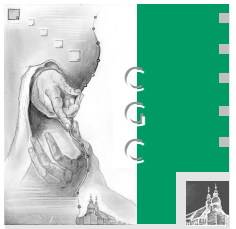


Virtuelle Realität
Kollisionserkennung

G. Zachmann

Clausthal University, Germany

cg.in.tu-clausthal.de

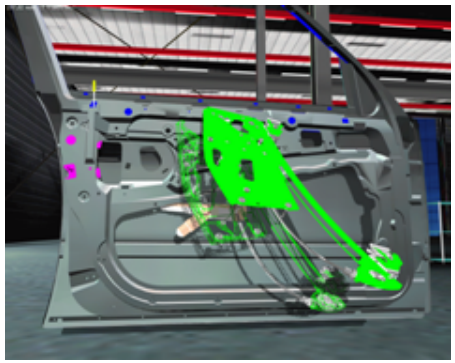
Anwendungsbeispiele

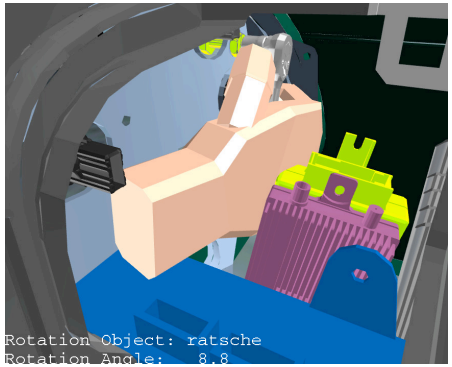
Virtual Prototyping

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

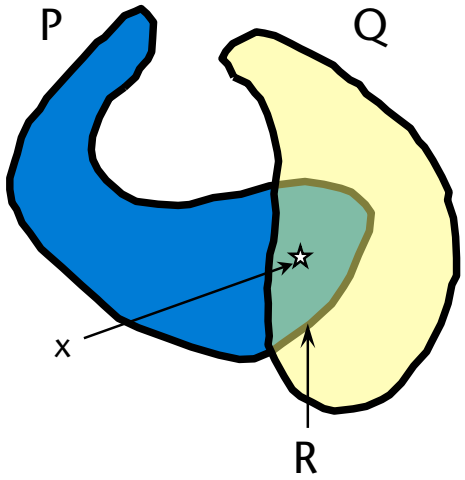
- Kollisionen stellen zwei Probleme:
  - Kollisionserkennung
  - Kollisionsbehandlung
- Im folgenden: Kollisionserkennung

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:

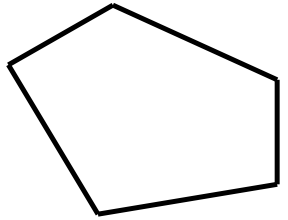
  P,Q kollidieren $\Leftrightarrow$

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

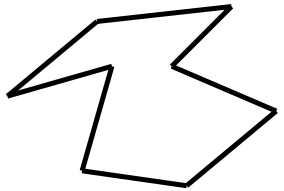
- Andere Definition in der Spielebranche

Objekt-Klassen

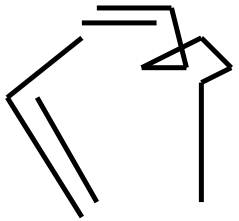
- konvex
- geschlossen, einfach
  (keine Selbstdurchdringung)
- Sack Polygone (“*polygon soup*”)
  - nicht notwendig geschlossen
  - doppelte Polygone
  - koplanare Polygone
  - Selbstdurchdringungen
  - degenerierte Polygone
  - Löcher
- starr / flexibel



