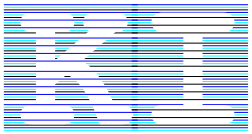
Game Physik
Simulating Physics

Dipl. Physiker
Dr. Christian Herta

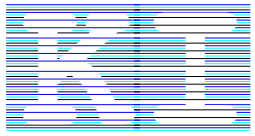
23. November 2004



Motivation

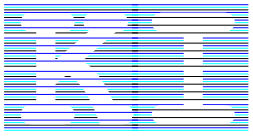
Warum Game Physik?

- Realistisches Verhalten *look and feel*: Aktion und Reaktion
- Simulations Spiele brauchen Game Physik: z.B. Autorennen, Flugsimulatoren
- Charakter Physik
- Simulationen von Ketten, Seilen, Kleidung, Stoffen, Flüssigkeiten etc.
  (in der Zukunft: Haare, Muskeln, Haut ?)
- Eigenschaften von Objekten müssen nicht detailiert programmiert werden.
  Nur Erzeugen eines Objektes plus Antrieb, den Rest macht die Physik.



Anwendungen

- Character Physics and Dynamics: Bewegungen von Avataren
- Vehicle Dynamics: Autorennen etc.
- Environmental Dynamics
- Sport Simulationen: Fussball, Ski-Rennen ..

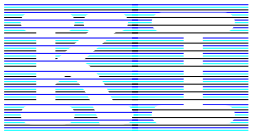


Klassische Mechanik

Game Physik - klassische Mechanik

Zwei Arten von Game Physik:

- Physik starrer Körper ($rigid\ body\ physics$) nicht deformierbarer Körper
- Nicht elastische Physik mit Deformationen



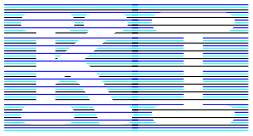
Physik starrer Körper *Rigid Body Mechanics*

Zwei prinzipielle Betrachtungsweisen werden verwendet:

- Newtonsche Mechanik
- Lagrangesche Mechanik

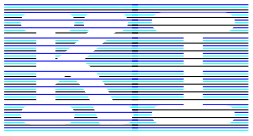
(Bewegungs-)Gleichungen beschreiben die Physik

- Aufstellen der (Differential-)Gleichungen für ein System
- Anfangs- und Randbedingungen
- Lösen der Gleichungen



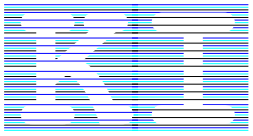
Effizienz statt Exaktheit

- Kompromiss zwischen Rechenzeit und Rechengenauigkeit.
- Game Physik braucht nicht so exakt zu sein, wie eine Ingenieur-Simulation.
- Sie soll vor allem schnell sein und realistisch wirken.



Physik

- Unconstrained Motion: Freie Bewegung der Körper
- Constrained Motion: Körper dürfen sich nicht gegenseitig durchdringen



Ungebundene Bewegung

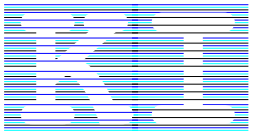
(unconstrained motion)

Keine Einschränkung der Bewegungsfreiheit der Körper.

Keine Wechselwirkung zwischen den Körpern.

Zur Beschreibung geeignet: Newtonsche Mechanik.

Kräfte und Drehmomente wirken auf die Körper und verändern deren Zustände (Lage im Raum, Geschwindigkeit).



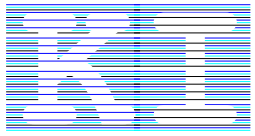
Bewegung eines Massenpunktes

Die freie Bewegung eines Massepunktes wird mit dem 2. Newtonschen Gesetz beschrieben:

$$\vec{F}(t) = d\vec{p}/dt = d(m * \vec{v})/dt = dm/dt * \vec{v} + m * d\vec{v}/dt = \dot{m}\vec{v} + m\dot{\vec{v}} \qquad (1)$$

$\vec{p} = \vec{p}(t) = m * \vec{v}$: Impuls des Massepunktes
m: Masse
$\vec{F}(t)$: zeitabhängige Kraft
$\vec{r} = \vec{r}(t)$: Ortsvektor
$\vec{v} = \vec{v}(t) = d\vec{r}/dt = \dot{\vec{r}}$: Geschwindigkeit

Der Punkt über den Größen ist eine Abkürzung für die Ableitung nach der Zeit.



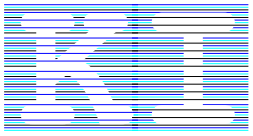
Bewegung eines Massenpunktes II

Bei konstanter Masse m reduziert sich das Newtonsche Gesetz auf

$$\vec{F}(t) = m\dot{\vec{v}}(t) = m\vec{a}(t) = m\ddot{\vec{r}}(t) \tag{2}$$

mit der Beschleunigung $\vec{a} = \vec{a}(t) = d\vec{v}/dt = d^2\vec{r}/dt^2 = \ddot{\vec{r}}$

Gleichung 2 ist eine Differentialgleichung 2.ter Ordnung in $\vec{r}$
(2. Ableitung nach der Zeit t).



Äquivalenz zu Gleichungen 1. Ordnung

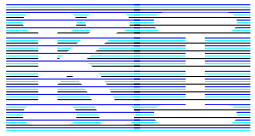
Numerische Differentialgleichungs-Solver (DLG-Solver) arbeiten meist mit Systemen von Differtialgleichungen 1.Ordung.

Eine Differentialgleichungen der n-ten Ordung lässt sich in ein System von n Differentialgleichungen der 1. Ordnung überführen.

Das 2. Newtonsche Gesetz (Gl. 2) ist äquivalent zu den beiden Differentialgleichungen 1. Ordung:

$$\dot{\vec{r}} = \vec{v} \tag{3}$$

$$\dot{\vec{v}} = \vec{F}/m \tag{4}$$



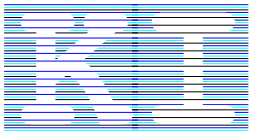
Zustandsvektor

$\vec{r}(t)$ und $\vec{v}(t)$ lässt sich zum Zustandvektor ($state\ vector$) $\vec{S}$ zusammenfassen:

$$\vec{S}(t) = \begin{pmatrix} \vec{r}(t) \\ \vec{v}(t) \end{pmatrix} \tag{5}$$

Damit lassen sich die beiden Gleichungen 3 und 4 schreiben als:

$$\frac{d\vec{S}}{dt} = \frac{d}{dt}\begin{pmatrix} \vec{r} \\ \vec{v} \end{pmatrix} = \begin{pmatrix} \dot{\vec{r}} \\ \dot{\vec{v}} \end{pmatrix} = \begin{pmatrix} \vec{v} \\ \vec{F}/m \end{pmatrix} \tag{6}$$



