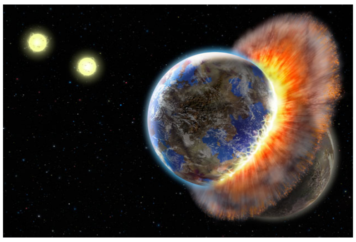
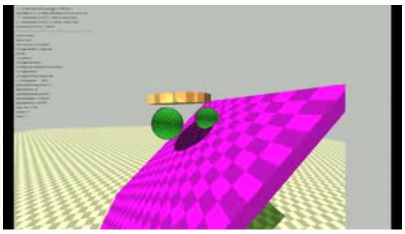
Virtuelle Realität
Kollisionsdetektion



G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de

Anwendungsbeispiele

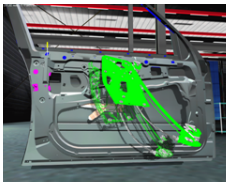
Virtual Prototyping



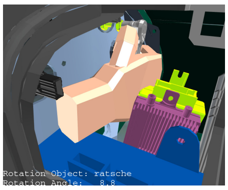
Physikalisch basierte Simulation

Einsatzgebiete von Kollisionserkennung

- Grundlegende Operation:
  - Physikalisch-basierte Simulation
  - Interaktion in VR
  - Haptisches Rendering
- Anwendungsfelder:
  - Spiele, Animation, Medizin, Virtual Prototyping, Pfadplanung, Teleoperation, Roboter-Kollisionsvermeidung, …



Hierarchische Kollisionserk.



Koll.erkennung innerhalb einer Simulation

- Main *loop*:
  - Objekte bewegen
  - Kollisionen checken
  - Kollisionen behandeln, z.B.: Objekte zurückbewegen, Kräfte bestimmen
- Kollisionen stellen zwei Probleme:
  - Kollisionserkennung
  - Kollisionsbehandlung
- Im folgenden: Kollisionserkennung

Definitionen

- Gegeben $P, Q \subseteq \mathbb{R}^3$
- Erkennungsproblem ("*detection problem*"):
  "P und Q Kollidieren": $\Leftrightarrow$

  $$P \cap Q \neq \varnothing \Leftrightarrow$$

  $$\exists x \in^3 : x \in P \wedge x \in Q$$

- Konstruktionsproblem ("*construction problem*"):

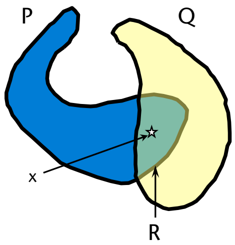
  $$R := P \cap Q$$

- Definition "Kollision" für polygonale Objekte:

  P,Q kollidieren $\Leftrightarrow$

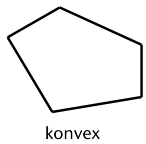
  $$\exists f \in F^P \exists f' \in F^Q : f \cap f' \neq \varnothing$$

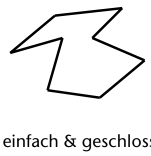
- Andere Definition in der Spielebranche

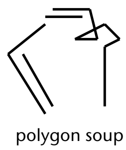


Objekt-Klassen

- konvex
- geschlossen, einfach
  (keine Selbstdurchdringung)
- Sack Polygone ("*polygon soup*")
  - nicht notwendig geschlossen
  - doppelte Polygone
  - koplanare Polygone
  - Selbstdurchdringungen
  - degenerierte Polygone
  - Löcher
- starr / flexibel



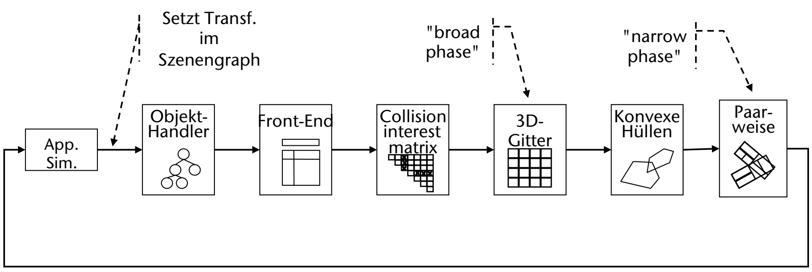




Anforderungen an Kollisionserkennung

- Möglichst große Klasse von Objekten
- Viele bewegliche Objekte (einige 1000)
- Schnell, damit physikalische Simulation iterieren kann
  (wenigstens 2x 100,000 Polygone in <1 Millisek.)
- Kollisionspunkt (“*witness*”), falls Kollision;
  und optional: alle Kollisionspunkte
- Nicht zu große zusätzliche Datenstrukturen (<2x);
  der Aufbau dieser Datenstrukturen sollte nicht zu lange dauern,
  damit man das zur Ladezeit machen kann (< 5sec / Objekt)

Kollisions-Pipeline



Die Collision-Interest Matrix

- Anwendungsspezifisch:
  - Verschiedene Module interessieren sich für verschiedene Paare;
  - manche Objekt-Paare kollidieren immer, manche Paare können nicht kollidieren;
- Vermeide unnötige Kollisionstests
  ⇒ *Collision-Interest* Matrix
- Dreiecks-Matrix, Elemente enthalten:
  - Flag, ob Kollisionserkennung
  - bei inkrementellen Algos
    Infos zum Status beim letzten Frame
    (z.B. bei Algo S die separierende Ebene)
  - *Callbacks* in die Module

