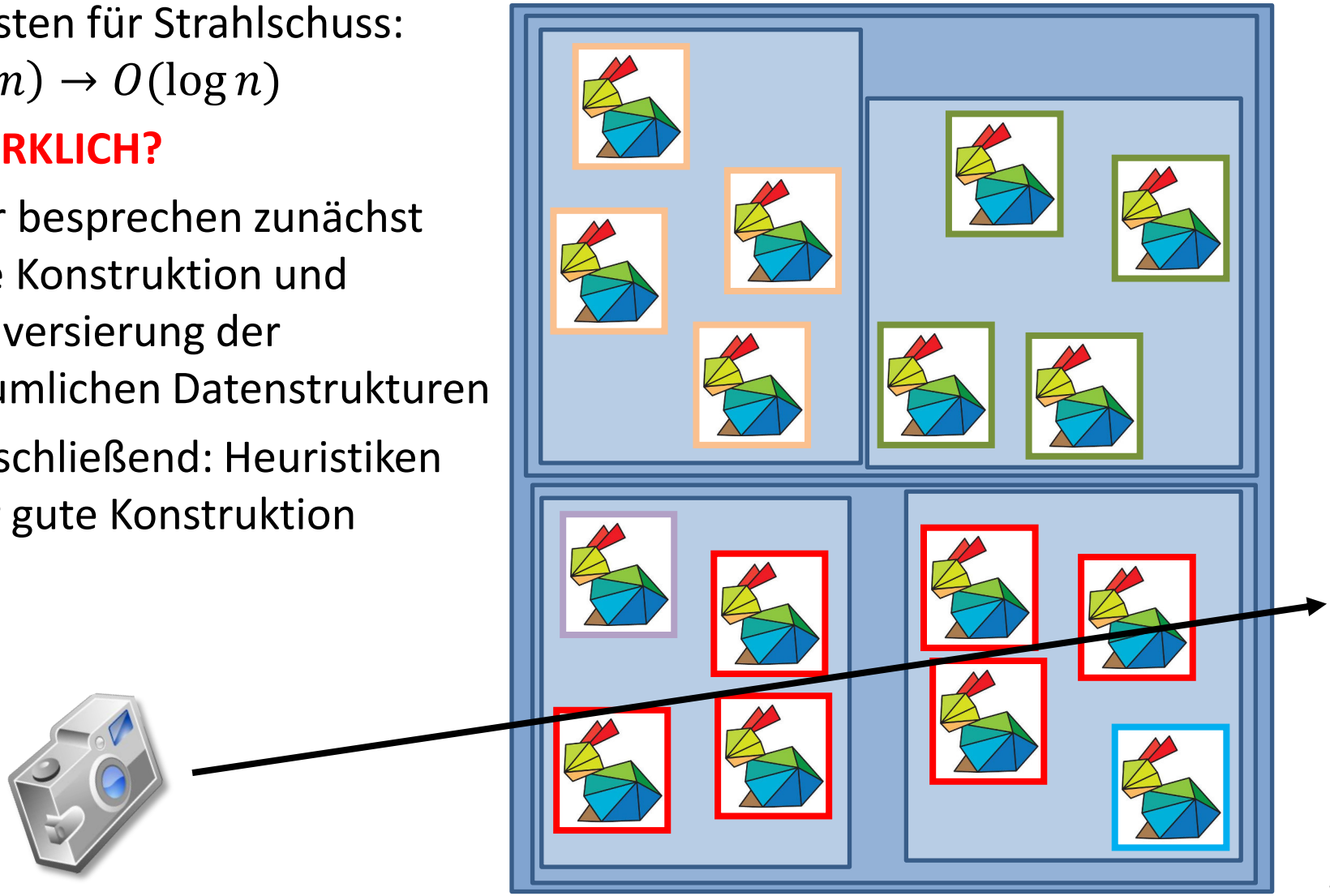
- Tiefensuche im Baum,
  Kosten für Strahlschuss:
  $O(n) \rightarrow O(\log n)$



Beschleunigung des Raycasting

Grundidee: Hierarchie von einschließenden Hüllkörpern

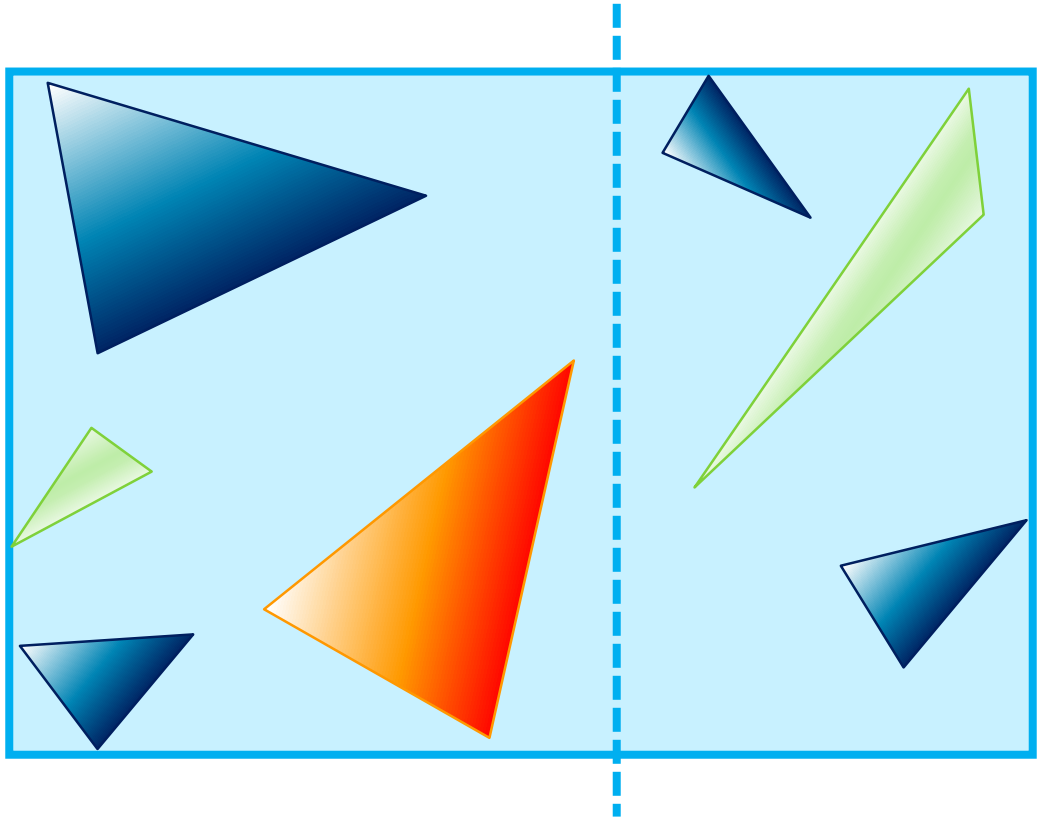
- Tiefensuche im Baum, Kosten für Strahlschuss: $O(n) \rightarrow O(\log n)$
- **WIRKLICH?**
- wir besprechen zunächst die Konstruktion und Traversierung der räumlichen Datenstrukturen
- anschließend: Heuristiken für gute Konstruktion



Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

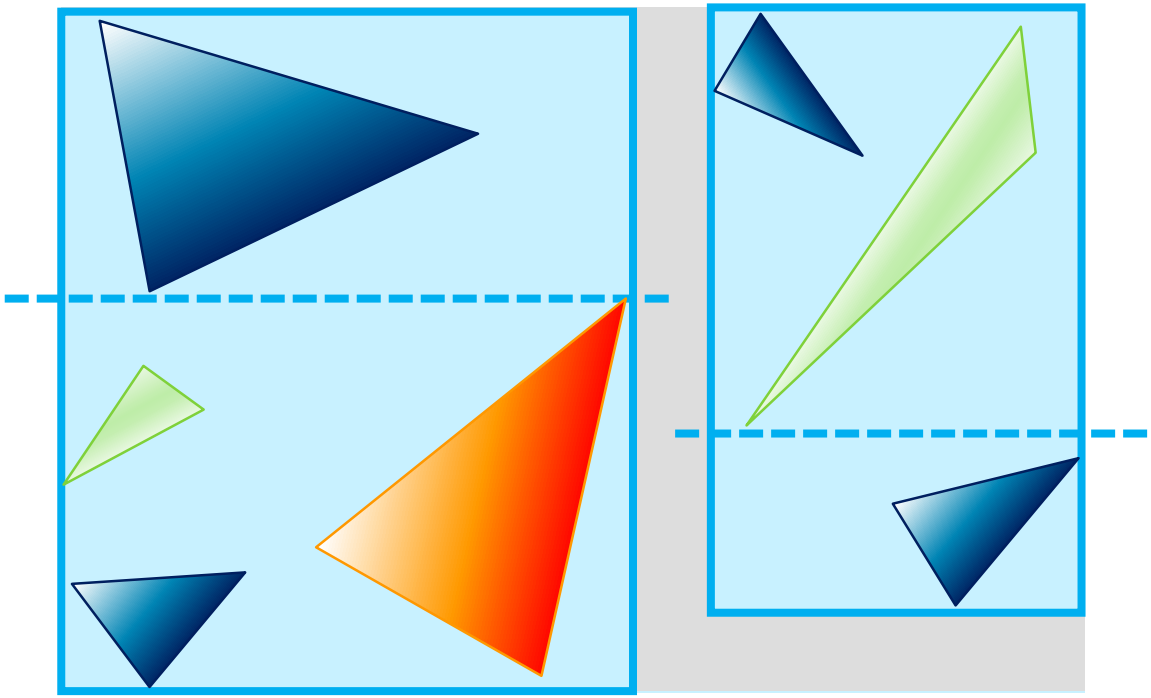
- bestimme den Hüllquader der Primitive jeder Gruppe (anfangs: alle Δ)
- teile die Primitive in **zwei Gruppen** auf
- die Linie ---- dient zur Illustration der Aufteilung



Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

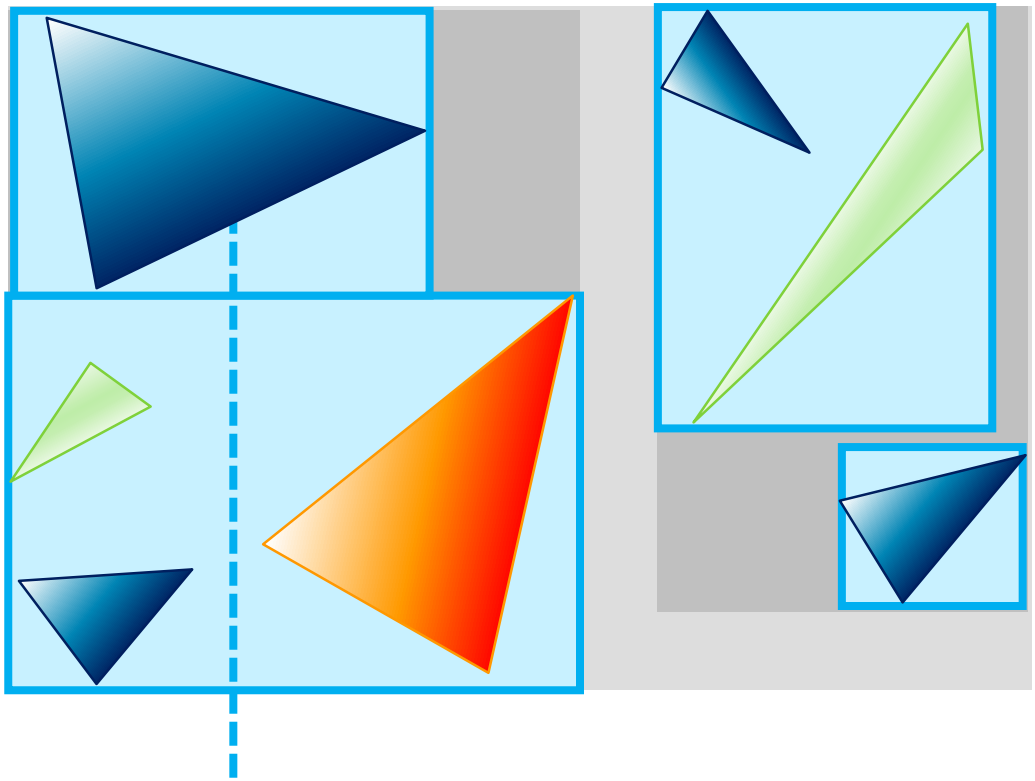
- bestimme den Hüllquader der Primitive jeder Gruppe
- teile die Primitive jeweils wieder in **zwei Gruppen** auf
- verfahre so rekursiv weiter



Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

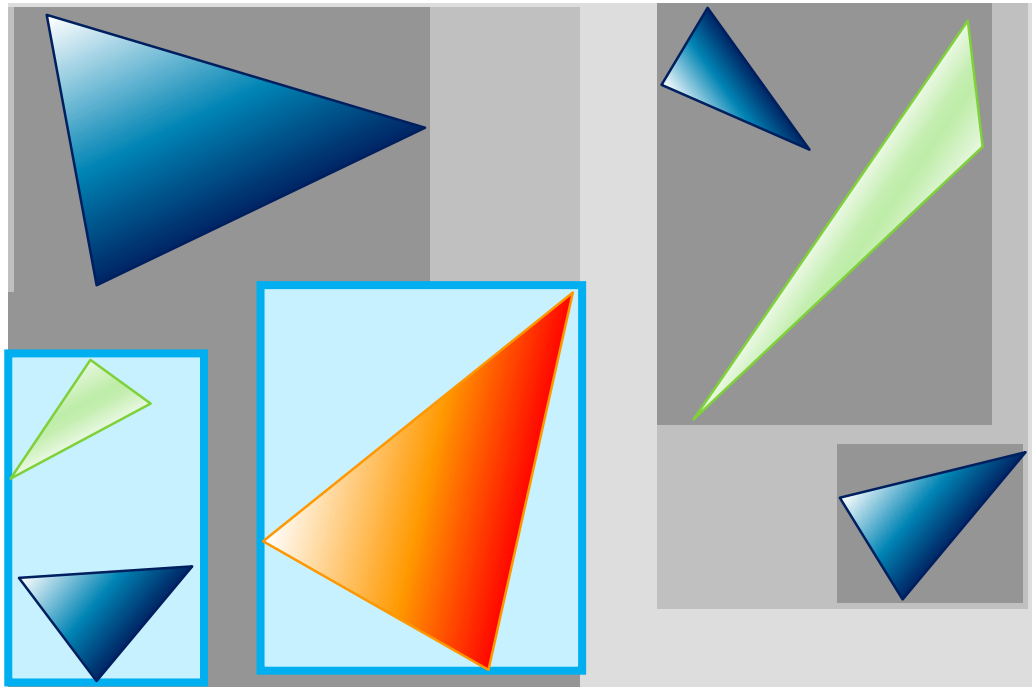
- bestimme den Hüllquader der Primitive jeder Gruppe
- teile die Primitive jeweils wieder in **zwei Gruppen** auf
- verfahre so rekursiv weiter



Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

- bestimme den Hüllquader der Primitive jeder Gruppe
- teile die Primitive jeweils wieder in **zwei Gruppen** auf
- verfahre so rekursiv weiter

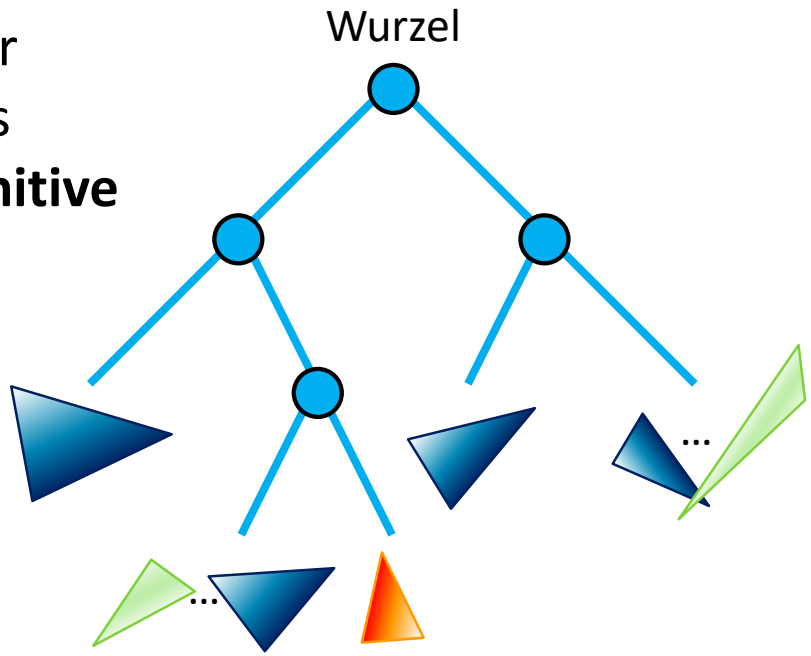


Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

- bestimme den Hüllquader der Primitive jeder Gruppe
- teile die Primitive jeweils wieder in **zwei Gruppen** auf
- verfahre so rekursiv weiter

- jeder Knoten speichert die Hüllkörper seiner Kindknoten und einen Verweis auf diese bzw. auf eine **Liste der Primitive**

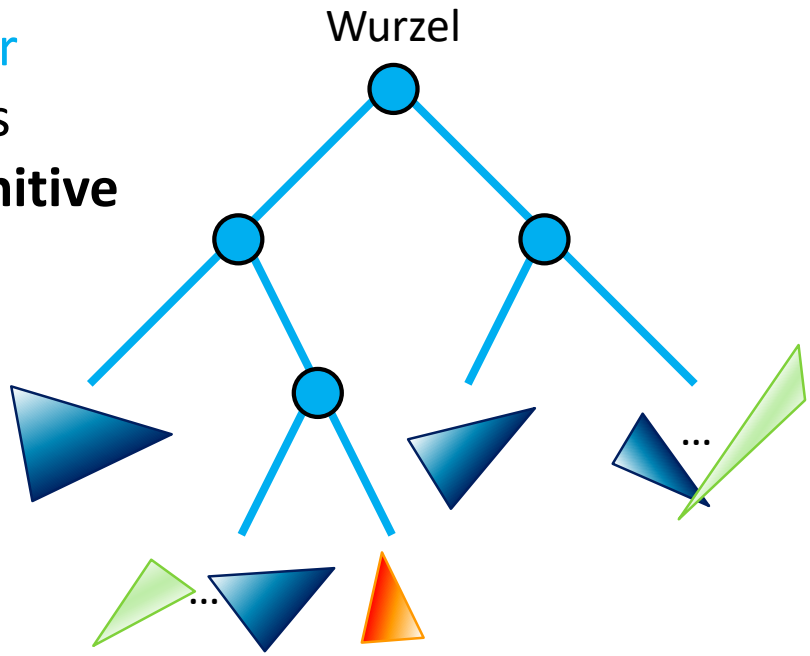


Hüllkörper-Hierarchie

Aufbau der Datenstruktur (top-down)

- bestimme den Hüllquader der Primitive jeder Gruppe
- teile die Primitive jeweils wieder in **zwei Gruppen** auf
- verfahre so rekursiv weiter

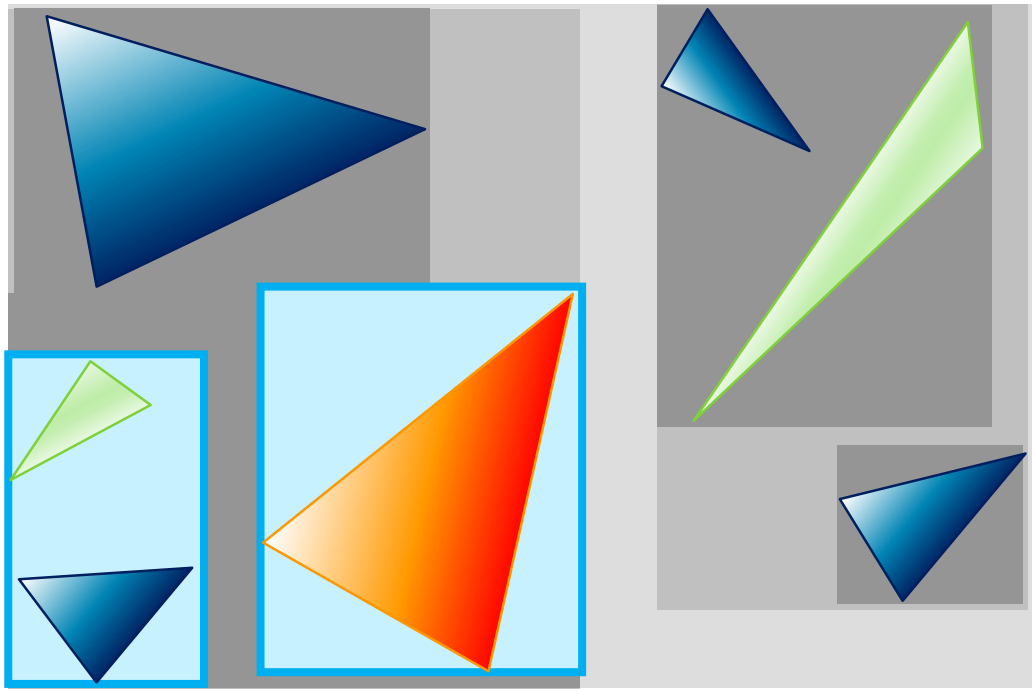
- jeder Knoten speichert die Hüllkörper seiner Kindknoten und einen Verweis auf diese bzw. auf eine **Liste der Primitive**
  - Information wird zusammen benötigt und entsprechend gespeichert (Caching!)



Bounding Volume-Hierarchie

Unterschiedliche Unterteilungskriterien (später mehr)

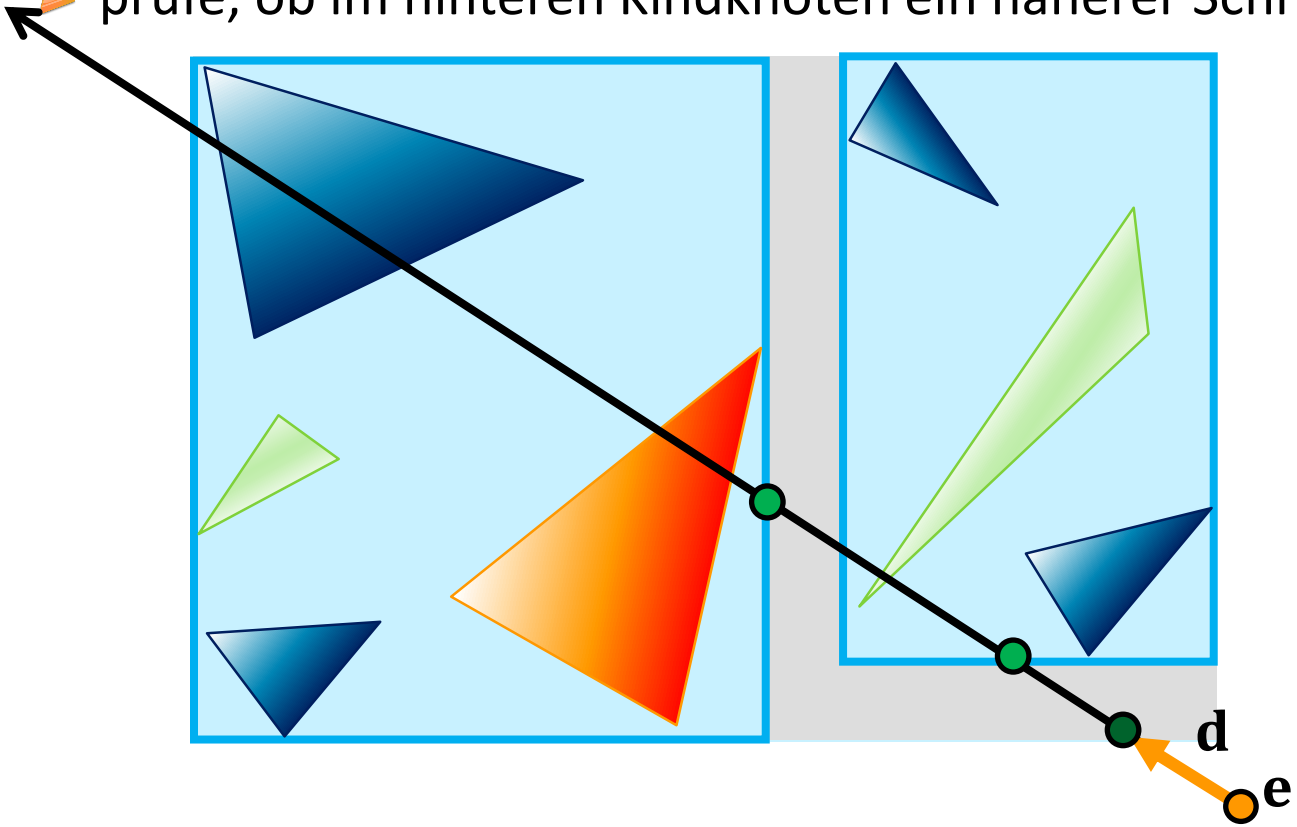
- unterteile in der Mitte senkrecht zur Achse der größten Ausdehnung
- unterteile einmal entlang der x-Achse, dann y-Achse, dann z-Achse, …
- unterteile so, dass in jeder Teilmenge etwa die gleiche Anzahl Objekte ist
- verwende Szenengraph bzw. Modellhierarchie
- minimiere eine Kostenfunktion (z.B. Surface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

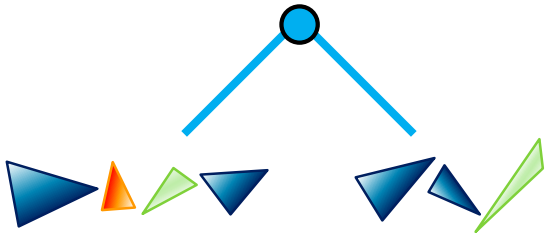


Schnittberechnung mit Hüllkörper-Hierarchien

Traversierung, Schnittberechnung

- überprüfe auf Schnitt mit der Wurzel ●
- teste Hüllkörper der Kindknoten, traversiere näheren Kindknoten zuerst
  - innere Knoten: teste auf Schnitt mit Hüllkörpern weiterer Kindknoten
  - in Blattknoten: teste auf Schnitt mit Primitiven
- prüfe, ob im hinteren Kindknoten ein näherer Schnitt existieren kann

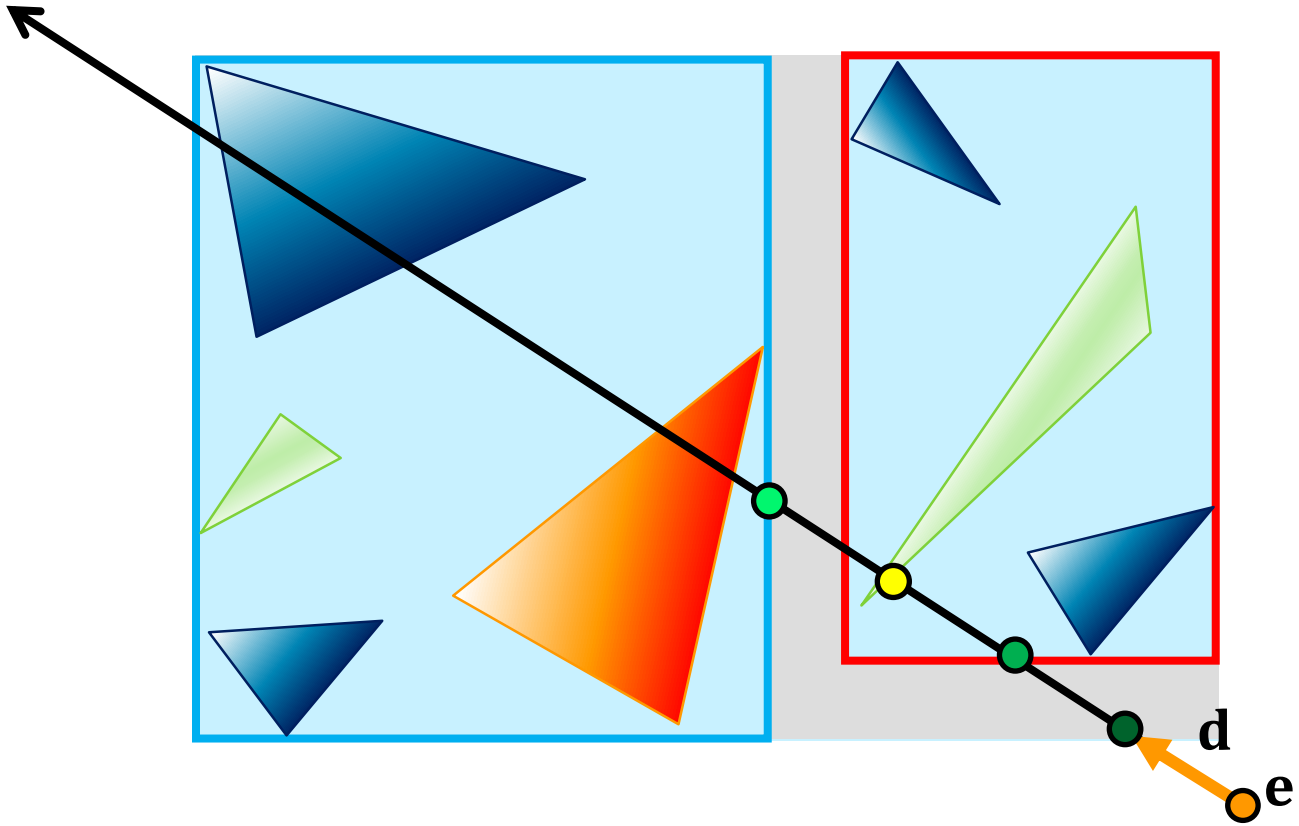


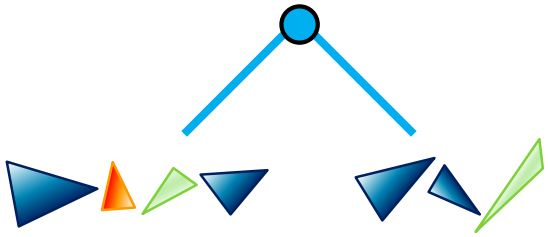


Schnittberechnung mit Hüllkörper-Hierarchien

Traversierung, Schnittberechnung

- teste auf Schnitt mit Primitiven in Blattknoten
  (hier kein weiterer Abstieg, wir sind schon in einem Blattknoten)
- Schnittpunkt ● wurde gefunden
- Schnittpunkt ist näher als der hintere Hüllkörper (Schnittpunkt ●)
  → keine weiteren Tests in diesem Ast des Baums notwendig