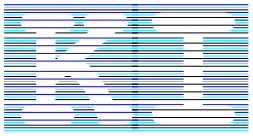
Verallgemeinerung auf n Teilchen

Für n Teilchen lautet der Zustandsvektor: $S = [\vec{r}_1 \vec{v}_1 ... \vec{r}_n \vec{v}_n]$

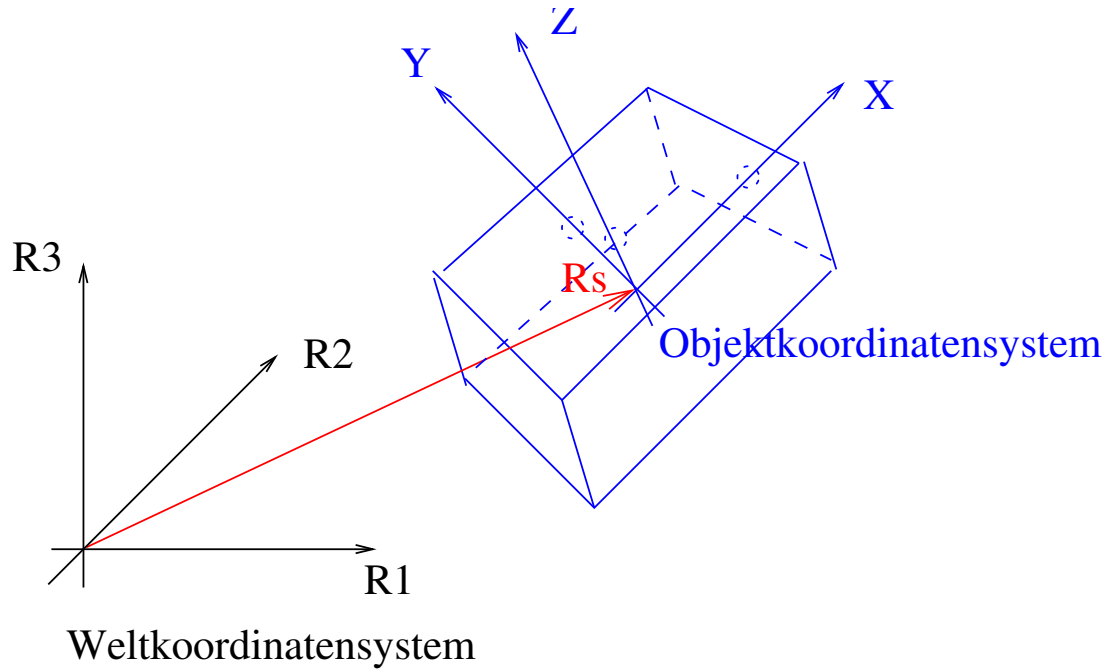
Damit lässt sich Gleichung 6 für n Teilen (1,2...n) verallgemeinern:

$$\frac{d\vec{S}}{dt} = \frac{d}{dt} \begin{pmatrix} \vec{r}_1 \\ \vec{v}_1 \\ \vec{r}_2 \\ \vec{v}_2 \\ . \\ . \\ \vec{r}_n \\ \vec{v}_n \end{pmatrix} = \begin{pmatrix} \dot{\vec{r}}_1 \\ \dot{\vec{v}}_1 \\ \dot{\vec{r}}_2 \\ \dot{\vec{v}}_2 \\ . \\ . \\ \dot{\vec{r}}_n \\ \dot{\vec{v}}_n \end{pmatrix} = \begin{pmatrix} \vec{v}_1 \\ \vec{F}_1/m_1 \\ \vec{v}_2 \\ \vec{F}_2/m_2 \\ . \\ . \\ \vec{v}_n \\ \vec{F}_n/m_n \end{pmatrix} \tag{7}$$

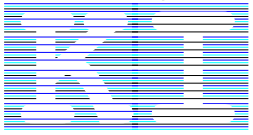
Beachte: Keine Wechselwirkung zwischen den Teilchen.



Koordinatensysteme



Konvertion der verwendeten Koordinatensysteme:
Weltkoordinatensystem: $\vec{r}$
Objektkoordinatensystem (Ursprung im Schwerpunkt): $\vec{x}$



Starrer Körper *(rigid body)*

Reale Körper haben eine Ausdehnung. D.h. zusätzlich zur Translationsbewegung können sie rotieren.
Freie Körper rotieren um ihren Schwerpunkt.

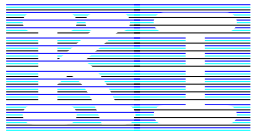
Um die Rotation in Gleichungen beschreiben zu können, benutzt man, analog zur Masse für die Translation, das Trägheitsmoment $\hat{J}$ für die Rotation.

$\hat{J}$ ist ein Tensor 2. Stufe und kann somit durch eine Matrix ausgedrückt werden.

Definition von $\hat{J}$:

$$\hat{J} = \int_K \left(|\vec{r}|^2 \hat{I} - \vec{r}\vec{r}^T \right) dm \tag{8}$$

mit der Einheitsmatrix I und dem Spaltenvektor $\vec{r} = (r_1, r_2, r_3)^T$, d.h. $\vec{r}\vec{r}^T$ ist eine Matrix.



Schwerpunktsbewegung

Man zerlegt die Bewegungsgleichungen in eine Schwerpunktsbewegung und eine Rotationsbewegung.

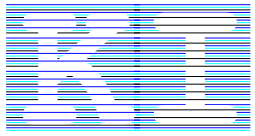
Es gilt das 2. Newtonsche Gesetz für die Schwerpunktsbewegung:

$$\vec{F}(t) = m_{ges}\vec{a_S}(t) = m\dot{\vec{v}}_S(t) = m\ddot{\vec{r}}_S(t) \tag{9}$$

Der Index S macht deutlich, dass sich die Größen auf den Schwerpunkt beziehen.

Der Schwerpunkt($center\ of\ mass$) ist dabei definiert durch

$$\vec{r}_S = \frac{\int_K \vec{r}\rho(\vec{r})dr^3}{m_{ges}} \tag{10}$$



Rotationsbewegung

Die Drehmomente verändern die Drehung um den Schwerpunkt. Mit der Definition des Drehimpuls $L(t)$:

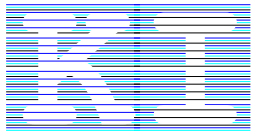
$$\vec{L}(t) = \hat{J}(t)\vec{\omega}(t) \tag{11}$$

$\hat{J}(t)$: Zeitabhängige Trägheitstensor (durch die Drehung ändert sich seine Form im Weltkoordinatensystem)
$\vec{\omega}$: Winkelgeschwindigkeit

Gilt analog dem Newtonschen Gesetz:

$$\frac{d\vec{L}(t)}{dt} = \dot{\vec{L}}(t) = \dot{\hat{J}}(t)\vec{\omega}(t) + \hat{J}(t)\dot{\vec{\omega}}(t) = \vec{M}(t) \tag{12}$$

$\vec{M} := \vec{x} \times \vec{F}$: Drehmoment

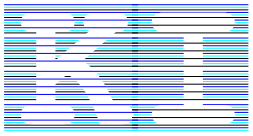


Gebundene Bewegung - Kollisionen

Körper sollen sich im Normalfall nicht durchdringen können. Daher muss auf Kollision von Objekten so reagiert werden, dass dies der Fall ist.