Obj	1	2	3	4	5	6	7	8
1			x	x	x	x		
2						x		
3						x		x
4								x
5								x
6								x
7								x
8								

Mehrkörper-Kollisionserkennung

- Kollisionserkennung vieler Objekte:
  Schließe jedes Objekt in eine Bounding-Box ein;
  vergleiche deren BBoxes vor dem exakten Koll.test
- n Objekte bewegen sich

→ *brute-force* Methode muss $O(n^2)$ BBoxes vergleichen.

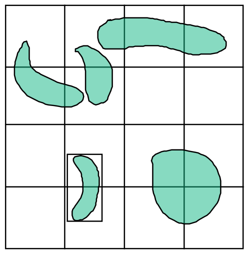
- Idee:
  versuche zum Objekt P schnell die "Nachbarn" zu finden und mache nur mit diesen Bbox-Vergleiche

→ Raum-Gitter, *Sweep-Plane*, etc.

Raum-Gitter

Idee:

1. Teile die “Welt” in regelmäßiges Gitter.
2. Objekte “benachbart”, wenn sie gemeinsame Gitterzelle(n) belegen.
3. Bestimme Zellen-Belegung anhand der BBoxes (exakte Bestimmung zu teuer)
4. Objekt bewegt sich
   - → Gitter updaten.

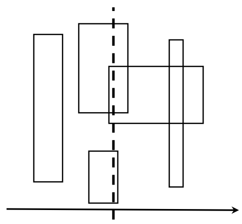


Trade-Off:

- weniger Zellen = größere Zellen
  - → entfernte Objekte sind “benachbart”;
- mehr Zellen = kleinere Zellen
  - → Objekte belegen mehr Zell
  - → Aufwand zum Update wird größer

Plane-Sweep

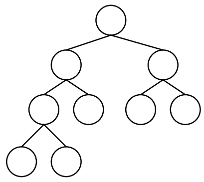
- Idee:
  Lasse eine Ebene senkrecht zur x-Achse durch den Raum streichen (“*sweep*”)
- Algo:
  sortiere x-Koord. der Box-Ränder
  starte mit der linken Box
  führe Liste “aktiver” Boxes
  springe von Box-Rand zu Box-Rand:
  if aktueller Box-Rand ist “linke” Seite einer Box
  füge diese Box zur aktiven Liste hinzu
  checke neue Box gegen alle anderen in der aktiven Liste (2D)
  else
  entferne diese Box aus der aktiven Liste



Szenengraph

Idee:

- Verwende die Hierarchie des Szenen-Graphen,
- Analog zu hierarchischer Koll.erkennung.



Unterschiede:

- Nur 1 Hierarchie statt 2;
- Blätter = Objekte (statt Blätter = Polygone);
- Alle Blätter bewegen sich.

Probleme:

- Hierarchie wird schnell ineffizient,
- Oft keine Hierarchie von außen vorgegeben.
  (Bsp.: Auto-Daten)

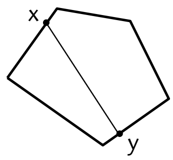
Frame-to-Frame Coherence

- Beobachtung:
  Zwei aufeinander folgende Bilder in einer zusammenhängenden Sequenz unterscheiden sich (meistens) wenig.
- Beispiele
  - Kamerabewegung
  - Objektbewegung
- Anwendungen
  - Computer Vision (Bsp.: Tracking von Markern)
  - MPEG
  - Kollisionserkennung
  - Ray Tracing von Animationen (?)
- Algorithmen basierend auf frame-to-frame coherence heißen “inkrementell”, manchmal auch “dynamisch” oder “on-line”

Konvexe Objekte

- Definition "konvexer Polyeder":

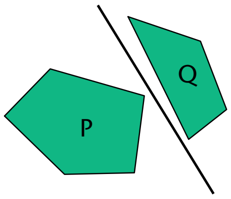
$$P \subset \mathbb{R}^3 \text{ konvex } \Leftrightarrow$$
$$\forall x, y \in P : \overline{xy} \subset P \Leftrightarrow$$
$$P = \bigcap_{i=1,\cdots,n} H_i \quad , H_i = \text{Halbräume}$$



- Bedingung für Nicht-Kollision: "linear separierbar"

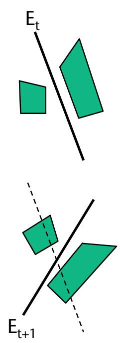
$$P \cap Q = \bigcap_{i=1}^{n1+n2} H_i = \varnothing \Leftrightarrow$$
$$\exists i : P \subseteq H_i \wedge Q \subseteq H_i^c$$

("P liegt ganz auf der einen Seite von H_i,
Q liegt ganz auf der anderen Seite")



Algorithmus "separierende Ebene"

- Idee:
wenn im letzten Frame E eine trennende Ebene zwischen P und Q war, dann ist in diesem Frame die trennende Ebene "in der Nähe" von E (evtl. ist es sogar dieselbe).