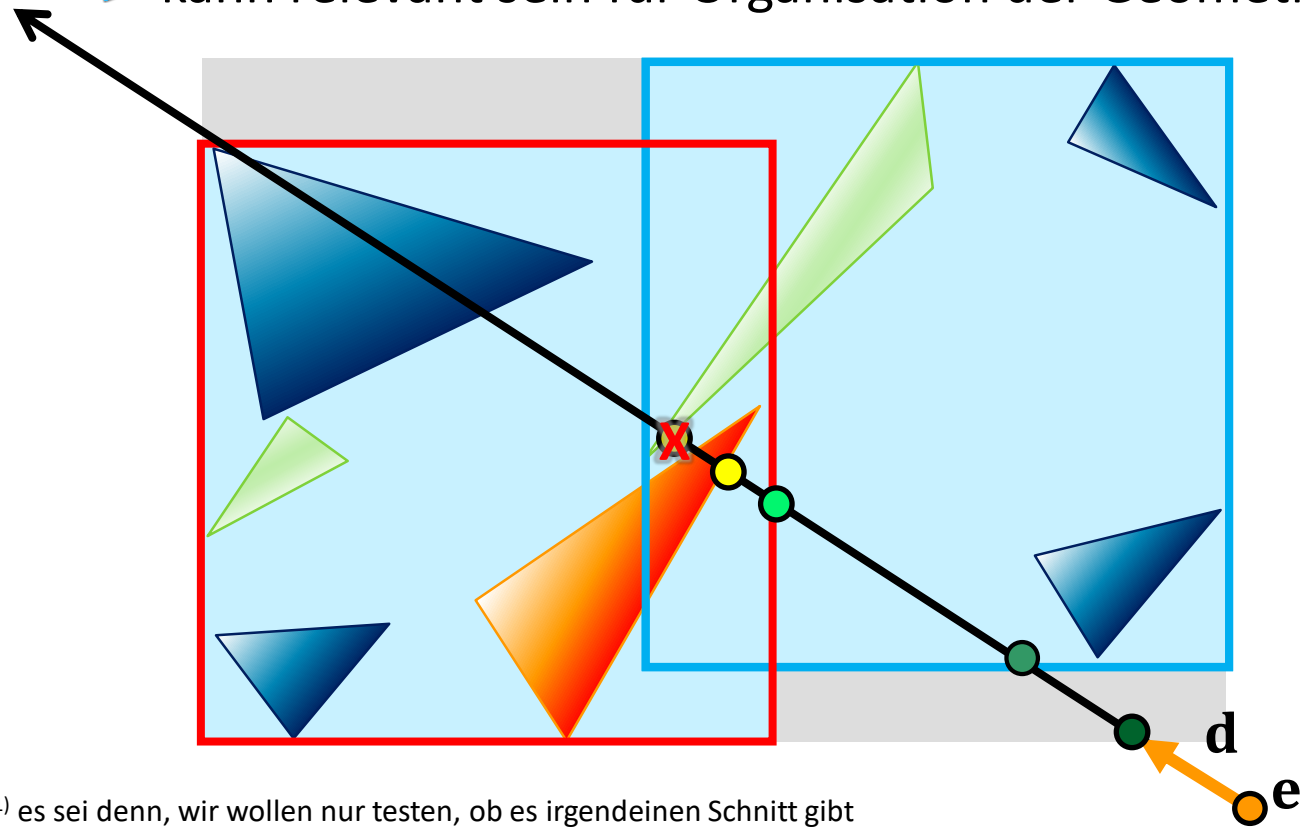


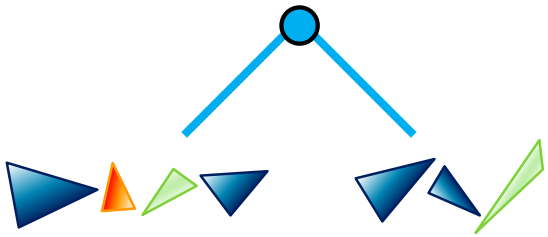


Schnittberechnung mit Hüllkörper-Hierarchien

Traversierung, Schnittberechnung

- Vorsicht: liefere einen gefundenen Schnittpunkt nicht sofort zurück [1], es könnte einen näheren in einem anderen Knoten geben
- **Grund: Hüllkörper können sich überlappen** 
  - ebenfalls eine der Ursachen für Aufwand schlechter als $O(\log n)$
  - kann relevant sein für Organisation der Geometrie bei GPU-Raytracing





[1] es sei denn, wir wollen nur testen, ob es irgendeinen Schnitt gibt

(vorläufiges) Fazit zu Bounding Volume Hierarchy

Vorteile

- Konstruktion und Traversierung ist prinzipiell einfach
- resultiert in einem Binärbaum mit fixer, geringer Verzweigung
  (in der Praxis populär: 1-zu-4 Verzweigung, also jeweils 4 Kindknoten)
- 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 aufwändig und es ist nicht unbedingt klar ist, was eigentlich gut ist: wir wissen nicht vorab, welche Strahlen beim Raytracing verschossen werden
- ungeschickte Unterteilung kann zu schlechter Performanz führen

Performance von BVHs

Wie schnell kann ein Raytracer denn sein? Performance-Zahlen…

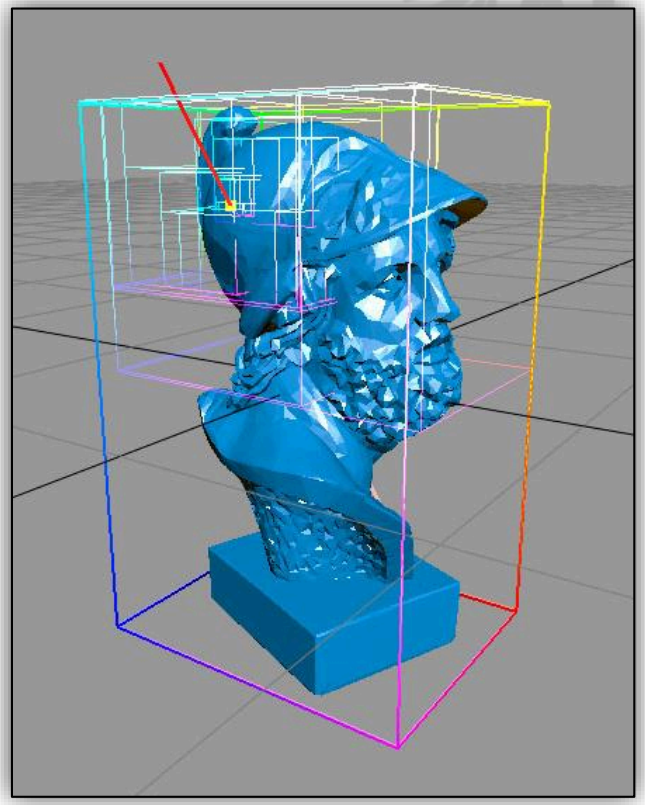
- Datenstrukturen optimiert für besonders schnellen Strahlschuss
  - moderne GPUs: Milliarden kohärente (= ähnliche) Strahlen pro Sekunde, Faktor 5 bis 10 weniger inkohärente Strahlen
  - Aufbau: einige Hundert Millisekunden für ca. 10 Mio. Dreiecke
- Datenstrukturen optimiert für besonders schnellen Aufbau
  - Aufbau: ~600 Mio. Dreiecke pro Sekunde, deutlich mehr bei „refitting“
  - Raytracing-Performanz ca. 50%-80% von optimalen Datenstrukturen
- CPU-GPU schwer zu vergleichen, je nach Anwendung etwa gleich
- Disclaimer: rule of thumb! Abhängigkeit von der Beschaffenheit der 3D-Szene und des Bildsyntheseverfahrens macht genaue Aussagen unmöglich
- etwas mehr Details in aktuellen Forschungsarbeiten, z.B.

  H-PLOC Hierarchical Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction, Benthin et al., ACM on Computer Graphics and Interactive Techniques 2024

  HIPRT: A Ray Tracing Framework in HIP, Meister et al., ACM on Computer Graphics and Interactive Techniques 2024

Inhalt: Räumliche Datenstrukturen

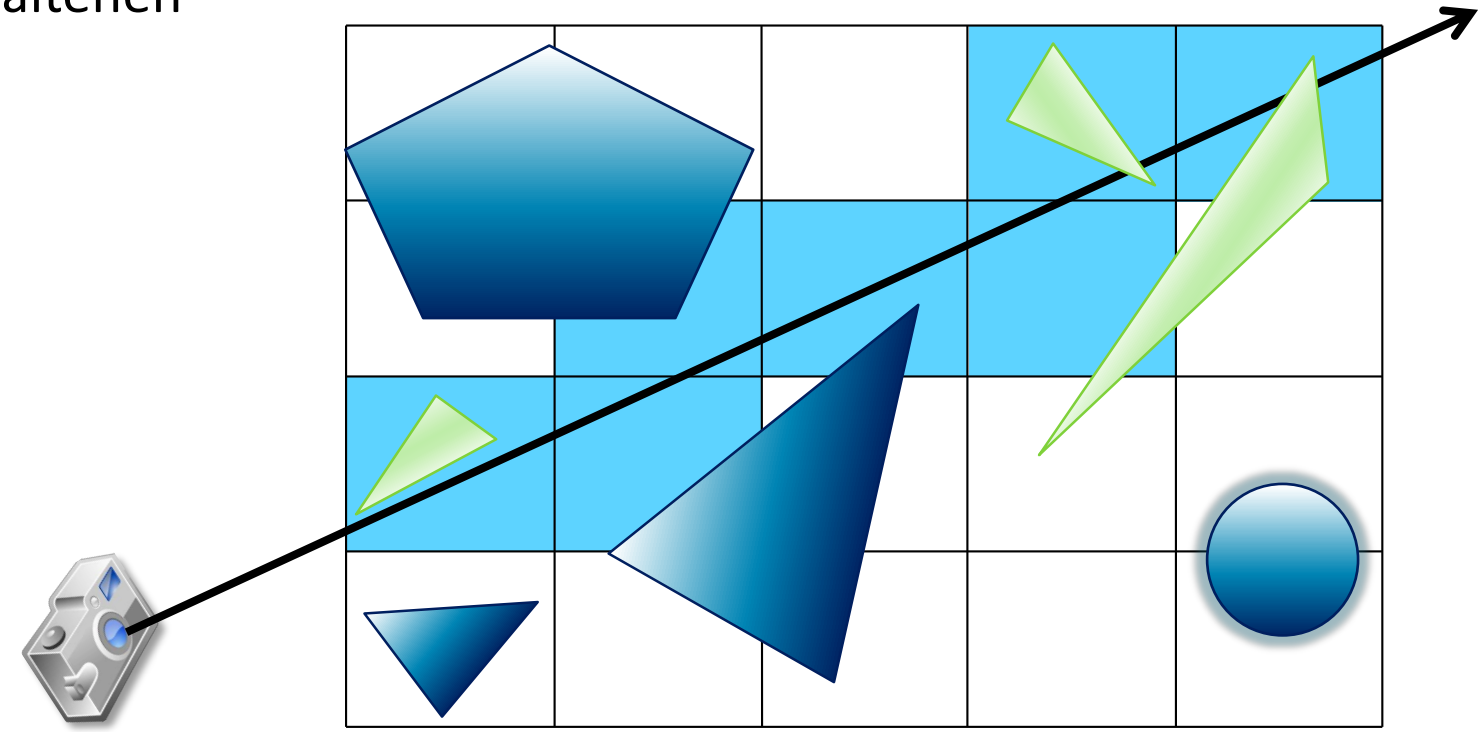
- Analyse der Kosten bei Raytracing
- Ansätze zur Beschleunigung von Raytracing
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - **Bounding Volume Hierarchies**
  - **reguläre und adaptive Gitter**
    (nur Überblick in der Vorlesung,
    mehr in der Übung)
  - BSP-Bäume und **kD-Bäume**



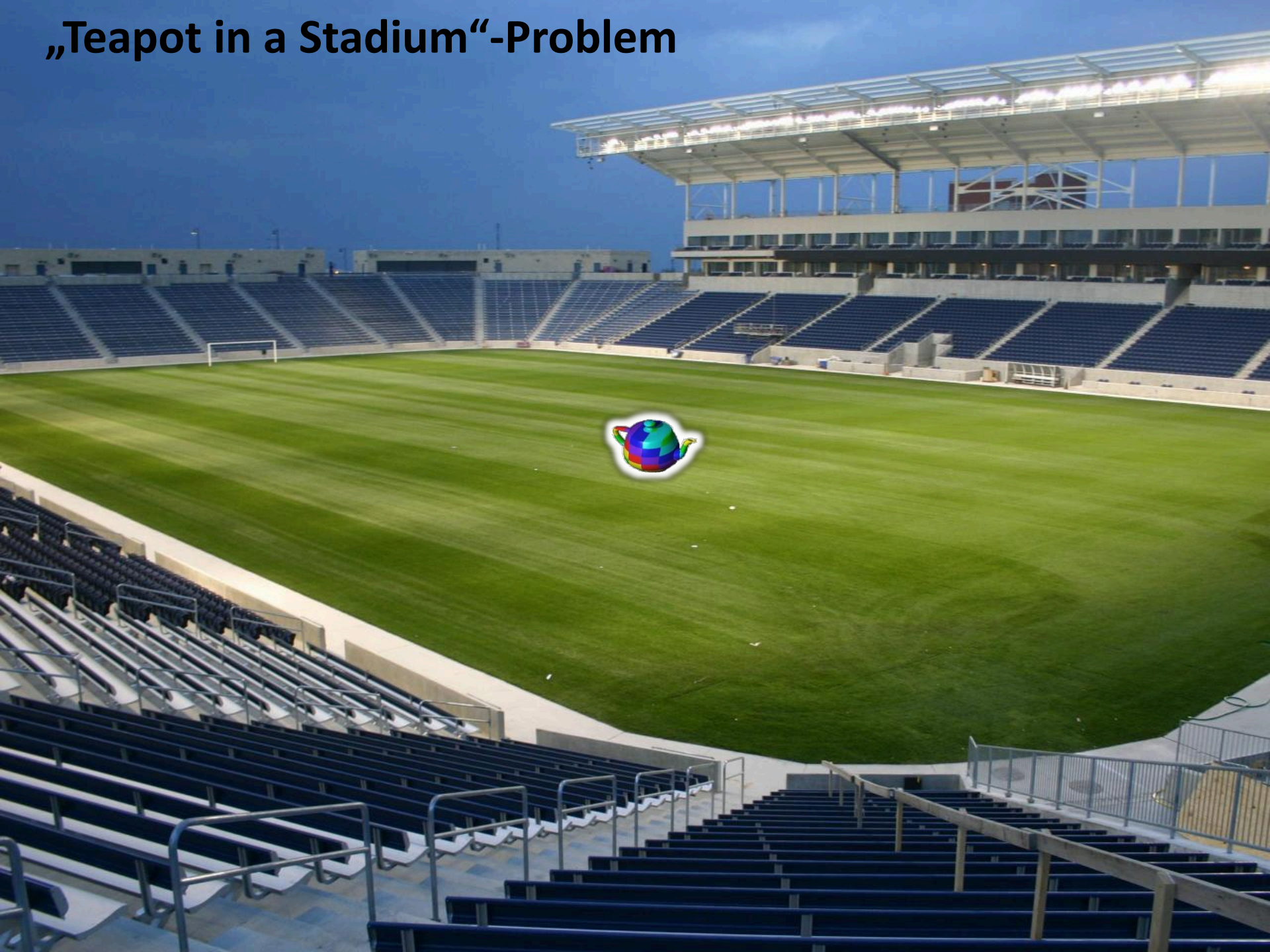
Raumunterteilung durch reguläre Gitter

Grundidee, Prinzip

- Unterteilung des Raums 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



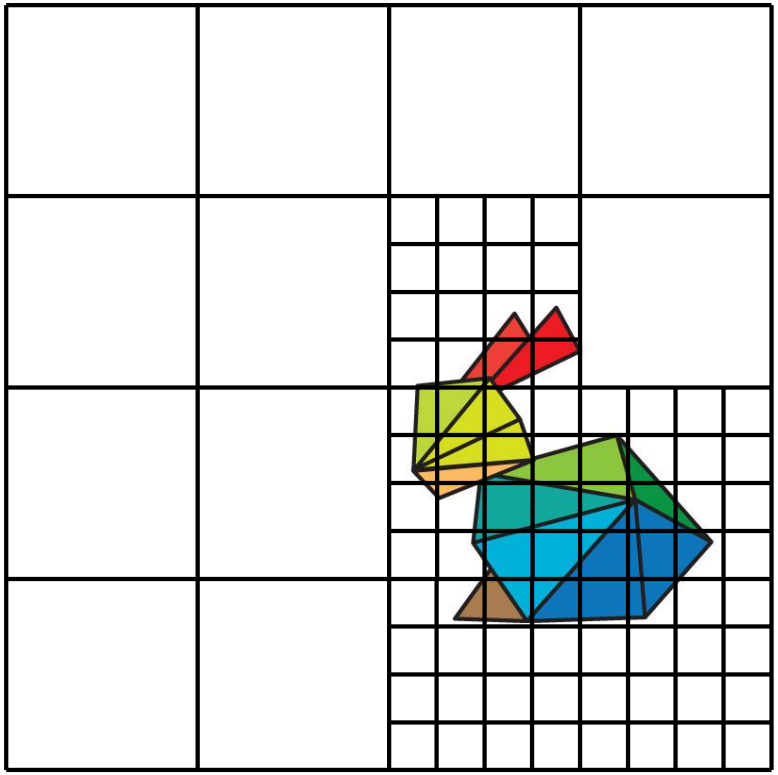
„Teapot in a Stadium“-Problem



Raumunterteilung durch adaptive Gitter

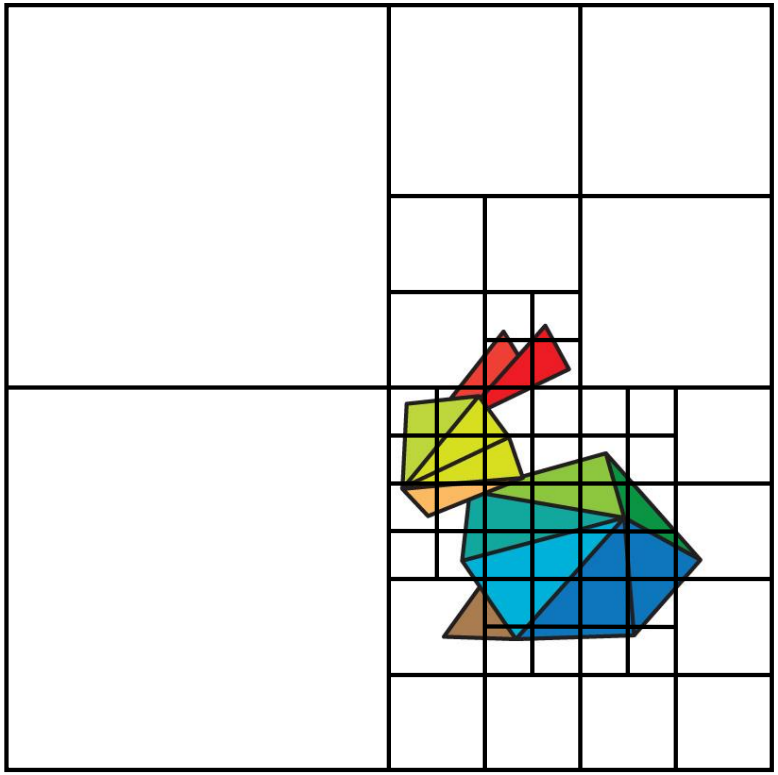
Grundidee: kombiniere Regelmäßigkeit mit Adaptivität

- rekursive regelmäßige Unterteilung von Zellen (übliche Abbruchkrit.)
- … und alle möglichen Variationen, z.B.
  GPU Ray Tracing using Irregular Grids, Pérard-Gayot et al., 2017 (onlinelibrary.wiley.com/doi/10.1111/cgf.13142)



Verschachtelte Gitter (Nested Grids)

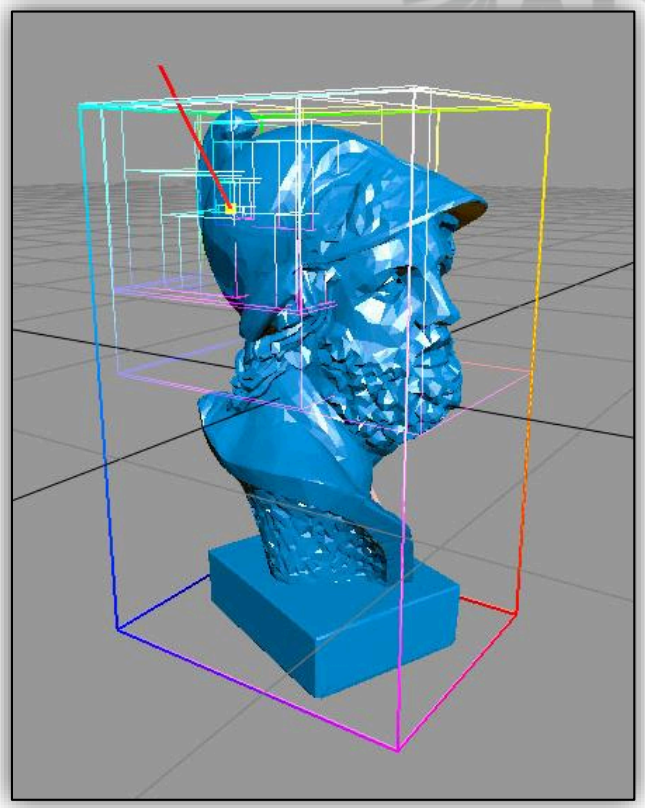
Bounding Box zur Illustration größer als notwendig!

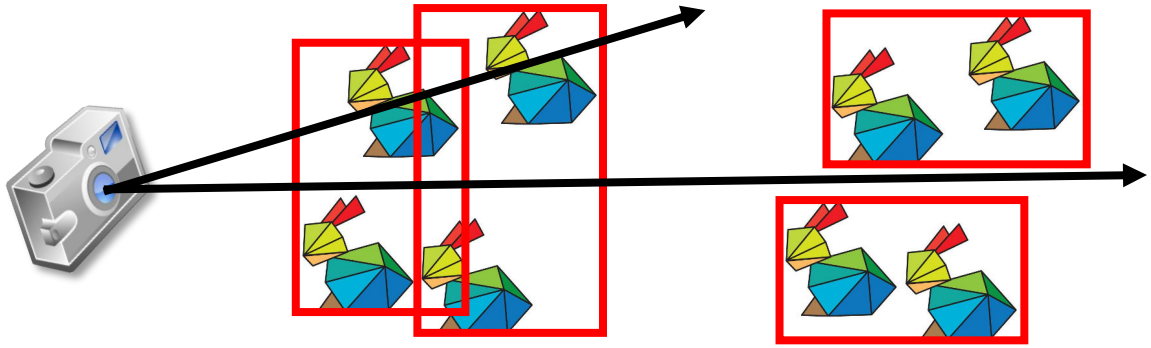


Oktalbaum (Octree), bzw. hier in 2D ein Quadtree (jeder Knoten hat 4 Kinder)

Inhalt: Räumliche Datenstrukturen

- Analyse der Kosten bei Raytracing
- Ansätze zur Beschleunigung von Raytracing
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - Bounding Volume-Hierarchies
  - reguläre und adaptive Gitter
  - BSP-Bäume und **kD-Bäume**
- geschickter Aufbau von BVHs und kD-Bäumen

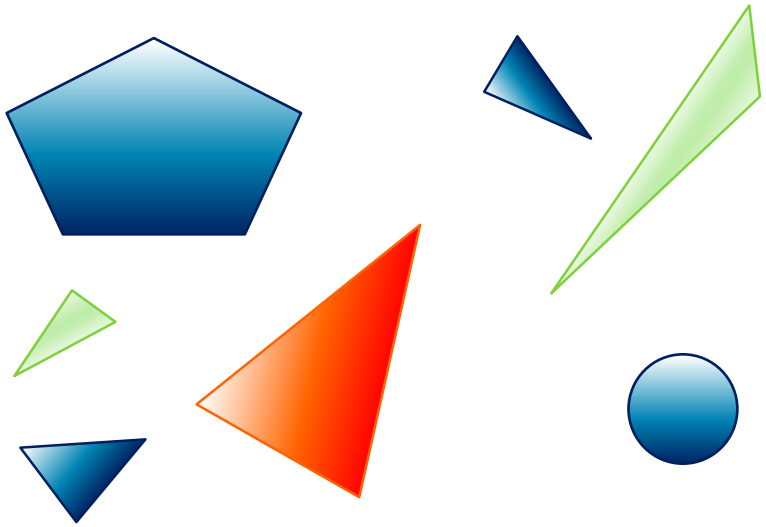




BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

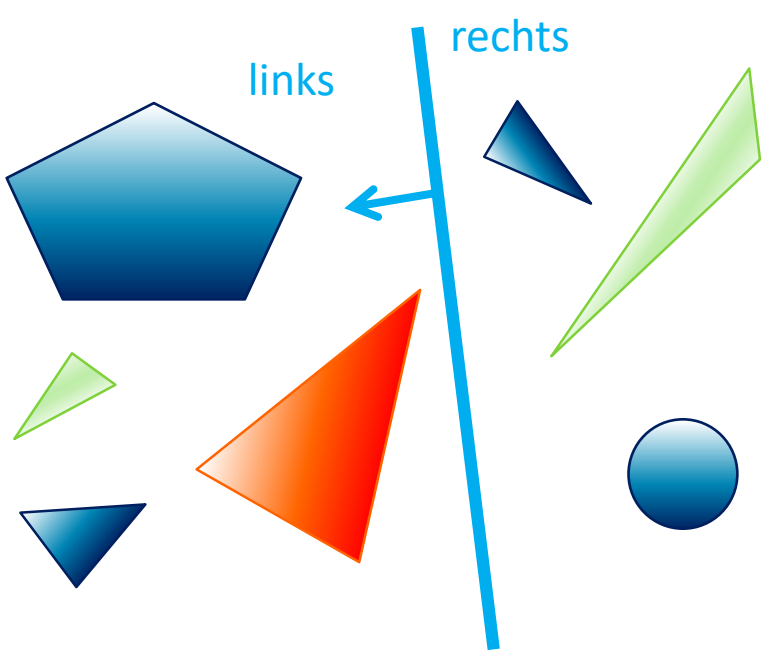
- Erweiterung von Binärbäumen auf k Dimensionen
- verwende Ebenen um **den Raum rekursiv zu unterteilen**
  → ergibt Binärbaumstruktur