Verbreitet:
Einführung von *impulsiven* Kräften, die beim Kontakt gesetzt werden. Diese verhindern ein Durchdringen der Körper.

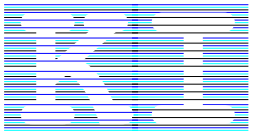
Aber auch andere Ansätze sind möglich, wie z.B. die Langragesche Mechanik.



Kontakte zwischen zwei Körpern

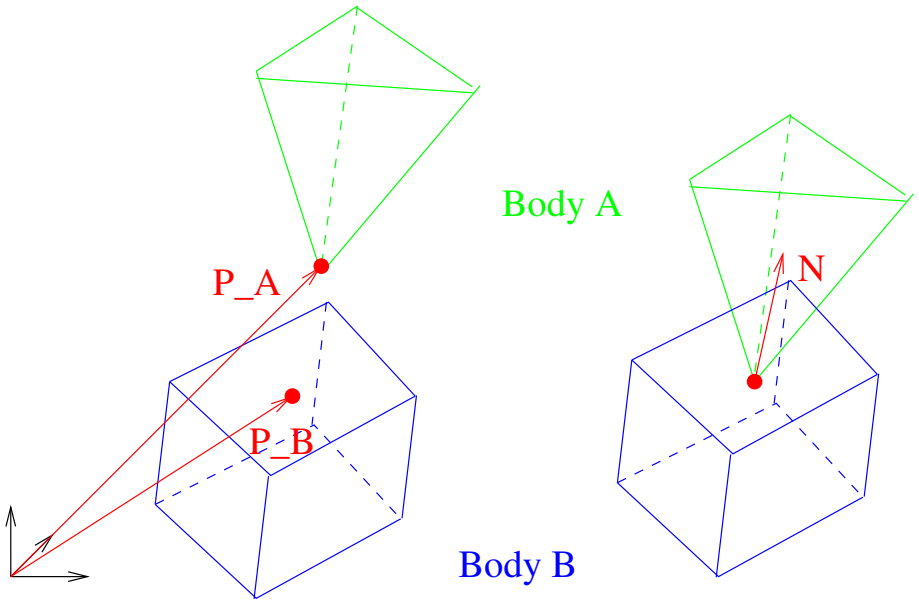
Es gibt zwei Arten von Kontakten zwischen zwei Körpern.

- Colliding Contact: Die relative Geschwindigkeit zwischen zwei Körpern bewirkt, dass sie sich durchdringen würden, wenn man sie frei bewegen lässt.
- Resting Contact: Die beiden Körper berühren sich, aber sie würden sich - aufgrund ihrer Geschwindigkeiten - weder durchdringen noch voneinander entfernen.

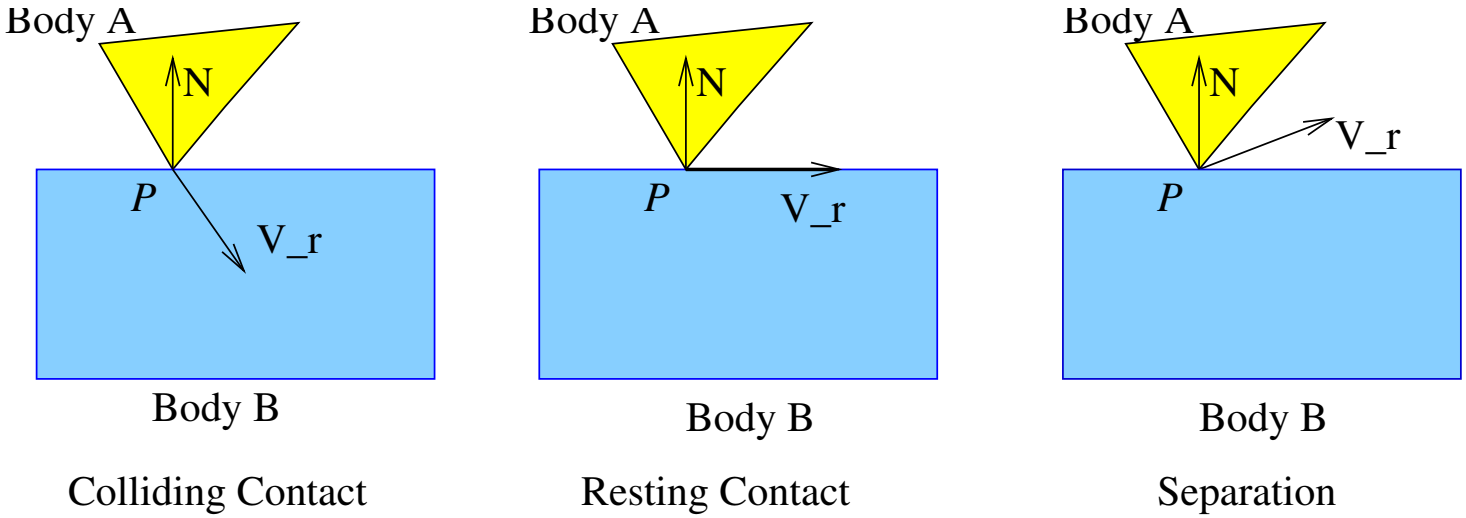


Einfacher Kontakt

Die beiden Körper A und B berühren sich beim Kontakt in einem Punkt. Die beiden Punkte auf den Körpern, die sich beim Kontakt berühren, werden im Folgenden mit $\vec{P}_A$ und $\vec{P}_B$ bezeichnet.

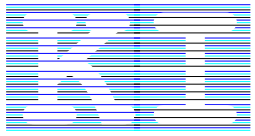


Unterscheidung zwischen Kontakten



$\vec{N} * \vec{V}_r < 0$ Colliding Contact
$\vec{N} * \vec{V}_r = 0$ Resting Contact
$\vec{N} * \vec{V}_r > 0$ Separation

$\vec{N}$: Normale der Kontakt-Fläche von Körper B
$\vec{V}_r = \dot{\vec{P}}_A - \dot{\vec{P}}_B$: relative Geschwindigkeit zwischen den Punkten $\vec{P}_A$ und $\vec{P}_B$

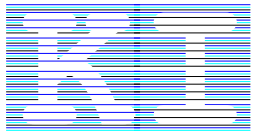


Colliding Contacts - Geschwindigkeitsänderung beim Kontakt

Man will verhindern, dass sich die beiden Körper durchdringen. Dies erreicht man indem man Geschwindigkeiten und Winkelgeschwindigkeiten bei der Kontaktzeit t_c ändert.