E_t war separierende Ebene zu P,Q im Frame t gewesen

E := E_t

repeat max n times

if es ex. $v \in V^P$ auf der Rückseite von E

drehe/verschiebe E, so daß v auf der Vorderseite

if es ex. $v \in V^Q$ auf der Vorderseite von E

drehe/verschiebe E, so daß v auf der Rückseite

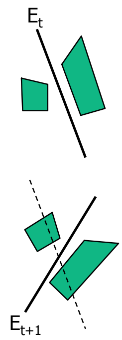
if kein $v \in V^P$ und kein $v \in V^Q$ auf der "falschen" Seite

return "keine Kollision"

es gibt immer noch v's auf der "falschen" Seite

return "Kollision" {kann manchmal falsch sein}

speichere E_{t+1} := E für's nächste Frame

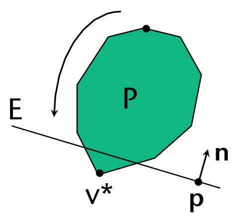


Finde schnell eine Ecke auf der "falschen" Seite

- Brute-Force: teste für alle **v** ob

$$f(\mathbf{v}) = (\mathbf{v} - \mathbf{p}) \cdot \mathbf{n} > 0$$

- Beobachtungen:
  1. f ist linear,
  2. $\exists^1 \mathbf{v}^* \ : \ f(\mathbf{v}^*) = \min$
  3. P konvex ⇒ f(x) hat genau ein lokales Minimum über allen Punkten x auf der Oberfläche von P (oder mehrere und diese liegen parallel zu E)



- Algo (minimales **v** bzgl. *f* suchen)
  - starte mit irgendeiner Ecke **v**
  - gehe zu demjenigen Nachbarpunkt **v'** von **v**, für das *f* am kleinsten ist
  - fertig, falls es kein "kleineres" **v'** gibt

Eigenschaften des Algorithmus'

+ Erwartete Laufzeit ist O(1)!
  Der Algo nützt *frame-to-frame coherence* aus:
  wenn sich die Objekte wenig bewegt haben, dann muss man nur überprüfen, dass die separierende Ebene immer noch eine ist; wenn die trennende Ebene ein bisschen verschoben werden muss, dann ist man meistens nach wenigen Iterationen fertig.
+ Funktioniert auch für sich verformende Objekte, solange sie konvex bleiben
– Funktioniert nur für konvexe Objekte
– Terminiert nicht notwendigerweise;
  also bricht man die Schleife nach **`max`** erfolglosen Versuchen ab; dann kann der Algo ein falsches Ergebnis liefern.
- *Frage: Gibt es eine deterministische Variante ?!*

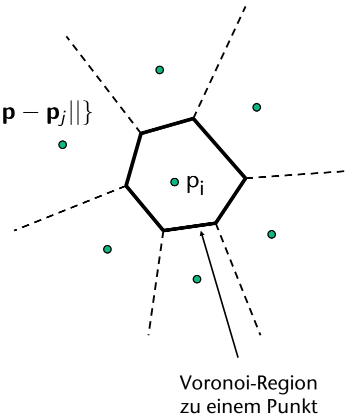
Closest Feature Tracking

- Vorgestellt von Lin & Canny 1992
  (→ Lin-Canny-Algorithmus)
- Idee
  - Verfolge Minimalabstand zwischen beiden Objekten
  - Wird realisiert durch je ein Punkt auf der Oberfläche
  - Bei kontinuierlicher Bewegung der Objekte wandern diese Punkte kontinuierlich über die Oberfläche
- Zugrunde liegende Verfahren
  - Voronoi-Diagramme
  - "closest features"

Voronoi-Diagramme zu Punkten

- Gegeben eine Menge Punkte $S = \{\mathbf{p}_i\}$
- Definition Voronoi-Region :

  $V_i := \{\mathbf{p} \in \mathbb{R}^2 \mid \forall j \neq i : ||\mathbf{p} - \mathbf{p}_i|| < ||\mathbf{p} - \mathbf{p}_j||\}$
- Definition Voronoi-Diagramm :
  Menge aller Voronoi-Regionen
  zu den Punkten in S.
- Partition der Ebene in Kanten und
  Voronoi-Regionen