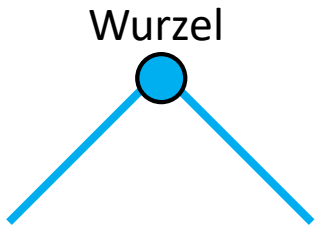


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

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

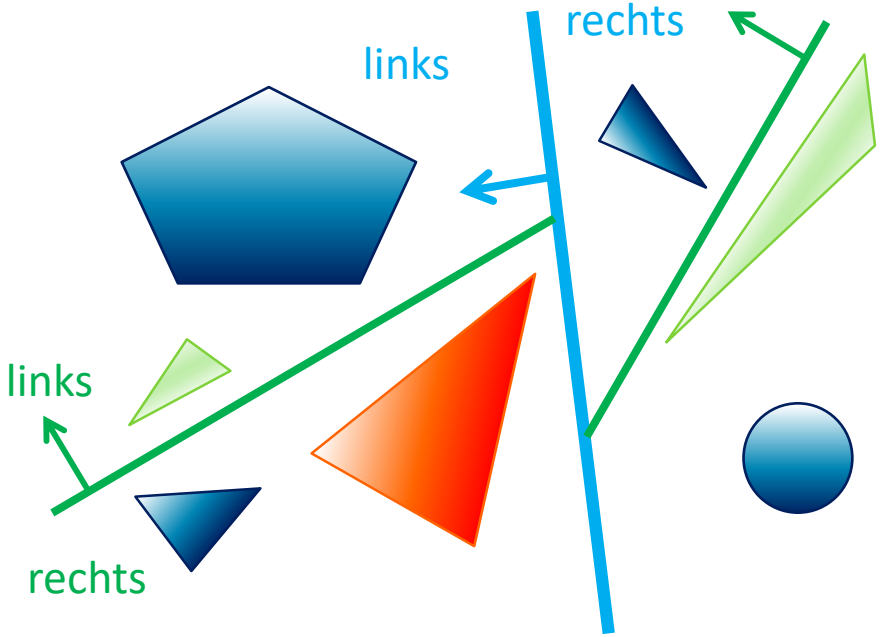


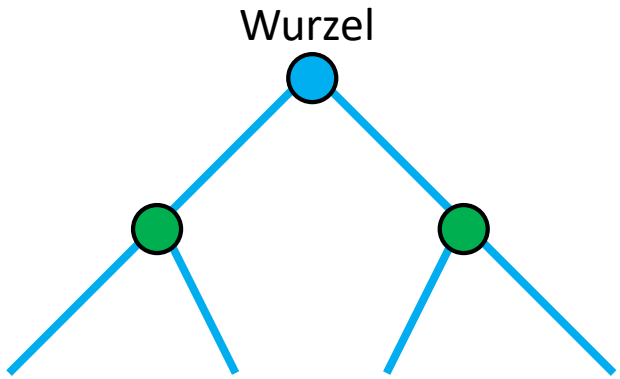


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

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

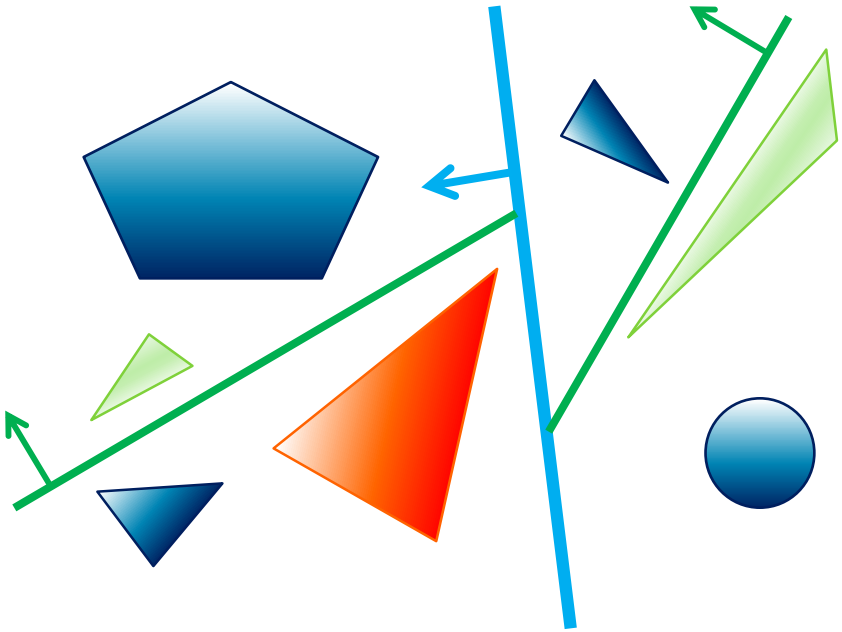


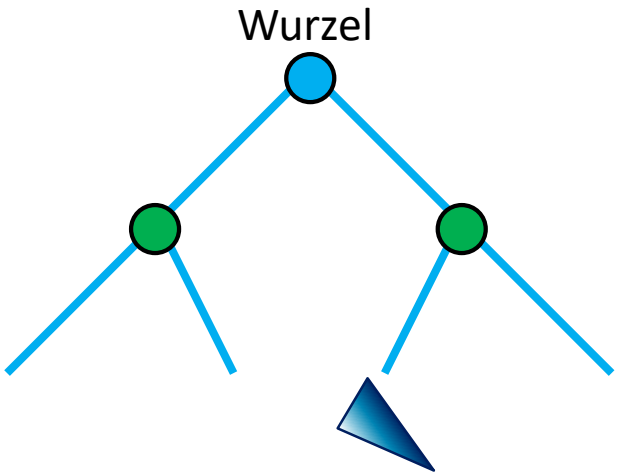


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

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

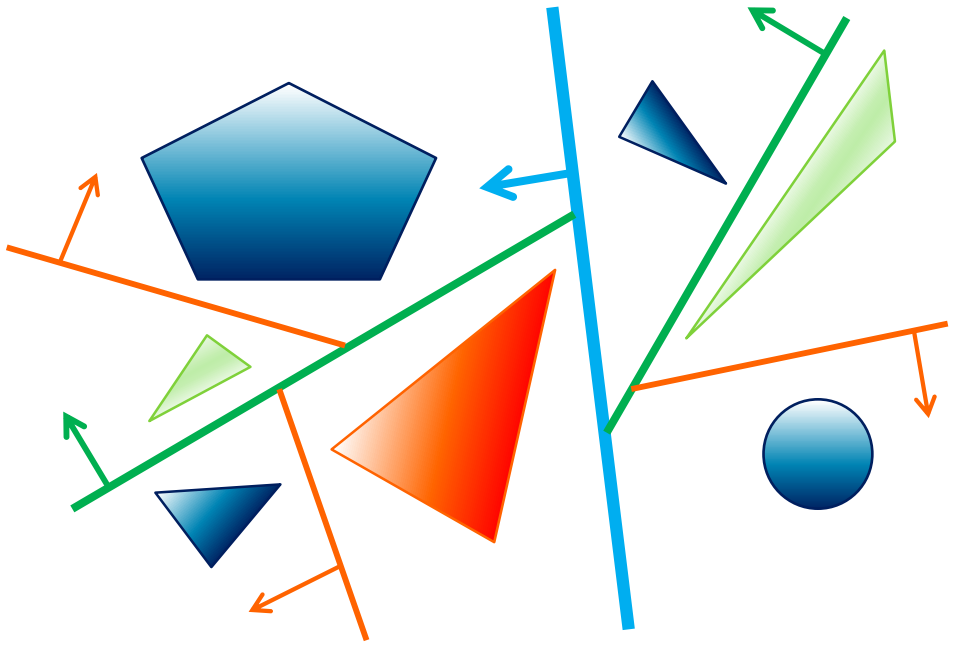


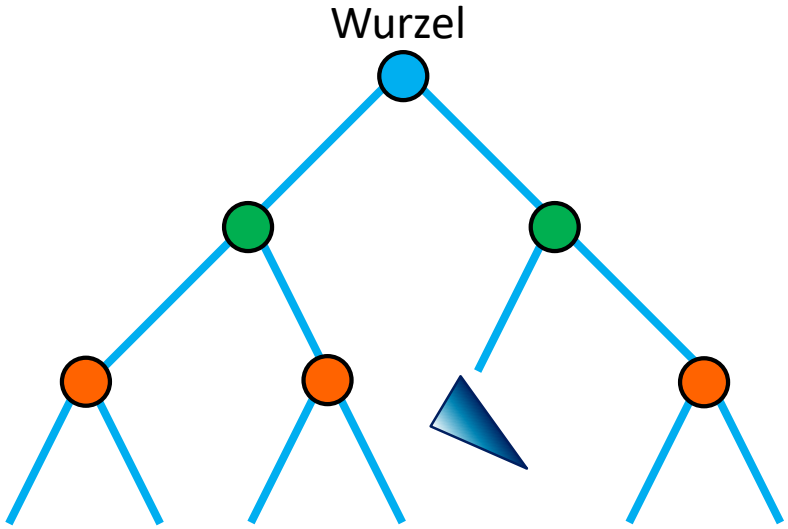


BSP-Baum und kD-Baum

Grundidee (Binary Space Partitioning Tree)

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

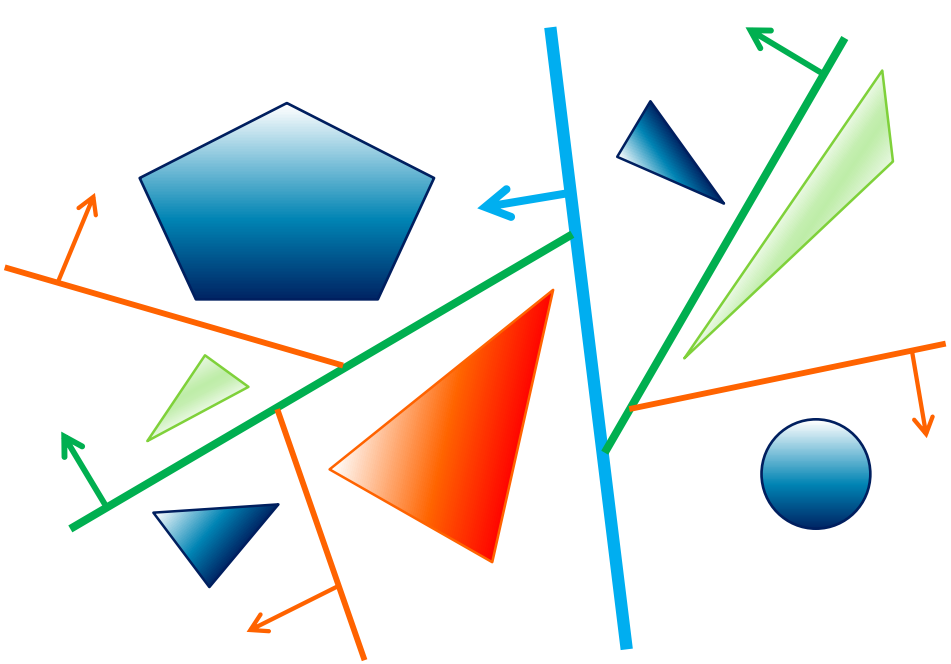


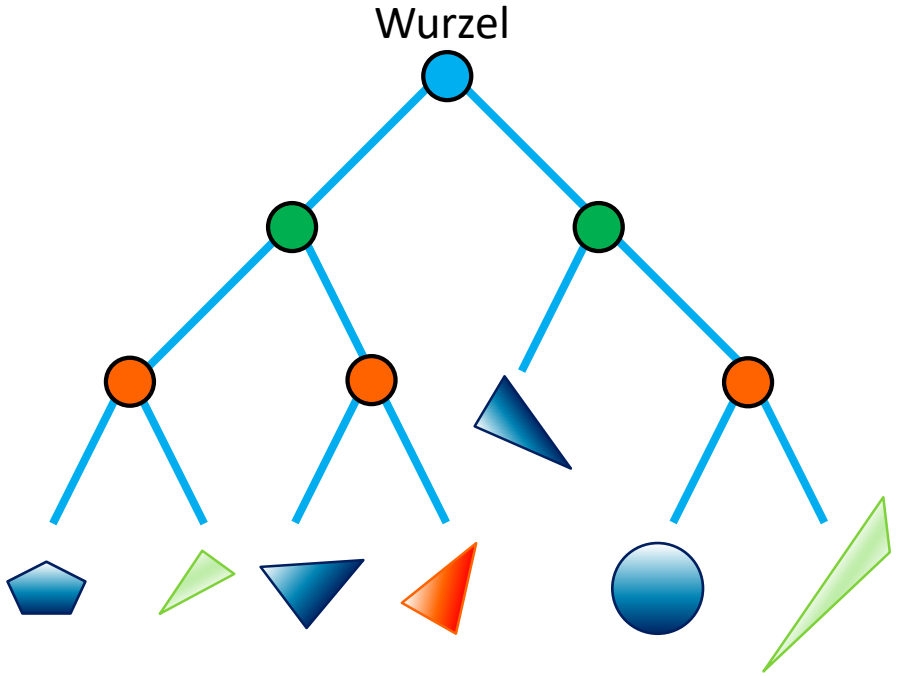


BSP-Baum und kD-Baum

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

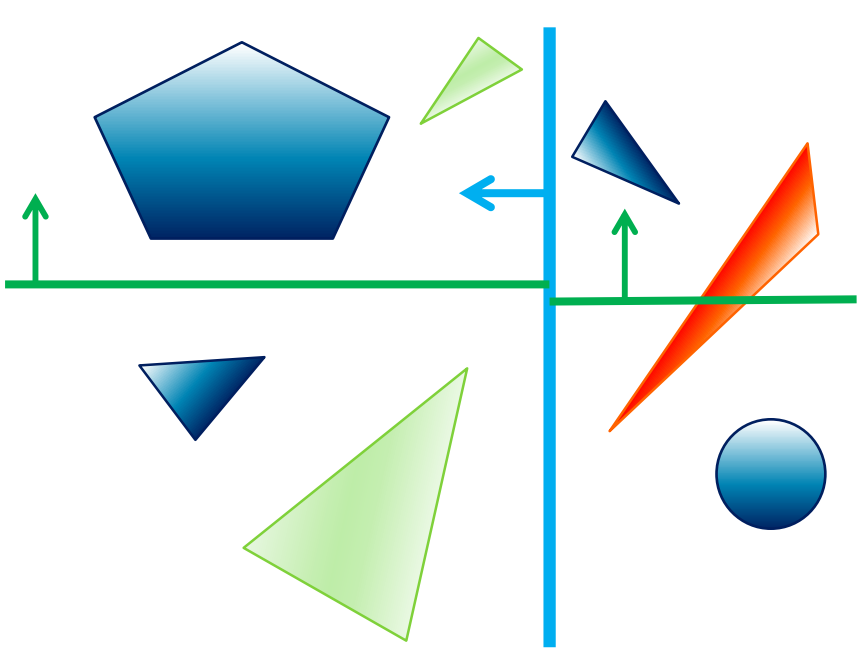


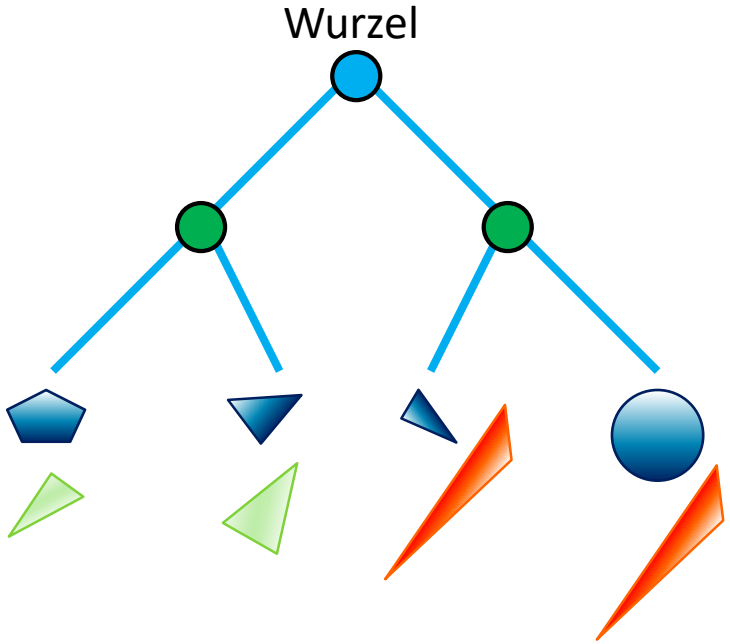


kD-Bäume

Eigenschaften

- Split-Ebenen senkrecht zur x-, y- oder z-Achse
- Primitive die eine Split-Ebene schneiden (auch bei BSP-Bäumen) werden meist in beide Kindknoten eingefügt: ein Primitiv kann also – im Gegensatz zu BVHs – in mehreren Knoten vorkommen
  → Dreiecke werden mehrfach indiziert





BSP-Baum und kD-Baum

Konstruktion (sehr ähnlich wie bei BVH)

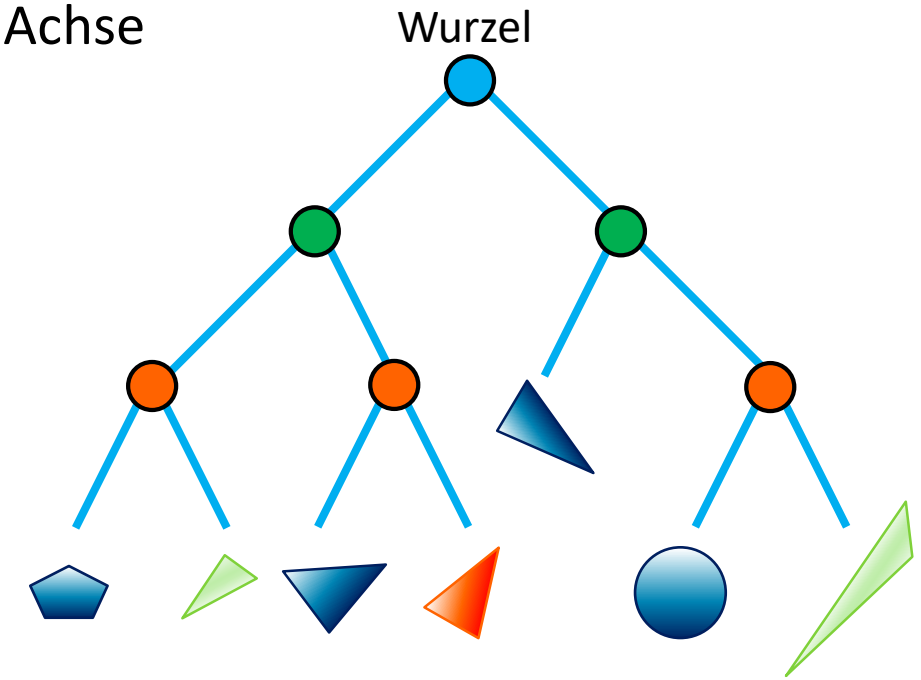
- initialisiere Wurzelknoten: enthält alle Objekte/Primitive der Szene
- unterteile den (Wurzel-)Knoten rekursiv…
  - …bis ein Knoten nur noch eine vorgegebene maximale Anzahl Primitive enthält
  - …oder eine maximale Rekursionstiefe erreicht ist
- **unterteile den Raum** und ordne die Primitive einem „linken“ und einem „rechten“ Kindknoten zu
  (vgl. BVH: teilt die Primitive auf, die Hüllkörper ergeben sich daraus)

BSP-Baum und kD-Baum

Eigenschaften

- echte Raumunterteilung: Knoten überlappen sich nicht (vgl. BVH)
- gute Anpassung an Geometrie möglich (v.a. bei BSP-Bäumen)
- die Blattknoten speichern die Primitive bzw. Verweise/Indizes darauf
- 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

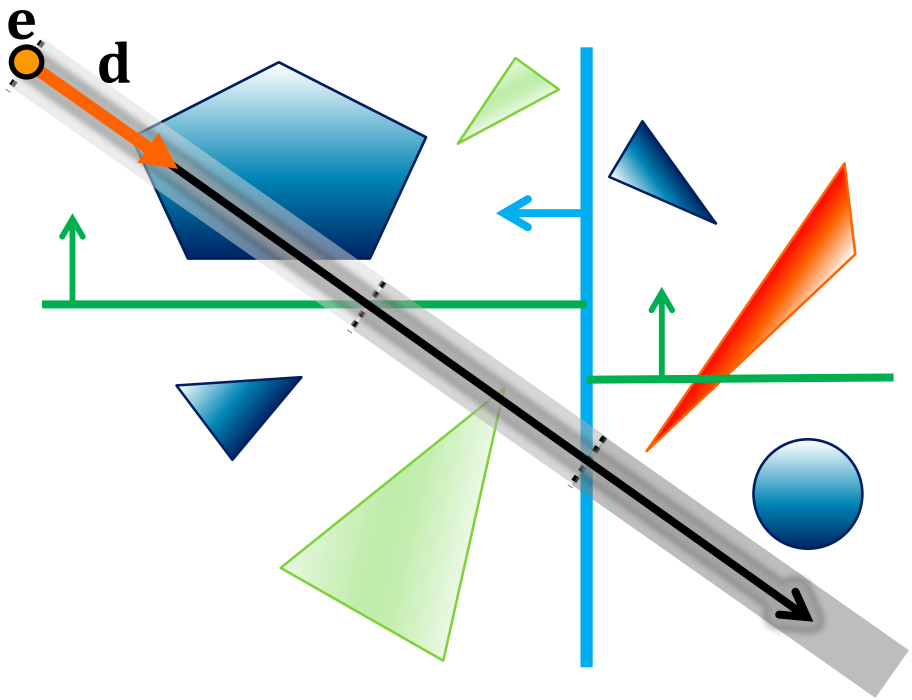
- kD-Bäume sind einfacher zu konstruieren (keine Ebenen mit freier Orientierung) und werden daher häufiger eingesetzt

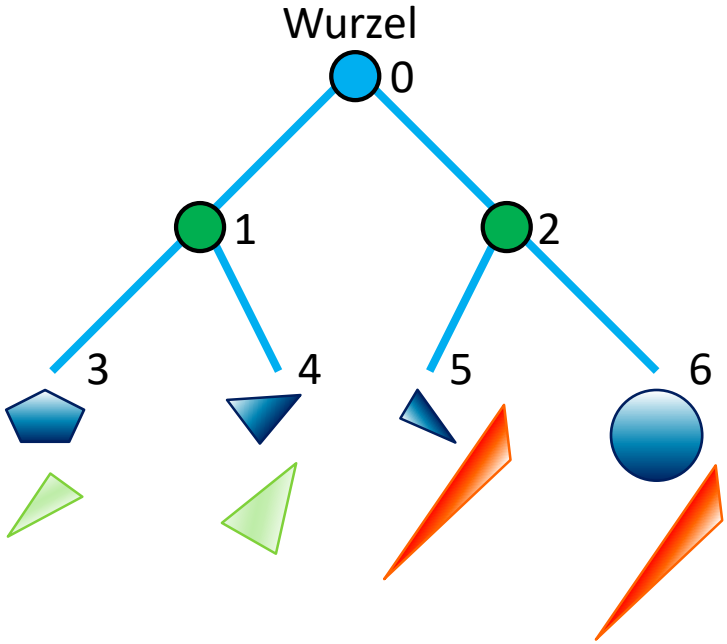


BSP-Baum und kD-Baum

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)



