



# Einführung in die Spieleprogrammierung

Physik und Audio



# Audio

- Audio in Spielen am ehesten vergleichbar zum Medium Film
- *Aber:* Interaktivität erfordert dynamische Anpassung
  - Einzelne Bestandteile von Audio werden in Echtzeit modifiziert und wiedergegeben
  - Gesamtergebnisse ist schwerer kontrollierbar, es müssen viele mögliche Situationen abgedeckt werden
  - Oft zusätzliche Informationen verfügbar (Spielername, bisherige Entscheidungen, ...)



- Musik 

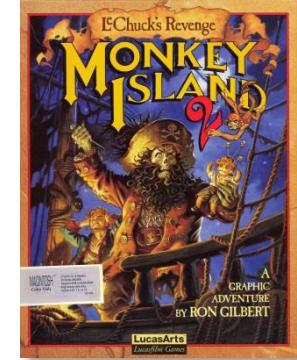
  - Meist im Hintergrund um eine Stimmung wieder zu geben oder bestimmte Ereignisse einzuleiten → oft dynamisch angepasst (siehe z.B. iMUSE ab Monkey Island 2 in der SCUMM Engine)
  - Titelmelodie als Erkennungsmerkmal

- Soundeffekte 

  - Simulation von beliebigen Geräuschen in der virtuellen Umgebung (Fußschritte, Explosionen, ...)

- Sprachausgabe 

  - Reale Aufnahme oder synthetische Sprache = Text-to-Speech
  - Bei virtuellen Charakteren ist **Lipsyncing** nötig



- Meist an externe Firma vergeben (Outsourcing)
- Professionelle Sprecher
- Sound-Designer
- Musiker/Komponisten von Einzelkünstler bis Symphonieorchester



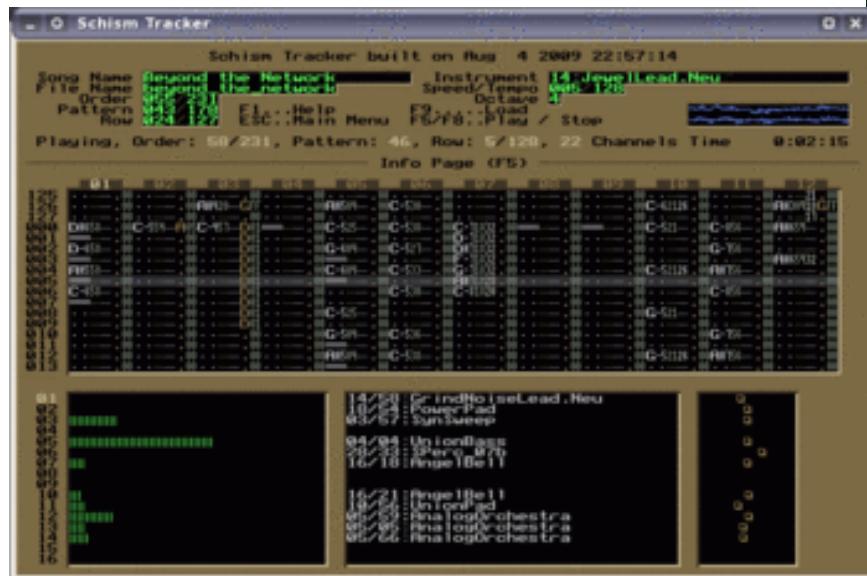


- Mehrere Spuren (Tracks) die zur Laufzeit gemischt werden können
  - Z.B. Instrumentvariation
- Mehrere überblendbare (Dovetails) Segmente
- Motive dynamisch bei Events spielen (Stringers)
- <http://gamesounddesign.com/making-interactive-music-for-games-part-two.html>



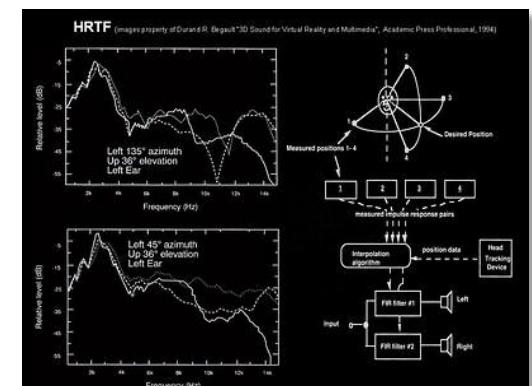
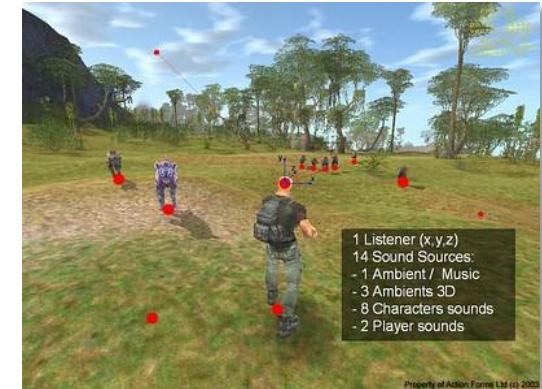
- Midi
- Tracker vs. DAWs
  - Vgl. Motioncapture vs. Handanimation
  - Starre Aufnahmen hoher Qualität (.wav) vs
  - Dynamisch anpassbare Kompositionen
  - Früher Hardware-Synthesizer, heute Samples
  - VSTi virtuelle Instrumente zur Erstellzeit
- Oft Multitrack Aufzeichnungen
  - -> mehr Kontrolle

