Virtual Reality & Physically-Based Simulation

Sound Rendering

G. Zachmann

University of Bremen, Germany

cgvr.cs.uni-bremen.de

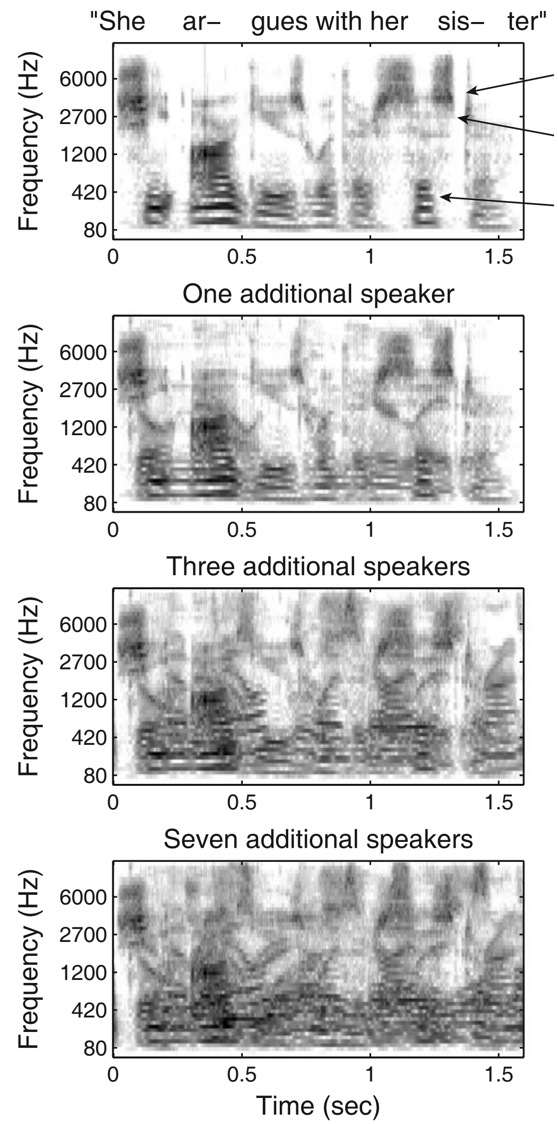
Sound in VR

- Reminder: complete immersion = *all* human senses are being stimulated *consistently*
  - (Counter-example: watch a thriller on TV without sound! ☺)
- Some of the functions of our auditory sense:
  - Localization of not (yet) visible sound sources (e.g., predator)
  - Our second most important sense for navigation (e.g., in buildings)
  - Allows us to obtain a sense of the space around us
- Auditory impressions (Höreindrücke):
  - "Big" (late reverberations),
  - "cavernous" (lots of echos),
  - "muffled" (no reverberations),
  - "outside" (no echos, buts many other sounds).
- How to render sound: in the following, only "*sound propagation*" (no *sound synthesis*)

Digression: the Cocktail Party Effect

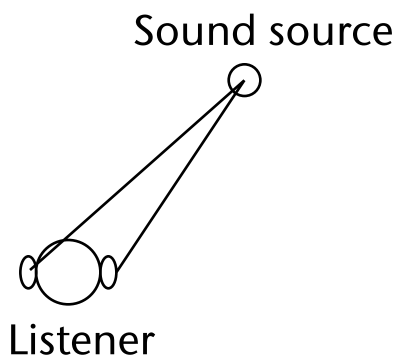
- Describes the ability of humans to extract a single sound source from a complex auditory environment (e.g., cocktail party)
- Challenges:
  - Sound segregation
  - Directing attention to sound source of interest

Breakfast at Tiffany's (Paramount Pictures)



Naïve Methods

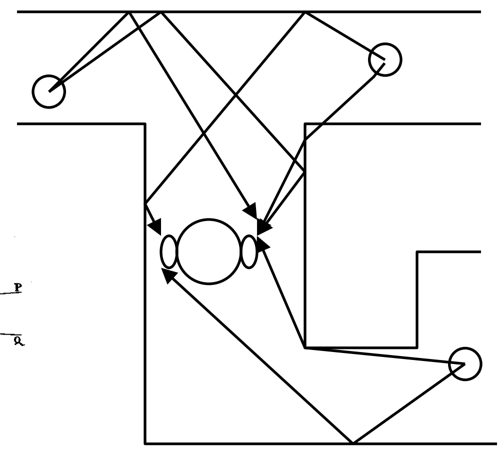
- Sound in games, e.g., noise from motors or guns
  - Sound = function of speed, kind of car/gun, centrifugal force, orientation of plane, etc.
  - Sounds have been sampled in advance
- Simple "3D" sound:
  - Compute difference in volume of sound that reaches both ears
  - Possibly add some reverberation (Nachhall)

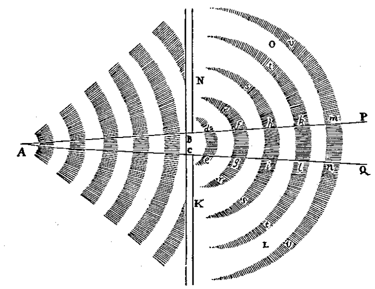


- But how to render sound *realistically* and in *real-time*?
- Differences between light and sound:
  - Velocity of propagation (sound = 343 m/s in air)
  - Wavelength
- ➢ Makes rendering of sound so difficult