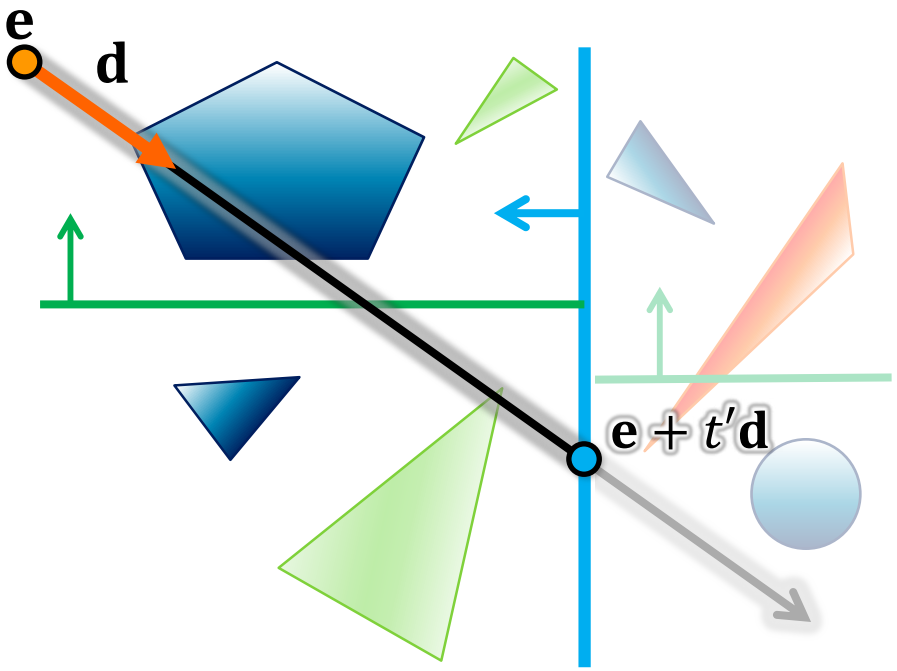


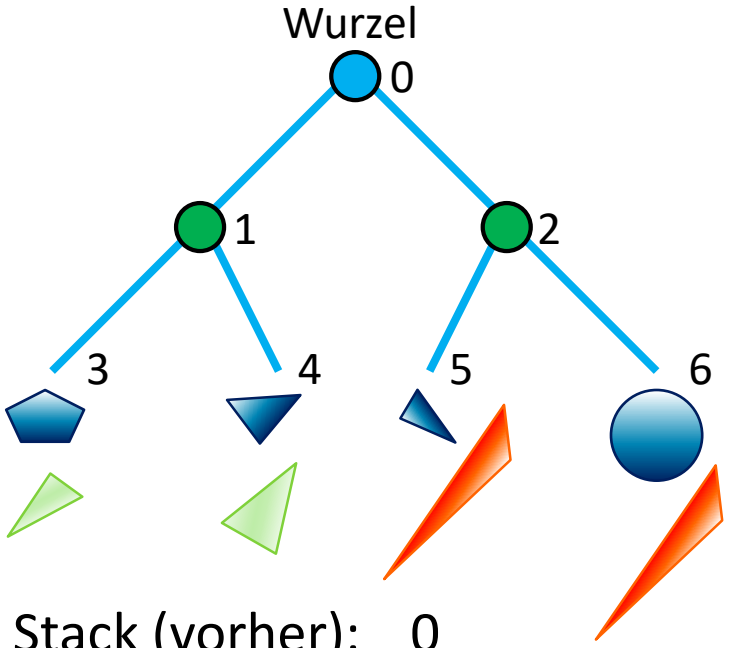
Stack: 0

BSP-Baum und kD-Baum

Traversierung

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





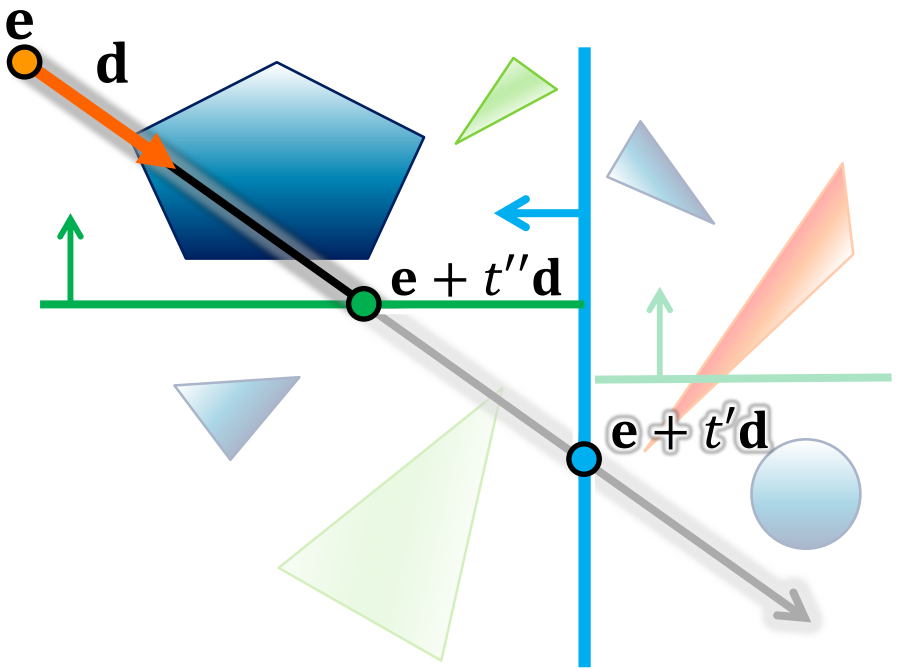
Stack (vorher): 0

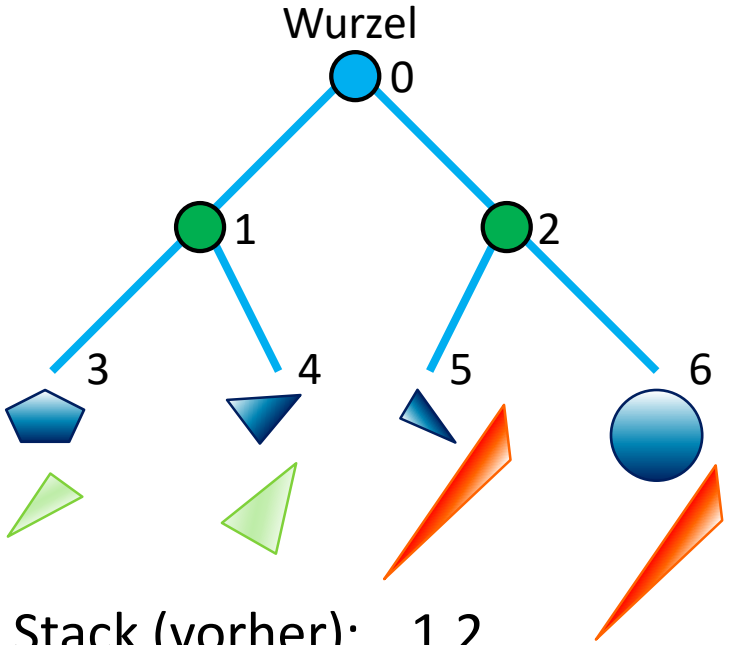
Stack (nachher): 1,2

BSP-Baum und kD-Baum

Traversierung

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





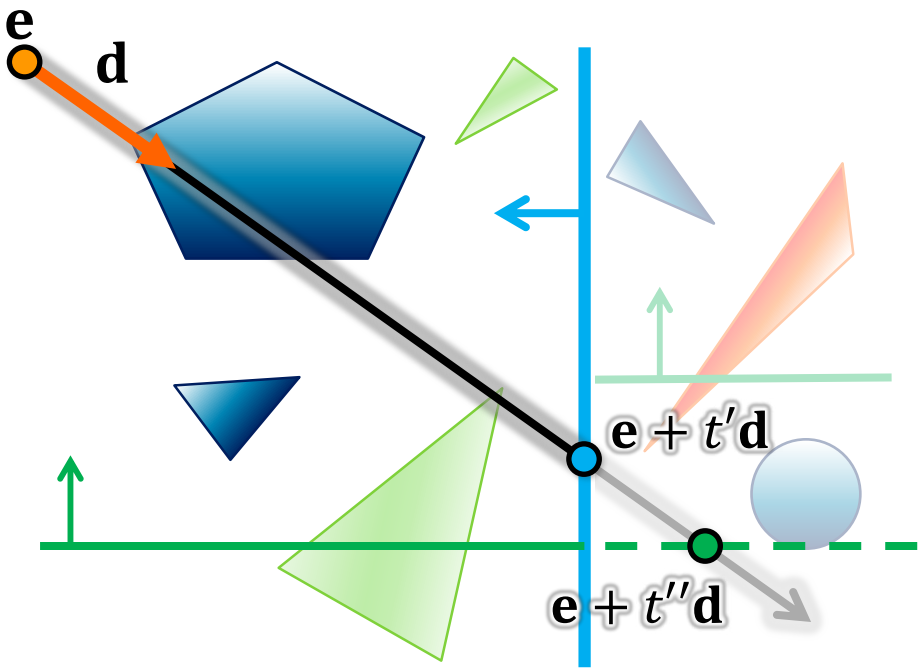
Stack (vorher): 1,2

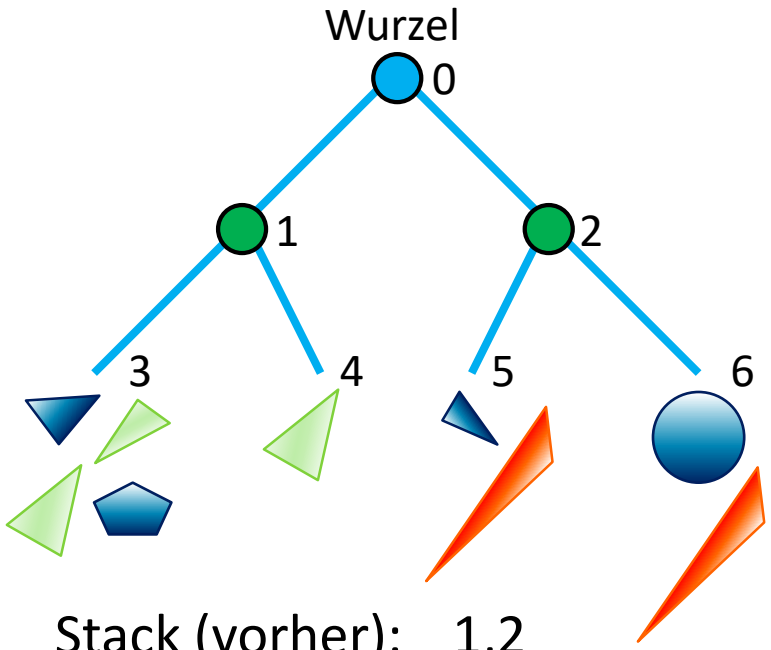
Stack (nachher): 3,4,2

BSP-Baum und kD-Baum

Traversierung: „der andere Fall“

- **Achtung: nur zur Illustration, hat nichts mit dem Beispiel eben 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





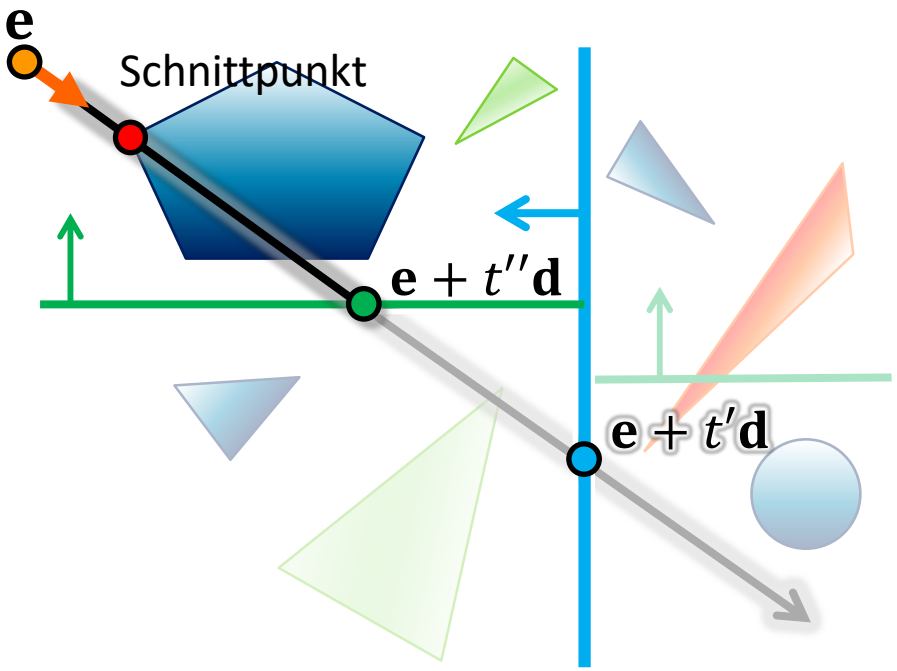
Stack (vorher): 1,2

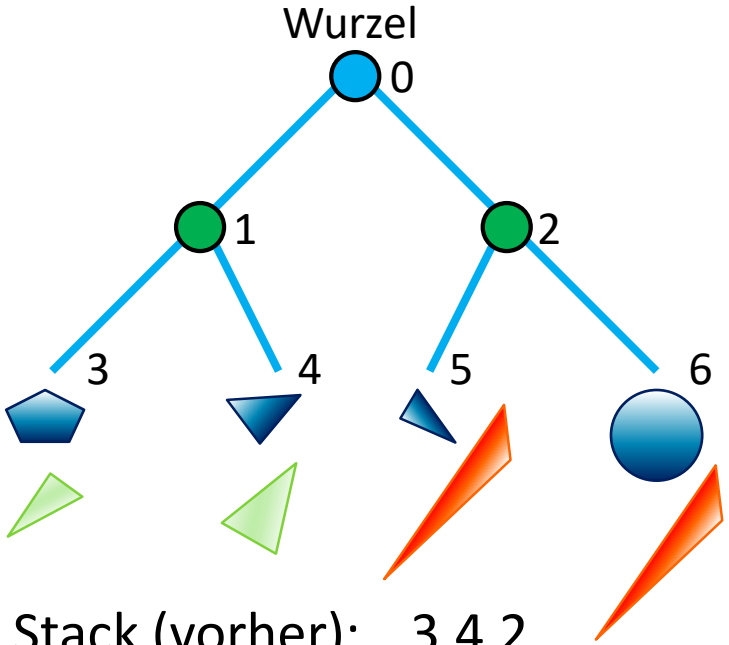
Stack (nachher): 3,2

BSP-Baum und kD-Baum

Traversierung

- wenn Blattknoten erreicht, dann teste Schnittpunkte mit Primitiven
- gebe nahsten Schnittpunkt zurück, wenn innerhalb $[t_{min}, t'')$ (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!





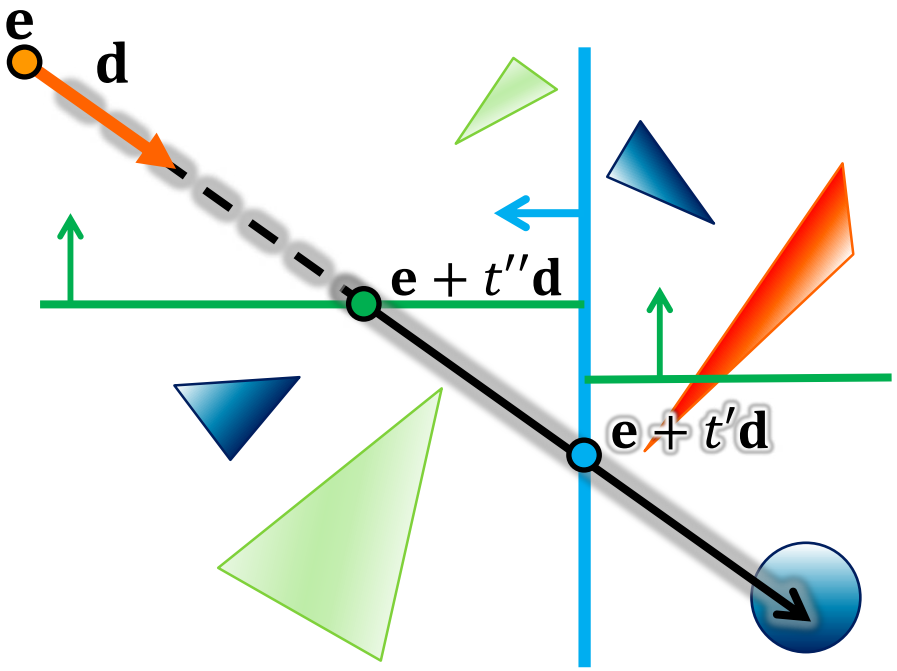
Stack (vorher): 3,4,2

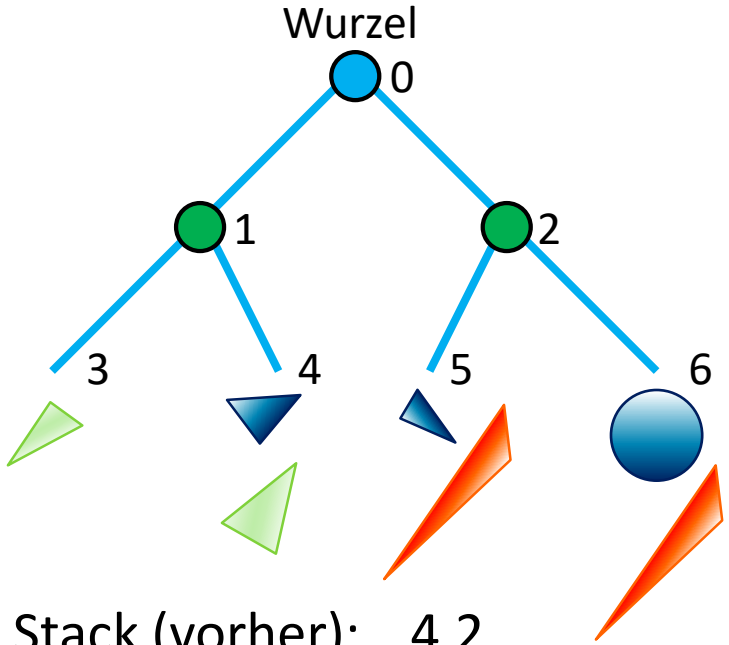
Stack (nachher): 4,2

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 $t'' \leq t \leq t'$





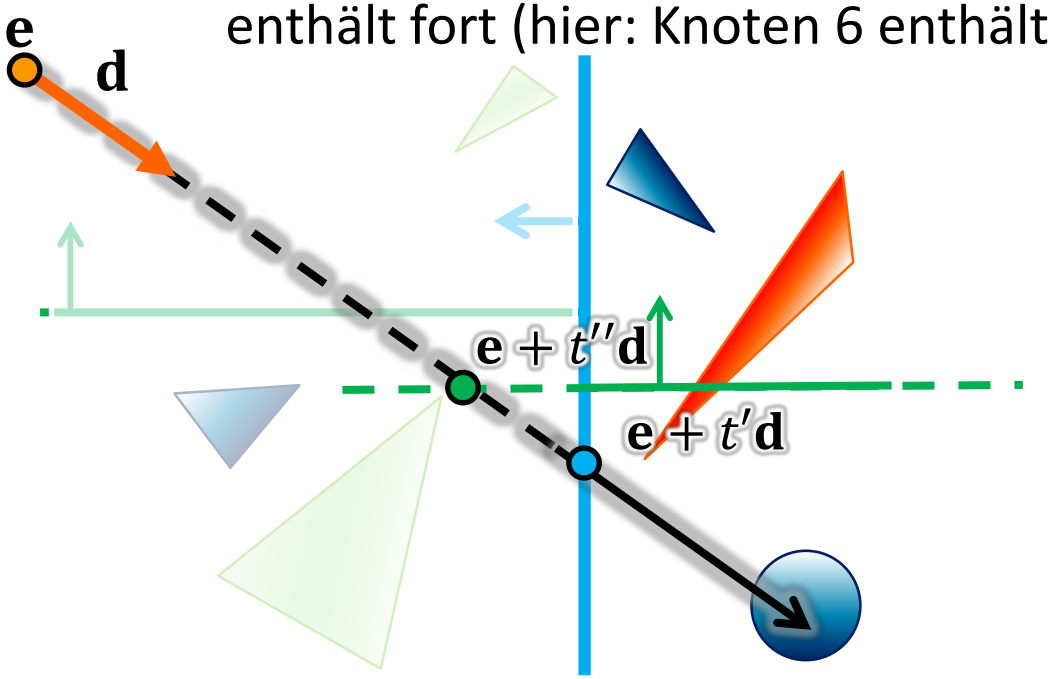
Stack (vorher): 4,2

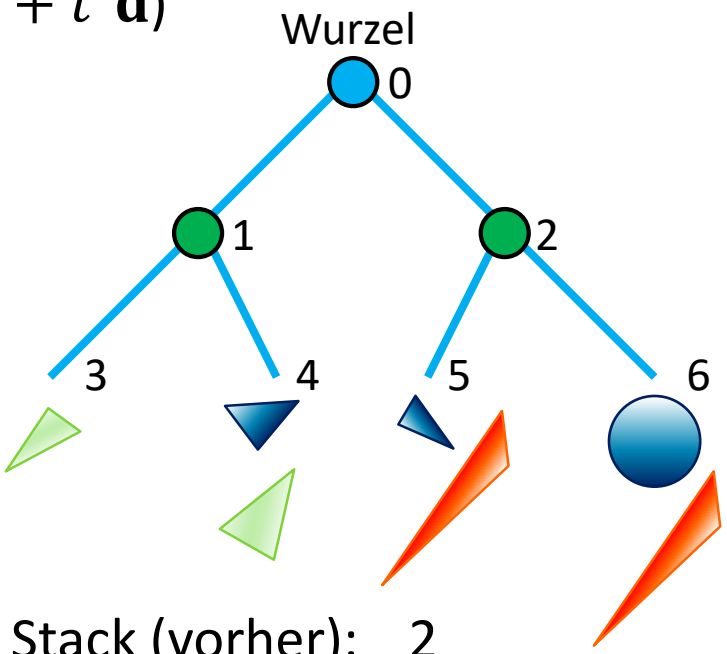
Stack (nachher): 2

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

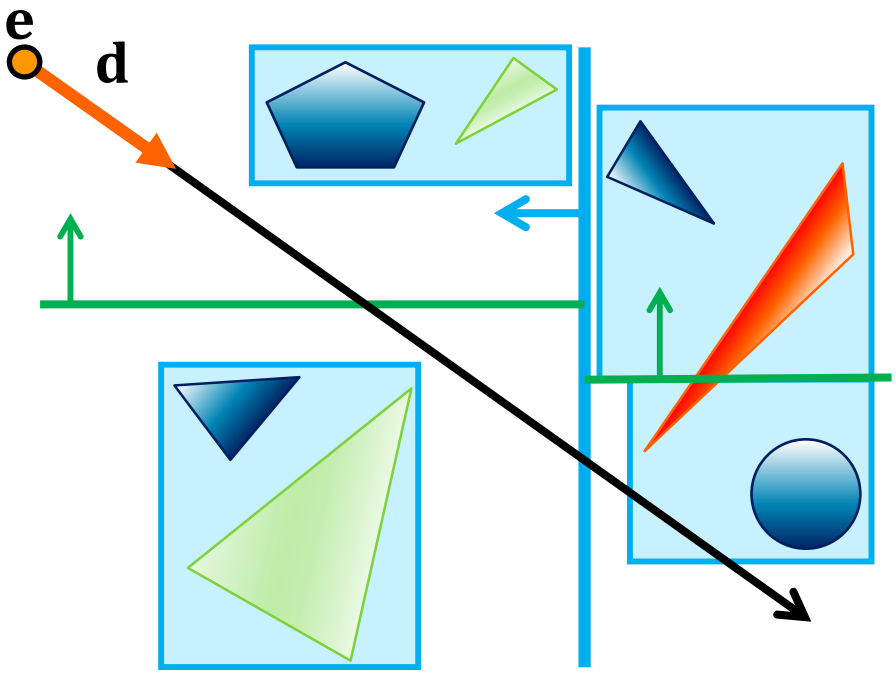
Traversierung

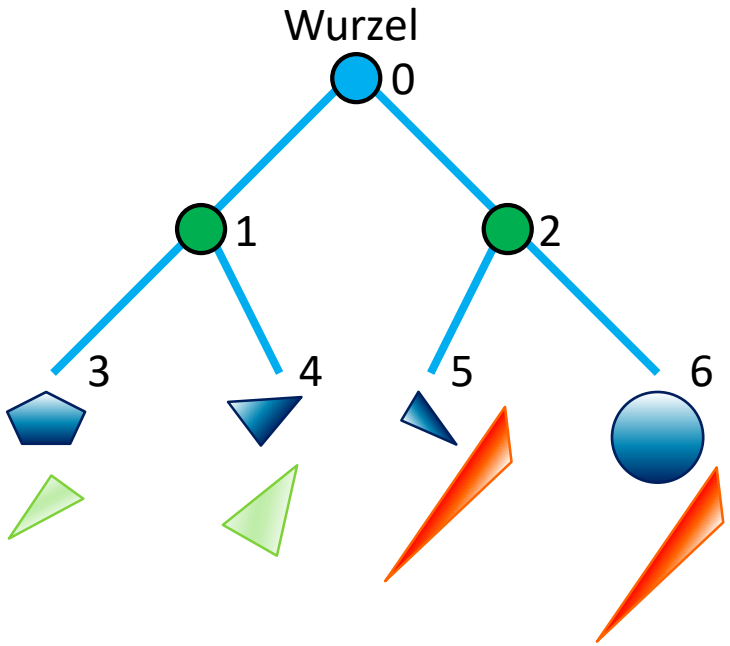
- Strahl $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$, mit $0 \leq t \leq t_{max}$
- schneide Strahl mit der Split-Ebene (beginnend beim Wurzelknoten)
  - Schnitt mit der Split-Ebene bei t'
    - wenn t' auf dem Strahlsegment liegt:
      traversiere beide Kinder rekursiv, zuerst das Kind, das $\mathbf{e}$ enthält
    - liegt t' nicht auf dem Strahlsegment:
      dann fahre nur mit dem Kind, das geschnitten wird fort
    - unterteile das Strahlsegment weiter mit t'', t''', …
  - wenn Blattknoten erreicht:
    - teste Schnittpunkte mit Primitiven
    - gebe Schnittpunkt zurück, wenn innerhalb des aktuellen Strahlsegments (= innerhalb des Teilraums)
  - Reihenfolge der Traversierung der Knoten: von vorne nach hinten!
- Komplexität: $O(\log n)$ Schnitttests für n Objekte/Primitive in der Szene
- Anm. Traversierung auch ohne Stack möglich, siehe z.B.
  Efficient stackless hierarchy traversal on GPUs with backtracking in constant time,
  Binder und Keller, High Performance Graphics'16

BSP-Baum und kD-Baum

Kombination mit AABBs

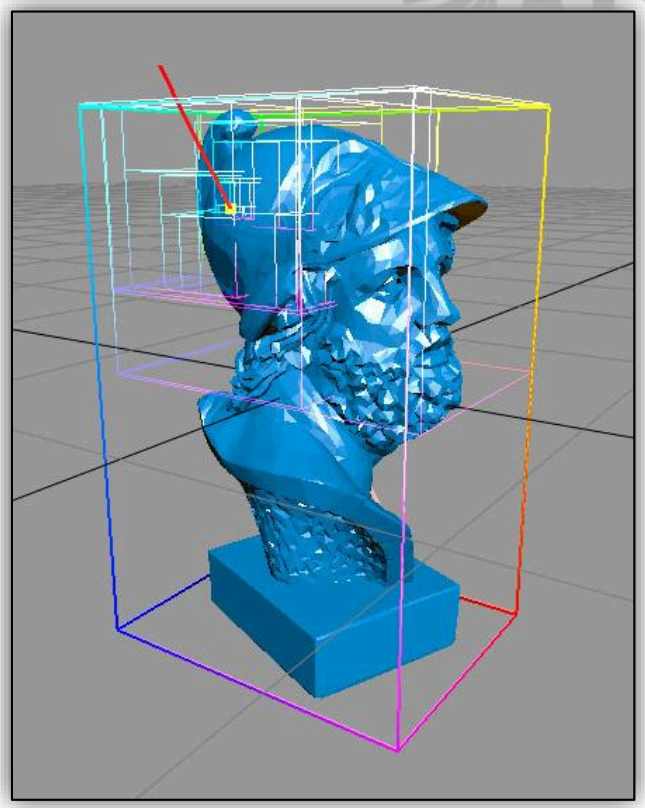
- manchmal speichert man für die Primitive eines Knotens zus. eine AABB
- hilfreich, wenn Primitive nur einen kleinen Teil des Raums einnehmen
- für Schnitttest pro Knoten bedeutet das: teste nur auf Schnitt mit Primitiven, wenn die AABB geschnitten wird
- (Isolation dünnbesetzter Bereiche daher wünschenswert)

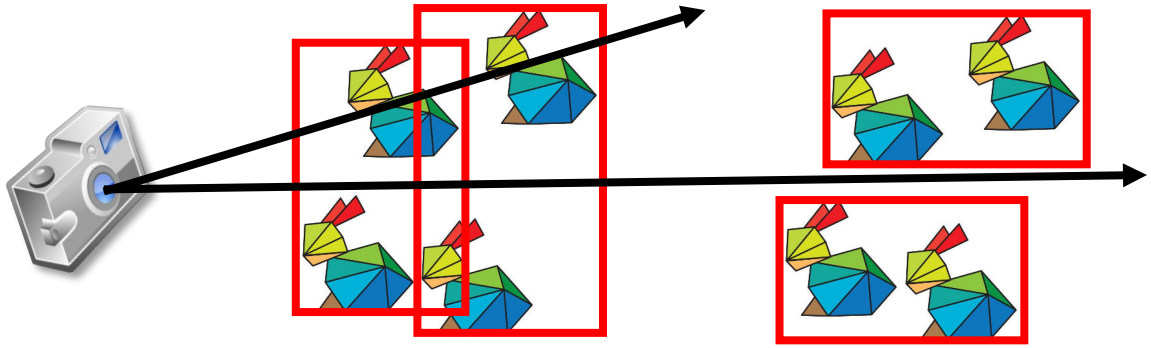




Inhalt: Räumliche Datenstrukturen

- Analyse der Kosten bei Raytracing
- Ansätze zur Beschleunigung von Raytracing
- Bounding Volumes (Hüllkörper)
- Räumliche Datenstrukturen
  - Bounding Volume-Hierarchies
  - reguläre und adaptive Gitter
  - BSP-Bäume und kD-Bäume
- geschickter Aufbau von BVHs und kD-Bäumen

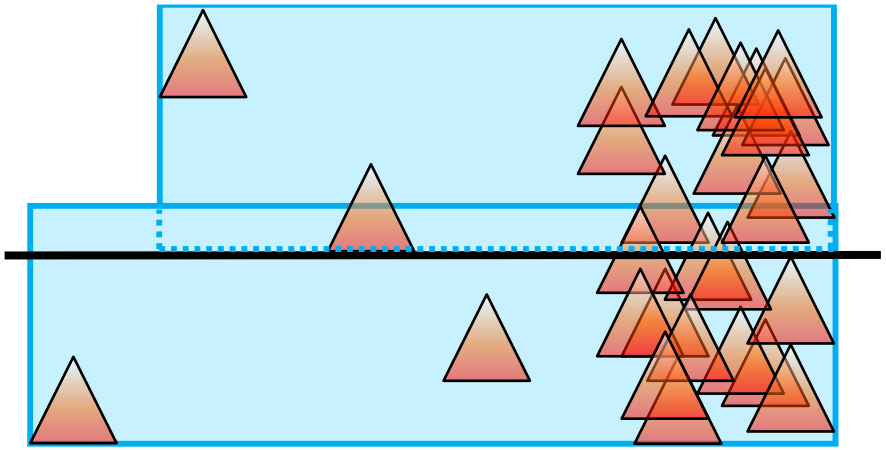




kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält

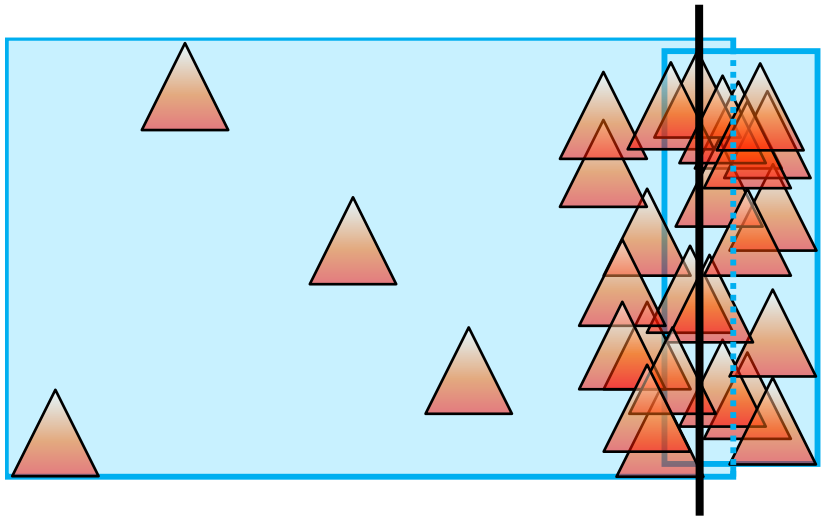


Objektmedian mit
Tiefe $O(\log n)$

kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält
  - … senkrecht zu welcher Achse?