konvex



einfach & geschlossen

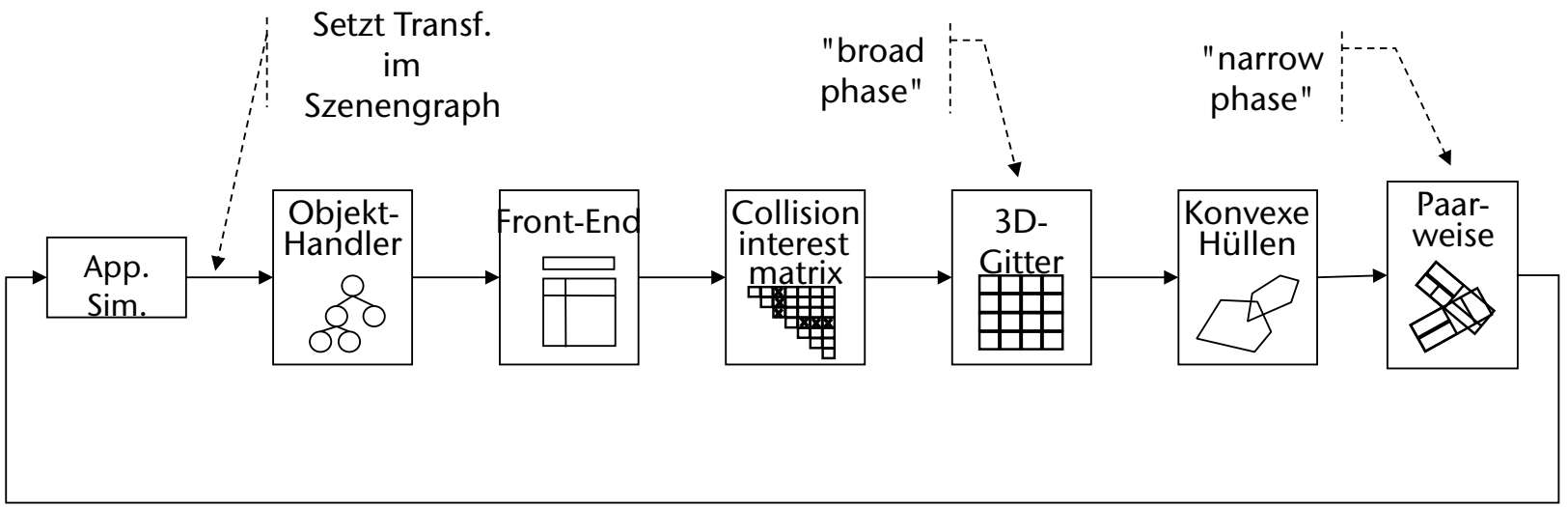


polygon soup

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.

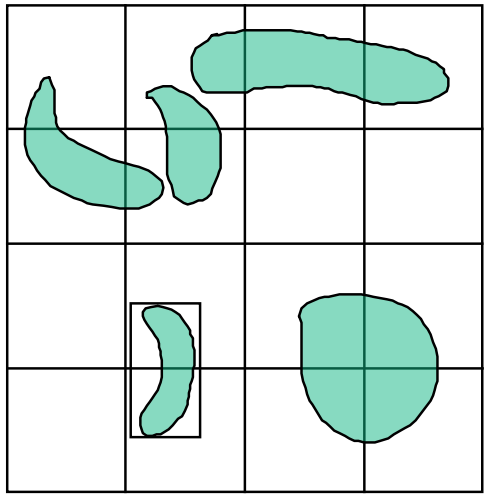
- Idee:
  versuche zum Objekt P schnell die "Nachbarn" zu finden und mache nur mit diesen Bbox-Vergleiche

→Raum-Gitter, *Sweep-Plane*, etc.

Raum-Gitter

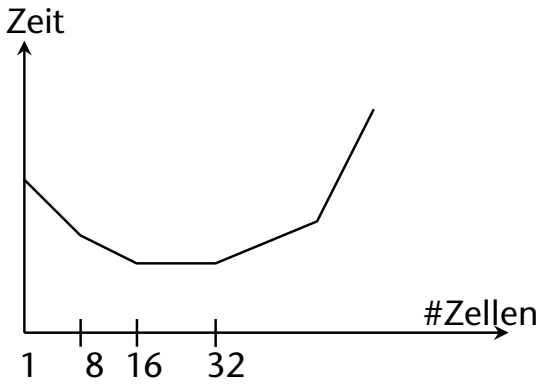
Idee:

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.