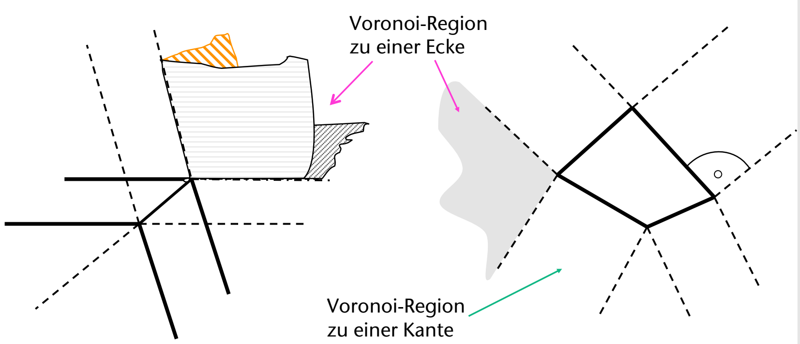


- Interaktive Demo: http://web.cs.uni-bonn.de/I/GeomLab/VoroGlide/

Voronoi-Diagramme zu Polyedern

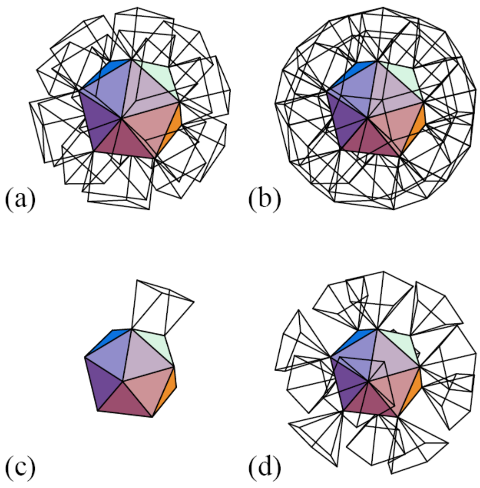
Voronoi-Regionen in 3D

Voronoi-Regionen in 2D



Äußere Voronoi-Regionen sind für konvexe Objekte
sehr einfach zu konstruieren!
(Innere Voronoi-Regionen brauchen wir nicht.)

Äußere Voronoi-Regionen eines Polyeders

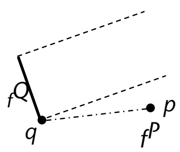


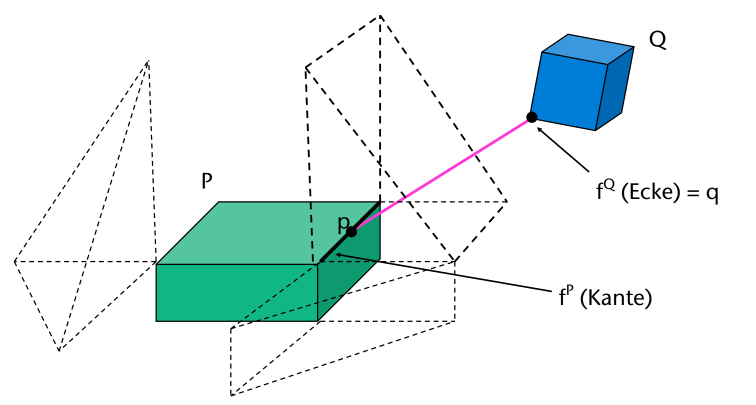
The external Voronoi regions of ...

(a) faces
(b) edges
(c) a single edge
(d) vertices

Closest Features

- Definition *Feature* f^P :=
  Ecke, Kante oder Polygon eines Polyeders P.
- Definition "*Closest Feature*":
  Seien f^P und f^Q zwei Features auf P bzw. Q, und seien p, q Punkte auf f^P bzw. f^Q die den minimalem Abstand von P und Q realisieren, d.h., $d(P,Q) = d\left(f^P, f^Q\right) = ||p - q||$.
  Dann heißen "*closest features*".
- Lemma:
  Sei $V(f)$ die Voronoi-Region zu einem Feature f;
  f^P, f^Q sind "closest features" :⇔
  p liegt in $V(f^Q)$, q liegt in $V(f^P)$.





Algorithmus

starte mit zwei beliebigen Features f^P, f^Q auf P bzw. Q

while (f^P, f^Q) sind noch nicht closest features && d(f^P, f^Q) > 0

    if (f^P,f^Q) wurde schon einmal betrachtet

    return "Kollision" (weil Zyklus)

    bestimme p und q, die den Abstand zwischen f^P,f^Q realisieren

    if $p \in V_q$ und $q \in V_p$

    return "keine Kollision", (f^P,f^Q) sind closest features

    if ex. eine Seite von V_q bzgl. der p auf der falschen Seite liegt

    $f^P \leftarrow$ das Feature der "dahinter" liegenden Voronoi-Region

    analog für q, falls $q \notin V_p$

if d(f^P, f^Q) > 0

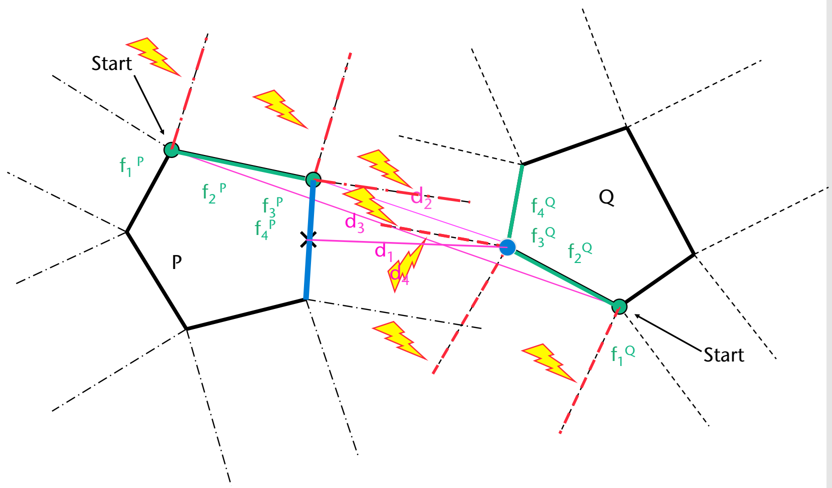
    return "keine Kollision"

else

    return "Kollision"