Human Factors

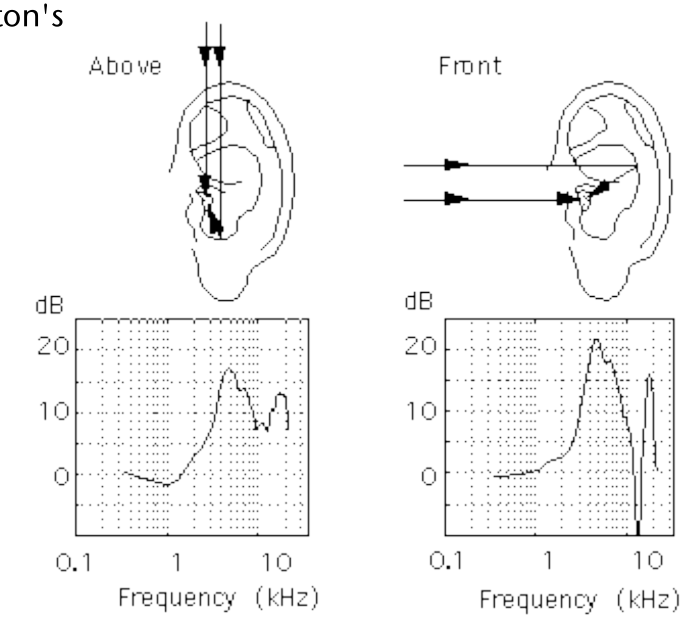
- Akustische Effekte:
  - Reflexion (*reflection*),
  - Brechung (*refraction*),
  - Streuung,
  - Beugung (*diffraction*),
  - Interferenz.

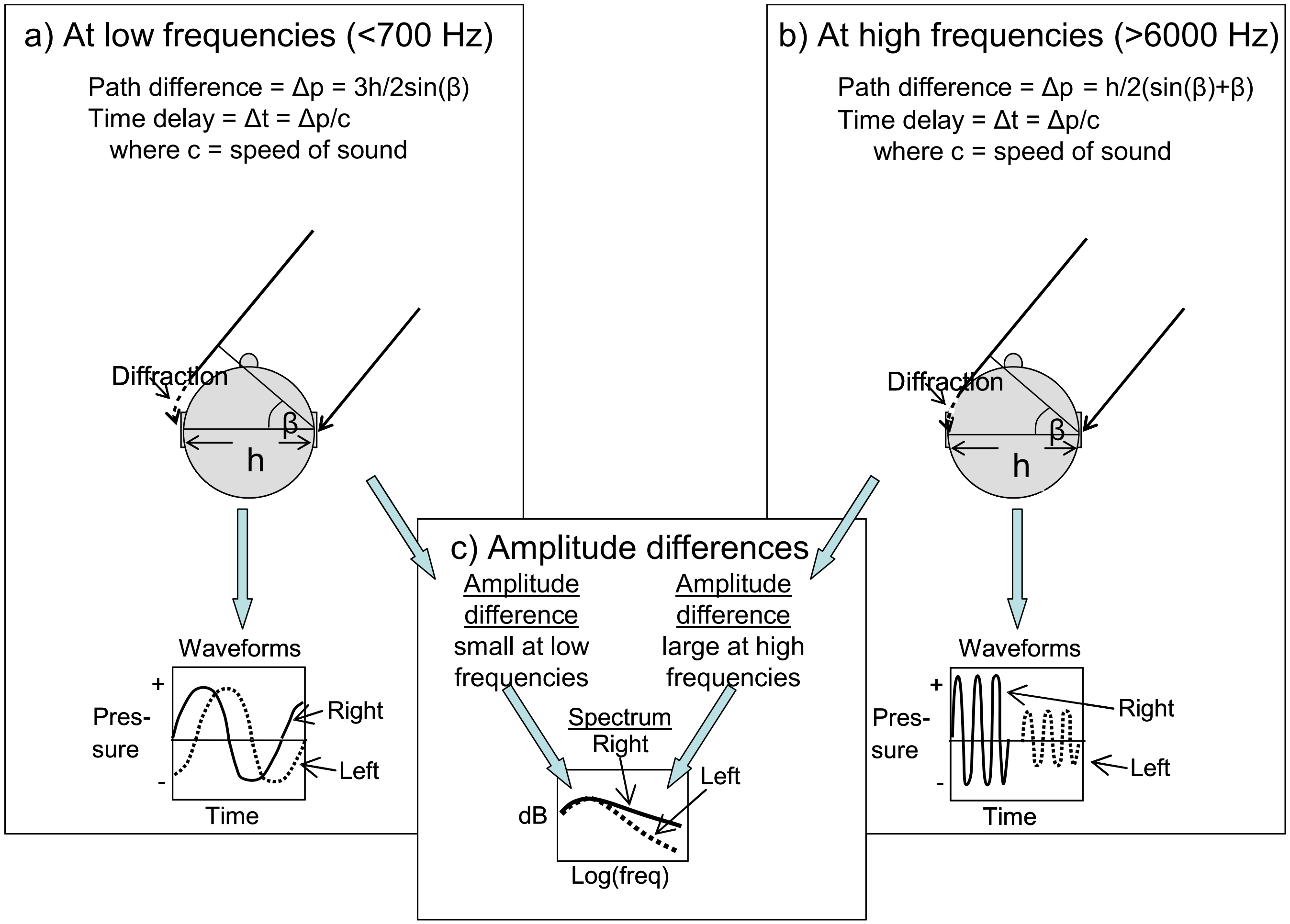




from Isaac Newton's
Principia (1686)