- Tracker



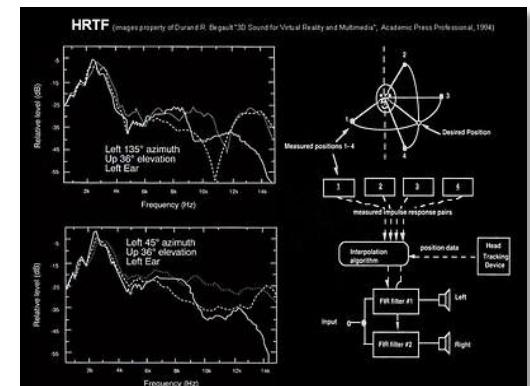
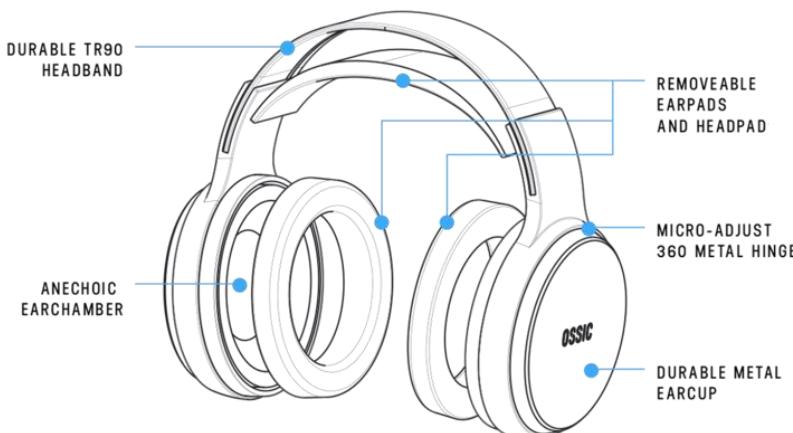
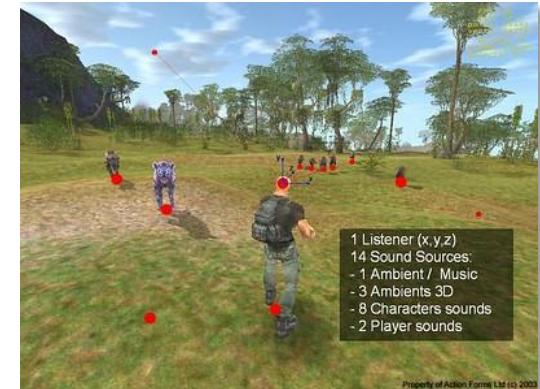
- <http://modarchive.org/>
- <https://github.com/Artefact2/libxm>
- <http://milkytracker.titandemo.org/>
- <http://rainwarrior.ca/music/moon8.htm>

- Mono / Stereo / Raumklang
  - *Software*: Anzahl Kanäle der Audiodatei
  - *Hardware*: Anzahl Lautsprecher oder virtueller Raumklang
- 3D-Sound
  - Üblicherweise für Soundeffekte und Sprachausgabe
  - Anpassung von Lautstärke, Aufteilung auf die Lautsprecher, Verzögerungen und weitere Filtereffekte (z.B. Head-Related Transfer Function =**HRTF** für Beugung von Schallwellen um den Kopf) je nach Position/Orientierung der Audioquelle und des Hörers
    - Räumliche Ortung von Audio in der virtuellen Szene, Dopplereffekt, ...

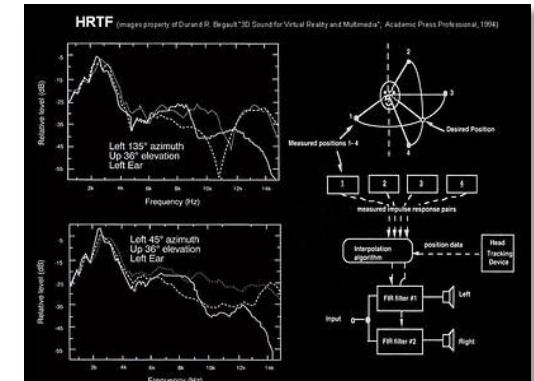
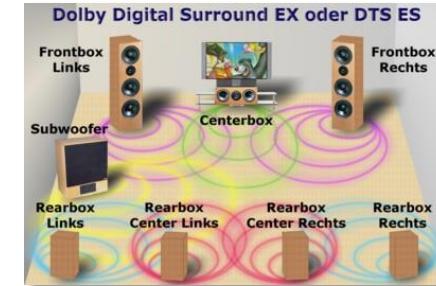


## HRTF:

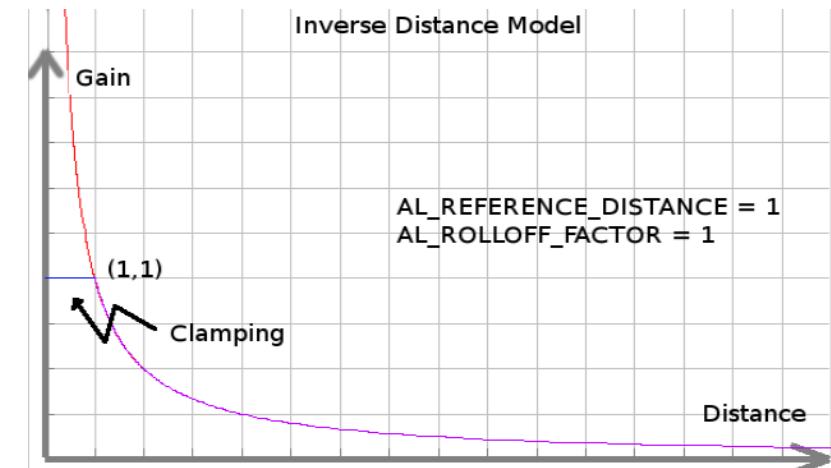
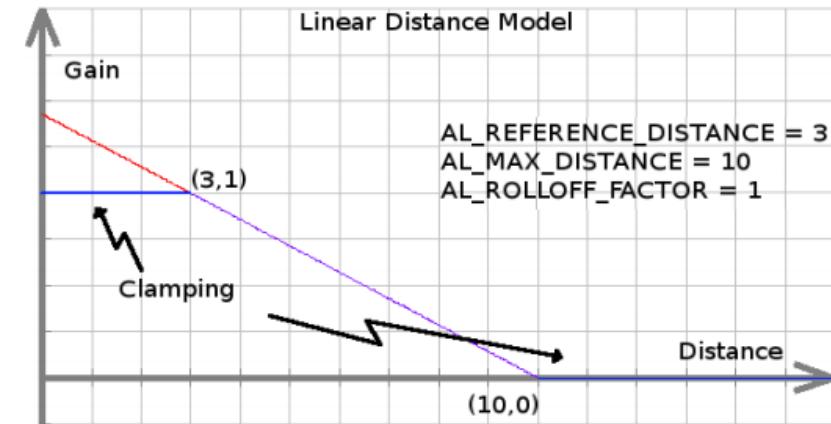
- Kopfgröße (Zeitunterschied links/rechts)
- Form der Ohrmuschel
- Eigentlich personenspezifisch:
  - Kopfhörer passt HRTF an:



- 3D-Sound
  - Objekte sind im 3D-Raum nur mit guten Kopfhörern ortbar!
  - Audioeffekte sind rechenaufwendig, dedizierte Soundkarten selten