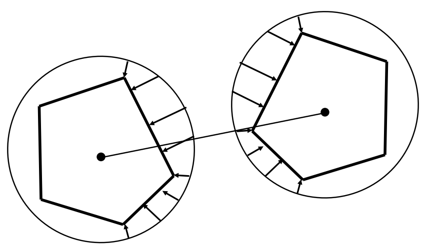
Achtung: bei Kollision befinden sich einige Features im Innern des anderen Objektes, aber im Innern ex. keine Voronoi-Regionen!

Visualisierung des Algorithmus'

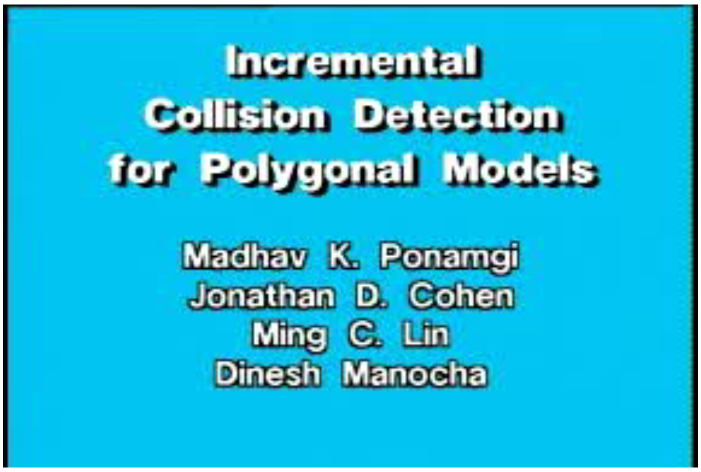


Anmerkungen

- Kleine Denkaufgabe:
  Das *Voronoi-Diagramm* braucht man eigentlich nicht!
  (aber mit *Voronoi-Diagramm* ist der Algo schneller)
- Berechnungsdauer hängt ab vom "Maß" der zeitlichen Kohärenz
- Verbesserung durch *Lookup-Table*:
  trage sphärische Koordinaten der Features
  in Tabelle ein

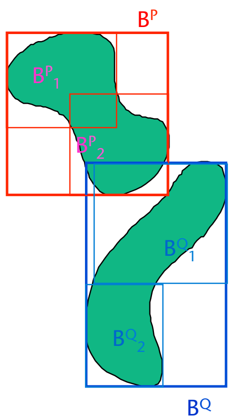


Movie



Hierarchische Kollisionserkennung

- Für *"Polygon soups"*
- Algorithmentechnik:
  Divide & Conquer

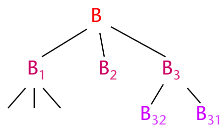


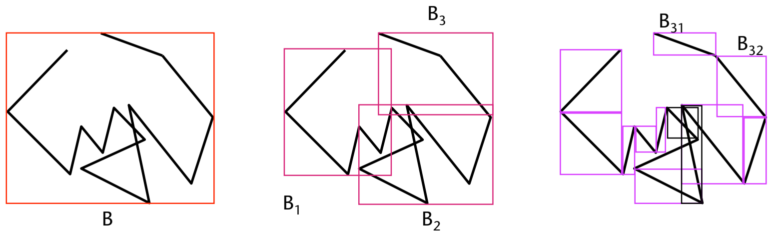
Bounding Volume Hierarchy (BVH)

- Schließe alle Polygone aus P in ein Hüllvolumen (*bounding volume*) BV(P) ein
- Teile P auf in $P_1, P_2, P_3, \ldots, P_n$ mit

  $$P_1 \cup P_2 \cup P_3 \cup \ldots \cup P_n = P$$
- Rekursiv für die P_i.

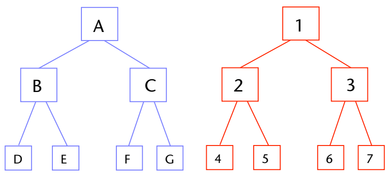
→ *bounding volume hierarchy*

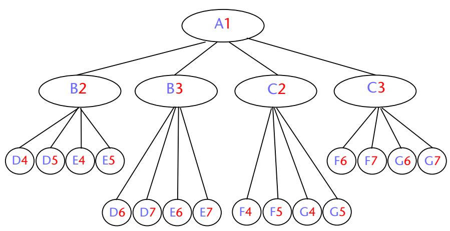




Simultane Traversierung

traverse(X, Y)
if X,Y do not overlap **then**
  return
if X,Y are leaves **then**
  check polygons
else
  for all children pairs **do**
  traverse(X_i, Y_j)