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





Wellenlänge groß relativ zur Kopfgröße →
starke Beugung um Kopf →
nur Zeitverschiebung zwischen den Ohren

Wellenlänge klein relativ zur Kopfgröße →
schwache Beugung um Kopf →
Zeitverschiebung **und Lautstärkedifferenz** zwischen den Ohren

Mischen von Schallquellen

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

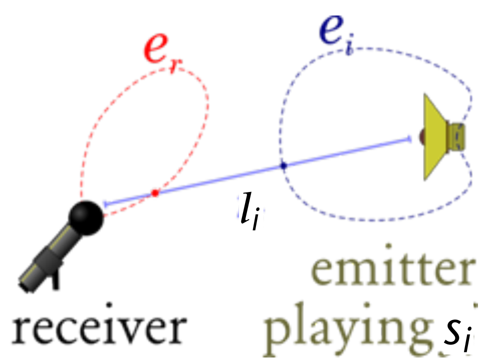
$$a_i = \frac{e_i(\phi_i, \omega_i) e_r(\phi_i, \omega_i) v_i}{l_i}$$

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

$$e_r = \text{Empfängerchar.}$$

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

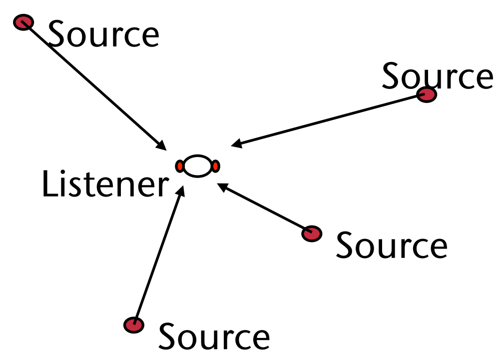
$$l_i = \text{Entfernung}$$



- Summe:

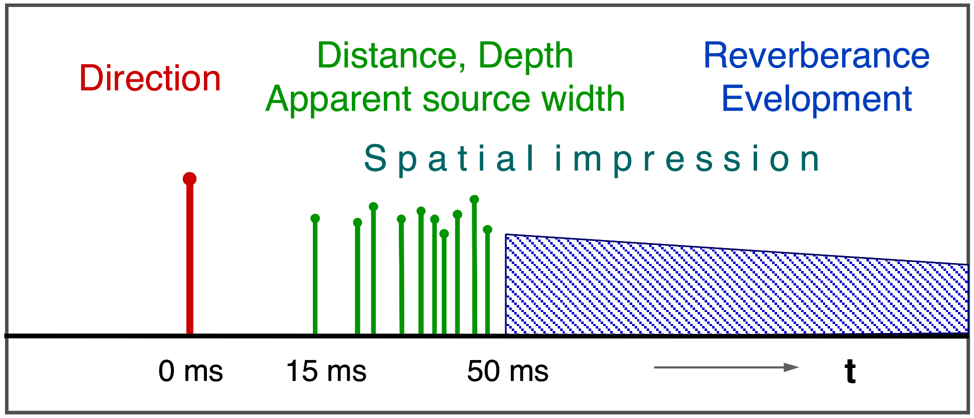
$$s(t) = \sum_i a_i s_i(t - \tau_i)$$

$$\tau_i = l_i / c \ , \qquad c = \text{Schallgeschwindigkeit}$$



Eine psychoakustische Erkenntnis

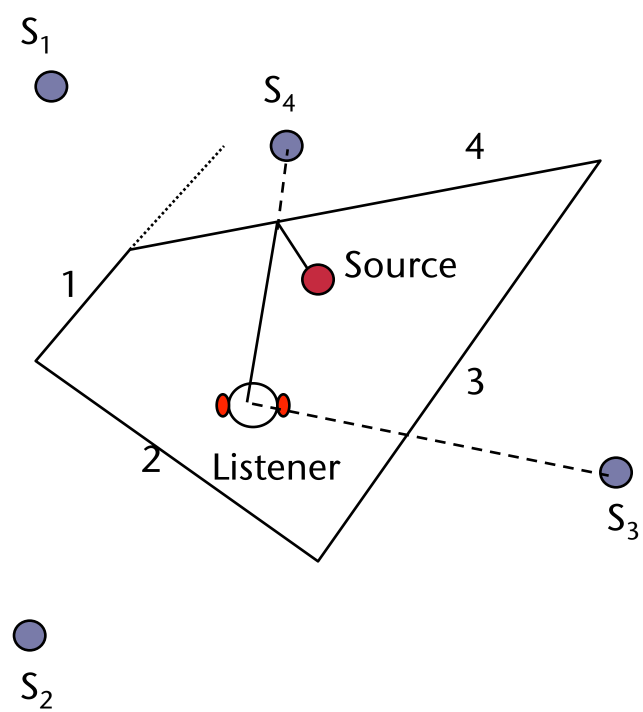
- Der Einfluß von "frühem" und "spätem" Schall:
  - Direkter Schall → Aufschluß über Richtung der Schallquelle
  - Early reflections (nur wenige Reflexionen) → Aufschluß über Entfernung & "Breite" der Source
  - Late reflections (reverberations) → Informationen über den Raum



- Fazit: man muss eine große Zahl von Schall-Pfaden berechnen!