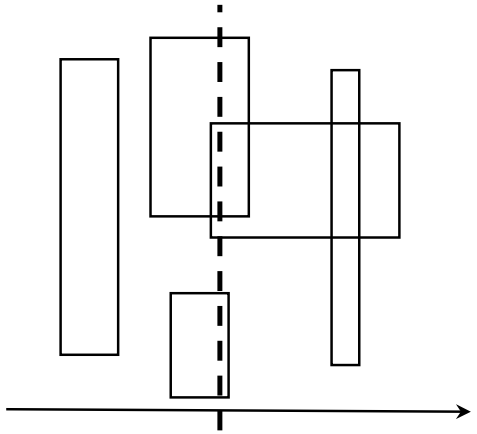
Trade-Off:

- 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)

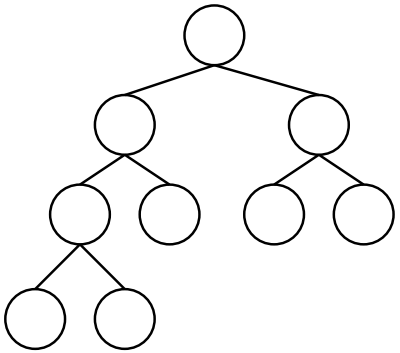
  else

    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”

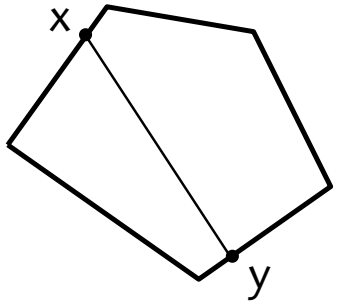
Konvexe Objekte

- Definition “konvexer Polyeder”:

$$P \subset \mathbb{R}^3 \text{ konvex} \Leftrightarrow$$

$$\forall x, y \in P : \overline{xy} \subset P \Leftrightarrow$$

$$P = \bigcap_{i=1,\cdots,n} H_i \quad , H_i = \text{Halbräume}$$



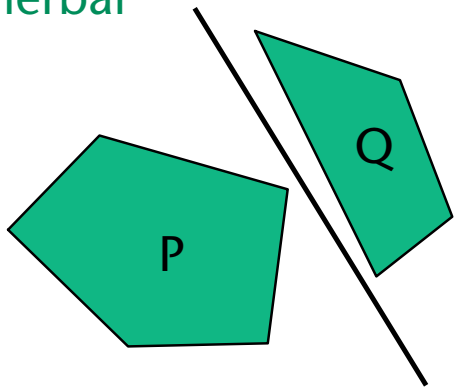
- Bedingung für Nicht-Kollision: “linear separierbar”

$$P \cap Q = \bigcap_{i=1}^{n1+n2} H_i = \varnothing \Leftrightarrow$$

$$\exists i : P \subseteq H_i \wedge Q \subseteq H_i^c$$

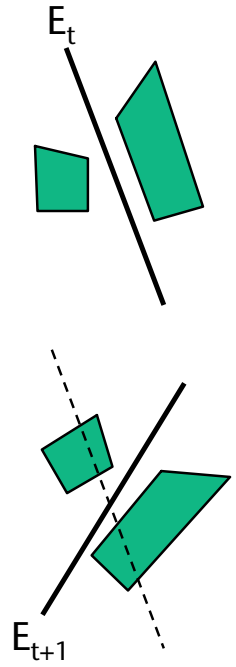
(“P liegt ganz auf der einen Seite von H_i,