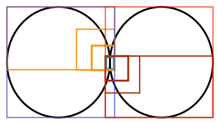


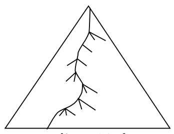


Bounding Volume Test Tree (BVTT)

Einfache Laufzeit-Abschätzung

- *Best-case*: $O(\log n)$





Bounding Volume Test Tree (BVTT)

- Einfache *average-case* Abschätzung:
  - P[*k*] = Wahrsch.keit daß genau *k* Kinderpaare überlappen, $k \in [0,\ldots,4]$

$$P[k] = \frac{1}{16}\binom{4}{k}$$

  - Annahme: alle Ereignisse sind gleich wahrscheinlich
  - Erwartete Laufzeit :

$$T(n) = \tfrac{1}{16}\cdot 0 + \tfrac{4}{16}\cdot T(\tfrac{n}{2}) + \tfrac{6}{16}\cdot 2T(\tfrac{n}{2}) + \tfrac{4}{16}\cdot 3T(\tfrac{n}{2}) + \tfrac{1}{16}\cdot 4T(\tfrac{n}{2})$$

$$T(n) = 2T(\tfrac{n}{2}) \in O(n)$$

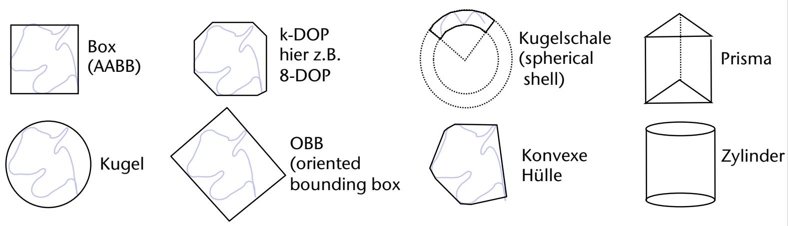
- In der Praxis besser

Bounding Volumes

Anforderungen:

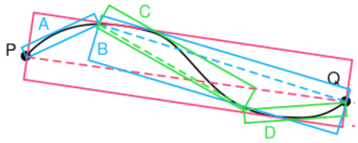
- *sehr* schneller Überlappungstest
- auch dann, wenn die *Bounding Volumes* rotiert oder transl. sind!
  → "einfache" *Bounding Volumes*
- eine Überdeckung des ganzen Raumes sollte möglichst wenig mehrfach belegten Raum haben → "*tight BVs*"

Einige mögliche *Bounding Volumes*:



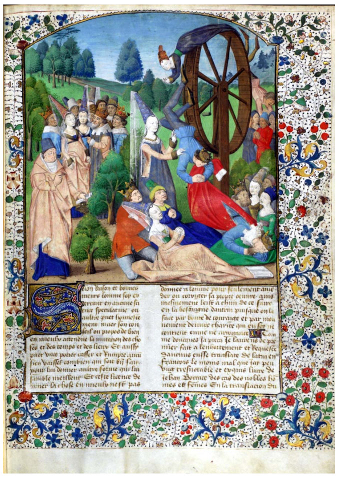
The Wheel of Re-Invention

- OBB-Trees: wurden für 2D-Kurven in 1981 schon von Dana Ballard vorgeschlagen, nannten diese "strip trees"



- AABB-Hierarchien: wurden in den 80er Jahren auch in der Datenbank-Gemeinde erfunden, heißen dort "R-Tree", R*-Tree", "X-Tree", etc.

Exkurs: Das Rad der Fortuna



Boccaccio De Casibus Virorum Illustrium Paris: 1467

Codex Buranus

Schnittest für konvexe BVs (Polyeder)

- Hermann Minkowski (1864 – 1909), deutscher Mathematiker und Physiker
- Definition (*Minkowski-Summe*): Seien *A* und *B* Teilmengen eines Vektorraums; die Minkowski-Summe von *A* und *B* ist

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \,|\, \mathbf{a} \in A,\ \mathbf{b} \in B\}$$

- Entsprechend die *Minkowski-Differenz*:

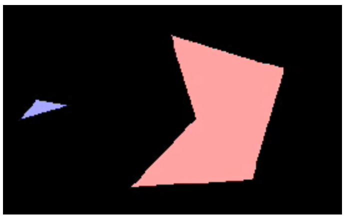
$$A \ominus B = \{\mathbf{a} - \mathbf{b} \,|\, \mathbf{a} \in A,\ \mathbf{b} \in B\}$$

- Zusammenhang zwischen *Minkowski-Summe* und *-Differenz*:

$$A \ominus B = A \oplus (-B)$$

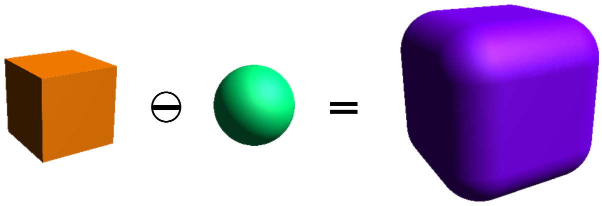
- Anwendungen: Computergraphik, Bildverarbeitung, Lineare Optimierung, Roboter-Pfadplanung, ...

