Virtuelle Realität und Simulation
Sound-Rendering

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

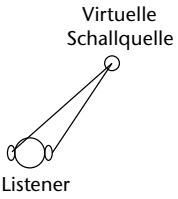
Sound in VR

- Vollständige Immersion nur, wenn alle Sinne richtig u. konsistent stimuliert werden.
  (Schau Thriller im Fernsehen ohne Ton! ☺)
- Wozu wird Hören gebraucht:
  - Lokalisieren von (noch) nicht sichtbaren Geräuschquellen,
  - Zweitwichtigstes Sinnesorgan zum Zurechtfinden / zur Navigation,
  - Erzeugung eines Raumeindrucks
- Höreindrücke:
  - "groß" (späte Reflexionen),
  - "unterirdisch" (viele Echos),
  - "dumpf" (keine Reflexionen),
  - "draußen" (keine Echos, aber Fremdgeräusche).
- Wie rendert man Sound?

Simple Methoden

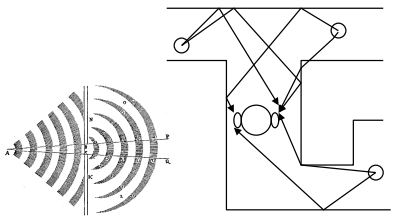
- Motoren-Geräusch (Spiele, Flugsimulator, ..)
  - Sound = Funktion von Geschwindigkeit, Benzinverbrauch, Gas, Fliehkräfte, Orientierung, etc.
  - Sound ist vorab aufgenommen
- "3D"-Sound:
  - Lautstärke-Differenz
  - Hall-Effekt ("*reverberation*")
- Wie rendert man Sound *realistisch* und in *Echtzeit* ?
- Unterschiede zwischen Licht und Schall:
  - Geschwindigkeit
  - Wellenlänge

→Macht Rendern von Sound so schwierig



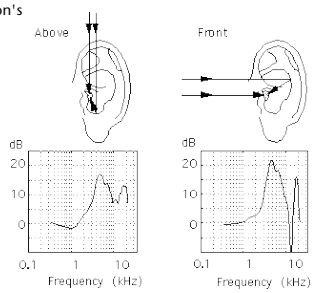
Human Factors

- Akustische Effekte:
  - Reflexion (*reflection*),
  - Brechung (*refraction*),
  - Streuung,
  - Beugung (*diffraction*),
  - Interferenz.



from Isaac Newton's *Principia* (1686)

- Ortungsfähigkeit des Ohrs:
  - Amplitudendifferenz,
  - Laufzeitunterschiede,
  - Änderungen des Spektrums durch Ohrmuschel und Kopf.



Mischen von Schallquellen

- n Quellen aus verschiedenen Richtungen, jede sendet ein Signal $s_i(t)$
- Dämpfung ("*attenuation*") a_i pro Quelle s_i:

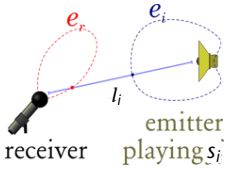
$$a_i = \frac{e_i(\phi_i, \omega_i) e_r(\phi_i, \omega_i) v_i}{l_i}$$

$e_i(\phi, \omega) =$ Abstrahlchar. z.B. $\frac{1}{2}(1 + \cos(\phi)^\alpha)$

$e_r =$ Empfängerchar.

$v_i =$ visibility, incl. Beugung, $\in [0, 1]$

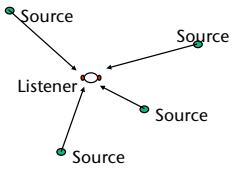
$l_i =$ Entfernung



- Summe:

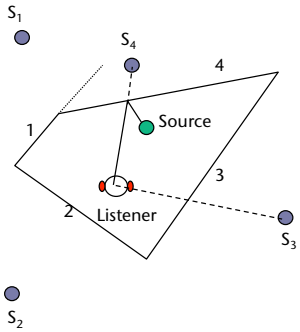
$$s(t) = \sum_i a_i s_i(t - \tau_i)$$

$\tau_i = l_i / c$, $c =$ Schallgeschwindigkeit

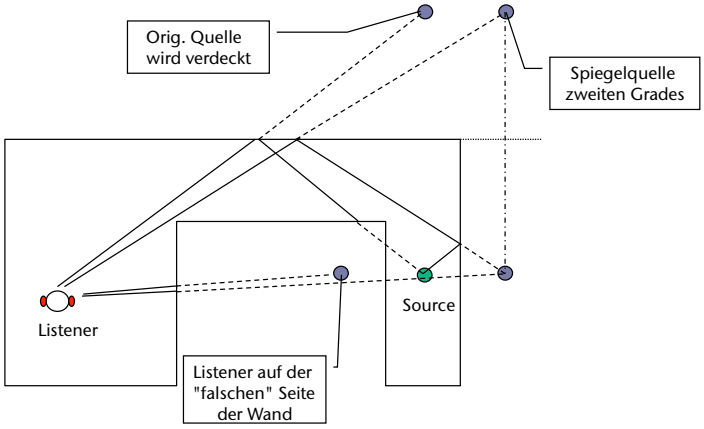


Spiegelquellen-Methode ("*image source*")