Wie ändert man die (Winkel-)Geschwindigkeiten in Anhängigkeiten von den den Massen, Trägheitstensoren, (Winkel-)Geschwindigkeiten, Lage der Kontaktpunkte auf den Körpern und der Elastizität?

Antwort: Einführung von impulsiven Kräften



Impulsive Kräfte

Die Änderung der Geschwindigkeiten entspricht dem Anwenden einer Kraft bei t_c. Durch Integration von Gl. 1 erhält man:

$$\vec{p}_A(t_1) - \vec{p}_A(t_0) = \int_{t_0}^{t_1} \vec{F}_A(\tau) d\tau \tag{13}$$

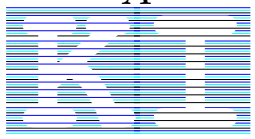
Führt man eine Kontakt-Kraft ($contact\ force$)

$$F_A(t) = (f\vec{N}\delta(t - t_c)) \tag{14}$$

ein, erreicht man eine geforderte Geschwindigkeitsänderung bei t_c:

$$\vec{v_A}(t_1) - \vec{v_A}(t_0) = m_A^{-1} \int_{t_0}^{t_1} (f\vec{N}\delta(\tau - t_c)) d\tau = m_A^{-1} f\vec{N} \tag{15}$$

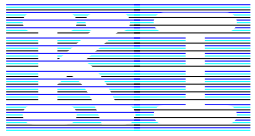
mit $t_0 < t_c < t_1$
$\vec{v_A}$: Geschwindigkeit des Schwerpunktes von A



Änderung der Winkelgeschwindigkeit

Die impulsive Kraft bewirkt ein Drehmoment $\delta(t-t_c)(\vec{r}_A \times f\vec{N})$ und somit eine Änderung der Winkelgeschwindigkeit:

$$\omega_A(t_1) - \omega_A(t_0) = \hat{J}_A^{-1}(\vec{r}_A \times f\vec{N}) \tag{16}$$



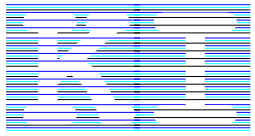
Elastischer Stoß - Unelastischer Stoß

Ein Massenpunkt stößt zur Zeit t_c an eine unbewegliche Wand mit der Normalen $\vec{N}$:

Ist die Geschwindigkeit parallel zur Normalen vor dem Stoß $\vec{v}(t_0)\vec{N}$ gleich $-\vec{v}(t_1)\vec{N}$ (Geschwindigkeit nach dem Stoß parallel zur Normalen), so ist der Stoß vollelastisch. Es wurde keine Energie beim Stoß absorbiert.

Wie elastisch ein Stoß ist hängt von der Stoßzahl (*coefficient of restitution*) ϵ ab.
$0 \leq \epsilon \leq 1$
$\epsilon = 0$: völlig unelastischer Stoß: $v(t_1)\vec{N} = 0$
$\epsilon = 1$: völlig elastischer Stoß

Geschwindigkeit nach dem Stoß: $-\epsilon\vec{v}(t_1)\vec{N}$

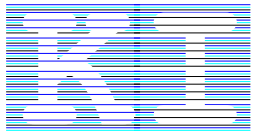


Impulsive Kräfte: Konstante f

Setzt man Gl. 15 und 16 in die Geschwindigkeitsänderung der Punkte (z.B.: $\dot{\vec{P}}_A = \vec{v}_A + \vec{\omega}_A \times \vec{x}_A$) ein, ergibt sich für die Konstante f nach längerer Rechnung (siehe z.B. [1]):

$$f = \frac{-(1+\epsilon)(\vec{N}(\vec{v}_A(t_0) - \vec{v}_B(t_0)) + (\vec{\omega}_A(t_0)(\vec{r}_A \times \vec{N}) - (\vec{\omega}_B(t_0)(\vec{r}_B \times \vec{N})))}{m_A^{-1} + m_B^{-1} + (\vec{r}_A \times \vec{N})^T \hat{J}_A^{-1}(\vec{r}_A \times \vec{N}) + (\vec{r}_B \times \vec{N})^T \hat{J}_B^{-1}(\vec{r}_B \times \vec{N})} \tag{17}$$

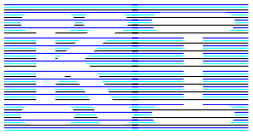
t_0 ist dabei die Zeit direkt vor dem Stoß.
Mit Hilfe der Gleichung 17 lässt sich die Konstante f und so die Kontakt-Kraft beim Stoß (Gl. 14) ausrechnen. Über Gl. 15 und 16 kann man somit die Geschwindigkeitsänderung und die Winkelgeschwindigkeitsänderungen der Körper A und B berechnen.



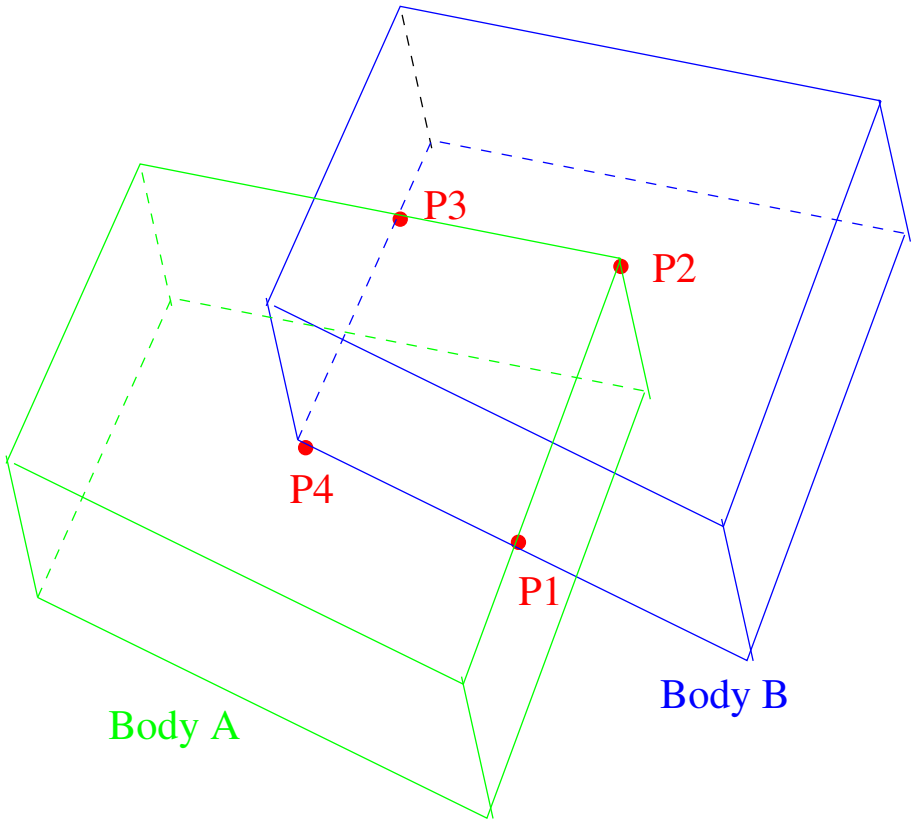
Reduzierte Kontaktmenge

(reduced contact set)