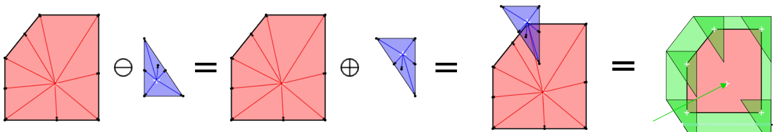
- Intuitive "Berechnung" der Minkowski-Summe:



  - Achtung: das gelbe Polygon zeigt die Minkowsi-Summe **modulo**(!) eventueller Translationen!

- Weitere Beispiele:





$A \ominus B = A \oplus -B = C$

- Dieses Objekt nennt man auch das *Configuration Space Obstacle (CSO)*

Eigenschaften

- *Minkowski-Summen* sind:
  - Kommutativ: $A \oplus B = B \oplus A$
  - Assoziativ: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
  - Distributiv bzgl. Vereinigung: $A \oplus (B \cup C) = (A \cup B) \oplus (A \cup C)$
  - Invariant (in gewissem Sinne) gegenüber Translation: $T(A) \oplus B = T(A \oplus B)$
- Minkowski-Differenz:

$$T(A) \ominus T(B) = A \ominus B$$

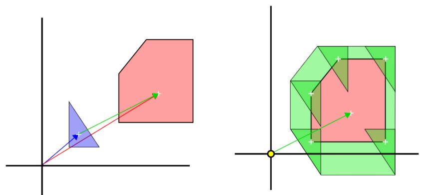
Exact Minkowski Sums of Convex Polyhedra

Efi Fogel and Dan Halperin

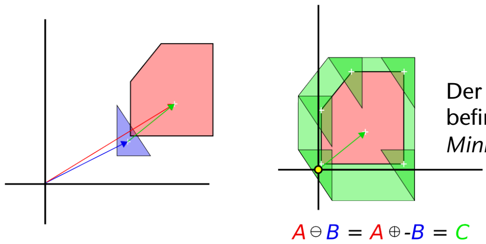
School of Computer Science
Tel Aviv University

Schnitttest für zwei konvexe BVs

- A und B schneiden sich $\Leftrightarrow$ $0 \in A \ominus B$
- Beispiel:



$A \ominus B = A \oplus -B = C$



Der Koordinatenursprung befindet sich in der *Minkowski-Differenz C*

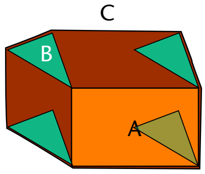
$A \ominus B = A \oplus -B = C$

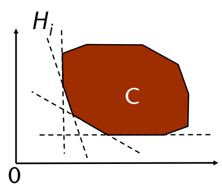
Schnittest für Oriented Bounding Boxes (OBB)

- Lemma "*Separating Axis Test*" (SAT):
  Seien A, B zwei konvexe Polytope (Polyeder).
  Wenn es eine separierende Ebene gibt,
  dann auch eine, die parallel zu einer Seite von A oder B ist,
  oder parallel zu mindestens einer Kante von A und einer von B.
  [Gottschalk, Lin, Manocha; 1996]
- Abwandlung des "*separating plane*" Lemmas
  ("*separating axis*" Lemma):
  Zwei konvexe Polyeder überlappen sich nicht ⇔
  es gibt eine Gerade, so daß die Projektion der beiden
  Objekte auf dieser Geraden sich nicht überlappen.
  Diese Achse heißt "separierende Achse".

Beweis des SAT-Lemmas

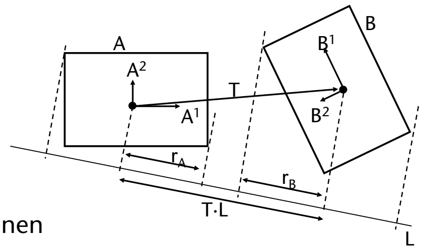
1. Annahme: A und B sind disjunkt
2. Betrachte Minkowski-Summe
3. Alle Faces von C sind entweder parallel zu einem Face von A, oder einem Face von B, oder parallel zu einer Kante von A *und* einer Kante von B
4. C ist konvex
5. $C = \bigcap_{i=1}^{m} H_i$
6. $A \cap B = \varnothing \Leftrightarrow (0,0,0) \notin C$
7. $\exists i : 0 \notin H_i$ (0 liegt außerhalb eines H_i)
8. Es gibt eine separierende Ebene für A und B, die parallel zu diesem H_i ist.