- Zur schnellen Berechnung der Reflexionspfade
- Idee: konstruiere für jede Schallquelle und jede Wand eine "*virtuelle*" Schallquelle ("*image source*")
- Länge des Strahls → Laufzeit, #Reflexionen → Dämpfung
- Aufwand: $O(n \cdot r)$
  n = # Schallquellen,
  r = # Wände.
- Probleme:
  - Alles neu berechnen, wenn sich Quellen bewegen
  - Nur Reflexionseffekt

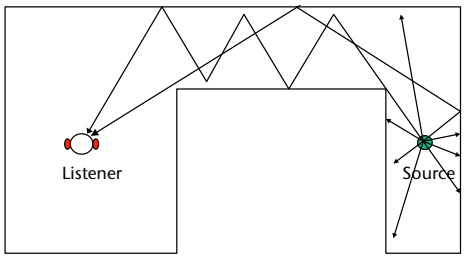


- Beispiel (nicht alle Spiegelquellen eingezeichnet):



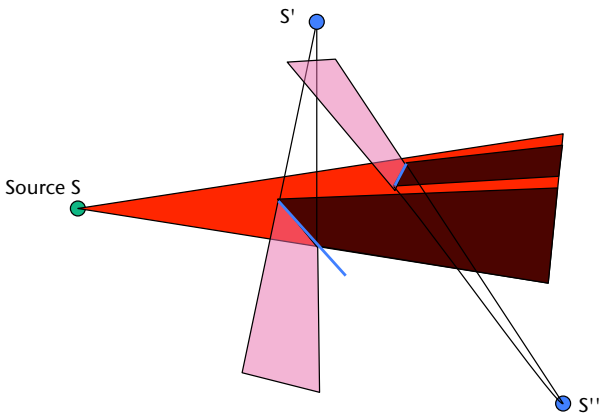
Ray Tracing Methods

- Verschieße Strahlen von Schallquelle (oder *Listener*).
- Probleme:
  - Viele Strahlen "umsonst",
  - *Listener* besteht eigtl. aus 2 sehr kleinen Punkten im Raum
  - Wie findet man garantiert die Hauptpfade?



Beam Tracing

- *Beam* = ausgedehnter, sich aufweitender Strahl
- Damit läßt sich viel vorausberechnen, zur Laufzeit Strahlrückverfolgung
- *Beam-Tracing*:
  1. Sortiere Polygone entlang *Beam*,
  2. Für jedes Polygon, das *Beam* schneidet: spalte *Beam* auf in 2 oder mehr, bestimme Spiegelquelle für reflektierten *Beam*
  3. *Rekursion*



Sound-Rendering mit Beams

- Alle *Beams* berechnen: zu jedem *Beam* speichere:
  - zugehörige Quelle/Spiegelquelle;
  - spiegelndes Polygon;
  - *Vater-Beam*; Für gegebenen *Listener*:

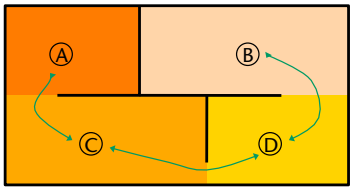
1. Bestimme alle enthaltenden *Beams*,
2. Verfolge Strahlen zurück in Richtung der Quellen/Spiegelquellen über die Spiegelpolygone bis zum Ursprung mittels *Image-Source-Methode*,
3. Signale aufaddieren.

  - Anzahl Reflexionen;
  - ursprüngliche Quelle;

→ "*Beam tree*".

Beschleunigung des *Beam-Tracings*

- Wie macht man *Beam-Tracing* schnell ?
- Idee: Berechne weitere Datenstruktur, die Szene in konvexe Zellen aufteilt.
  - Denn: Punkt in geschlossenem *Polyeder* "sieht" jedes Polygon;
- Definition *Zelle* :
  "Zelle" ist ein Volumen des Raumes, mit der Eigenschaft:
  1. Das Volumen ist konvex;
  2. Das Innere des Volumens enthält keine Polygone.
- Zu jeder Zelle speichere:
  - Nachbarzellen ("*cell adjacency graph*"),
  - Rand-Polygone.