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

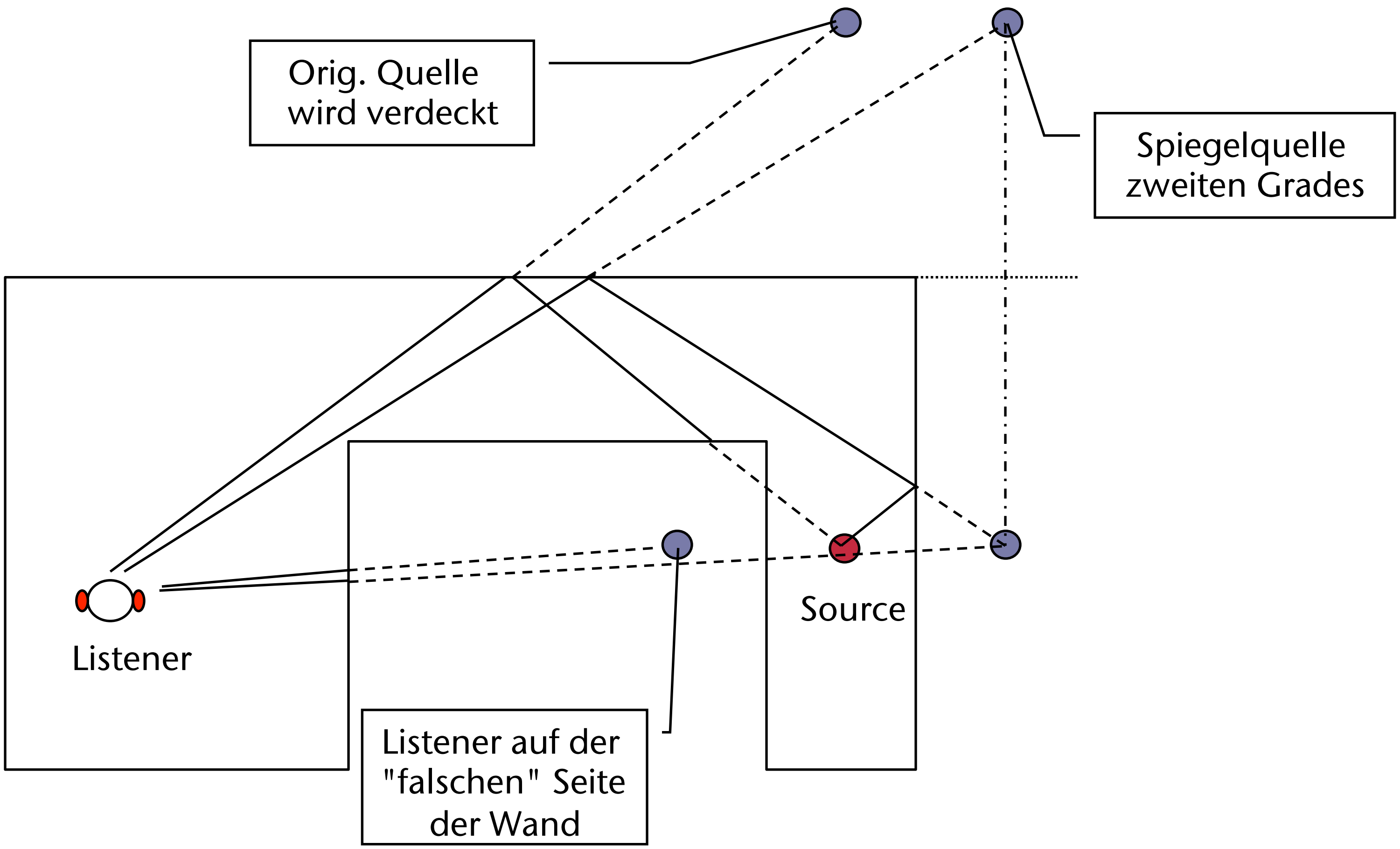


- Aufwand: $O(\,n{\cdot}r\,)$

  n = # Schallquellen,

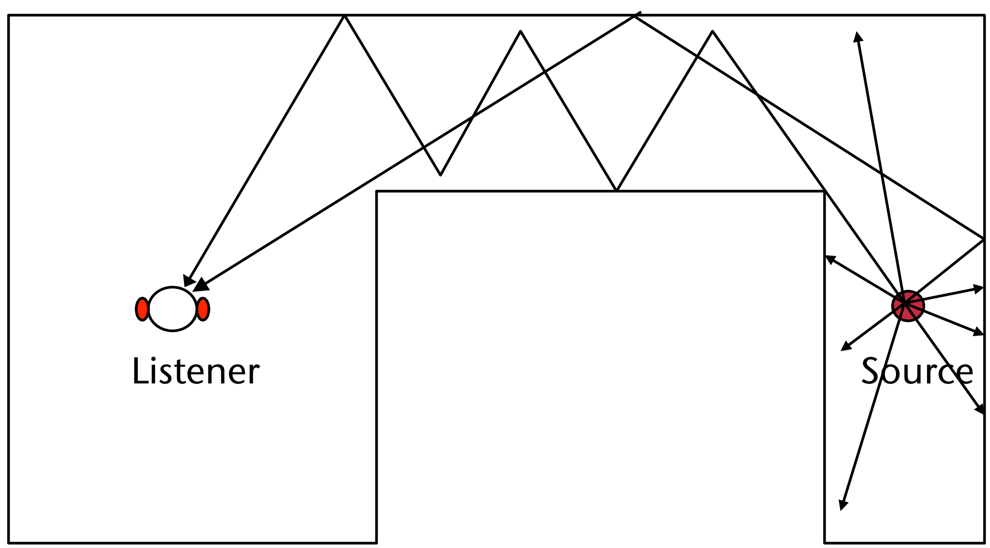
  r = # Wände.
- Probleme:
  - Alles neu berechnen, wenn sich Quellen bewegen
  - Nur Reflexionseffekt

- Beispiel (nicht alle Spiegelquellen eingezeichnet):



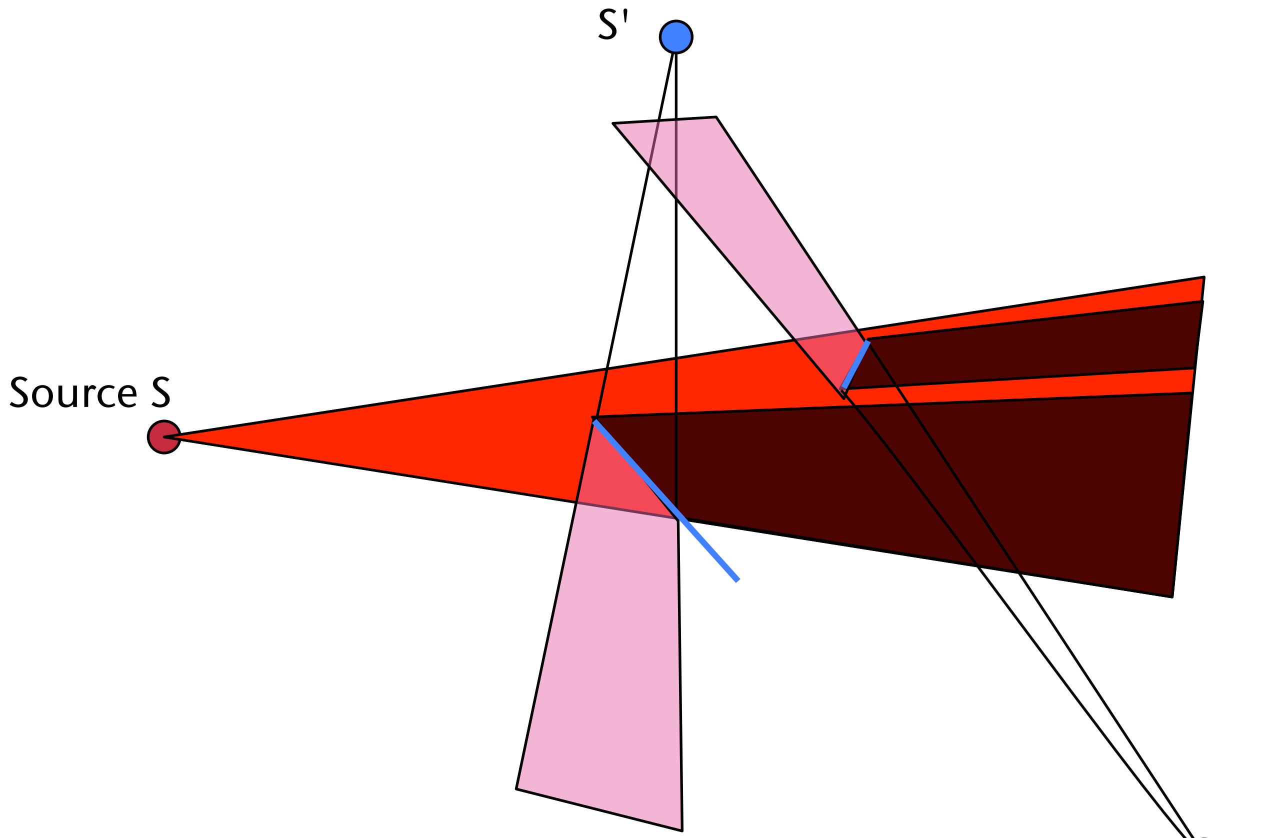
Ray Tracing Methods

- 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*


S'
Source S
S''