Der SAT für OBBs

- OBdA: rechne im Koord.system von Box A
- Box A definiert durch: C, a^1A^1, a^2A^2, a^3A^3
- Position von B relativ zu A ist definiert durch R & T
- Im Koord.system von A: B^i sind Spalten von R
- Gemäß Lemma müssen wir nur einige spezielle Ebenen betrachten, um die Separierung festzustellen

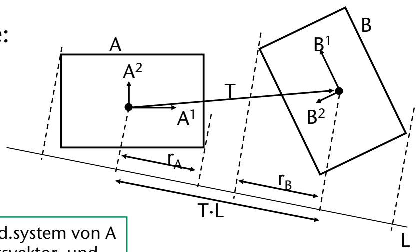


- A,B überlappen, wenn $|T \cdot L| < r_A + r_B$ für jede dieser Ebenen
  - L = Normale der Ebene
- Anzahl solcher "spezieller" Achsen bei Boxes = 15

- Bsp.: $L = A^1 \times B^2$
- Zu berechnen: $r_A = \sum_i a_i |A^i \cdot L|$ (und analog r_B)
- Bsp. 2-ter Term der Summe:

$$
\begin{aligned}
& a_2 A^2 \cdot \left(A^1 x B^2\right) \\
& = a_2 B^2 \cdot \left(A^2 x A^1\right) \\
& = a_2 B^2 \cdot A^3 \\
& = a_2 R_{32}
\end{aligned}
$$

Wir rechnen in Koord.system von A
→ A^3 ist 3-ter Einheitsvektor, und
B^2 ist 2-te Spalte von R



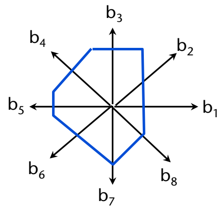
- Für jede der 15 Achsen hat man einen Test der Form

$$|T \cdot L| < a_2|R_{32}| + a_3|R_{22}| + b_1|R_{13}| + b_3|R_{11}|$$

Schnittest für Discretely Oriented Polytopes (k-DOPs)

- Definition:
  Wähle k Vektoren $\mathbf{b}_i \in \mathbb{R}^3$ fest, k gerade,
  mit b_i antiparallel zu $b_{i+k/2}$.
  k-DOPs sind als Volumen beschrieben durch

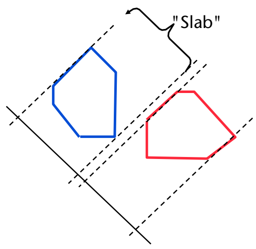
$$D = \bigcap_{i=1..k} H_i \quad , \quad H_i : \mathbf{b}_i \cdot x - d_i \leq 0$$



- Beschreibung eines k-DOP: $D = (d_1 ... d_k) \in \mathbb{R}^k$
- Überlappungstest:

$$D^1 \cap D^2 = \varnothing \Leftrightarrow$$

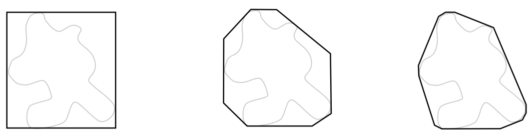
$$\forall i = 1, .., \frac{k}{2} : \left[d_i^1, d_{i+\frac{k}{2}}^1\right] \cap \left[d_i^2, d_{i+\frac{k}{2}}^2\right] = \varnothing$$



→ $k/2$ Intervall-Tests

Eigenschaften

- AABBs sind spezielle DOPs
- Überlappungstest $\in O(k)$, k = Anzahl Orientierungen
- Beliebig genaue Approximation der konvexen Hülle



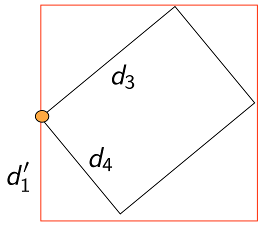
k=4 k=8 k=12

Overlap test of DOPs

- Algorithmus für "schiefe" DOPs:
  - Objektbewegung: Rotation R & Translation T
  - Neuer DOP nach affiner Transformation des Objektes:

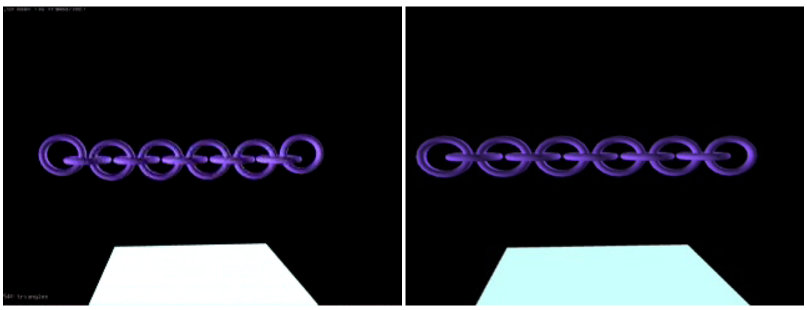
$$d'_i = \boxed{\mathbf{B}_i \begin{pmatrix} \mathbf{b}_{j'_1} \\ \mathbf{b}_{j'_1} \\ \mathbf{b}_{j'_1} \end{pmatrix}^{-1}} \begin{pmatrix} d_{j'_1} \\ d_{j'_1} \\ d_{j'_1} \end{pmatrix} + \boxed{\mathbf{B}_i T},$$

$$\mathbf{b}_j = \mathbf{B}_i R^{-1}$$



  - Korrespondenz j^i_l identisch für alle DOPs einer Hierarchie
  - Aufwand: O(k), früher O(k[2])

Performance-Vergleich zwischen AABB und DOP



24-DOPs AABBs

Fazit: die Geschwindigkeit der Kollisionsdetektion bestimmt die Geschwindigkeit der Simulation ...