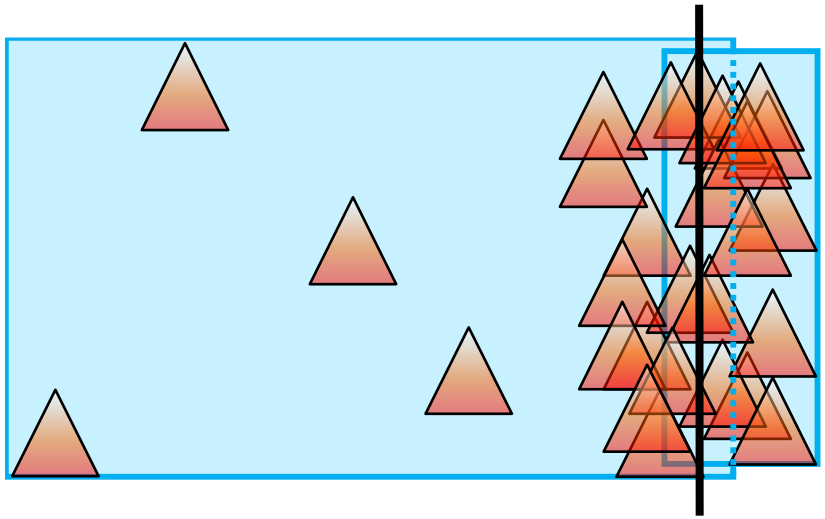


Objektmedian mit
Tiefe $O(\log n)$

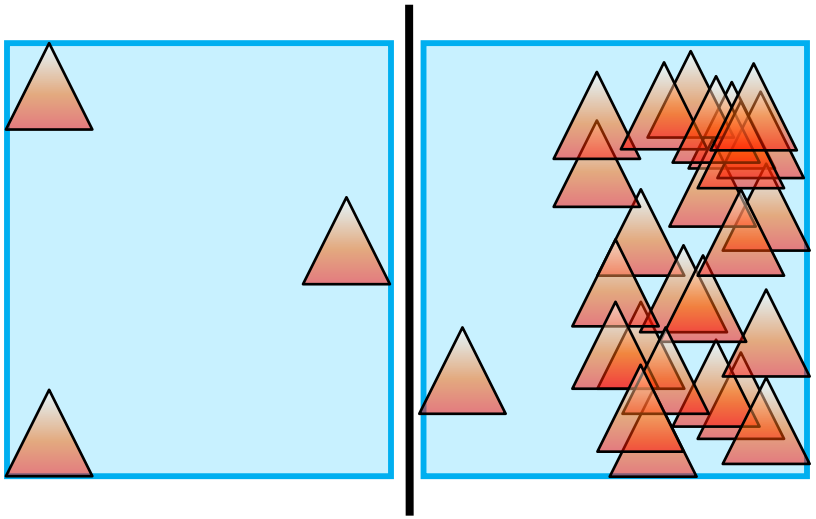
kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält
  - … senkrecht zu welcher Achse?
- … in der Mitte senkrecht zur Achse der größten Ausdehnung
  - … führt zu sehr unbalancierten Bäumen



Objektmedian mit
Tiefe $O(\log n)$

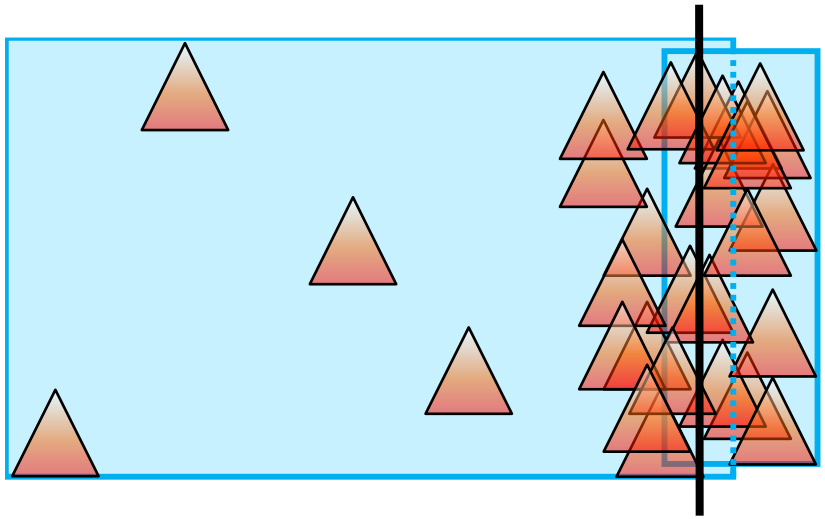


räumliches Mittel

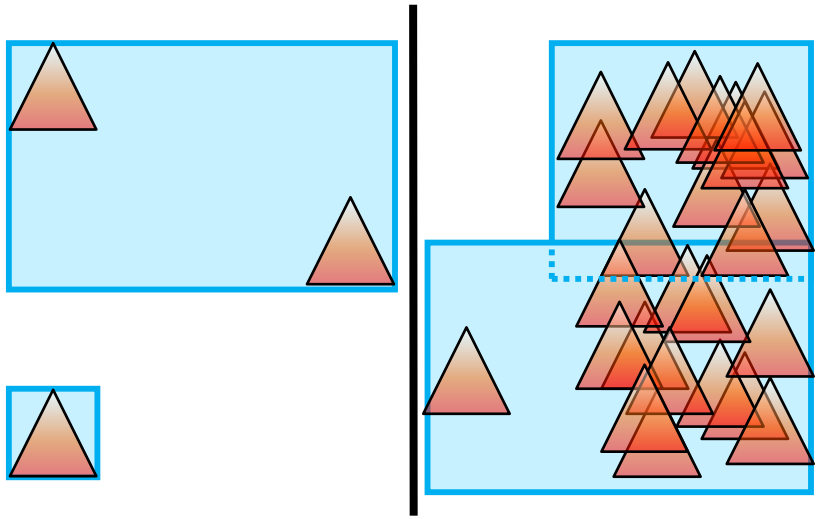
kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält
  - … senkrecht zu welcher Achse?
- … in der Mitte senkrecht zur Achse der größten Ausdehnung
  - … führt zu sehr unbalancierten Bäumen



Objektmedian mit
Tiefe $O(\log n)$

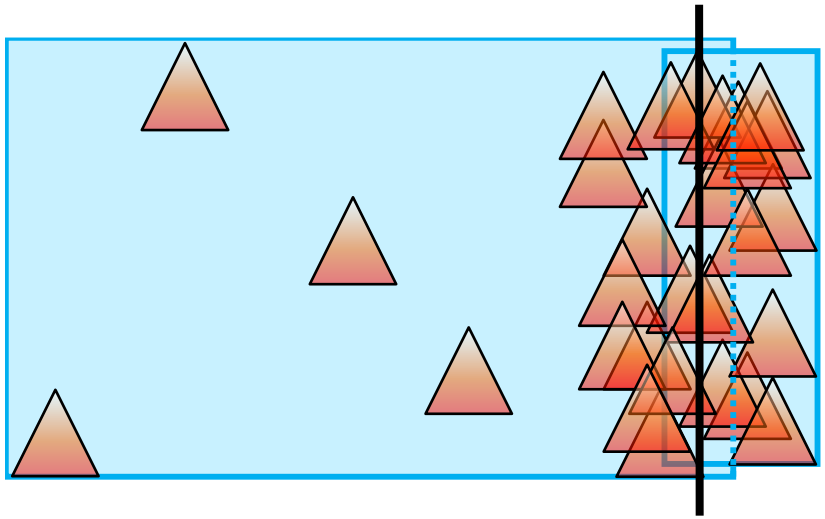


räumliches Mittel

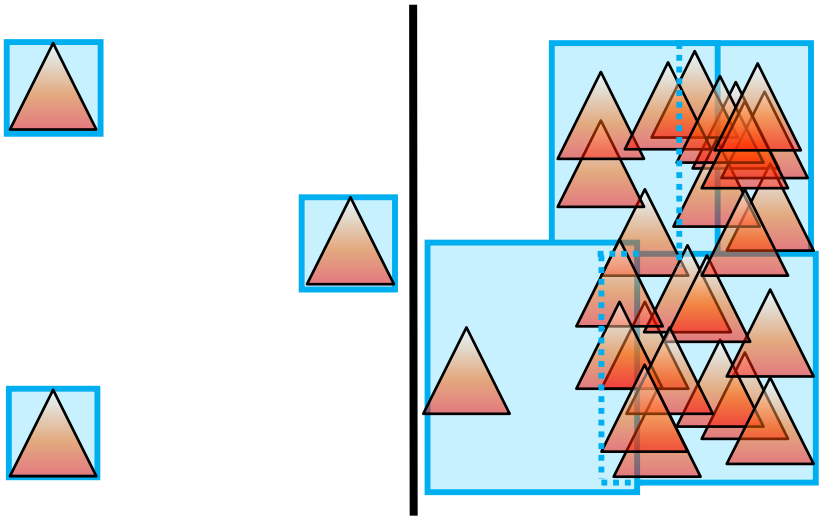
kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält
  - … senkrecht zu welcher Achse?
- … in der Mitte senkrecht zur Achse der größten Ausdehnung
  - … führt zu sehr unbalancierten Bäumen



Objektmedian mit
Tiefe $O(\log n)$

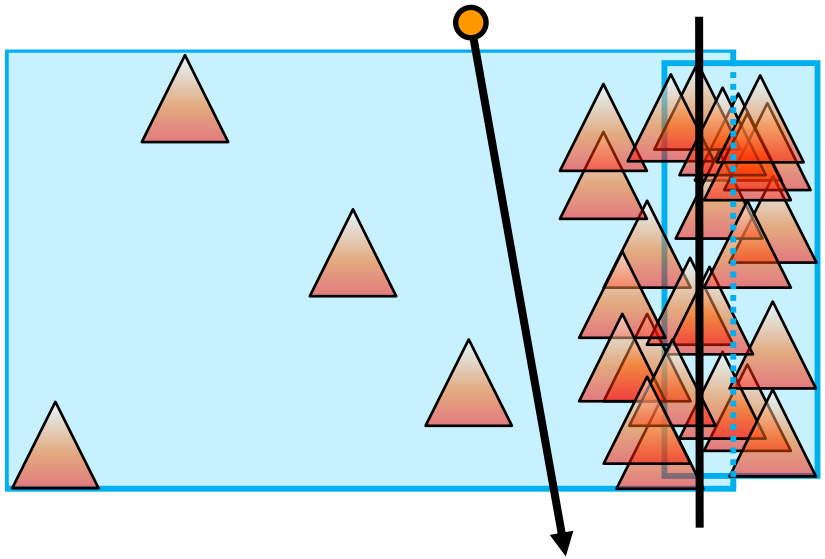


räumliches Mittel

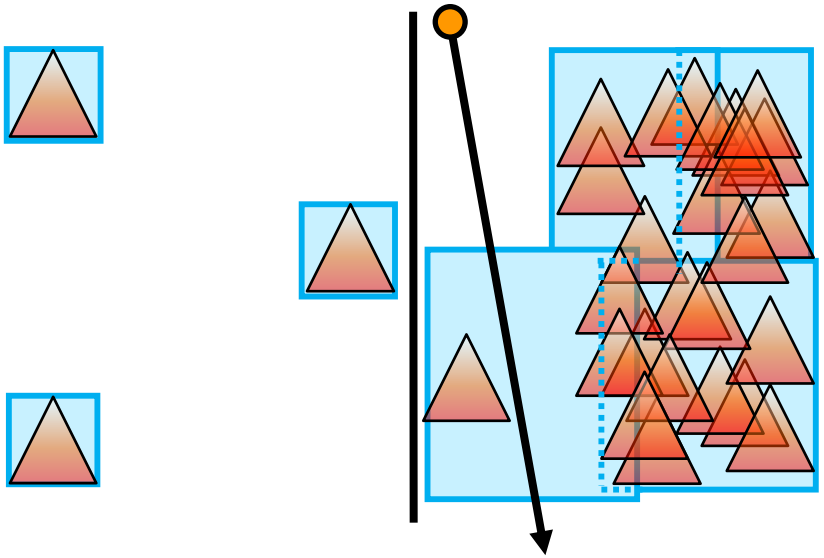
kD-Baum und BVH Konstruktion

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

- … so, dass jede Teilmenge etwa gleiche Anzahl Primitive enthält
  - … senkrecht zu welcher Achse?
- … in der Mitte senkrecht zur Achse der größten Ausdehnung
  - … führt zu sehr unbalancierten Bäumen
- in beiden Fällen: Hüllkörper werden nicht überall schnell kleiner



Objektmedian mit
Tiefe $O(\log n)$

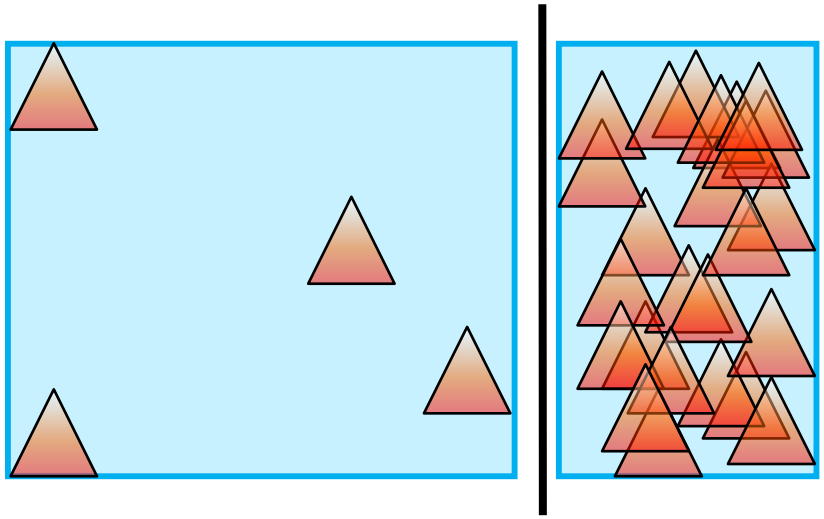


räumliches Mittel

kD-Baum und BVH Konstruktion

Zielformulierung

- Identifizieren von kompakten, dichtbesetzten Regionen
  - … großer Aufwand nur, wenn der Strahl diese auch wirklich trifft
- Abtrennen von dünnbesetzten Regionen
  - … wird in den nächsten Schritten schnell zu kleinen Hüllkörpern führen
- wie kann man das formalisieren?

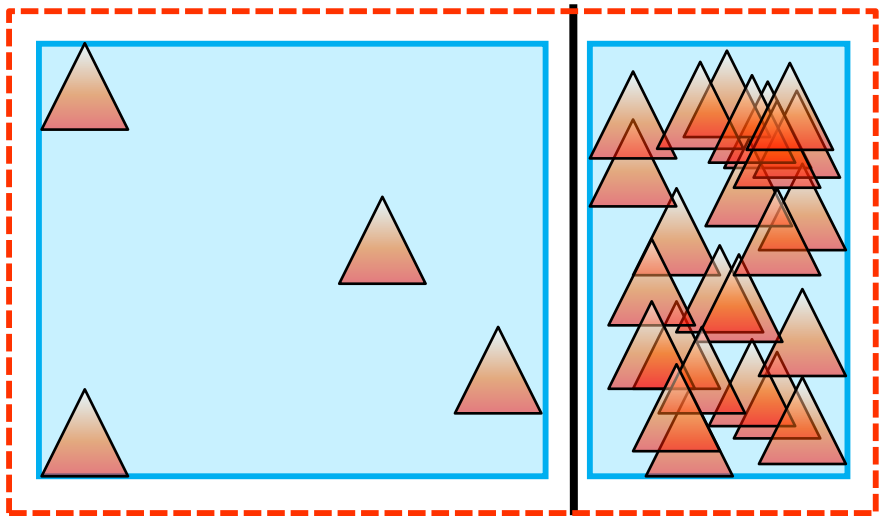


kD-Baum und BVH Konstruktion

Surface Area Heuristic: Kosten für Traversieren eines Knotens

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

- C_T: Kosten für Entscheidung mit welchem Kindknoten fortgefahren wird

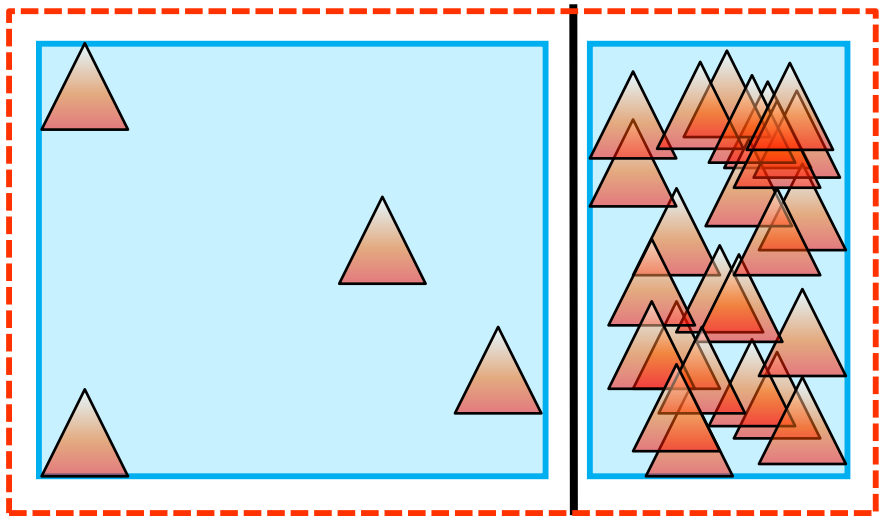


kD-Baum und BVH Konstruktion

Surface Area Heuristic: Kosten für Traversieren eines Knotens

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

- C_T: Kosten für Entscheidung mit welchem Kindknoten fortgefahren wird
- $P(\text{treffe } L), P(\text{treffe } R)$: Wahrscheinlichkeit, dass der Strahl den linken bzw. rechten Kindknoten trifft

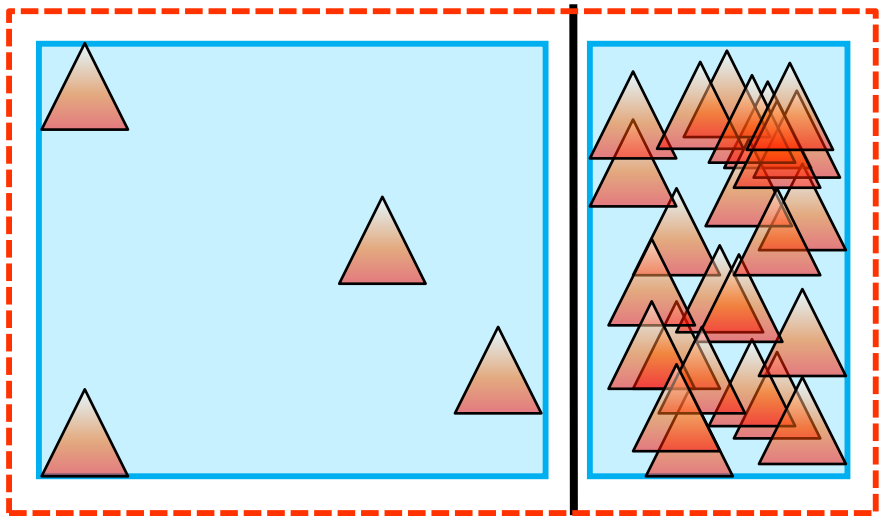


kD-Baum und BVH Konstruktion

Surface Area Heuristic: Kosten für Traversieren eines Knotens

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

- C_T: Kosten für Entscheidung mit welchem Kindknoten 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 → Annahme Primitivkosten × Anzahl

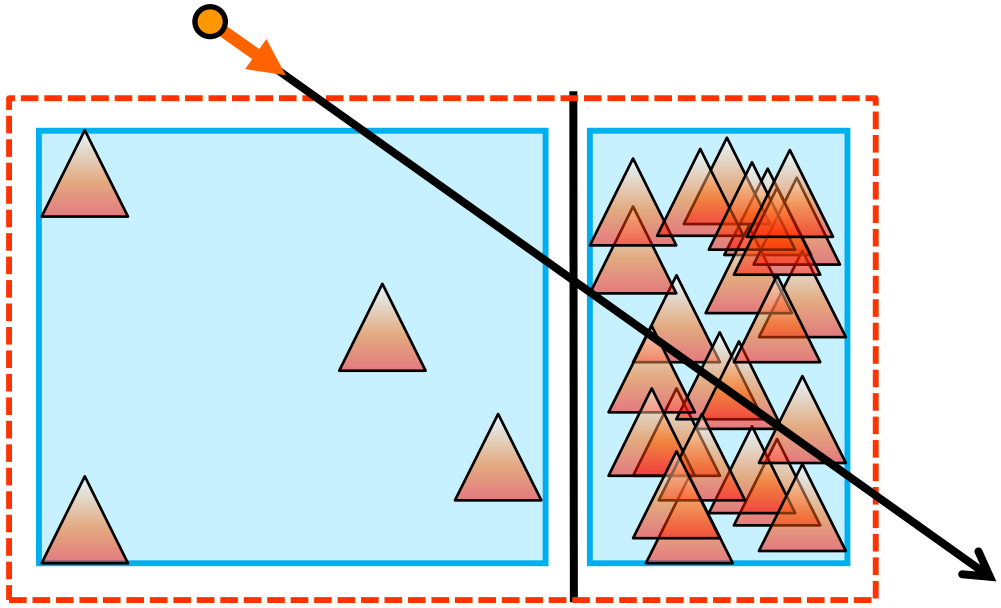


kD-Baum und BVH Konstruktion

Surface Area Heuristic: Kosten für Traversieren eines Knotens

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

- C_T: Kosten für Entscheidung mit welchem Kindknoten fortgefahren wird
- $P(\text{treffe } L), P(\text{treffe } R)$: Wkt., dass Strahl linkes/rechtes Kind trifft
  - eine Konsequenz aus Croftons Theorem (gilt allg. für konvexe Objekte): die Wahrscheinlichkeit, dass ein zufälliger Strahl von außerhalb, der [ ] schneidet, auch [ ] schneidet ist: $SA(\square)/SA(\square)$
  - $SA(.)$ ist die Oberfläche (surface area) einer Box



kD-Baum und BVH Konstruktion

Surface Area Heuristic

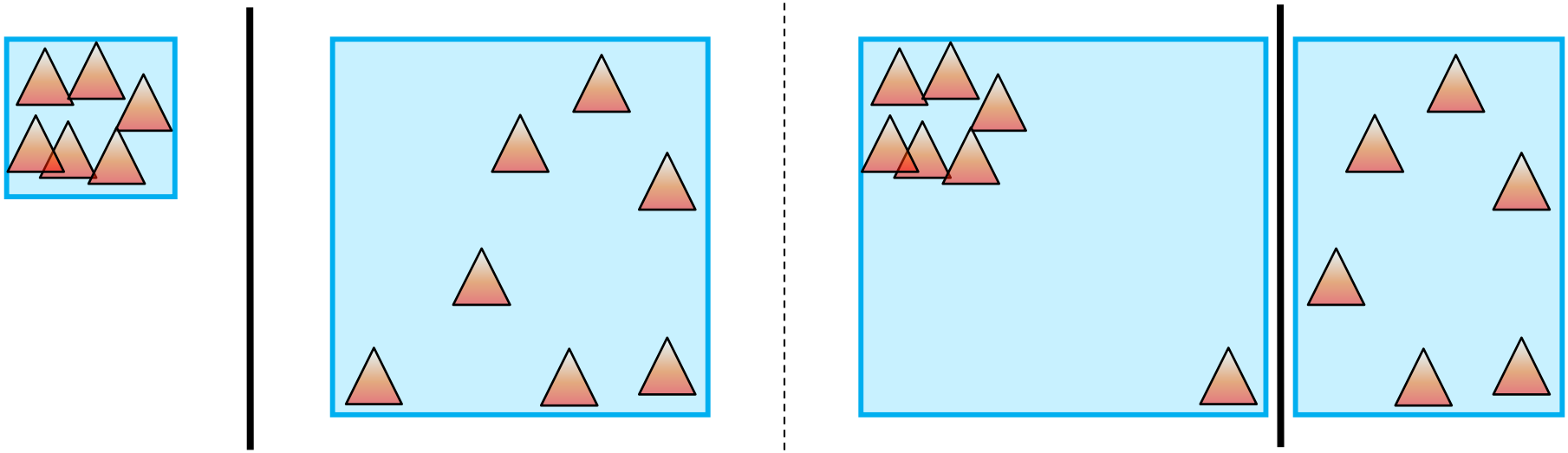
- Kostenfunktion für das Unterteilen eines kD-Baum/BVH Knotens p

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

  - 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
  - C_i sind die Kosten eines Strahl-Primitiv-Schnitttests
  - C_T sind die Kosten der Traversierung eines kD-Baum- bzw. BVH-Knoten
    - unterteile, wenn $C < (|P_l| + |P_r|)C_i$
    - durch den konstanten Overhead C_T lohnt sich Unterteilung bei kleineren Knoten irgendwann nicht mehr
- Ziel: finde die Unterteilung, die die Kostenfunktion minimiert

kD-Baum und BVH Konstruktion

Beispiel: akkurate Unterteilung spielt eine große Rolle



gut

schlecht

Surface Area Heuristics (SAH)

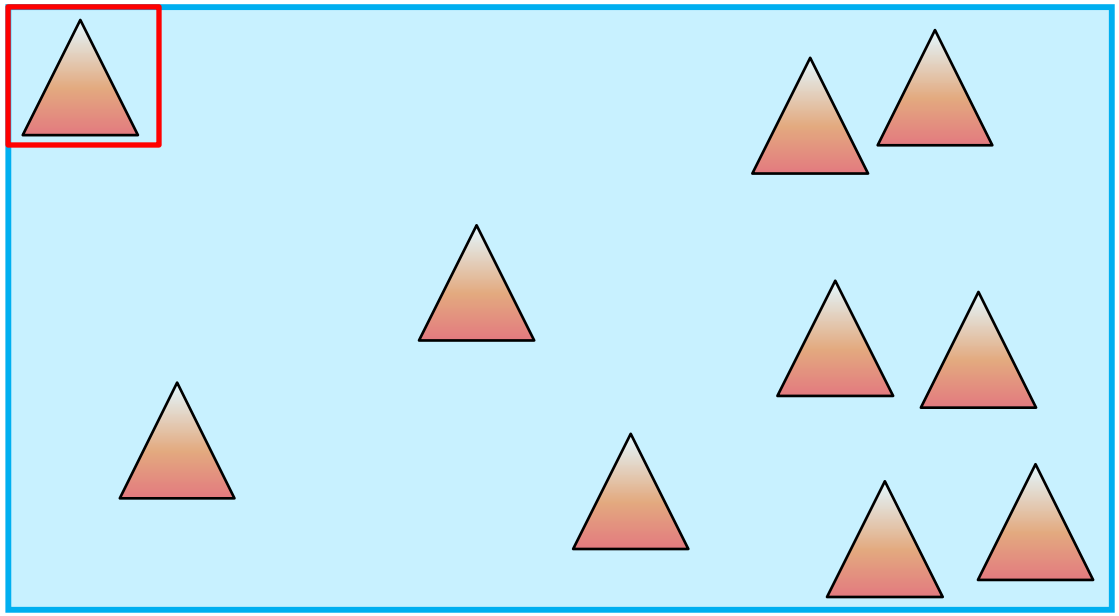
Optimierung/Bestimmung der Unterteilung

- aufwändige Suche nach der besten Unterteilung (niedrigste SAH-Kosten), da der Suchraum groß sein kann
- Variante 1: approximative Konstruktion (nicht elegant)
  - 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
- Variante 2: Konstruktion mit inkrementeller Berechnung (nächste Folie!)
  - sortiere die n Primitive nach x-, y- und z-Achse
  - teste jede der möglichen $3(n-1)$ Aufteilungen
  - inkrementelle Berechnung der AABBs und damit vergleichsweise effiziente Berechnung der SAH möglich
- Konstruktion mit SAH verursacht nicht-vernachlässigbare Kosten!
- gut konstruierte kD-/BSP-Bäume bzw. BVHs können dafür aber um ein Vielfaches schneller sein, als schlecht konstruierte

Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

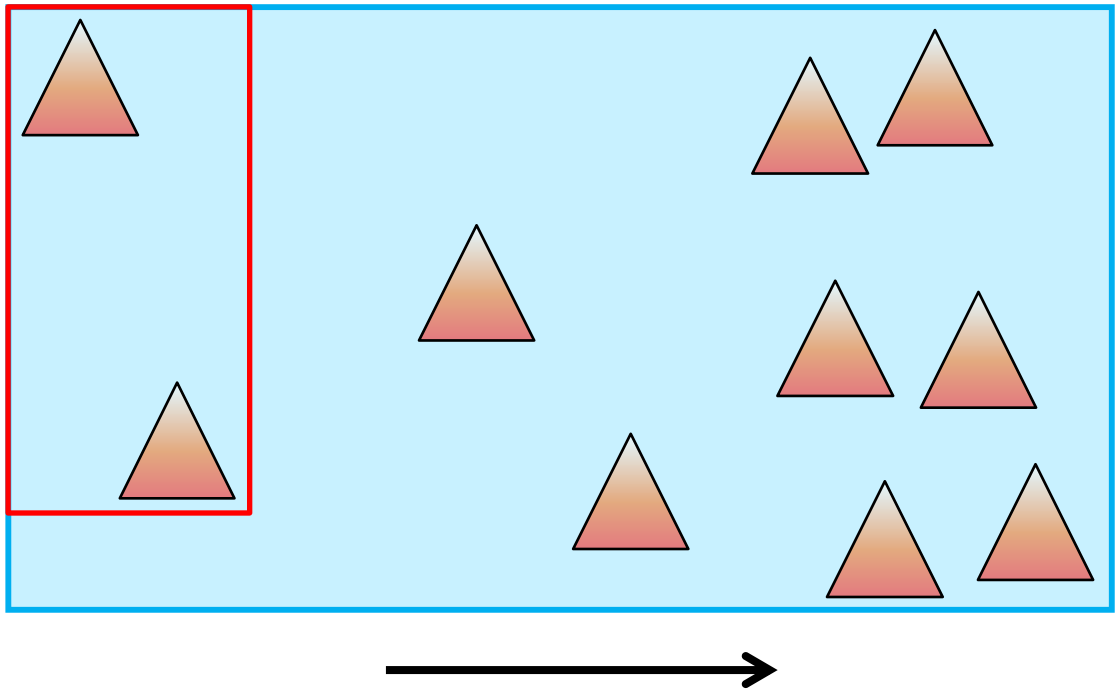
- AABB für linken Teilbaum für Primitiv 0



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

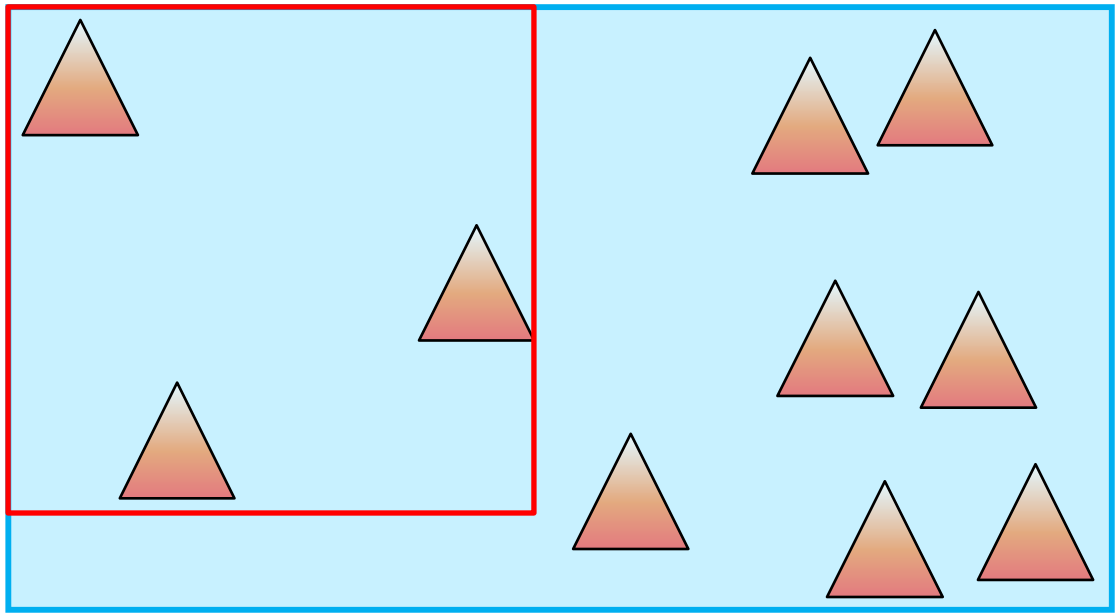
- AABB für linken Teilbaum für Primitiv $0..1$



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

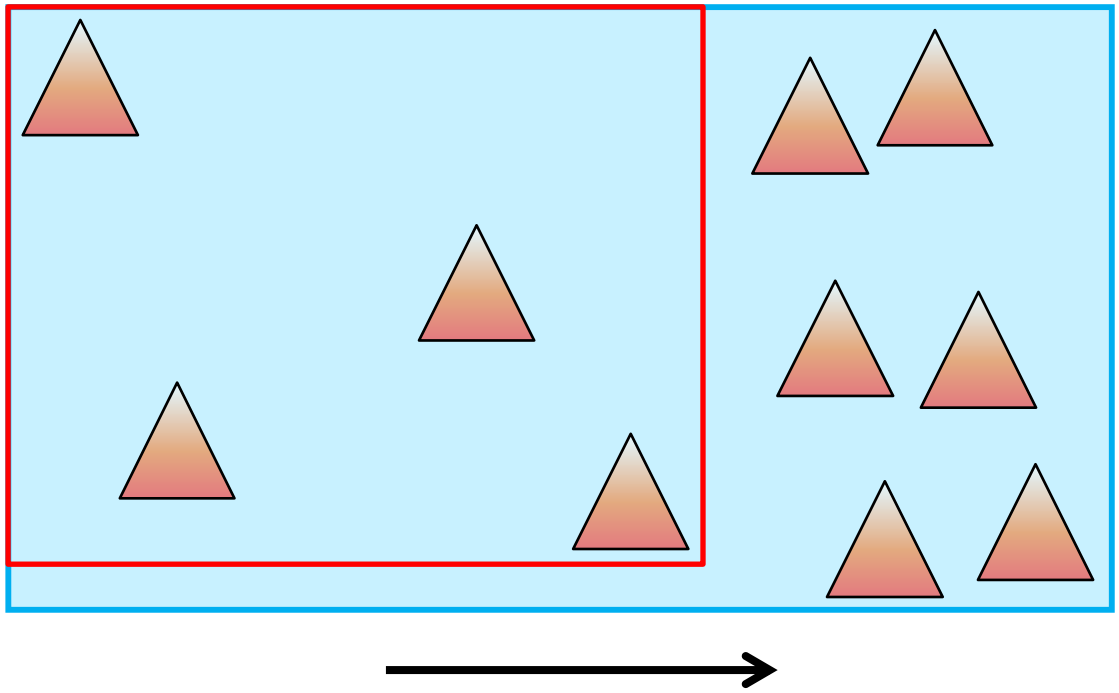
- AABB für linken Teilbaum für Primitiv $0..2$



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

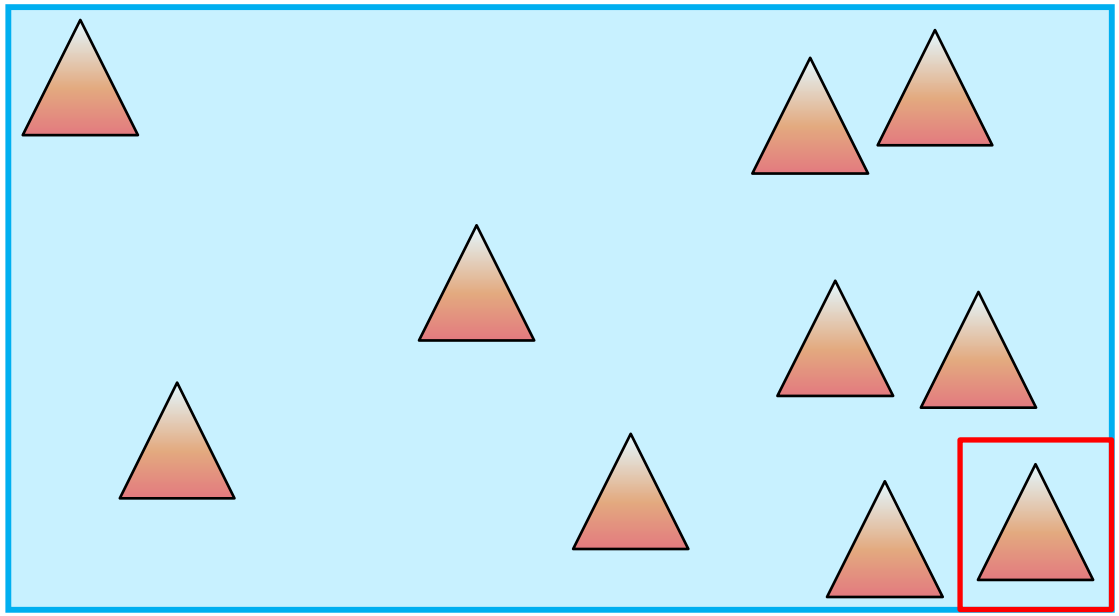
- AABB für linken Teilbaum für Primitiv $0..3$
- weiter bis $0..n-1$



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

- AABB für rechten Teilbaum für Primitiv $n - 1$

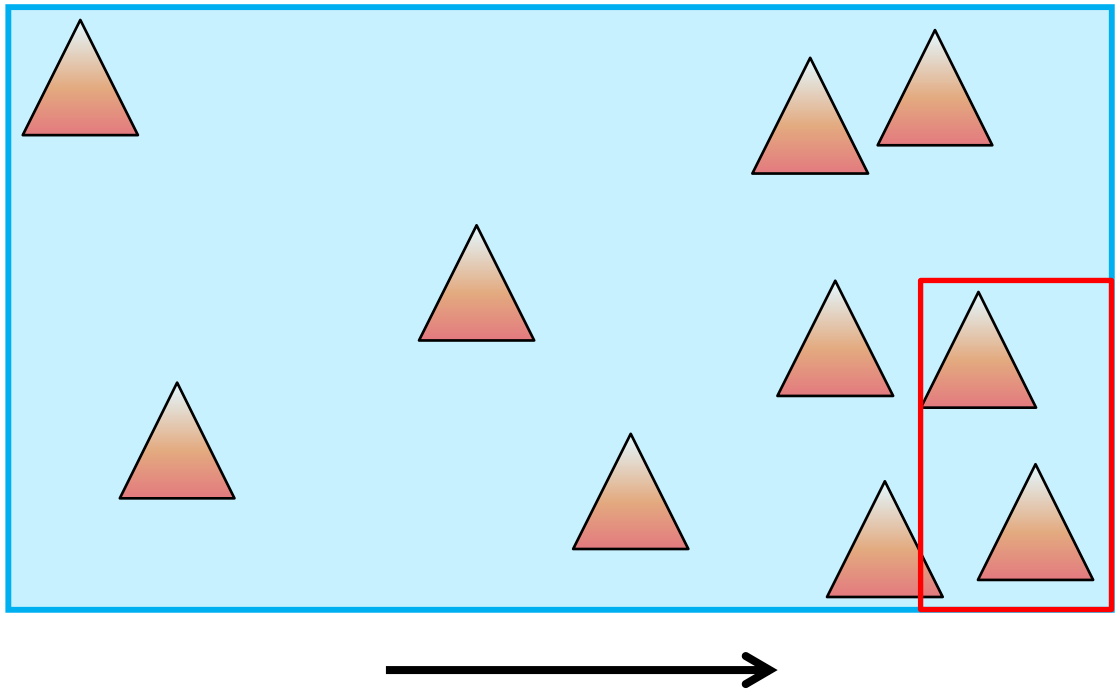


Sortierung der Objekte

Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

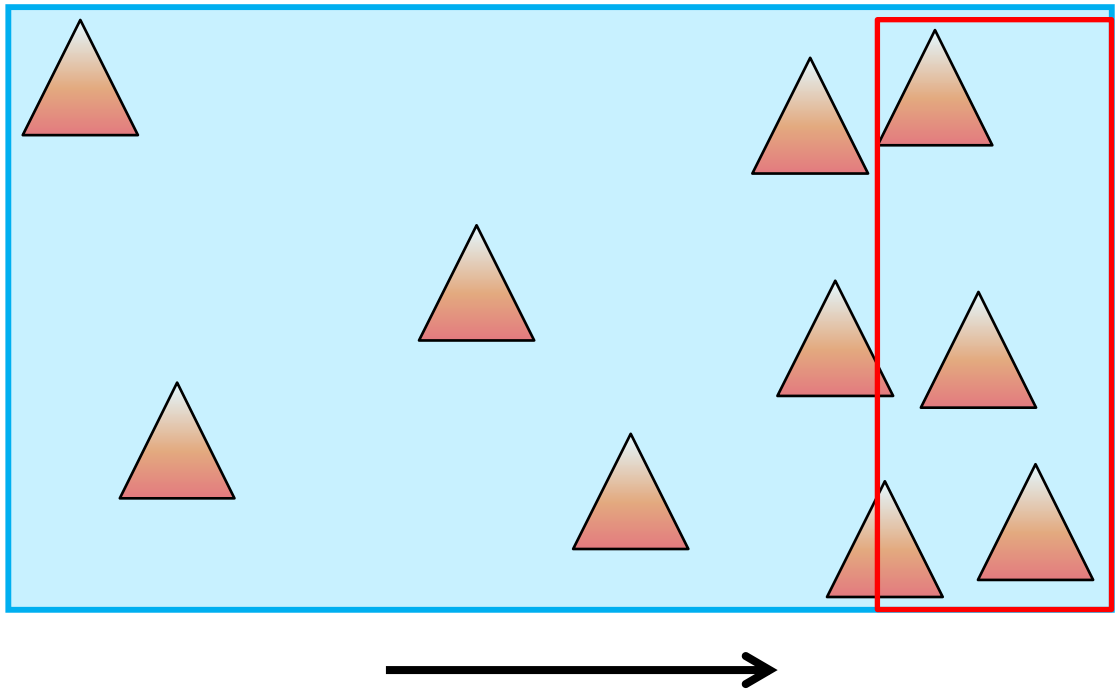
- AABB für rechten Teilbaum für Primitiv $n-2..n-1$



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

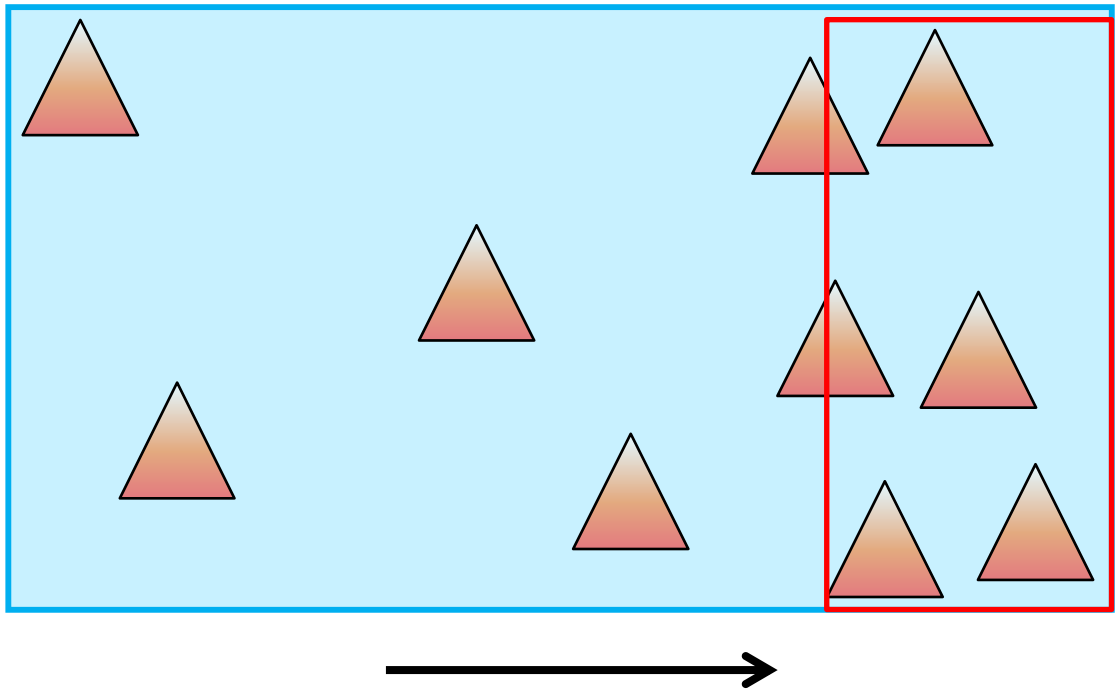
- AABB für rechten Teilbaum für Primitiv $n-3..n-1$



Surface Area Heuristics (SAH)

Inkrementelle Berechnung der AABBs

- AABB für rechten Teilbaum für Primitiv $n-4..n-1$
- die beiden inkrementellen Berechnungen der AABBs „von links“ und „von rechts“ erlauben eine schnelle Berechnung aller $n-1$ SAH-Werte
  - eine Richtung (z.B. von $n-1$ nach 1) berechnen und speichern
  - andere Richtung berechnen und direkt SAH auswerten



kD-Baum und BVH Konstruktion

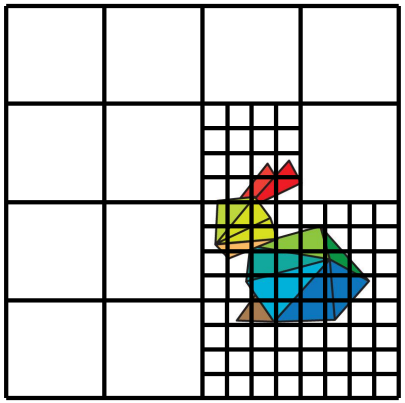
Zusammenfassung: Knotenaufteilung für kD-Baum- und BVH-Konstruktion

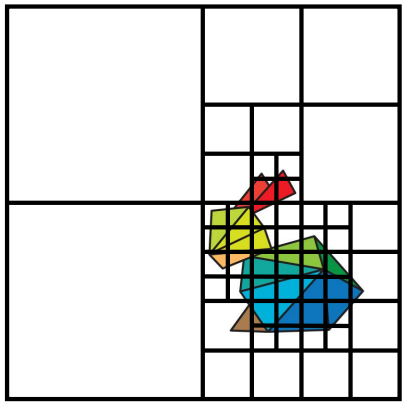
- Bestimmung der Split-Ebene
  - räumliches Mittel (spatial median):
    teile Knoten in der Mitte entlang der Achse der größten Ausdehnung **oder**
    teile erst entlang der x-, dann y-, dann z-, dann wieder x-Achse, usw.
    - Aufbau $O(n \log n)$: Tiefe des Baums $\log n$, je n Primitive zugeordnet
  - 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“ prinzipiell größer

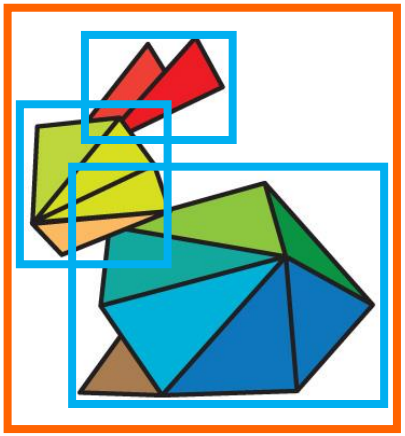
Beschleunigungsstrukturen

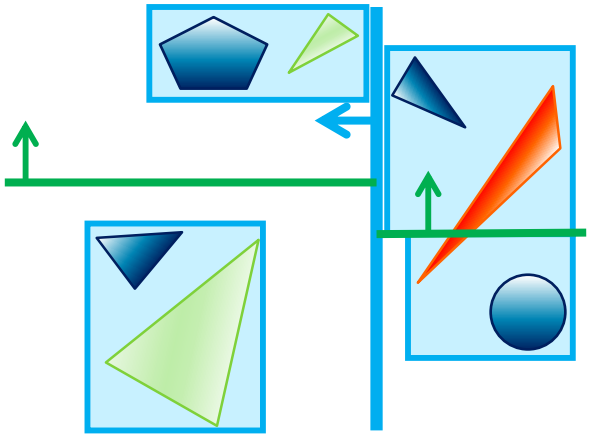
Ein ganzer Zoo und wichtige Aspekte…

- schneller Ausschluss von Schnitttests mit Primitiven erreicht durch Objekt- (BVH) oder Raumunterteilung (alle anderen)





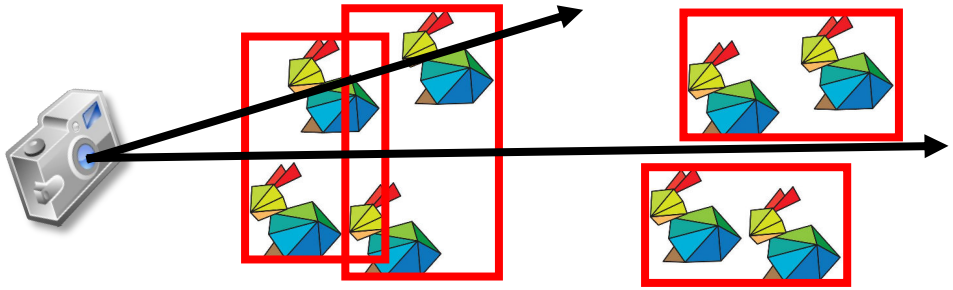


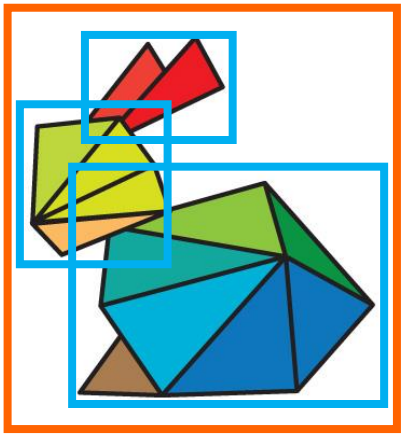


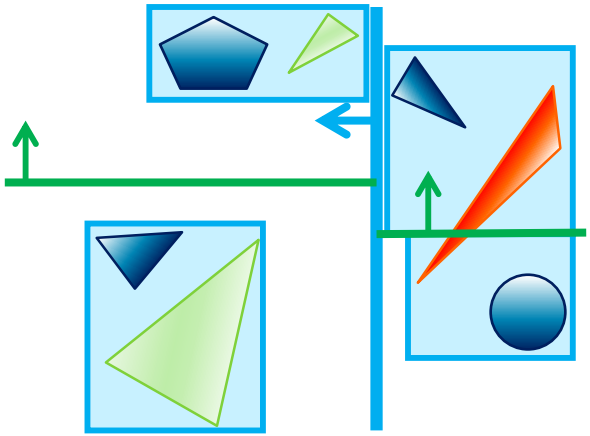
Beschleunigungsstrukturen

Ein ganzer Zoo und wichtige Aspekte…

- schneller Ausschluss von Schnitttests mit Primitiven erreicht durch Objekt- (BVH) oder Raumunterteilung (alle anderen)
  - für Raytracing relevant sind adaptive Beschleunigungsstrukturen, insb. der geschickte (und schnelle!) Aufbau von BVHs und kD-Bäumen