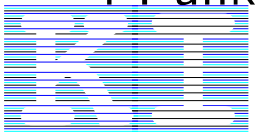
Es gibt im Allgemeinen nicht nur einen Kontakt-Punkt, sondern Kontakt-Flächen bzw. -Kanten, d.h. unendlich viele Kontakt-Punkte. Um Rechenzeit zu sparen arbeitet man mit der Näherung der reduzierten Kontaktmengen.



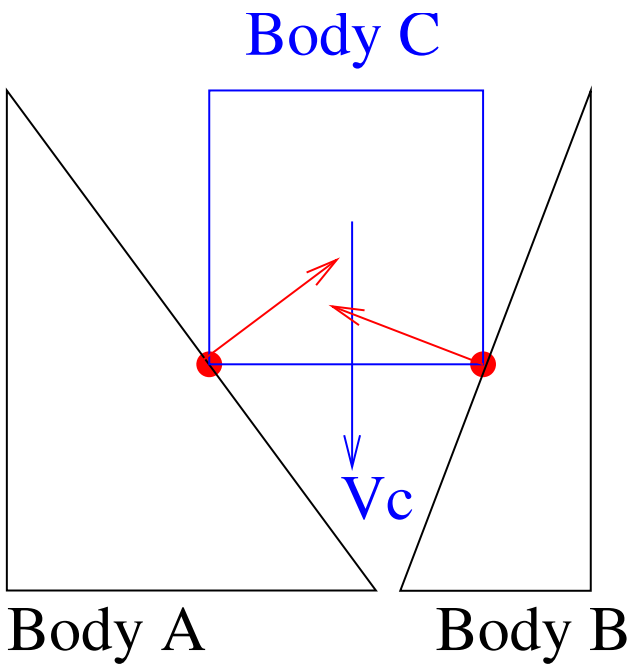
Beispiel eines *Reduced Contact Set*



Die beiden Körper berühren sich in einer Kontaktfläche. Der Kontakt wird mit 4 Punkten beschrieben.



Probleme bei mehreren Kontaktpunkten



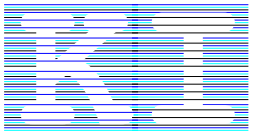
Probleme bei mehreren Kontaktpunkten

Wie sollen mehrere Kontaktpunkte behandelt werden?

Gleichzeitig in einem Zeitschritt des DLG-Solvers?

Hintereinander in unterschiedlichen Zeitschritt des DLG-Solvers?

Es ist i.A. nicht gewährleistet, dass die Körper nicht ineinander dringen.

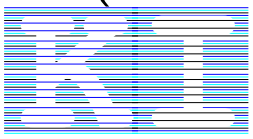


Simultane Behandlung von mehreren Kontaktpunkten

Das System muss garantieren, dass die Geschwindigkeiten und Beschleunigungen der Körper nicht dazu führen, dass sich die Körper durchdringen.

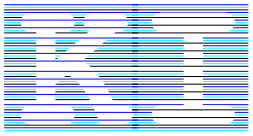
Die simultane Behandlung von mehreren Kontaktpunkten führt auf ein *convex quadratic programming problem*.
Siehe z.B. [1]:
arg min$_f|\hat{A}\vec{f} + \vec{b}|^2$
Nebenbedingungen:
$\vec{f} \geq \vec{0}$ (Kräfte sollen abstoßend sein)
$\hat{A}\vec{f} + \vec{b} \geq \vec{0}$ (kein Durchdringen der Körper aufgrund ihrer Geschwindigkeiten nach dem Stoß)
$\hat{A}\vec{f} + \vec{b} \leq \vec{c}$ (kein Gewinn kinetischer Energie beim Stoß)

Lösen des dualen Problems mit einem Linear Constraint Programming Solver (LCP-Solver).
(meist mit dem Lemke-Howson-Algorithmus)



Collision Response for Resting Contact

Relative Geschwindigkeit ist Null (Definition des resting Contacts). Aber die Beschleunigung kann dazu führen, dass sich die Körper durchdringen werden.
Überprüfen ob dies der Fall sein wird, d.h. berechnen der rel. Beschleunigung der ruhenden Kontaktpunkte.
Man hat als weitere Nebenbedingung für das *convex quadratic programming problem*, dass die rel. Beschleunigung größer gleich Null sein soll.



Numerische Lösung von Differentialgleichungen

Die Physik wird durch Differentialgleichungen beschrieben.

Startwert-Problem (*initial value problem*)

Zur Lösung der Differentialgleichungen wird die Zeit in diskrete Zeitschritte Δt eingeteilt.