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

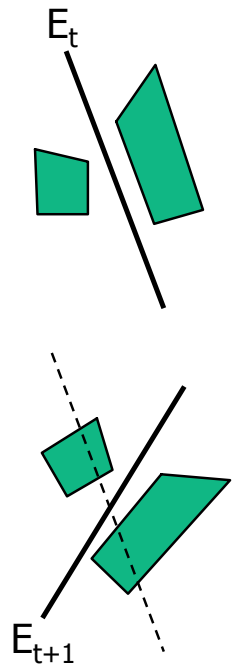
  if kein $v \in V^P$, kein $v \in V^Q$ auf der "falschen" Seite

    return "keine Kollision"

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

  return "Kollision" {kann manchmal falsch sein}

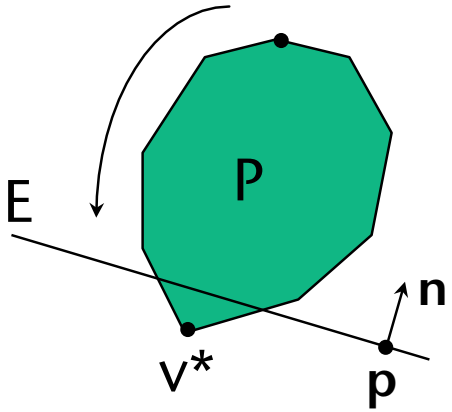
speichere $E_{t+1} := E$ für's nächste Frame



Finde schnell eine Ecke auf der “falschen” Seite

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

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



- Beobachtungen:
  1. f ist linear,
  2. $\exists^1 \mathbf{v}^* \; : \; f(\mathbf{v}^*) = \min$
  3. P konvex $\Rightarrow$ f(x) hat genau ein lokales Minimum über allen Punkten x auf der Oberfläche von P

- Algo (minimales **v** bzgl. *f* suchen)
  - starte mit irgendeiner Ecke **v**
  - gehe zu dem Nachbarpunkt **v’** von **v**, für das $f(\mathbf{v}') = f(\mathbf{v})$
  - 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

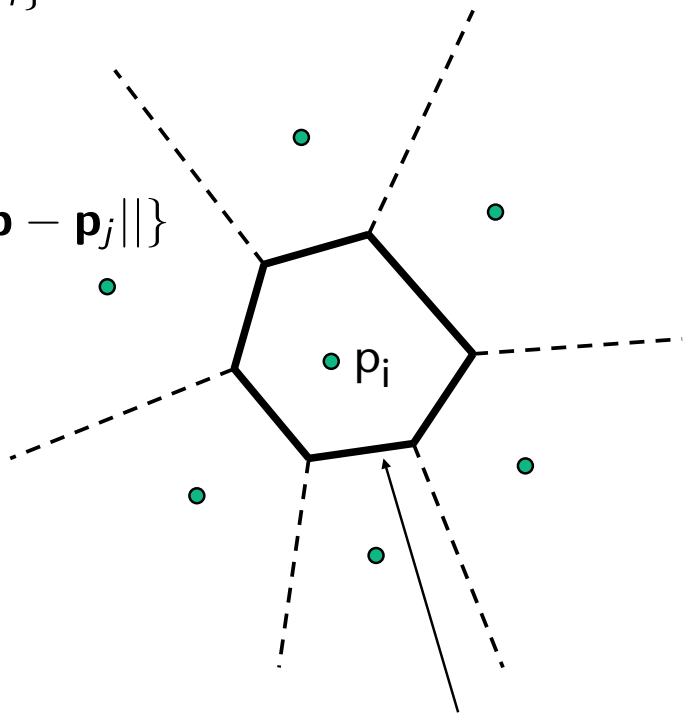
- Vorgestellt von Lin & Canny 1992
  (→ Lin-Canny-Algorithmus)
- Idee
  - Verfolge Minimalabstand zwischen beiden Objekten
  - Wird realisiert durch je ein Punkt auf der Oberfläche
  - Bei kontinuierlicher Bewegung der Objekte wandern diese Punkte kontinuierlich über die Oberfläche
- Zugrunde liegende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



Voronoi-Region zu einem Punkt

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