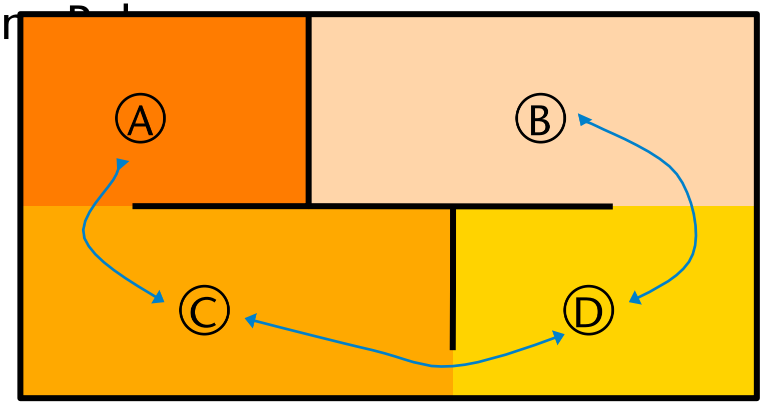
Sound-Rendering mit Beams

- Alle *Beams* berechnen: zu jedem *Beam* speichere:
  - zugehörige Quelle/Spiegelquelle;
  - spiegelndes Polygon;
  - *Vater-Beam*;
  - Anzahl Reflexionen;
  - ursprüngliche Quelle;

  → "*Beam tree*".
- 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.