- 3D-Sound
  - Abstandsmodelle (**Distance Models**) zur Anpassung der Lautstärke (**Gain**)
    - Linear, Logarithmisch, **Invers**, ...
    - Bei Abstand == **Referenzdistanz** → volle Lautstärke (Gain=1)
    - Bei größerem Abstand → Abnahme der Lautstärke gemäß Distanzmodell und **Rolloff Faktor**
    - Oft keine Lautstärkeerhöhung falls näher als Referenzdistanz und keine weitere Verringerung bei maximaler Distanz (→ Clamping)
    - Z. B. bei Inversem Distanzmodell (mit Clamping):

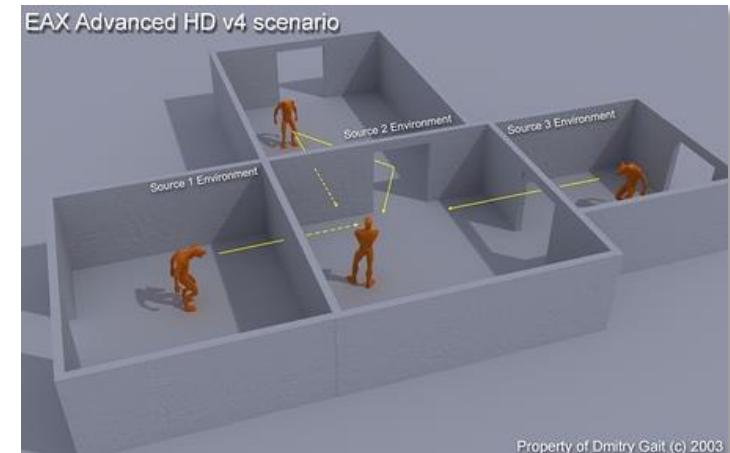
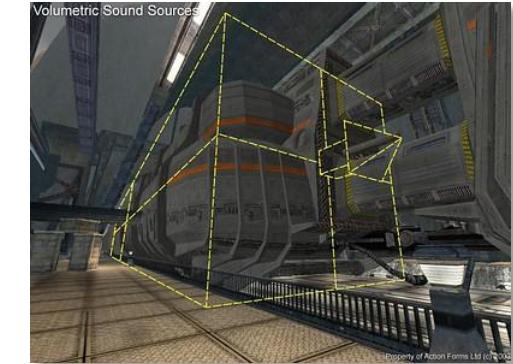


$$\text{Distanz} = \max(\text{Distanz}, \text{Referenzdistanz})$$

$$\text{Distanz} = \min(\text{Distanz}, \text{Maximaldistanz})$$

$$\text{Lautstärke} = \frac{\text{Referenzdistanz}}{\text{Referenzdistanz} + \text{RollOff} * (\text{Distanz} - \text{Referenzdistanz})}$$

- Audio-Komprimierung um Speicherplatz zu sparen
  - Proprietäre (z.B. MP3) oder freie Formate (z.B. Opus)
  - Wiedergabe durch dekodieren der kompletten Audiodateien in den Arbeitsspeicher oder Schritt-für-Schritt (**Streaming**)
- Filter/Effekte je Umgebung
  - Nachahmung von Akustik in Konzertsälen, Höhlen, ...
  - Verdeckung von Audio-Quellen, Volumetrische Audio-Quellen, Reflexionen, ...
  - Z. B. mit Environmental Audio Extensions (EAX)





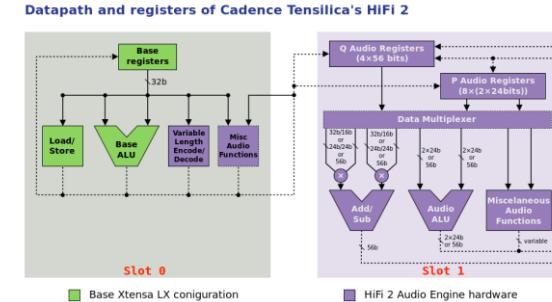
- Reverb (Hall):
  - Rückschlüsse auf
    - Entfernung
    - Umgebung (Material/Raum)
- Delay (Echo)
- Equalizer (Anpassung des Frequenzspektrums)
- Kompression (Reduktion der Dynamik)

- Schnittstelle vom Spiel zum Soundkartentreiber
  - Abspielen und Mischen von mehreren Audioquellen über Hardware-Puffer
  - 3D-Sound
  - Oft weitere Effekte, wie Reverb, Echo, ...
  - Meist keine Dekodierung von Audio und keine fortgeschrittenen Effekte
  - Da inzwischen nur noch eingeschränkte Hardware-Unterstützung vorhanden ist, implementieren eigentlich höher-levelige APIs oft auch direkt die Anbindung zum Treiber

- **DirectSound/XAudio2**
  - DirectSound 1995 von Microsoft als Teil von DirectX veröffentlicht → Nur für Windows
  - Nachfolger XAudio2 aus Fusion mit der Xbox360 Audio API ab DirectX 10.1 → auch für Windows Phone und Xbox
  - Keine direkte Hardware-Unterstützung mehr unter Windows Vista/7 wegen „Universal Audio Architecture“, auch unter Windows 8 noch stark eingeschränkt
    - → Creative ALchemy fängt DirectSound-Befehle ab und wandelt sie in OpenAL um
- **OpenAL**
  - 2000 von Loki Software und Creative Labs veröffentlichte plattformunabhängige Spezifikation
  - „OpenGL für Audio“, allerdings kam es nie zu einem Architecture Review Board
  - Quelloffene Beispielimplementierung für Windows; letzte stabile Version 1.1 allerdings von 2005, spätere Implementierungen nur noch proprietär
  - Offizielle Webseite seit längerem nur eingeschränkt online, Nutzung nur noch in wenigen Spielen
  - *Allerdings:* Parallel quelloffene (allerdings nicht hardware-beschleunigte) Weiterentwicklung in **OpenAL soft** (<http://kcat.strangesoft.net/openal.html>)
    - aktuell stabile Version: 1.18 vom September 2017; regelmäßig aktualisierte Version im Public Repository: <http://repo.or.cz/w/openal-soft.git>
    - Umsetzung der OpenAL Spezifikation in Software
    - Fortgeschrittene Effekte wie HRTFs, Verdeckung, Umgebungshall

## Trueaudio

- DSP zur Audiobeschleunigung
- Anspruchsvolle Audioeffekte fressen CPU (convolution reverb)
- Limitiert auf ausgewählte AMD-Hardware (Konsolen..)
- Nur mit kommerziellen Plugins verwendbar