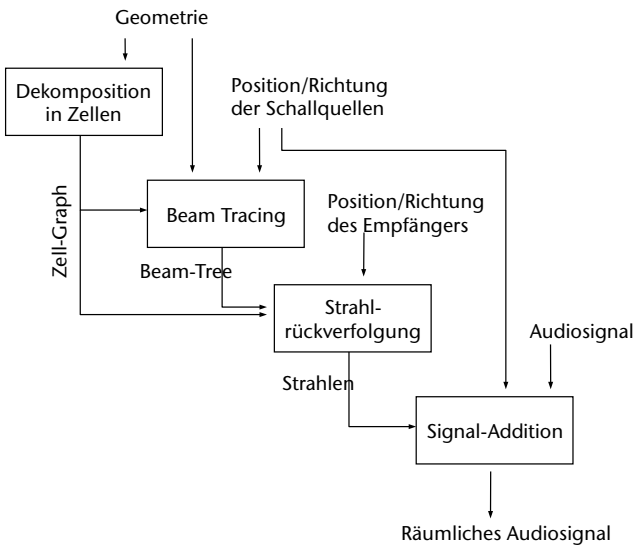


Beam-Tracing mit Zellen

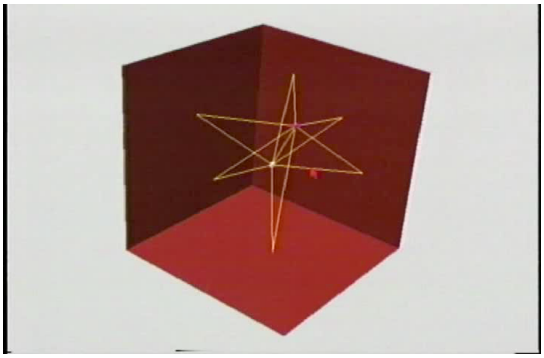
Gegeben Quelle, wie wird *Beam* "verschossen" und "beschnitten"?

1. Bestimme Zelle, in der der *Beam* anfängt (entweder von Punkt innerhalb der Zelle, oder vom Rand).
2. Für jedes Polygon am Rand: generiere gespiegelten *Beam*, falls Polygon getroffen, und schneide *Beam* ab.
3. Falls etwas "übrigbleibt" vom *Beam*, beginne wieder von vorne in der Nachbarzelle.
4. *Rekursion* mit den gespiegelten und den "übriggebliebenen" *Beams*.

Gesamtsystem

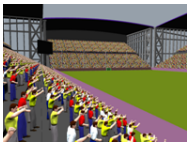


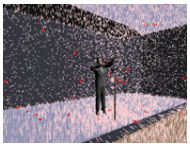
- Eigenschaften des Algorithmus':
  - Schnell
  - Auch Beugung schnell berechenbar
  - Gut parallelisierbar



Stochastisches Sound-Rendering

- Szenen mit vielen Schallquellen (100k)
  - Durch Spiegelquellenmethode
  - Stadium, Fußgängerzone, ...





- Mixing $$s(t) = \sum_i a_i s_i (t - \tau_i)$$

  läßt sich nicht mehr für alle mit 40 kHz durchführen

- Wähle k Samples aus n:

  $\pi : \{1, \cdots, k\} \rightarrow \{1, \cdots, n\}$

  schätze Gesamtsignal s(t) durch:

  $$\tilde{s}(t) = \frac{n}{k} \sum_{i=1}^{k} \frac{a_{\pi(i)} s_{\pi(i)} (t - \tau_i)}{p_{\pi(i)}}$$

  $p_{\pi(i)}$ = Wahrscheinlichkeit, dass Quelle i ausgewählt wird

Importance Sampling

- Wahl der Wahrscheinlichkeiten:
  - Optimal wäre:

    $$p_i = a_i s_i (t)$$

  - Vereinfachung:

    $$p_i = \frac{a_i}{\sum_{i=1}^{n} a_i}$$

- Dynamische Veränderungen in der Szene:
  - Listener bewegt sich
  - Schallquelle bewegt sich (oder rotiert und Abstrashlchar. nicht kugelförmig)
  - Occluder oder Reflektoren bewegen sich
  - Die a_i ändern sich, damit auch die p_i

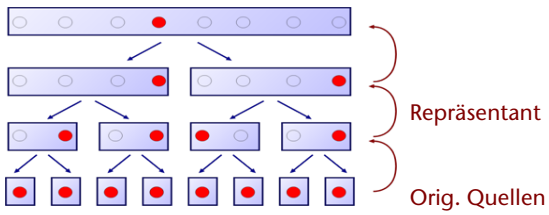
Dynamisches Importance Sampling

1. Volles Resampling:
   - Schätzung $\hat{s}$ ist nicht exakt
   - Umschalten auf neues *Sample-Set* hörbar
2. Stochastic diffusion:
   - Wechsle pro Zeiteinheit nur einen gewissen Anteil des *Sample-Sets*
   - *Fade-out*, dann ersetzen durch neue zufällige Quelle, dann Fade-in
   - Weniger, aber immer noch Artefakte
3. Adaptive diffusion:
   - Passe Änderungsrate des *Sample-Sets* der Änderungsrate der a_i an
   - Falls sich mehr als ε geändert hat:
     - Fall "zu klein": *fade-out*, ersetzen, fade-in
     - Fall "zu groß": anderes *Sample* ausfaden und ersetzen durch zufälliges *Sample*