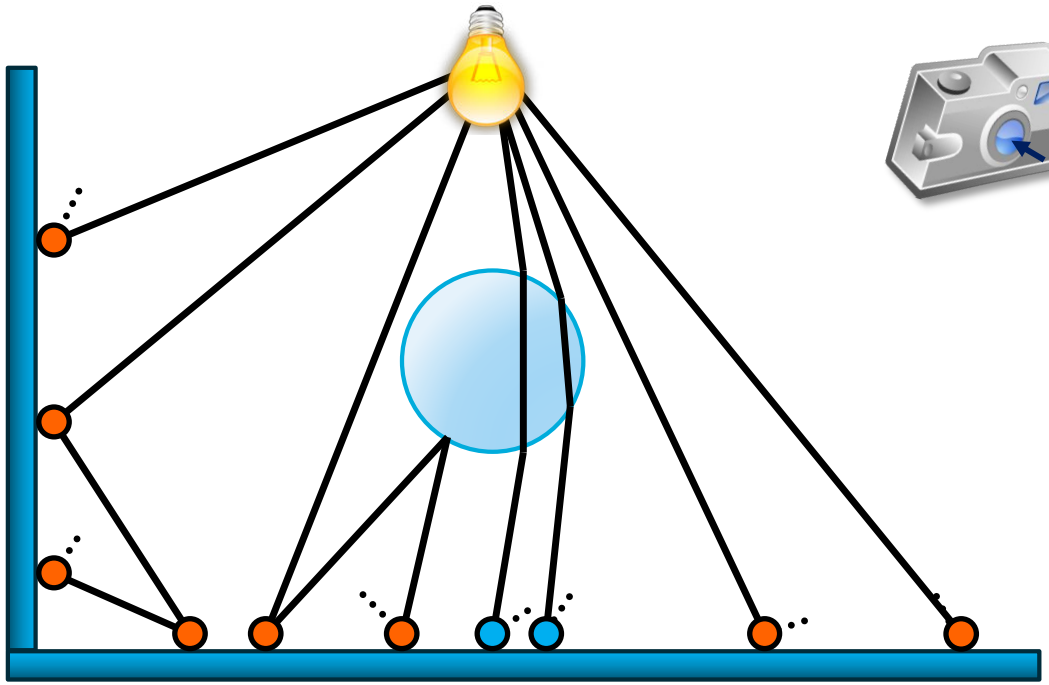


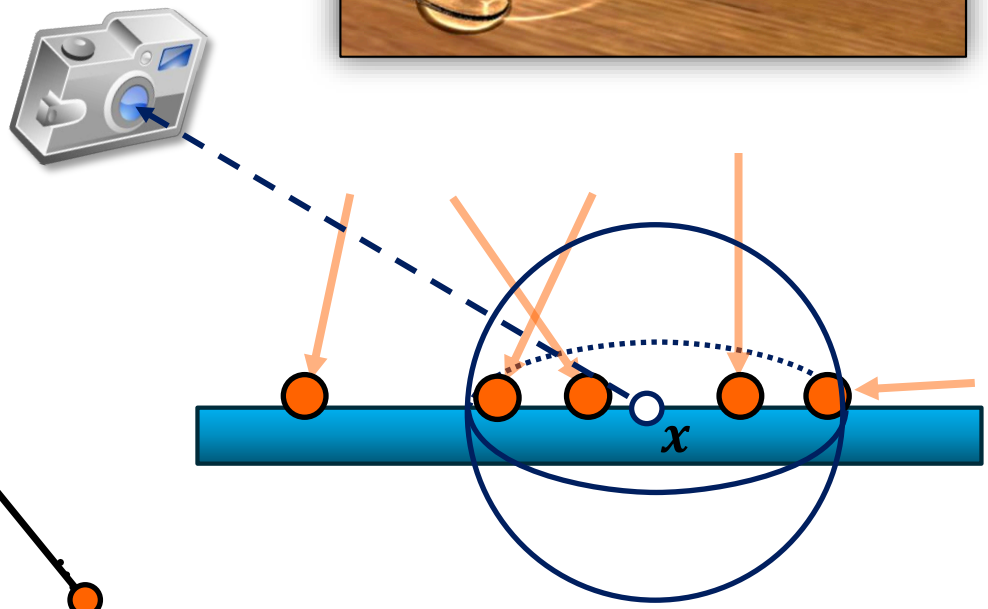


Exkurs: Anwendung in Photon Mapping

Grundidee

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

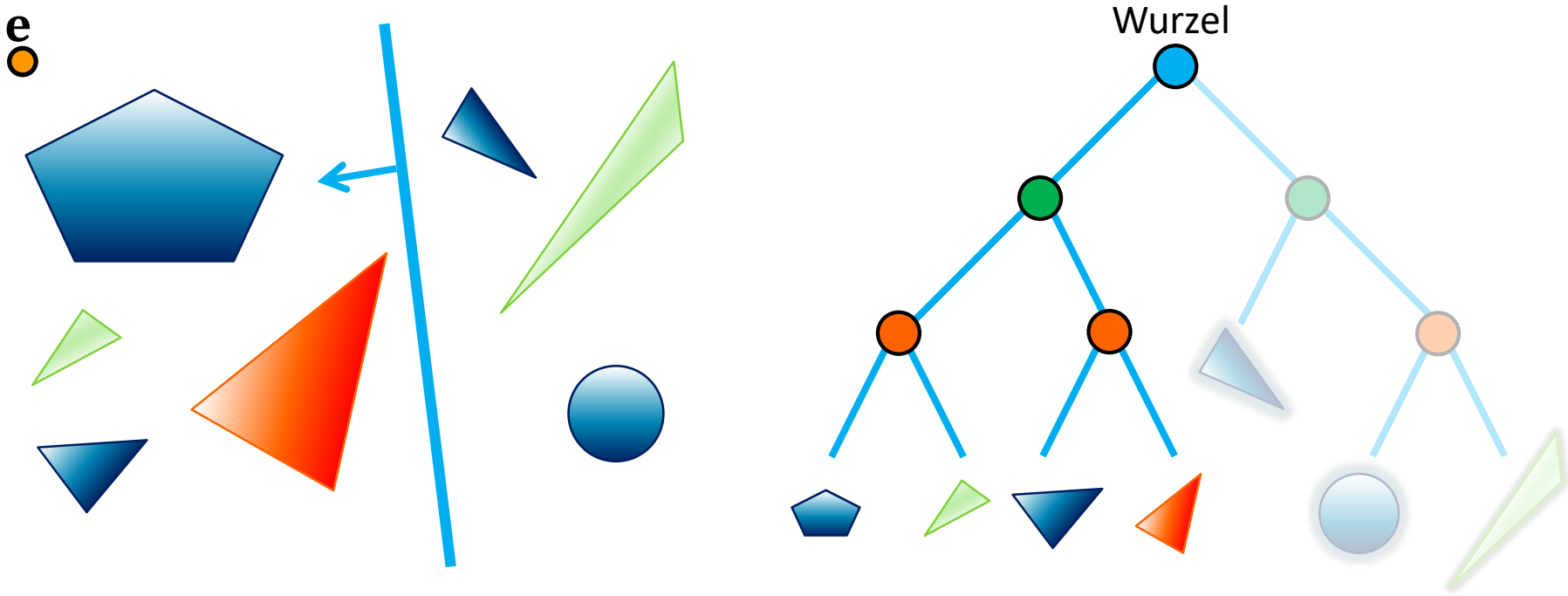




Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

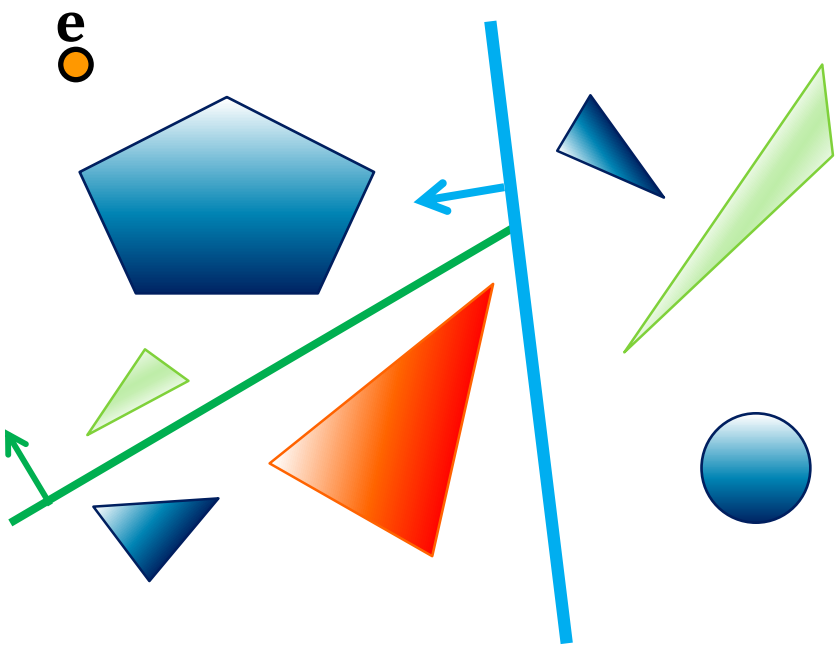
- **e** liegt im positiven Halbraum der Split-Ebene der Wurzel
  - d.h. alle Objekte im negativen Halbraum sind „weiter hinten“

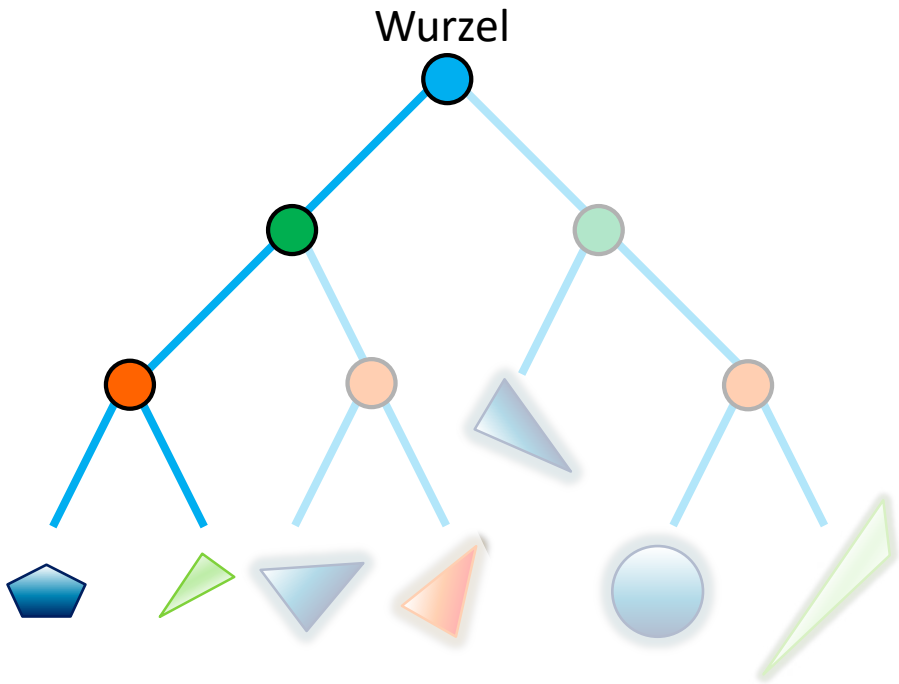


Suche im kD-/BSP-Baum

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“



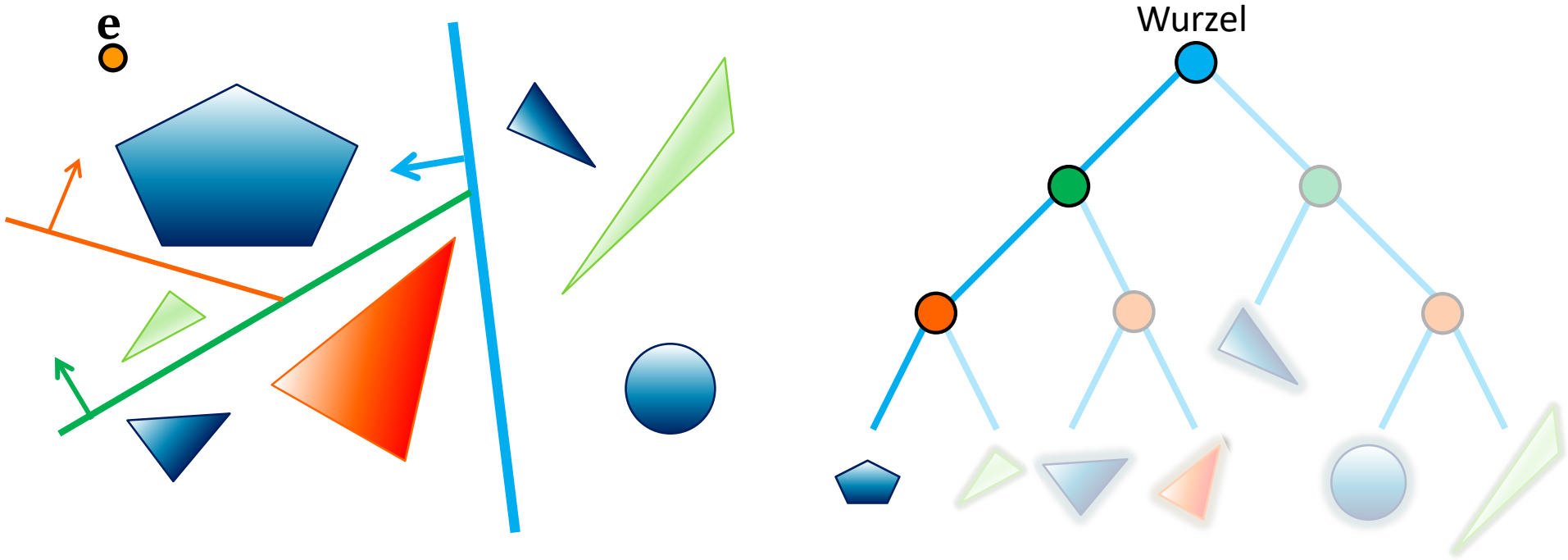


Suche im kD-/BSP-Baum

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“

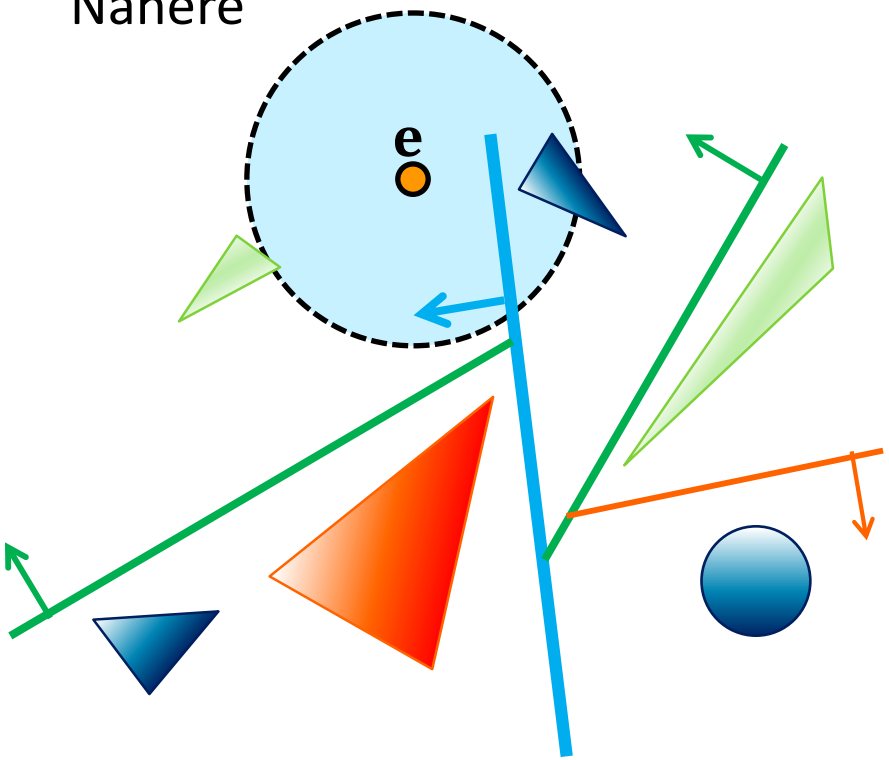
⚠ dieses Prozedere findet den Unterraum in dem sich **e** befindet, aber nicht immer das nahste Objekt!

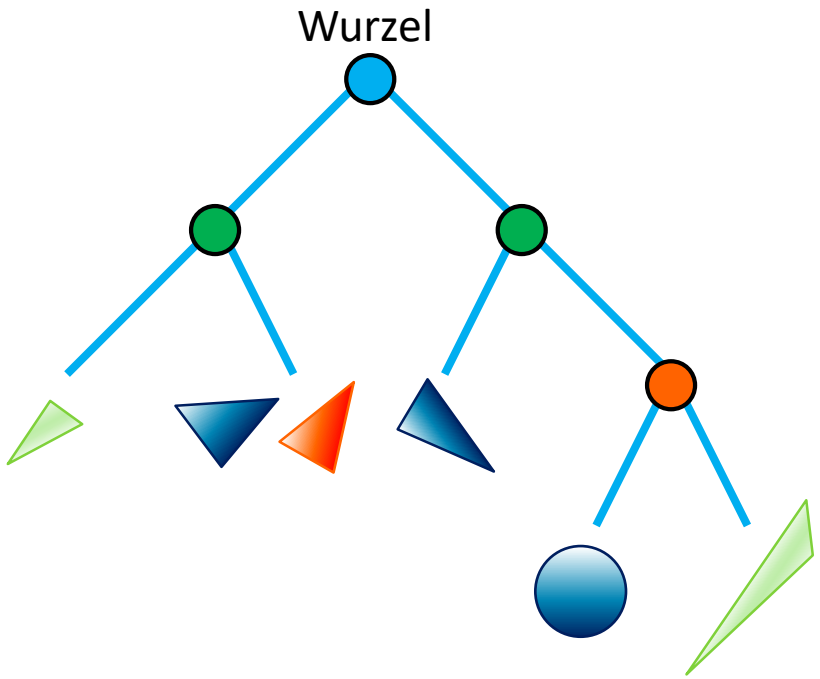


Suche im kD-/BSP-Baum

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

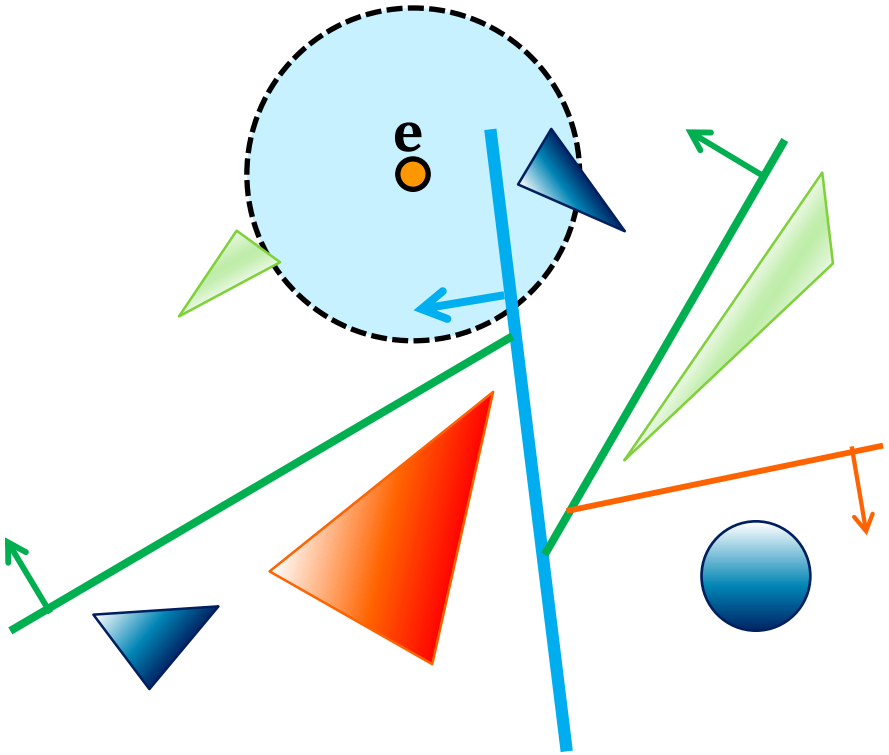


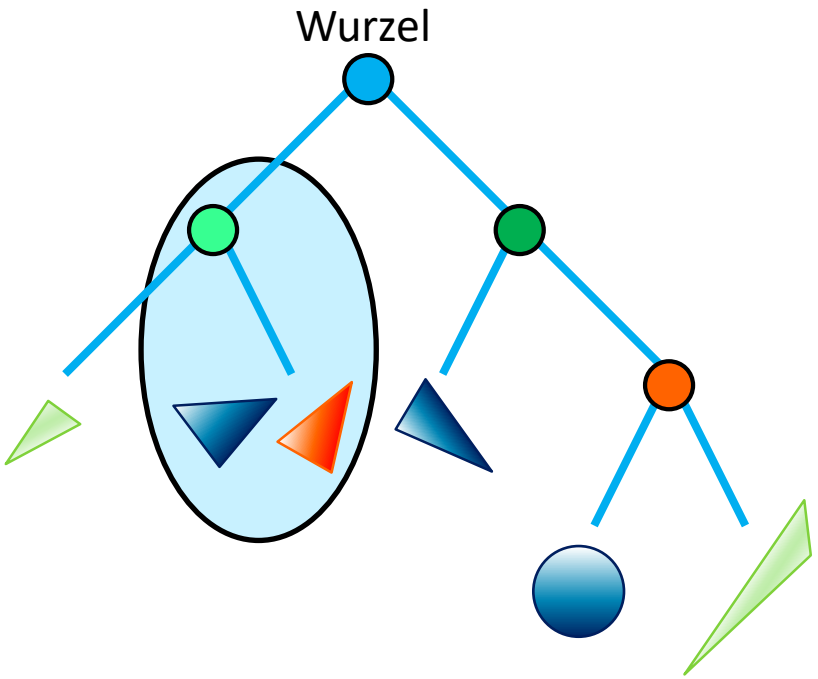


Suche im kD-/BSP-Baum

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)

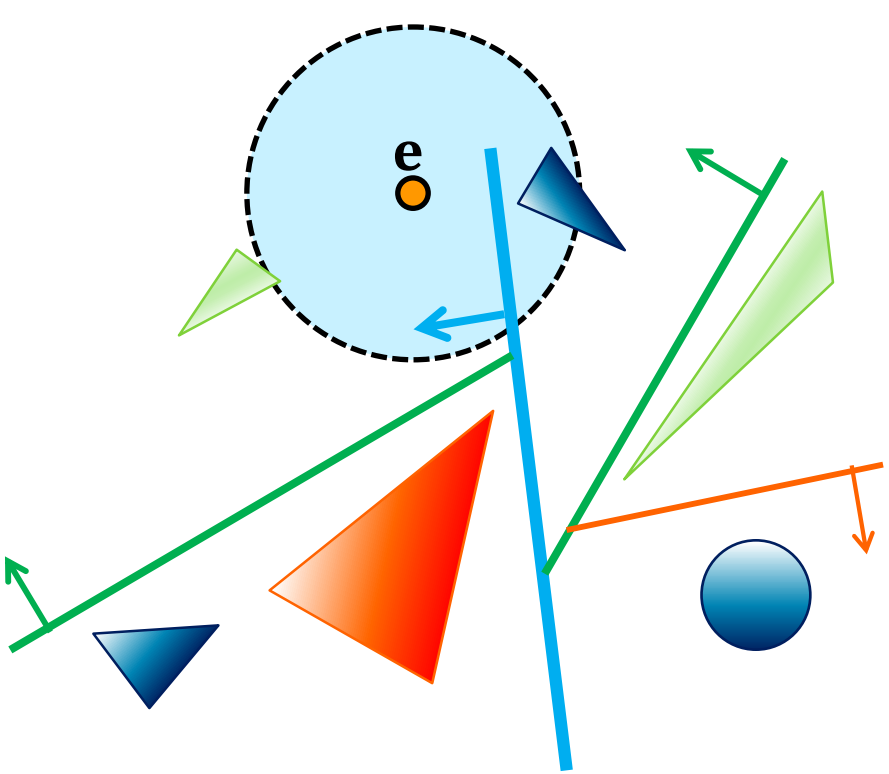


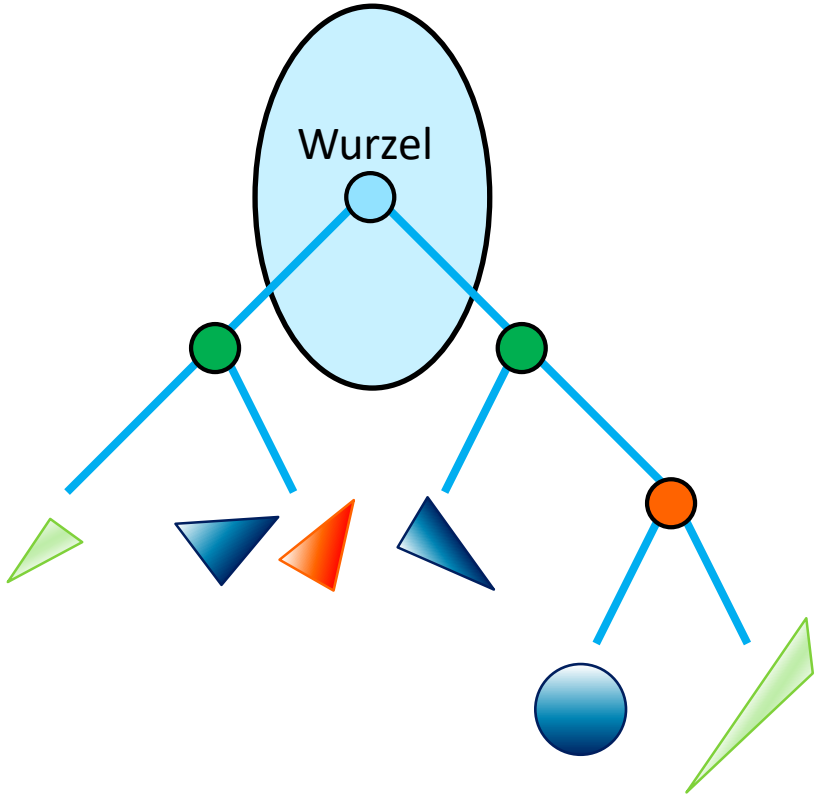


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

- das gesuchte Objekt kann in jedem Unterraum liegen, der näher an **e** liegt, als das bisher gefundene Primitiv
- gehe zum Eltern-Elternknoten ○ (hier schon die Wurzel) und prüfe ob der andere Kindknoten/Teilbaum in Frage kommt (hier der Fall)

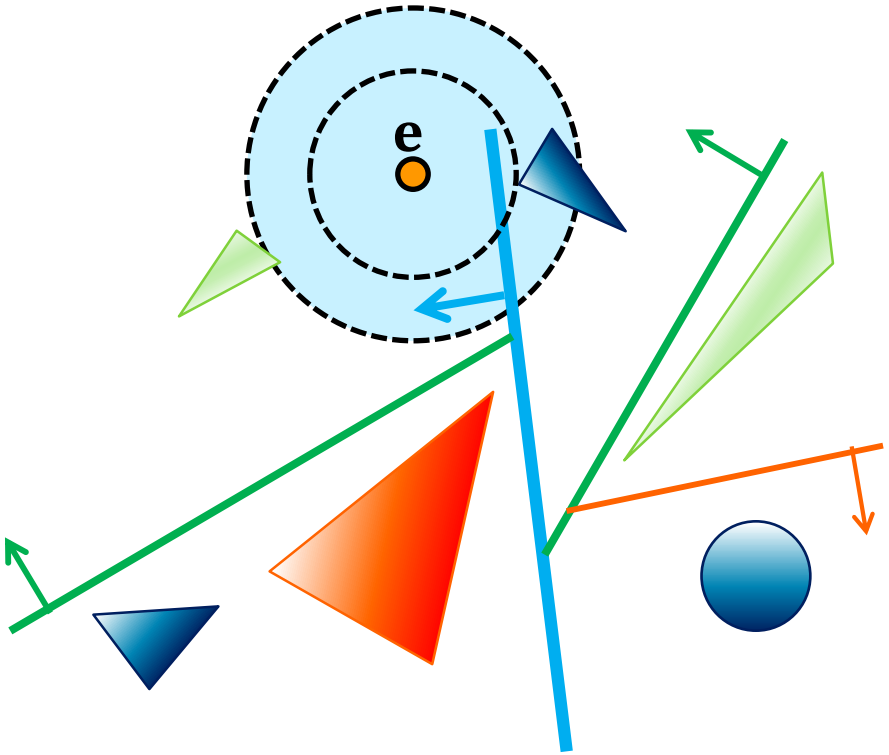


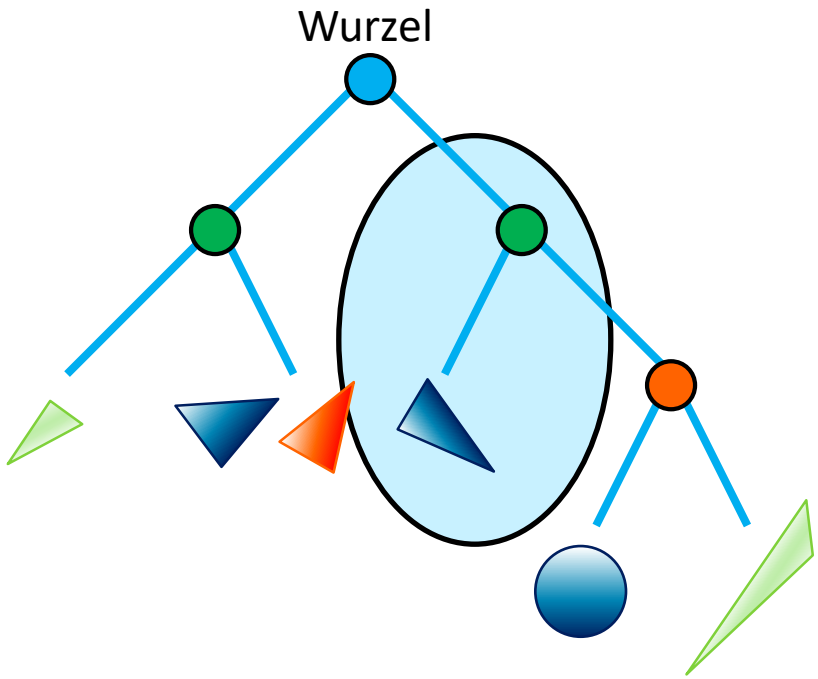


Suche im kD-/BSP-Baum

Suche das nahste Objekt/Primitiv zum Punkt e

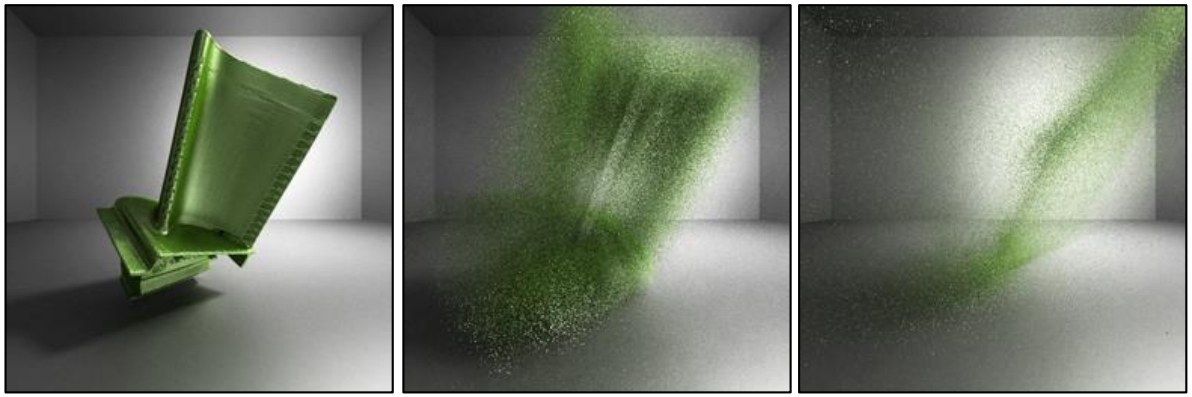
- steige in diesem Teilbaum wieder durch Tiefensuche ab (depth-first traversal): jeweils den Ast des näheren Teilbaums zuerst
- im Mittel $O(\log n)$, worst case $O(n)$ (n Objekte/Primitive)





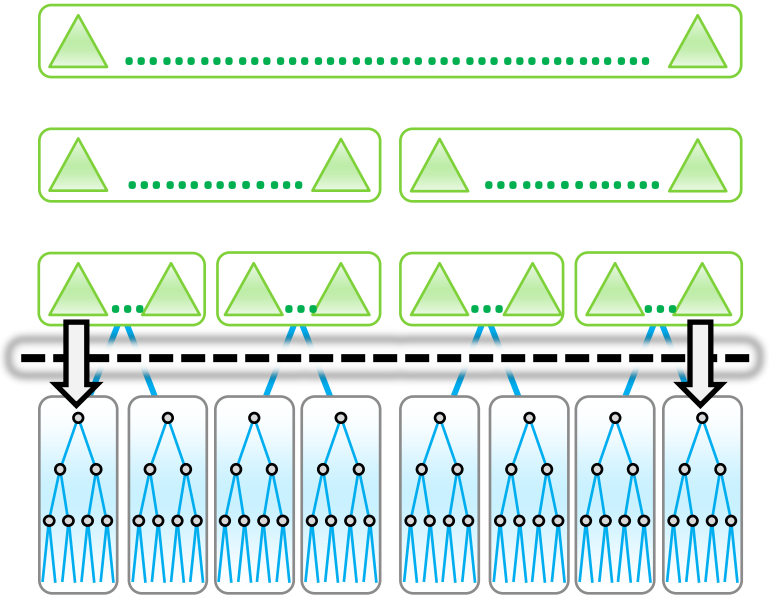
Parallele Konstruktion räumlicher Datenstrukturen

Neuberechnung und inkrementelle Updates in dynamischen Szenen?