Beschleunigung des *Beam-Tracings*

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

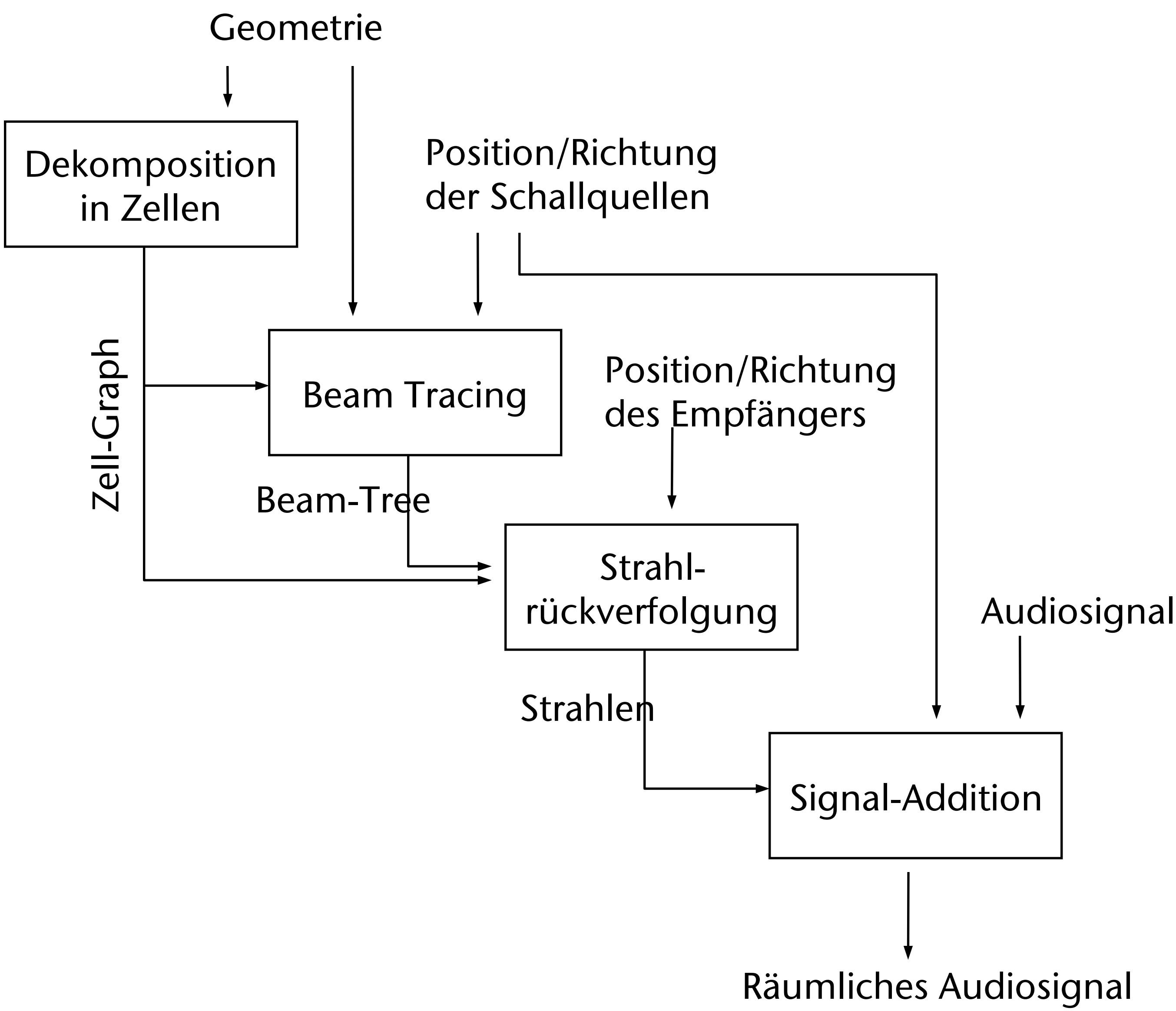


Beam-Tracing mit Zellen

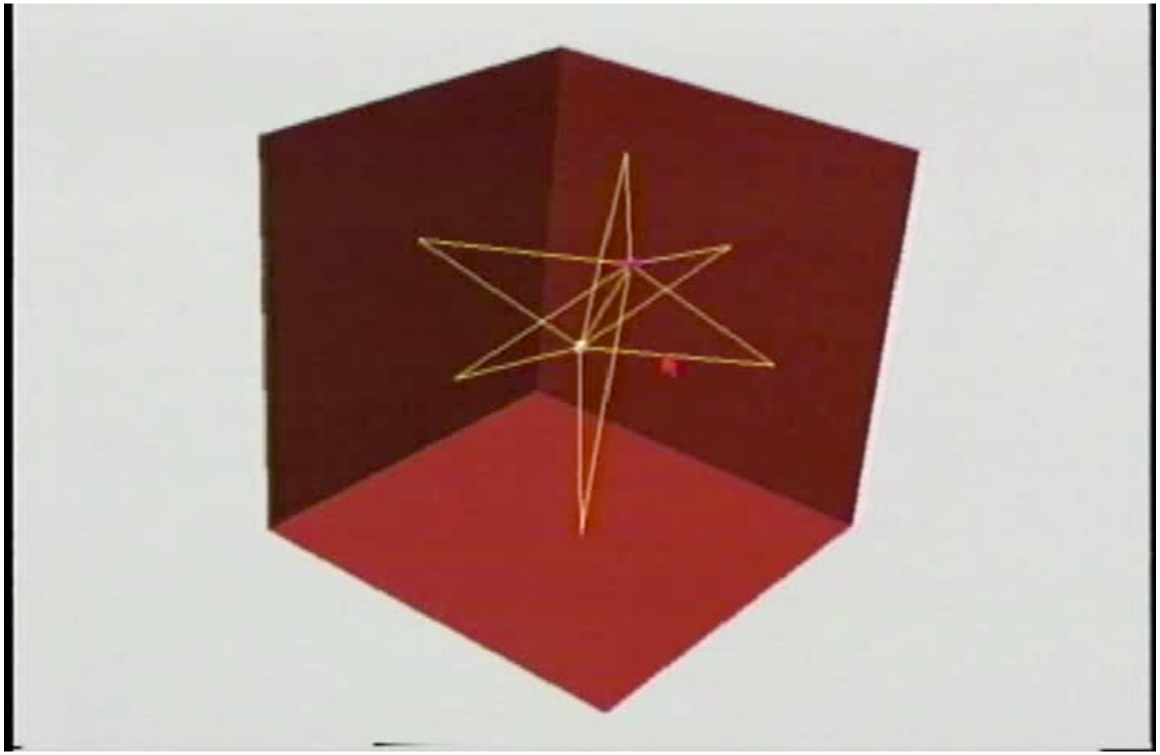
Gegeben Quelle, wie wird *Beam* "verschossen" und "beschnitten"?

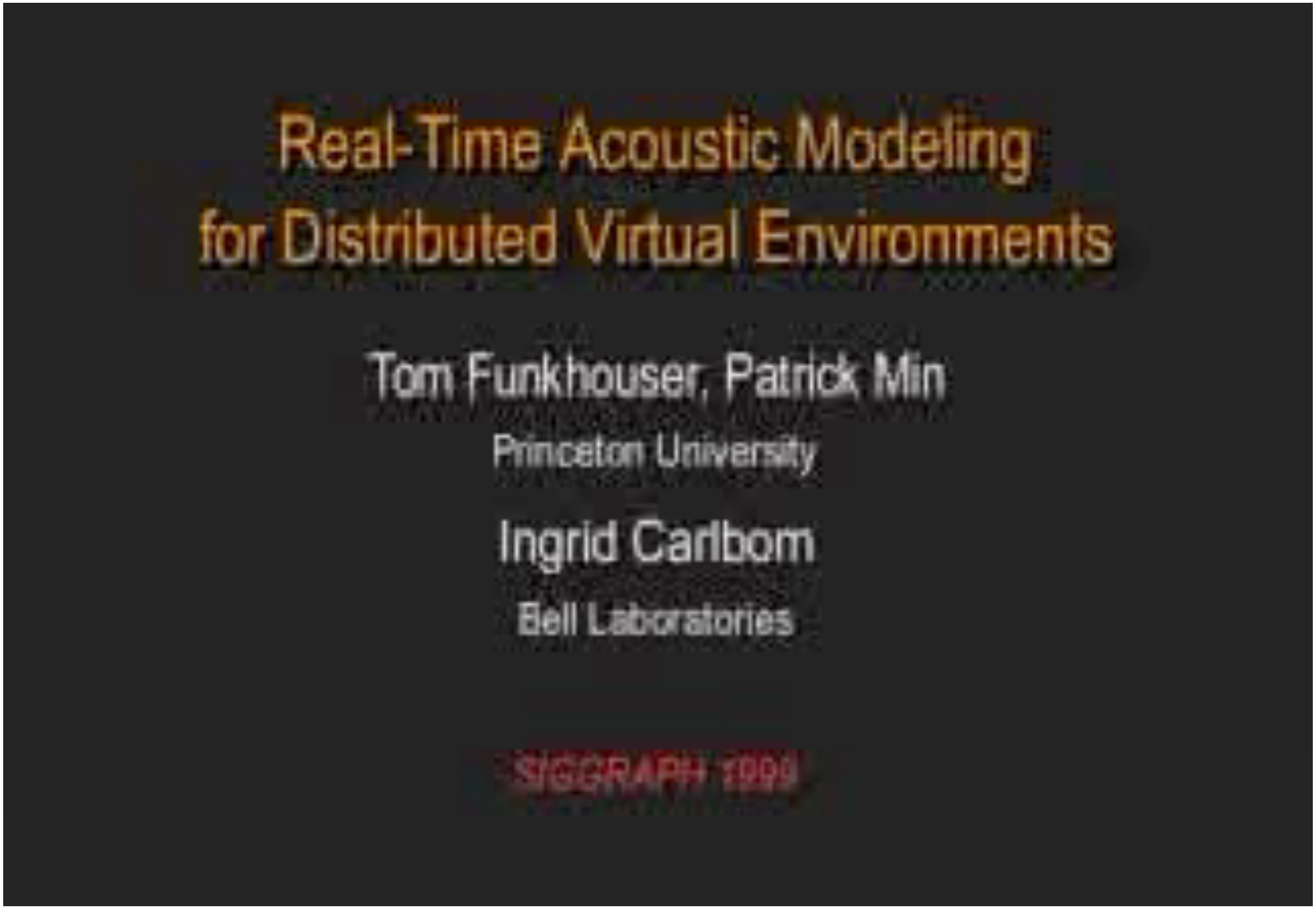
1. Bestimme Zelle, in der der *Beam* anfängt
   (entweder von Punkt innerhalb der Zelle, oder vom Rand).
2. Für jedes Polygon am Rand: generiere gespiegelten *Beam*, falls Polygon getroffen, und schneide *Beam* ab.
3. Falls etwas "übrigbleibt" vom *Beam*, beginne wieder von vorne in der Nachbarzelle.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