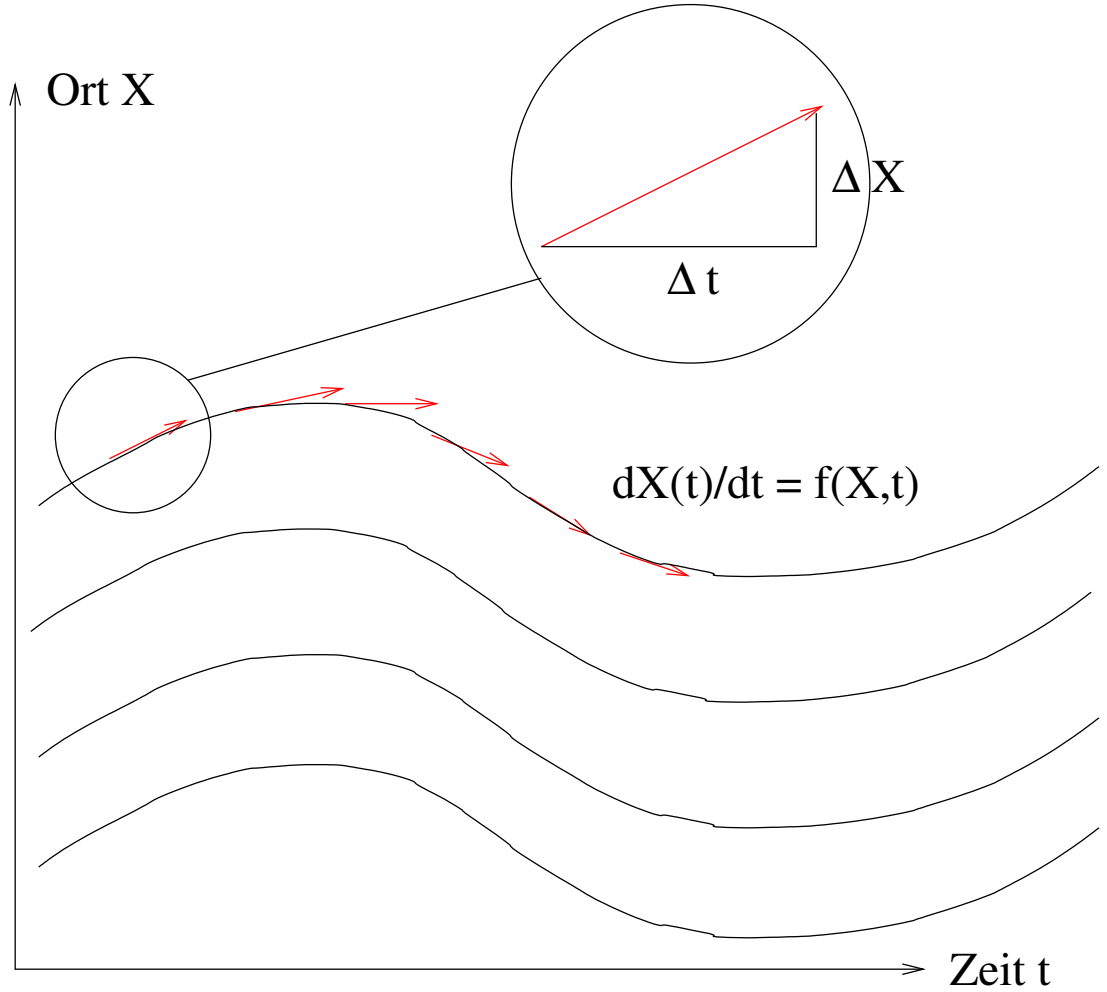
Schrittweite (*step size*) Δt beeinflusst die Ergebnisse der Simulation:

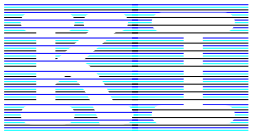
- Genauigkeit (*accuracy*)
- Stabilität (*stability*)

Genauigkeit ist nicht so wichtig für Game Physik, aber die Stabilität ist wichtig!

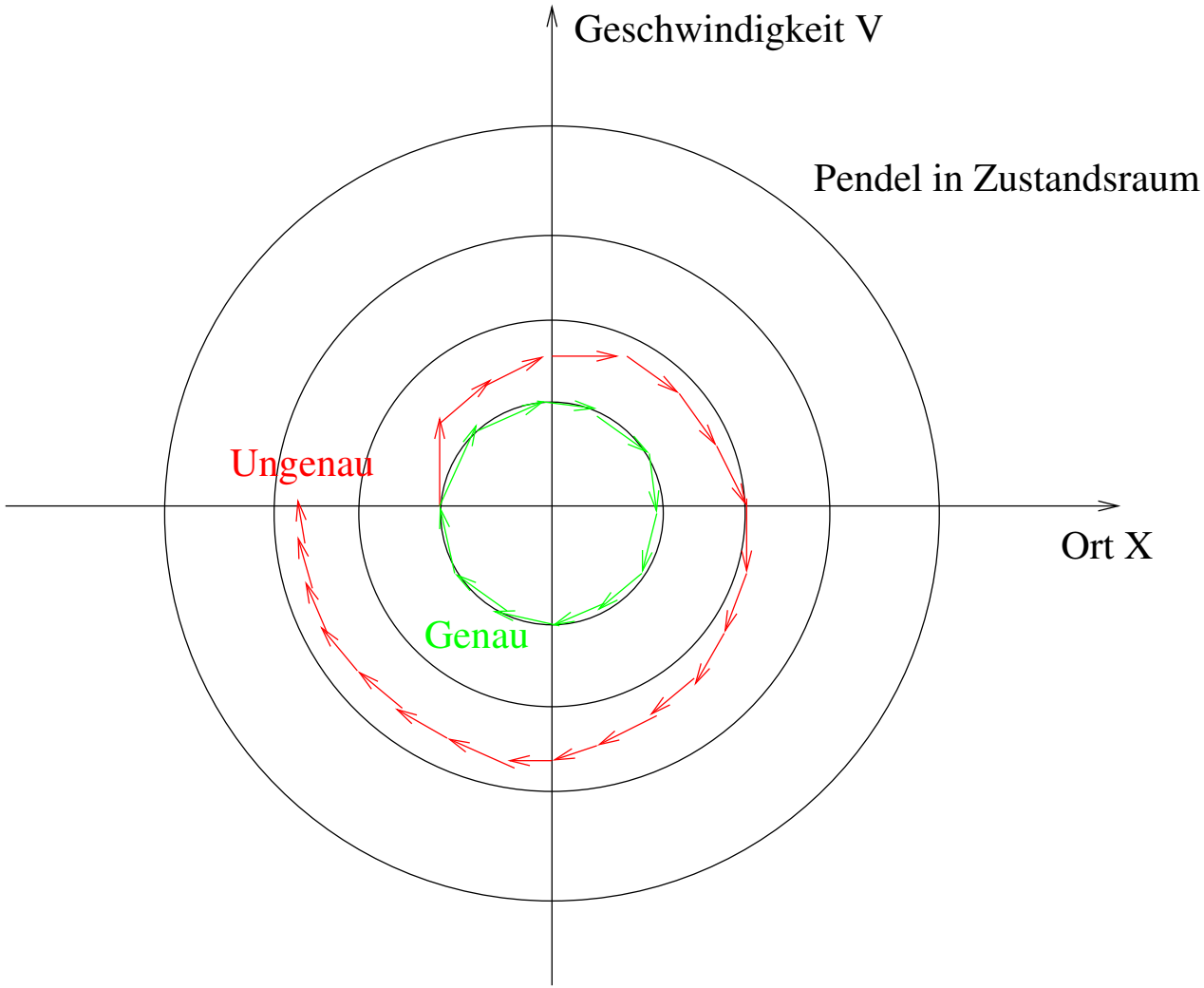
Größere Schrittweite: Schnellere Simulation, aber unstabiler und ungenauer. Und Vice Versa.