Bilder:
Pantaleoni und Luebke

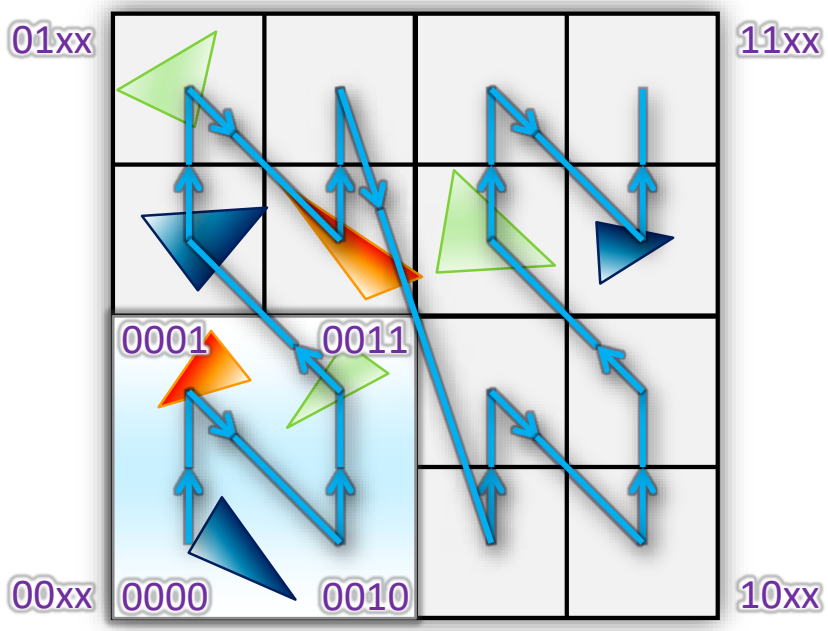
- paralleler Aufbau in Millisekunden für Millionen Dreiecke auf CPU und GPU
  - mehrere hundert Mio. Δ pro Sekunde mit und ohne SAH auf CPUs
  - Aufbaugeschwindigkeit und Qualität sind widersprüchliche Kriterien
  - ein weiteres Problem ist der temporäre Speicherbedarf bei der Konstruktion
- inkrementelle Updates in großen Szenen meist mit „two-level BVHs“

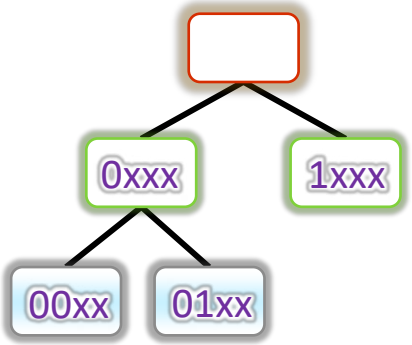


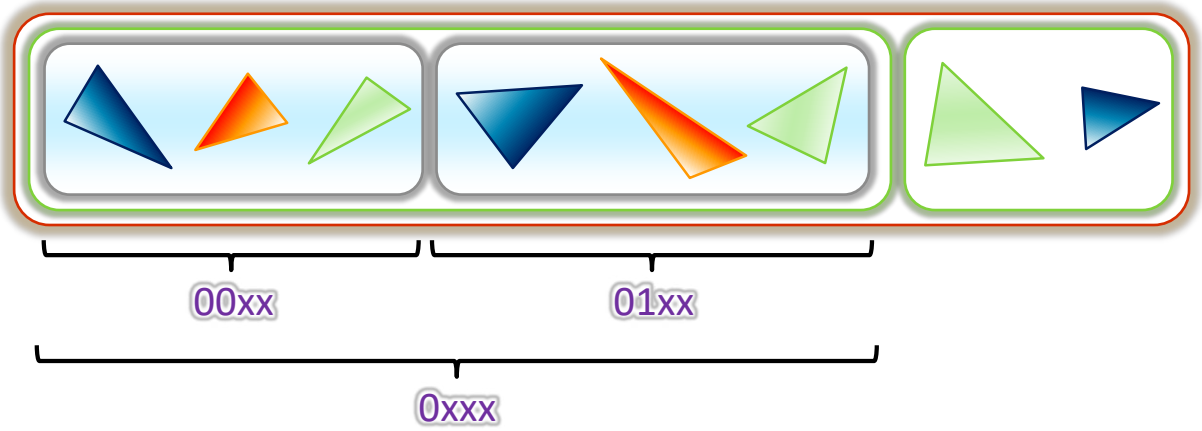
Parallele Konstruktion räumlicher Datenstrukturen

Grundidee: Vorsortierung durch Quantisierung und Morton Codes

- schnellstmögliche Konstruktion, zu Lasten der Qualität
- quantisiere Primitivschwerpunkte
- Zuweisung Morton Codes zu Zellen
- sortiere Primitive nach Zellcodes
- ergibt implizite Baumstruktur



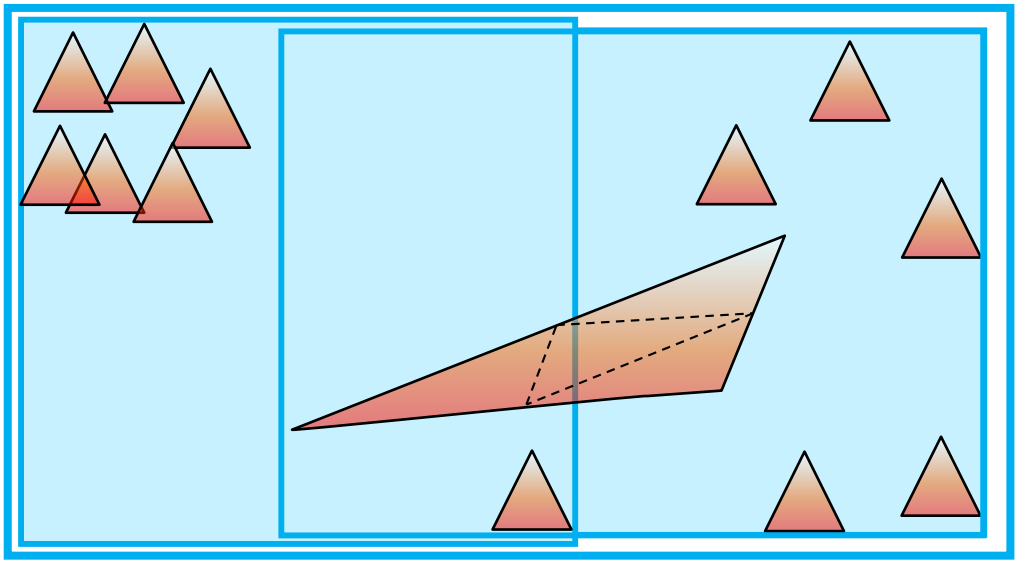




kD-Baum und BVH Konstruktion

Bestimmen der besten Unterteilung

- große Primitive verhindern eine gute Unterteilung (insb. bei BVH)

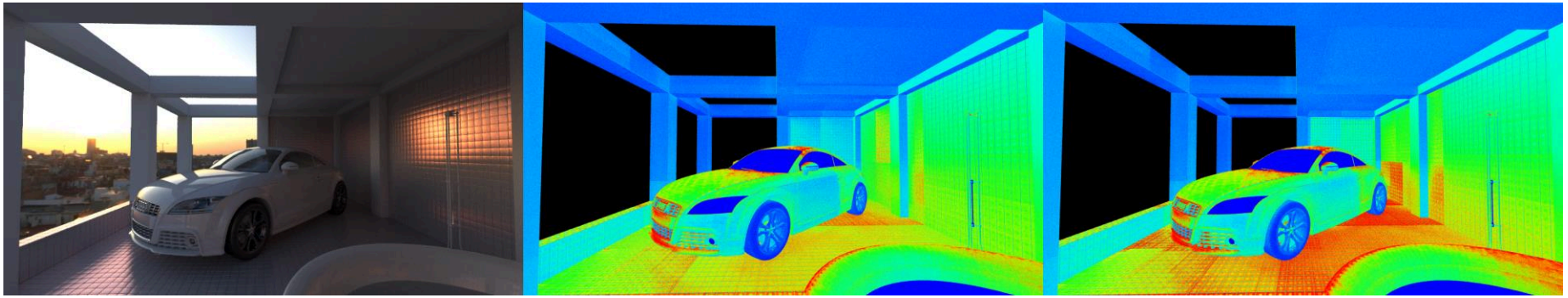


Optimierung

- 1-zu-4 Split → geringere Tiefe (Stack und Cache-Effekte)
  z.B. Multi Bounding Volume Hierarchy, Ernst und Greiner, 2008
- ...

Varianten und aktuelle Forschungsthemen

- Schattenstrahlen: weiche von Reihenfolge ab und teste zuerst nahe, große Dreiecke (nach Verdeckungswahrscheinlichkeit priorisiert)
  SATO: Surface Area Traversal Order for Shadow Raytracing, Nah et al.
  MBVH Child Node Sorting for Fast Occlusion Test, Ogaki et al.

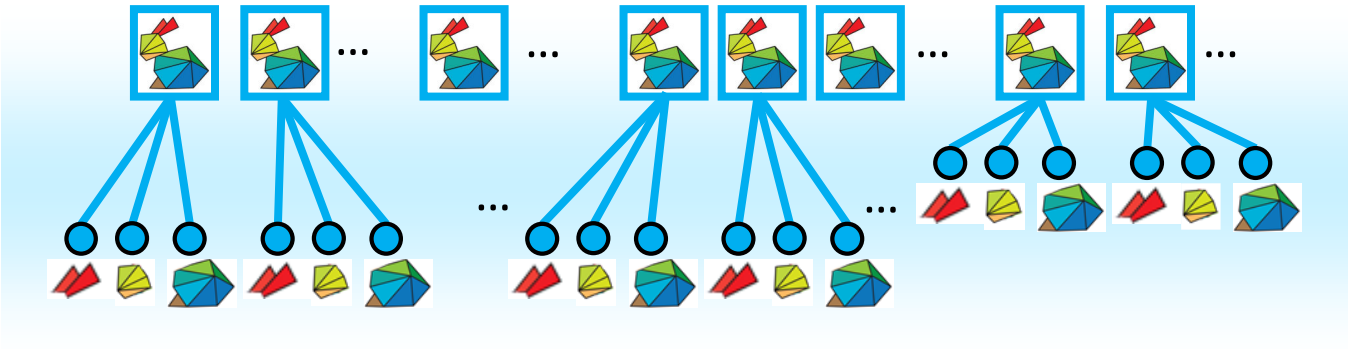


- Treelets: schneller paralleler Aufbau durch lokale „Umstrukturierung“
  Fast Parallel Construction of High-Quality Bounding Volume Hierarchies, Karras und Aila, 2013
- Konstruktion anhand repräsentativer Geometrie-Stichproben
  Stochastic Subsets for BVH Construction, Tessari et al., 2023
- immer noch aktuelle Forschung, manchmal ganze Konferenz-Sessions, z.B: www.highperformancegraphics.org/2024/program/index.html
  → Session 1: Acceleration Structures

Hüllkörperhierarchie 2-Level-BVH

2-Level-BVH: üblich bei teildynamischen Szenen und Grafik-Hardware

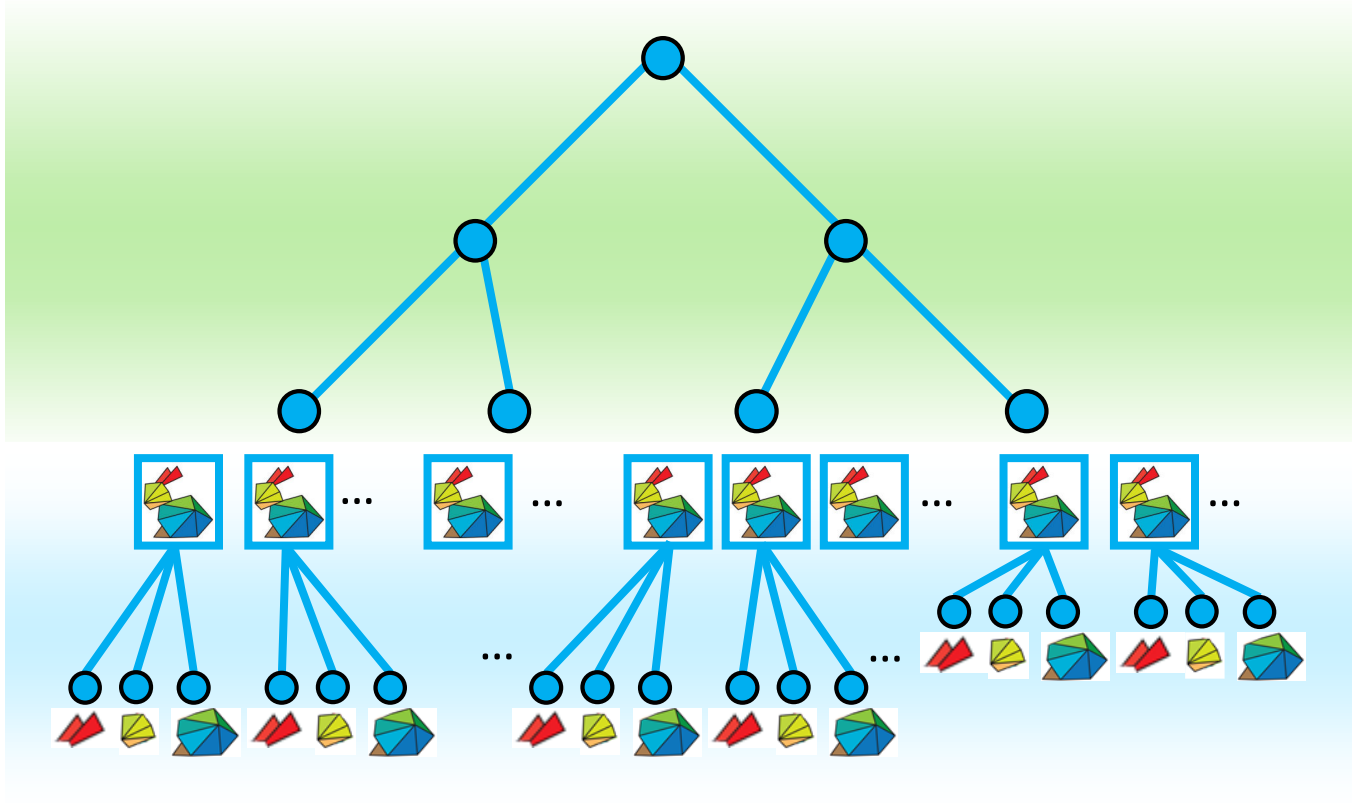
- BVHs pro (rigidem) 3D-Objekt



Hüllkörperhierarchie 2-Level-BVH

2-Level-BVH: üblich bei teildynamischen Szenen und Grafik-Hardware

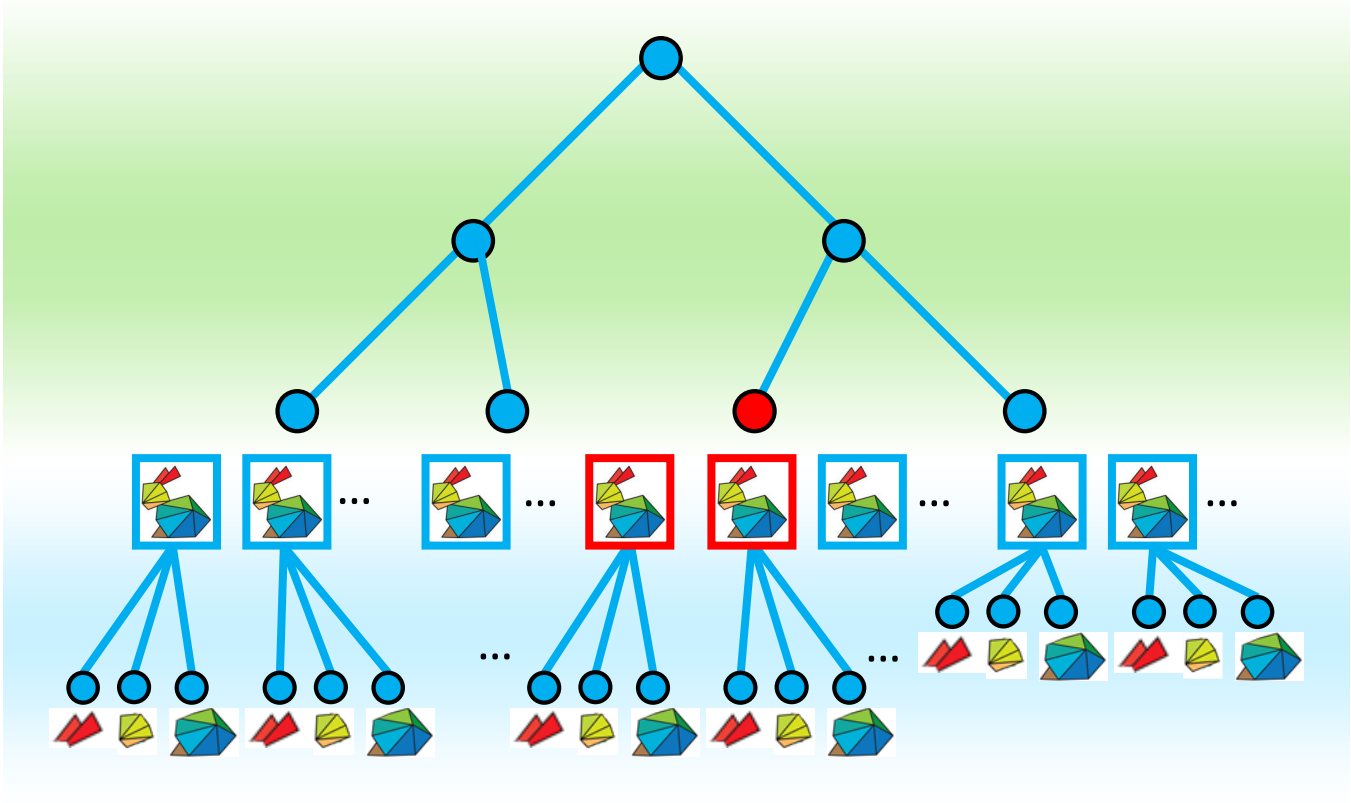
- BVHs pro (rigidem) 3D-Objekt
- … darüber eine zweite BVH



Hüllkörperhierarchie 2-Level-BVH

2-Level-BVH: üblich bei teildynamischen Szenen und Grafik-Hardware

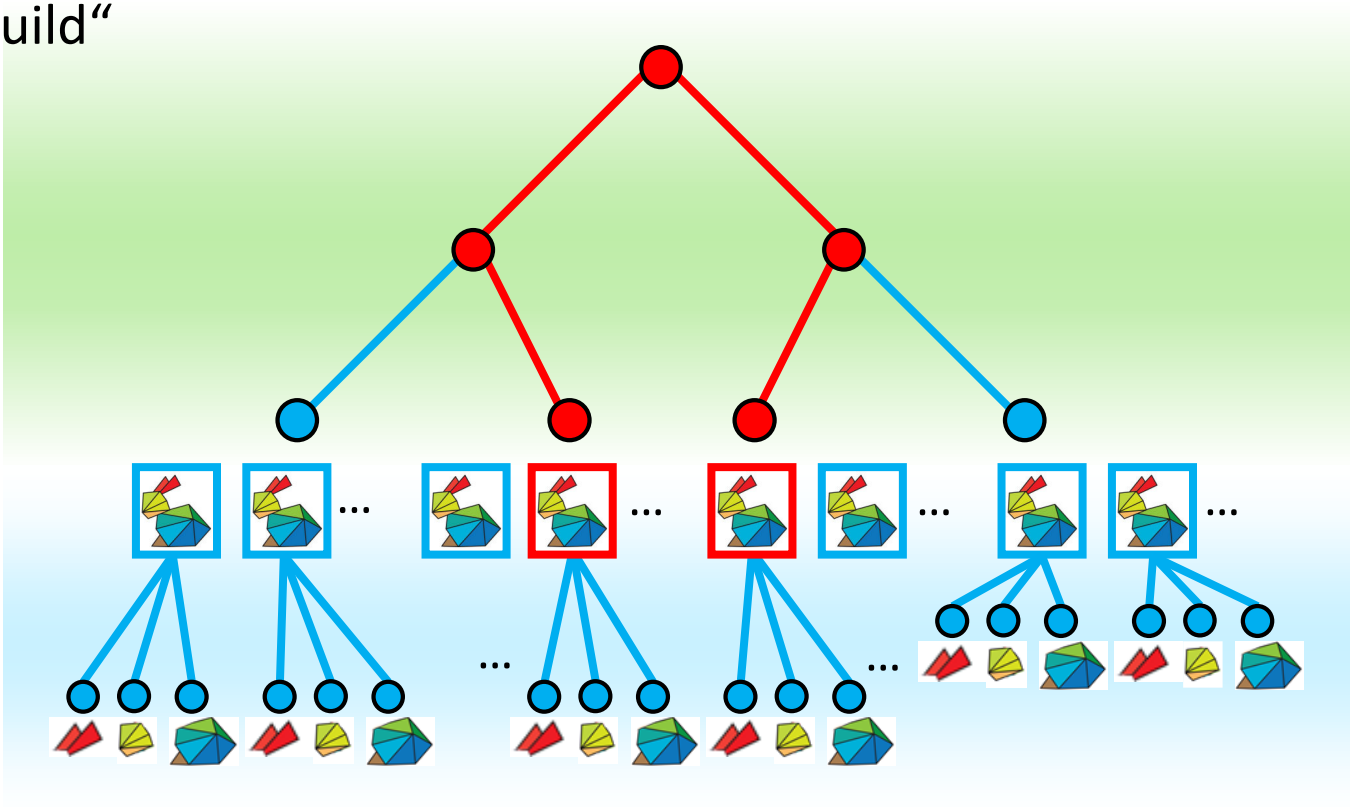
- BVHs pro (rigidem) 3D-Objekt
- … darüber eine zweite BVH
  → kleine Bewegungen: „refit“



Hüllkörperhierarchie 2-Level-BVH

2-Level-BVH: üblich bei teildynamischen Szenen und Grafik-Hardware

- BVHs pro (rigidem) 3D-Objekt
- … darüber eine zweite BVH
  → größere Änderungen:
  „rebuild“

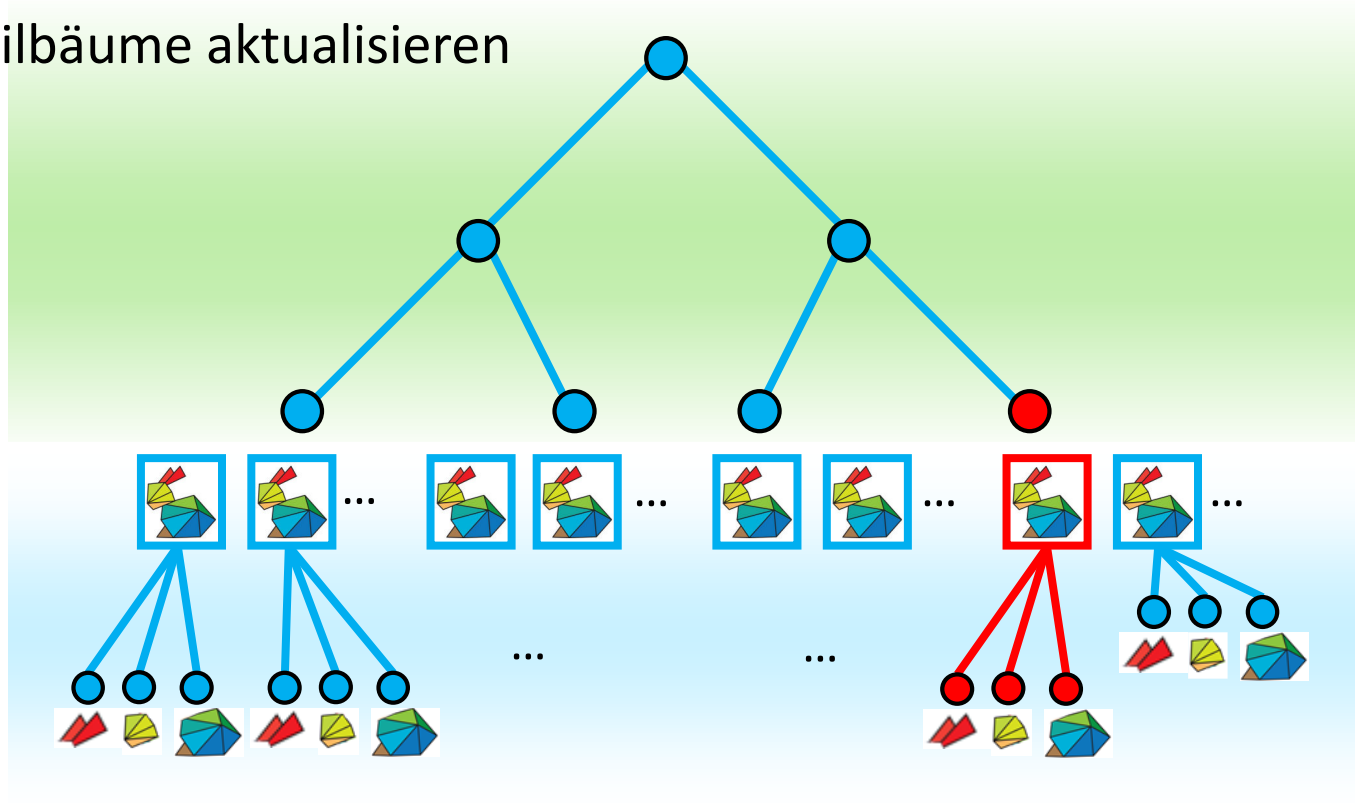


Hüllkörperhierarchie 2-Level-BVH

2-Level-BVH: üblich bei teildynamischen Szenen und Grafik-Hardware

- BVHs pro (rigidem) 3D-Objekt
- … darüber eine zweite BVH
- dynamische Geometrie:

  nur Teilbäume aktualisieren



Modernes Raytracing

- moderne Grafik-Hardware unterstützt nativ Strahlschnitte mit räuml. DS
  - Konstruktion der DS auf CPU möglich
  - „Echtzeit-Raytracing“ bedeutet oft noch: Hybridansatz aus Rasterisierung für direkt sichtbare Flächen und Raytracing für Sekundärstrahlen

- kohärente Strahlenpakete: Gruppen von Strahlen mit ähnlichem Ursprung und Richtung (z.B. Primärstrahlen, Schattenstrahlen etc.)
  - schnelle Verarbeitung: oft ähnliche Wege durch die Beschleunigungsstruktur
- inkohärente Pakete können Parallelität der HW weniger (gut) nutzen
  → es gibt sogar Versuche Sekundärstrahlen während des Raytracing sortieren

- CPU-Raytracing: SIMD-Architekturen und Strahlenpakete (typ. 4-32 Strahlen)
  - Bsp. der Schnitttest Strahl-AABB kann mit SIMD Befehlen einfach auf einen Test „Strahl gegen 4 AABBs“ erweitert werden

Fazit (Achtung: Erfahrungswerte!)

Vergleich Datenstrukturen

- BVH mit AABBs: schnelle Erzeugung, gute Hierarchie
  (tendenziell etwas Probleme mit großen Dreiecken)
- kD-Baum: aufwändiger zu konstruieren, manchmal einen
  Tick besser als BVH, optional kombiniert mit AABBs pro Knoten
- BSP-Baum: aufwändig zu konstruieren, oft nur ein marginal
  besser als ein gut konstruierter kD-Baum
- Gitter: für Raytracing nicht gut geeignet trotz einfachem Aufbau
  - oft als Suchstrukturen, z.B. in SPH-Simulationen verwendet
  - 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 (Physiksimulation, Robotik)