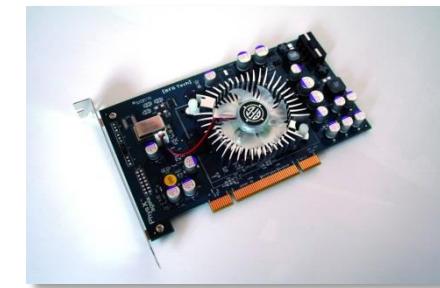
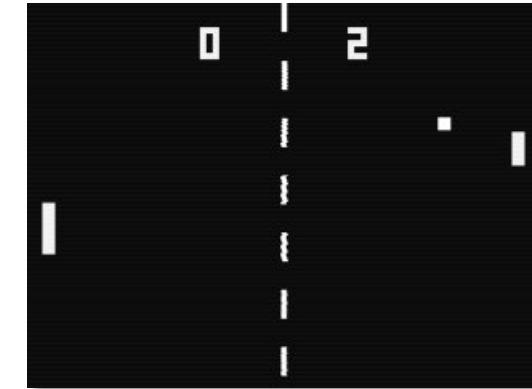


- **Miles Sound System** (RAD game tools)
  - Urspr. Audio Interface Library (AIL), eine Treiber-Bibliothek für DOS von 1991
  - Gilt auch heute noch als eine der performantesten und stabilsten Audio-Engines und unterstützt praktisch jegliche Plattformen
  - Fokus auf Audio-Dekodierung und DSP Effekte
  - Inzwischen aber auch grafische Tools zum Authoring
- **FMOD** (Firelight Technologies)
  - Größerer Fokus auf Authoring und grafische Tools, meist über gute Unterstützung von externer Software
  - In viele Engines eingebunden
  - Frei zur nicht kommerziellen Nutzung und für Indieproduktionen (< \$100k und max. 1 Spiel pro Jahr für eine Plattform)
- **Wwise** (audiokinetic)
  - Viele eigene grafische Tools
  - Frei zur nicht kommerziellen Nutzung
- **Unreal Sound System** (Epic)
  - Direkte Integration in den Level-Editor
- ...

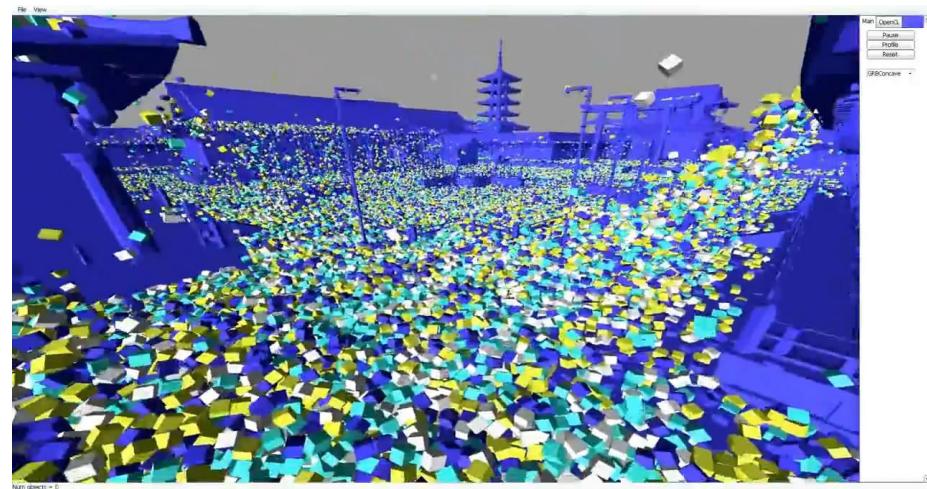
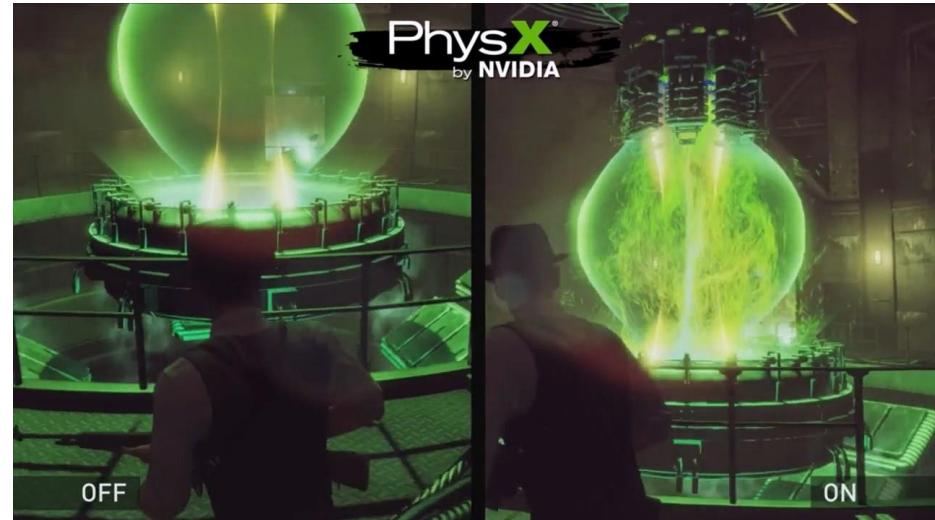


# Physik

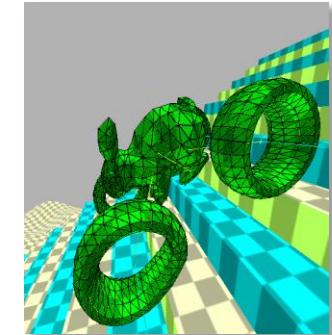
- Praktisch jedes Spiel braucht zumindest eine rudimentäre Physik
- Bei manchen ist sie besonders wichtig (Rennspiel, Flugsimulatoren, Sportspiele)
- Inzwischen meist mit Hardware-Beschleunigung
  - 2006 kamen separate Physikbeschleunigerkarten heraus (Ageia PhysX), die Aufgaben wurde aber kurz darauf von Grafikkarten übernommen (Übernahme von Ageia durch Nvidia 2008)
- Je mehr bzw. genauere Gesetze der Physik, desto höher der Realismusgrad, aber auch mehr Rechenleistung nötig
- Höhere Immersion, da realistischere Interaktion mit Gegenständen in der virtuellen Welt
- Physik vereinfacht Animationen, Interaktion
- Viele Physik-Effekte sind aber auch rein optisch (v. a. Stoff- u. Partikeleffekte)



# Beispielvideos Physik



- Feste/Starre Körper (**Rigid Bodys**)
  - Kinematik und Gravitation
  - Kollisionserkennung (u.a. Hüllkörper)
  - Ragdoll und Kettensimulation
- Deformierbare Körper (**Soft Bodys**)
  - elastische oder plastische Verformung
  - Stoff, Seile, Haare, ...
- Erweiterte **Partikeleffekte** zur Simulation von Wasser, Feuer, Rauch, Staub, Regen, ...