Differentialgleichungen

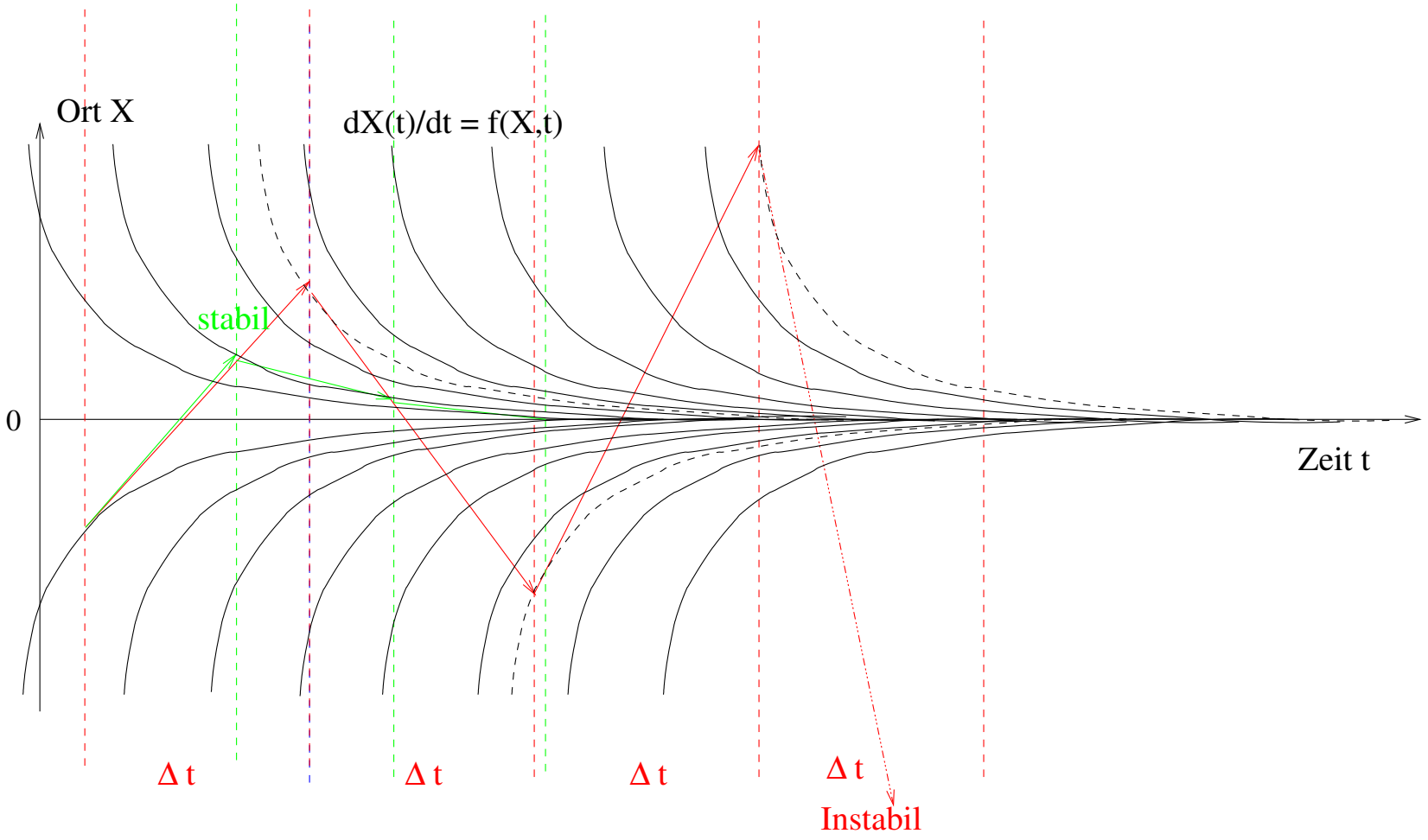




(Un-)Genauigkeit



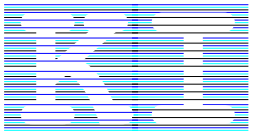
(In-)Stabilität



Numerische Lösung von Differentialgleichungen

Es existieren verschiedene Methoden zur Lösung von Differentialgleichungen:

- Newton-Methode (nicht zu empfehlen!)
- Runge-Kutta (einfach und robust)
- Leapfrog (speziell für Systeme der Form wie Gl. 5)
- Verlet Integration
- Spezielle Predictor-Corrector-Methoden (wie z.B. Bulirsch-Stoer)



Physik Engines: Das Herz der Game Physik

Game Engines bieten als externe Module (*Middleware*) die nötige Physik an. Analog den *Redering Engines* für die Visualisierung.

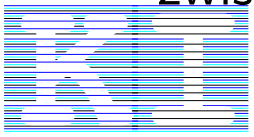
Rigid Body Simulation SDKs (Software Development Kits)

Vorteile:

- Bieten die nötige Physik und die Lösungen der Gleichungen.
- Robuste, optimierte und getestete Numerik

Nachteile:

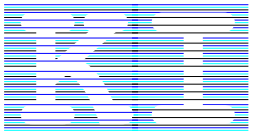
- Aufgrund ihrer universellen Einsatzmöglichkeiten ist ein komplexes Interface zwischen der *Game Engine* und der *Physik Engine* nötig.



- Langsamer als spezial Lösungen

Zwei Phasen:

- Collision Detection
- Collision Response

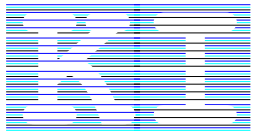


Collision Detection

3 Typen von Objekten

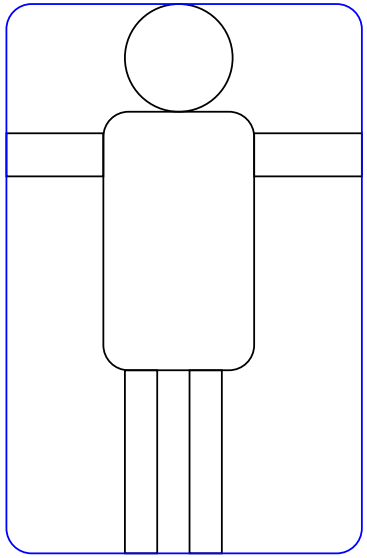
- Primitives (Kugeln, Ebenen, Zylinder)
- Konvexe Polyeder
- Beliebig komplizierte Objekte (*polygen soups*)

Haben verschiedene Geschwindigkeitsperformance.
Designer sollen einfache Objekte bevorzugen!