- Problem: immer noch O(n) Zeit, da Auswahl gemäß p_i irgendeine Form der Durchmusterung benötigt

Hierarchisches Sampling

- Für statische Szenen
- *Octree* über Schallquellen:



- Repräsentant = einer der Repräsentanten der 8 Kinder (Wahrscheinlichkeit ~ Volumen)
- Volumen des Repräsentanten = Summe der Volumen der Kinder

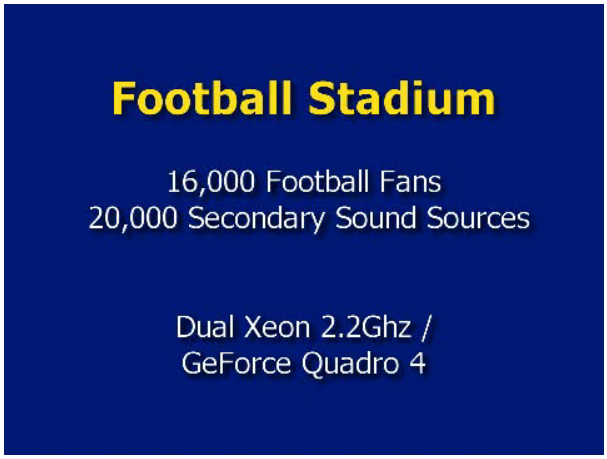
Prioritätsgesteuerte Traversierung

- Priorität = Lautstärke des Repräsentanten x Dämpfung
- Algorithmus:

```
repeat
    hole Top aus Queue
    füge Top zur Liste der k Samples
    füge Kinder von Top in Queue
until k Elemente in Liste
```

- Laufzeit: O(k log k)
- Implementiert *Importance Sampling*
- Und Stratifizierung ("*stratification*" = "gleichmäßigere Auswahl")
- Nachteile:
  - Nur Empfängercharakteristik und Dämpfung durch Distanz
  - Sichtbarkeit und Emissionscharakteristik werden ignoriert
- Lösung:
  - Wähle 10,000 Kandidaten mit hierarchischem *Sampling*
  - Wähle daraus mit dynamischem *Sampling* 100-1000 *Samples*

Beispiele



13 versch. *Soundtracks*, zufällige Phasenverschiebung,
16 bit, 44.1 kHz, Mono; Mixer erzeugt Stereo.
Sekunärquellen durch *Photon-Tracing* erzeugt (*diffuse Reflexion*)

Violin Performance

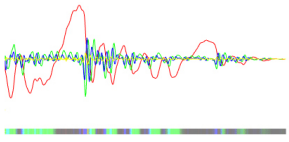
1 Primary Source
50,000 Secondary Sound Sources

Dual Xeon 2.2Ghz /
GeForce Quadro 4

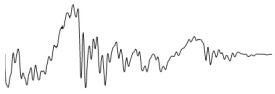
Sekundärquellen durch Spiegelquellenmethode

Audio-Processing on the GPU

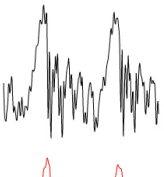
- Mixer:
  - Audiodaten verschieben (Zeitverzögerung)
  - Strecken oder Stauchen (Dopplereffekt)
  - n Frequenzbänder gewichten und aufsummieren (pro Schallquelle)
- Idee: verwende GPU
  - Audiodaten = 1D-Textur (4 Frequenzkanäle → RGBA)
  - Texturkoord.verschiebung = Zeitverzögerung
  - Farbtransformation = Frequenzmodulation
  - Skalierung der Texturkoord. = Dopplereffekt

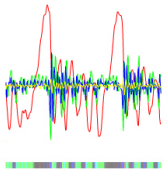


Gestreckte Textur



Neues Signal (Dopplerv.)





- HRTF ("*head-related transfer function*" = Empfängercharakteristik) = Cubemap
- Mixen = 4D-Skalarprodukt

Head related transfer function (HRTF) encoded as an RGBA cubemap for efficient access by the GPU.

- Vergleich:
  - SSE-Implementierung auf 3 GHz Pentium 4
  - GPU-Implementierung auf GeForce FX 5950 Ultra, AGP 8x
  - *Audio-Processing*: Frames der Länge 1024-Samples, 44.1 kHz, 32-bit floating point.
  - Resultat:
    - GPU ca. 20% langsamer,
    - Zukünftige GPUs evtl. ca. 50% schneller
    - Bus-Transfer unberücksichtigt