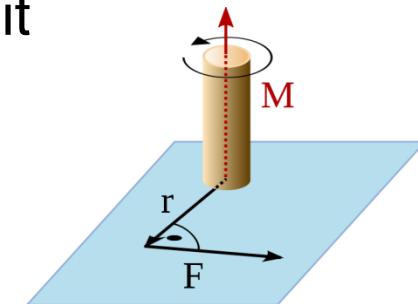
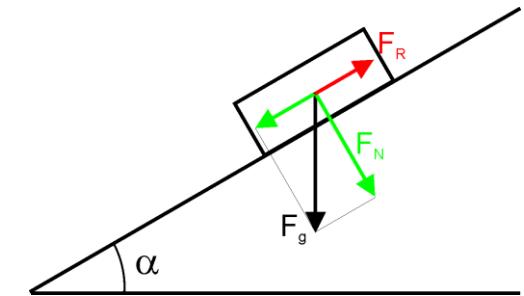


- **Vereinfachte Sichtweise:** Körper als Punktmassen, uneingeschränkte Bewegung
  - Körper haben **Masse** (konstant) und **Position** (änderbar → Bewegung)
  - Verwendung bei Partikelsystemen oder Gewehrkugeln
- Physik auf Basis der Newtonschen Gesetze:
  - **Trägheit:** Geschwindigkeit und Bewegungsrichtung bleiben ohne Krafteinwirkung gleich
  - **Aktionsprinzip:** Krafteinwirkung ändert Bewegung proportional  $\vec{F} = m\vec{a}$
  - **Wechselwirkungsprinzip:** Kräfte treten paarweise auf, „actio gleich reactio“ → Impulserhaltungssatz

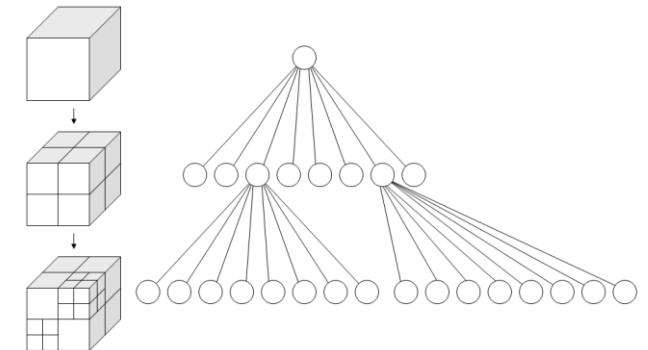
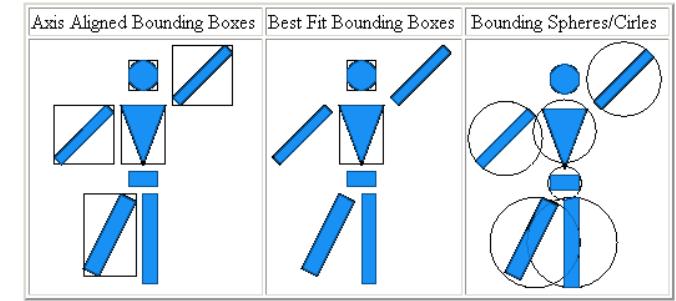


- Kräfte in Spielen meist bekannt:
  - Berechnung der **Beschleunigung** über  $\vec{a} = \frac{\vec{F}}{m}$ , bei Einwirkung mehrerer Kräfte werden diese addiert
  - Update der **Geschwindigkeit** über Integration:  $\vec{v} = \vec{v}_0 + \vec{a}t$
  - Update der **Position** über erneute Integration:  $\vec{p} = \vec{p}_0 + \vec{v}t$
  - Code: `v += a*t; p += v*t;`
- Manche Physik-Engines arbeiten nur mit **Kräften**, andere mit **Impulsen** (exakter Begriff: Kraftstoß = Impulsänderung = Kraftanwendung über gewisse Zeit), manche auch mit beidem
- **Sleeping**: Wenn die Bewegung eines Objektes unter eine gewisse Grenze fällt wird es deaktiviert ( $\rightarrow$  keine Integration der Bewegungsgleichungen mehr nötig)

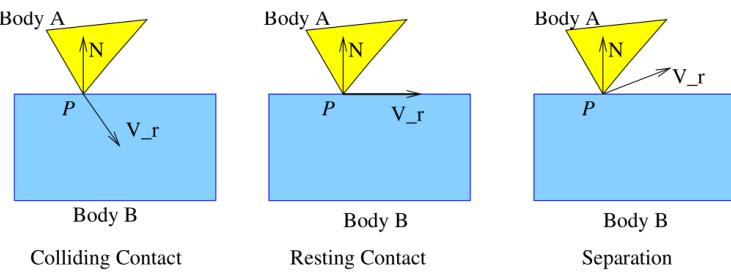
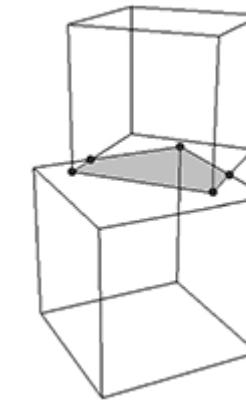
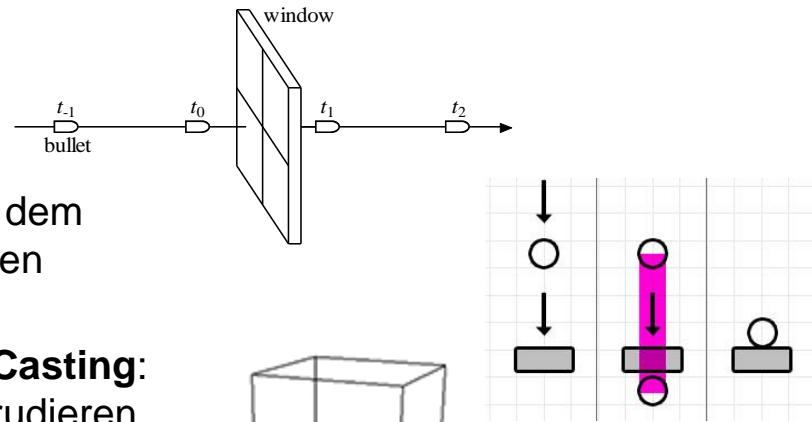
- Wichtige Kräfte:
  - **Gravitation** ( $\sim 9,8 \frac{m}{s^2}$  auf der Erde)
  - **(Luft-/Wasser-/-..)widerstand** bei Bewegung durch ein Medium
  - **(Haft-/Gleit-/Roll-)reibung** bei berührenden Flächen; evtl. anisotropisch (=unterschiedlich je Richtung)
- *Erweiterte Sicht:* Körper haben **Volumen** und damit eine **Orientierung**
  - Positionsbestimmung über den **Schwerpunkt**
  - Neben Kräften wirken auch **Drehmomente** → Drehbewegungen um Achse durch den Schwerpunkt
  - Masse bestimmt sich über **Volumen** und **Dichte**
  - Wechselwirkungen zwischen Körpern durch **Kollisionen**