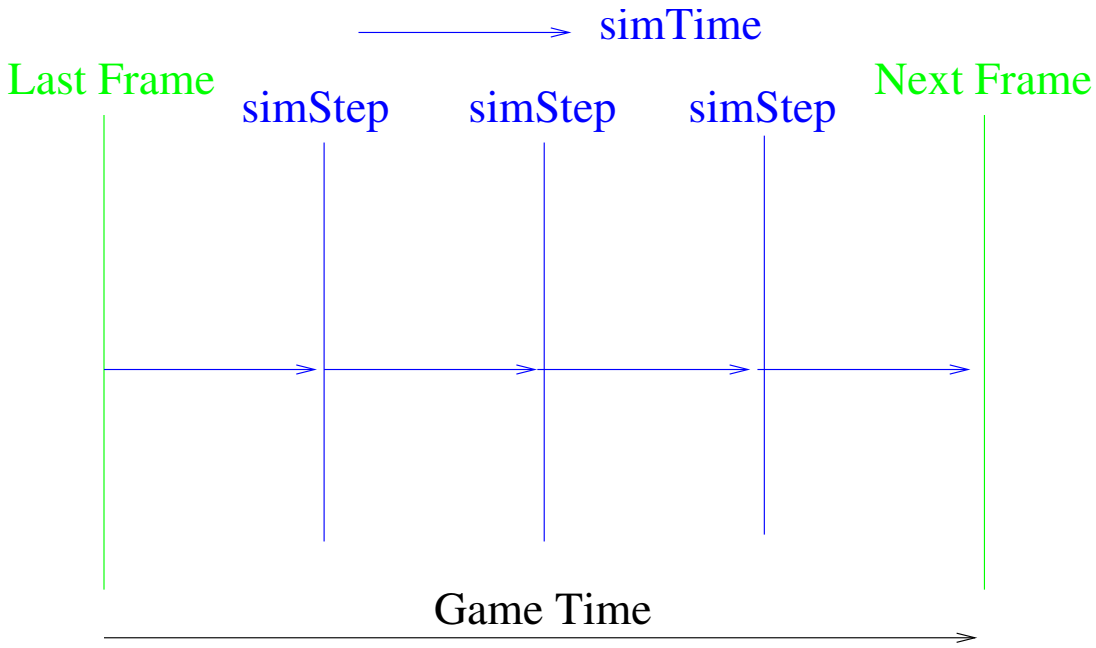
Space-Hierachien

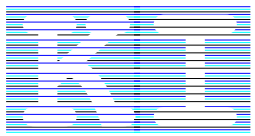
(Space-)Hierachien können zur Performance Steigerung der Collision Detection eingesetzt werden.



Zeiten

- Game Time
- Frame Time
- Simulation Time



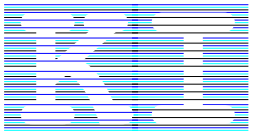


Feste Physikzeitschritte

Problem bei festen Physikzeitschritten:
Wenn die Famerate fällt, immer mehr Physikschritte $\longrightarrow$ weiteres Fallen der Framerate!

Damit sich die Pipline erholen kann: Einfrieren der Objekte.

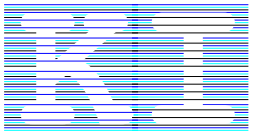
Oder adaptive Physikzeitschritte.



Objekte bewegen

Auf Objekte kann prinzipiell auf 3 Arten eingewirkt werden:

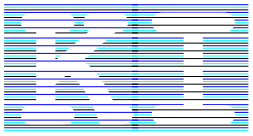
- Anwenden einer Kraft (Komplexität)
- Anwenden eines Impulses (Instantane Geschwindigkeitsänderung)
- Direktes Setzen der Geschwindigkeit



Game Engines

Kommerzielle Game-Physik Engines:

- Havok (www.havok.com)
- Tokamak(www.tokamakphysics.com)
- Meqon (www.meqon.com)
- Karma (now in Renderware)
- Newton (www.newtondynamics.com)
- Novodex (www.novodex.com)

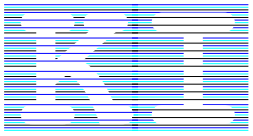


Kommerzielle Simulating Physics Engine

- CMLabs (www.cm-labs.com)

Kostenlose und Open-Source Physik Engines:

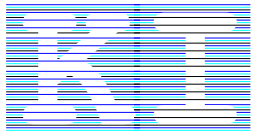
- ODE - Open Dynamics Engine (www.ode.org)
- Open Tissue (A low level application programming interface (API)) (www.diku.dk/forskning/image/research/opentissue/)



ODE - Open Dynamics Engine

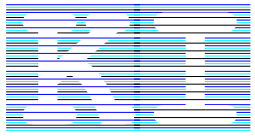
Unterscheidung zwischen *geometry*(Größe, Form, Position und Orientierung) und *rigid bodies*(Masse, Geschwindigkeit,..). Geometry und Body repräsentieren ein Objekt.

- *world*: Für die Physik
- *space*: Für die Collision Detection



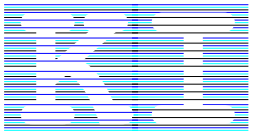
Verbindungstypen (*Joints*)

- ball and socket
- hinge
- slider
- universal
- hinge2
- contact



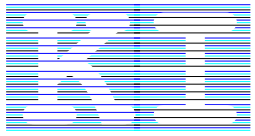
Erhöhen der Stabilität

- Vermeide steife Federn und Kräfte!
- Harte Nebenbedingungen (*hard constraints*)
- Massenverhältnisse (von verbundenen Objekten) klein halten



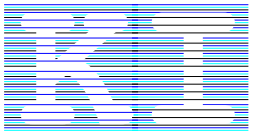
Zusammenfassung

- Physik wird durch (Differential-)Gleichungen beschrieben (plus Anfangsbedingungen)
- Lösen der Gleichungen $\longrightarrow$ Bewegungen
- Impulsive Kräfte verhindern Durchdringen
- Game Physik Engines - Middleware:
  - Lösen der Bewegungsgleichungen
  - Collision Detection
  - Collision Response



Meine Game Physik - Simulation Physik Seite finden Sie im Internet unter:

www.christianherta.de/index_physics.html



Literatur

[1] David H. Eberly :*Game Physics*, Elsevier - Morgan Kaufmann, ISBN: 1-55860-740-4

[2] Press, Teukolsky, Vetterling, Flannery: *Numerical Recipes in C*, Cambridge University Press 1992

[3] Matt McLaurin:*Outsourcing Reality: Integrating a Commercial Physics Engine*, Gamasutra Game Physics Resource Guide, www.gamasutra.com

[4] Tutorial: An Introduction to Physically Based Modeling
http://www-2.cs.cmu.edu/afs/cs/user/baraff/www/pbm/pbm.html

[5] Tutorial: Physically Based Modeling
http://www.pixar.com/companyinfo/research/pbm2001

[6] Open Dynamics Engine - v0.5 User Guide - Russell Smith
www.ode.org