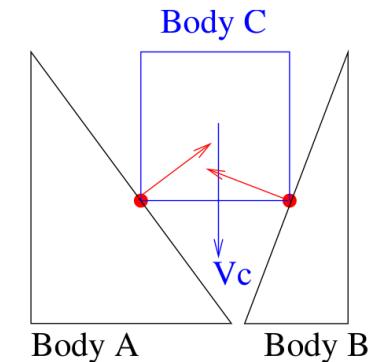
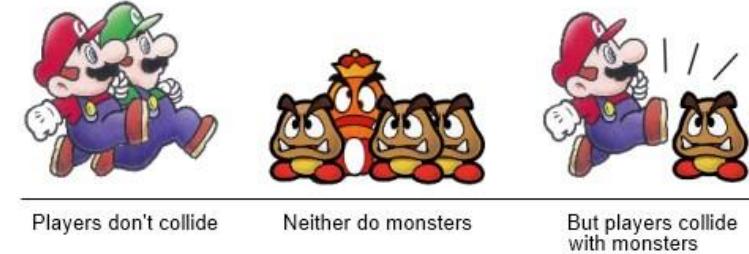
- Berechnung der Bewegung für das aktuelle Frame  
→ Kollisionserkennung → Auflösung
- **Kollisionserkennung** zunächst durch Testen der Überschneidungen von **Hüllkörper**, z. B. **AABB**, Kugel, abgerundeter Zylinder (=**Capsule**), ...\*  
**(Broad Phase)**
  - Überschneidungen einfacher **Primitive** oder zumindest **konvexe Hüllkörper** sind einfacher zu berechnen (Middle Phase)
  - Oft zunächst Einteilung des Raumes in **Grids**, **Occtrees**, Aufbau einer **Hierarchie** von Hüllkörpern, oder sonstige Gruppierung möglicher Kollisionspartner zur effizienteren Suche (Broad Phase)
- Später auf anhand der eigentlichen Geometrie Polygon für Polygon (**Narrow Phase**)
  - Aufwendiger, aber liefert präzise Kontaktpunkte
  - Muss trotzdem nicht der exakten (sichtbaren) Geometrie entsprechen bzw. kann entfallen, falls Hüllkörper das Objekt schon gut annähert



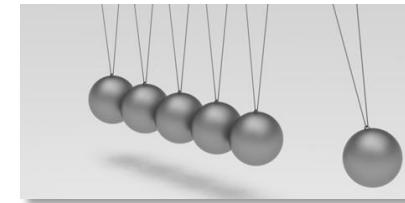
- **Tunneling** bei zu geringer Framerate
  - Lösung **Continuous Collision Detection**:  
Betrachtung des zurück gelegten Weges seit dem letzten Frame → Kollisionszeitpunkt bestimmen (**Time Of Impact**)
  - Andere Lösung **Swept Shapes** bzw. **Shape Casting**:  
Objekt entlang seines Bewegungspfades extrudieren → Kollisionserkennung mit neuem Objekt
  - Meist nur bei Objekten mit Mindestgeschwindigkeit
- Arten von **Kontaktpunkten**
  - Punkt-Punkt, Punkt-Kante, Punkt-Fläche, Kante-Kante, Kante-Fläche, Fläche-Fläche
    - Je nach Objekt nur bestimmte möglich (z.B. hat eine Kugel nur eine Fläche)
    - Bearbeitungsreihenfolge wichtig (Fläche-Fläche kann anderen beinhalten)
    - Auflösung unterschiedlich schwierig
  - **Kontaktpunkt, Normale und Eindringtiefe**
  - Kollidierende, ruhende oder sich trennende Kontaktpunkte



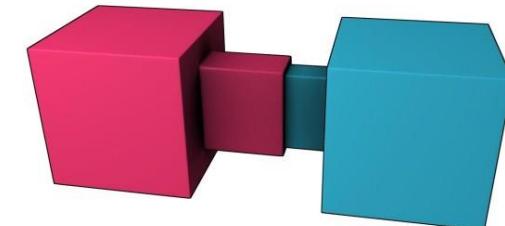
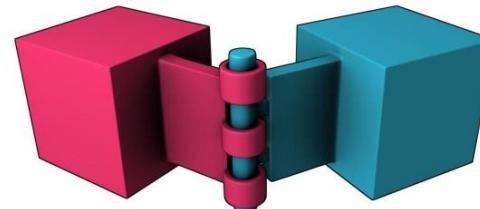
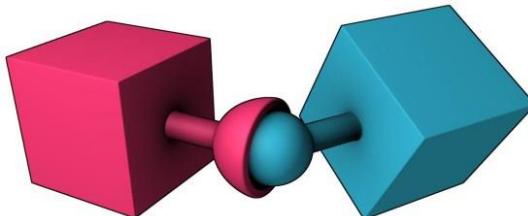
- **Kollisionsfilterung**
  - Oft können/sollen Kollisionen zwischen bestimmten Objekten deaktiviert werden
  - → Deaktivierung über **Collision Masks, Layers** oder **Callbacks**
- **Simulation Islands:** (potentiell) interagierende Objekte werden gruppiert und können separat (parallel) abgearbeitet werden
- **Kollisionsauflösung**
  - Ein Durchdringen der Körper soll verhindert werden
  - Anpassung von Position/Orientierung um Kollision entgegen zu wirken
  - Bei mehreren Kontaktpunkten: Abarbeitung aller Kontaktpunkte auf einmal (**Jacobian Method**, rechenaufwendig) oder iterativ nacheinander (ungenauer, Entstehen neuer Konflikte → Gegenständen kommen nicht zur Ruhe)



- Auflösen von Kollisionen durch Anwendung einer gegenwirkenden Kraft an den Kontaktpunkten
  - **Elastischer Stoß:** Körper prallt zurück mit Impulserhaltung  
 $\rightarrow m_1 v_1 + m_2 v_2 = m'_1 v'_1 + m'_2 v'_2$
  - **Unelastischer Stoß:** Bewegungsenergie wird (teilweise) absorbiert Körper  $\rightarrow$  im Extremfall bleibt der Körper am Kontaktpunkt „kleben“
  - Realer Stoß ist immer eine Mischung aus beiden (teilelastischer Stoß)  
 $\rightarrow$  **Restitutionskoeffizient** bestimmt Anteil der erhalten bleibenden Bewegungsenergie bei Kollision
  - Durch Ausdehnung des Körpers kann durch die gegenwirkende Kraft eine Drehbewegung um den Schwerpunkt folgen
    - Update der Winkelgeschwindigkeit ähnlich zu linearer Geschwindigkeit, aber statt Richtung gibt es eine Drehachse



- Bewegungseinschränkungen (**Constraints**):
  - Mehrere starre Körper hängen an Kontaktpunkten zusammen (=unendliche Anziehung und Reibung, aber keine Restitution)
  - Abknicken des Gesamtkörpers um andere Objekte herum → **Ragdoll Physik**
  - Mögliche Verbindungen: **Punkt-an-Punkt, Scharnier, Slider, Feder-Verbindung**
  - Auch mit eingeschränkten Freiheitsgraden (**DOF**), vgl. Skelettautomation
  - **Featherstone**-Algorithmus liefert effizientere Auflösung von Constraints bei einer Kette von zusammenhängenden Objekten durch reduzierte Koordinatendarstellung



- *Umgekehrt:* Auseinanderbrechen von Körpern bei Kollision
  - Vorherige Festlegung der Bruchstellen durch den 3D-Modellierer
  - Kann automatisiert werden, z.B.:
    - konvexe Dekomposition
    - Voronoi Shatter: Zufällige Verteilung von Punkten im 3D-Körper → Ebenen zwischen den Punkten liefern Schnittebenen
  - Zur Laufzeit animiert die Physik-Engine das eigentliche Auseinanderbrechen
    - Vor allem früher wurden solche Effekte meist vorberechnet und zur Laufzeit nur die entsprechende Animation abgespielt
    - Heute oft auch in Echtzeit, z. B. halten Constraints die einzelnen Bruchteile dann solange zusammen, bis eine zu hohe Kraft einwirkt (Breakable-Constraint)

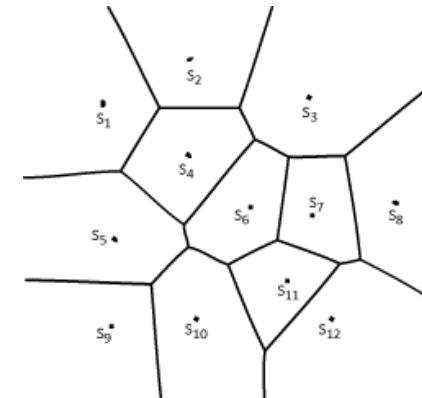
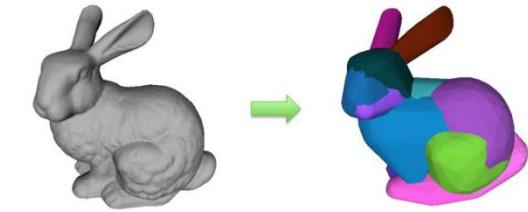
Geometry preparation and artist tools



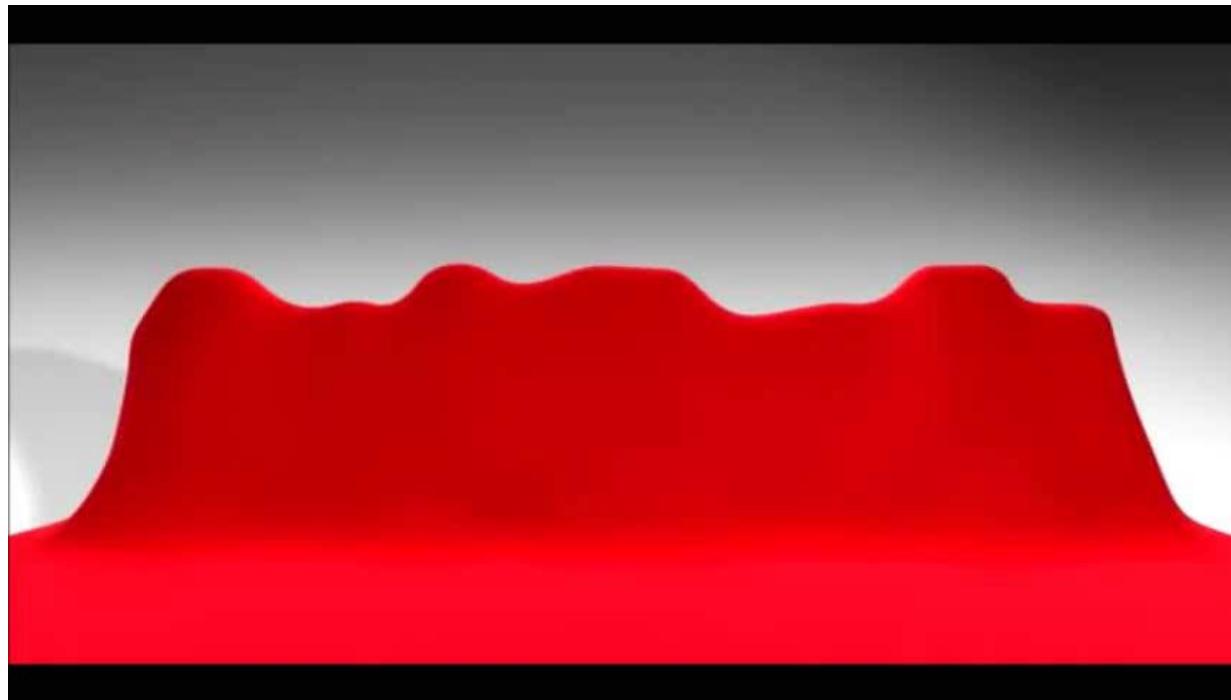
Runtime destruction methods



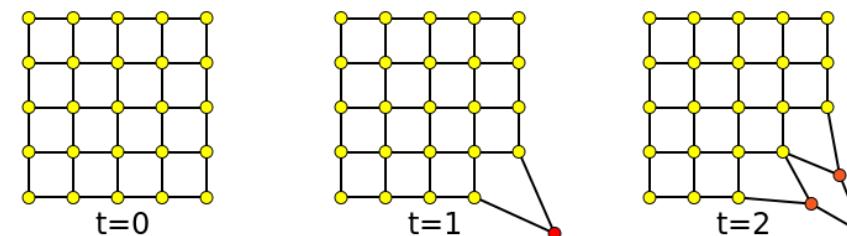
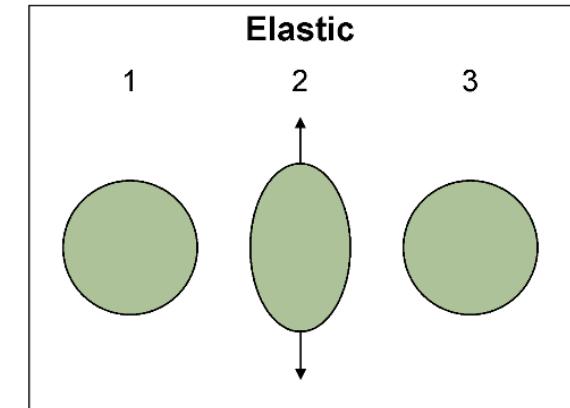
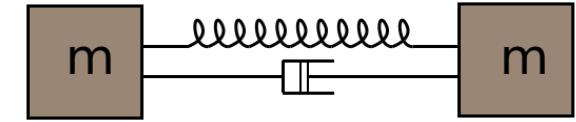
(Semi) Automatic physics shape generation



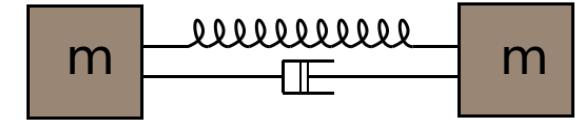
- **Grundprinzip von Soft Bodies:** Anwendung der Kräfte nicht auf den Gesamtkörper, sondern auf dessen einzelne Bestandteile (Vertices)
  - Verformung und Eigenkollisionen möglich
  - Innere Kräfte bzw. Constraints halten Objekt zusammen bzw. geben von außen einwirkende Kräfte weiter



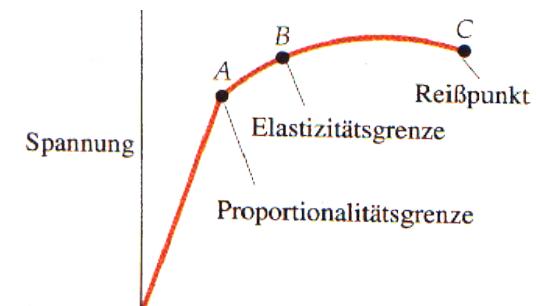
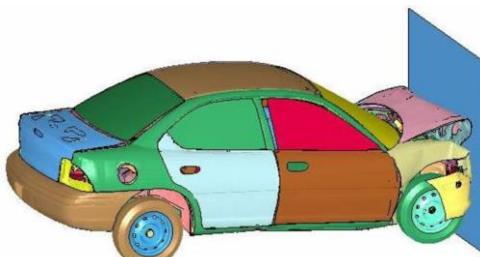
- Elastische Verformung
  - Vereinfachte Ansicht als Feder zwischen den einzelnen Vertices, welche bestimmte Masse zugewiesen bekommen (=Feder-Constraint)
    - Anwendung des Hookeschen Gesetzes (Linear-elastisches Verhalten)
    - Federkraft  $F_s = -kx$  mit  $k$  =Federkonstante,  $x$ =Längenunterschied zur Ruheposition



- Elastische Verformung
  - Um zu langes Schwingen zu verhindern kann zusätzlich ein (Stoß-) dämpfer hinzugefügt werden
    - Dämpfungskraft  $F_d = -cv = -c\frac{x}{t}$  mit  $c$  = Dämpfungskonstante
    - Gesamtkraft  $F_{ges} = F_s + F_d = -kx - cv = -kx - c\frac{x}{t}$
  - Bei Erreichen der maximalen Elastizität → plastische Verformung bis zu Bruch



- Plastische Verformung
  - Verformung bleibt nach Aussetzen der äußeren Kraft erhalten → "Zurücksetzen" der Feder
  - Ideal plastischer Körper:
    - verhält sich bis zu einer bestimmten einwirkenden Kraft bzw. Spannung wie ein starrer Körper
    - ab seiner Fließgrenze wird er irreversibel und unbegrenzt verformt
    - In der Realität aber immer zusammen mit elastischer Verformung



- Wasser, Feuer,..
- In der Regel Partikel



- **Havok** ([havok.com](http://havok.com)): Kommerzielles SDK mit sehr vielen Features, aber auch entsprechend teuer
- **PhysX** ([geforce.com/hardware/technology/physx](http://geforce.com/hardware/technology/physx)): Kommerzielles SDK von Nvidia mit speziellem Fokus auf GPU-Beschleunigung
- **Newton Game Dynamics** ([newtondynamics.com](http://newtondynamics.com)): Zunächst kommerzielles SDK, inzwischen quelloffen
- **Open Dynamics Engine** ([ode.org](http://ode.org)): Quelloffenes SDK
- Tokamak Physics Engine ([tokamakphysics.com](http://tokamakphysics.com)): Zwischenzeitlich kommerziell, inzwischen wieder quelloffen, wird aber nicht mehr weiter entwickelt
- **Bullet Physics** ([bulletphysics.com](http://bulletphysics.com)): Mit Abstand umfangreichstes quelloffenes SDK, dass in vielen kommerziellen Anwendungen, Spielen und Filmen verwendet wird
- **Physics Abstraction Layer** ([adrianboeing.com/pal](http://adrianboeing.com/pal)): Quelloffene Bibliothek um mit mehreren Physik SDKs gleichzeitig zu arbeiten bzw. diese falls nötig auszutauschen
- ...

- Bullet Physics Manual (Teil des SDKs) und weitere Präsentationen:  
<https://code.google.com/p/bullet/downloads/list>
- Web-basierte Physiksimulationen:
  - <http://www.wildbunny.co.uk/blog/2011/04/06/physics-engines-for-dummies/>
  - <http://www.myphysicslab.com/index.html>
- Ian Millington, Game Physics Development
- Christer Ericson, Real-Time Collision Detection