Gesamtsystem



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




Real-Time Acoustic Modeling
for Distributed Virtual Environments
Tom Funkhouser, Patrick Min
Princeton University
Ingrid Carlbom
Bell Laboratories
SIGGRAPH 1999

Exkurs: Monte-Carlo-Integration

- Aufgabe: bestimmtes Integral berechnen

$$E = \int_a^b f(x)\,\mathrm{d}x$$

(wobei *f* zu "schwierig" zum analytischen Berechnen des unbestimmten Integrals sei)

- Triviale numerische Näherung: *f* auf einem Gitter auswerten

$$E_N = \sum_{i=1}^{N} h \cdot f(a + i \cdot h)$$

wobei $h = \frac{b-a}{N}$

- Wenn $N \to \infty$, dann geht $E_N \to E$
- Alternative Betrachtung:

$$E_N = (b-a) \sum_{i=1}^{N} \frac{1}{N} f(x_i) \quad , x_i = a + i \cdot h$$

d.h., jedes $f(x_i)$ wird mit $\frac{1}{N}$ gewichtet

- Das selbe funktioniert auch für mehrdimensionale Integrale, d.h., dass $f : \mathbb{R}^d \to \mathbb{R}$:

$$E = \int_\Omega f(x)\,\mathrm{d}x \approx \mathrm{Vol}(\Omega) \sum_{i=1}^{N} \frac{1}{N} f(x_i)$$

wobei die x_i (insgesamt N Stück) wieder auf einem regelmäßigen Gitter innerhalb von Ω angeordnet sind
- Probleme:
  - Aufwand steigt exponentiell mit d, weil Vol(Ω) = Seitenlänge^d
  - Sampling von f ist schwierig, falls das Gebiet Ω eine nicht-kanonische Form hat
- Lösung: Monte-Carlo-Integration = sample f an zufälligen, gleichverteilten Positionen x_i :

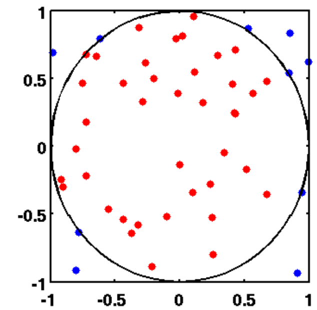
$$E_N = (b-a) \sum_{i=1}^{N} \frac{1}{N} f(x_i)$$

- Wie wahrscheinlich ist es, dass E_N "nah" bei E liegt?
- Beobachtung: wenn die $\{x_i\}$ zufällig gezogen sind, ist auch E_N eine Zufallsvariable, die eine bestimmte Wahrscheinlichkeits-dichtefunktion haben muß
- Zentraler Grenzwertsatz (*central limit theorem*) sagt:
  wenn N groß ist, dann folgt E_N ungefähr der Gauss'schen Normalverteilung, mit Mittelwert E, und Standardabweichung

$$\sigma_N^2 = \frac{\frac{b-a}{N} \sum_{i=1}^{N} f^2(x_i) - E_N^2}{N-1}$$

- Mit W'keit 0.95 liegt E_N im Intervall $[E-\sigma_N, E+\sigma_N]$
- Die Standardabweichung σ_N sagt also etwas über die Unsicherheit unserer Schätzung von E aus

- Vorteil von Monte-Carlo-Integration:
  - Man kann N beliebig steuern (muss nicht mehr n^d sein)
  - Beliebige Gebiete lassen sich leicht sampeln & integrieren, falls der Test x $\in \Omega$ einfach ist, z.B. Integration über eine Kugel



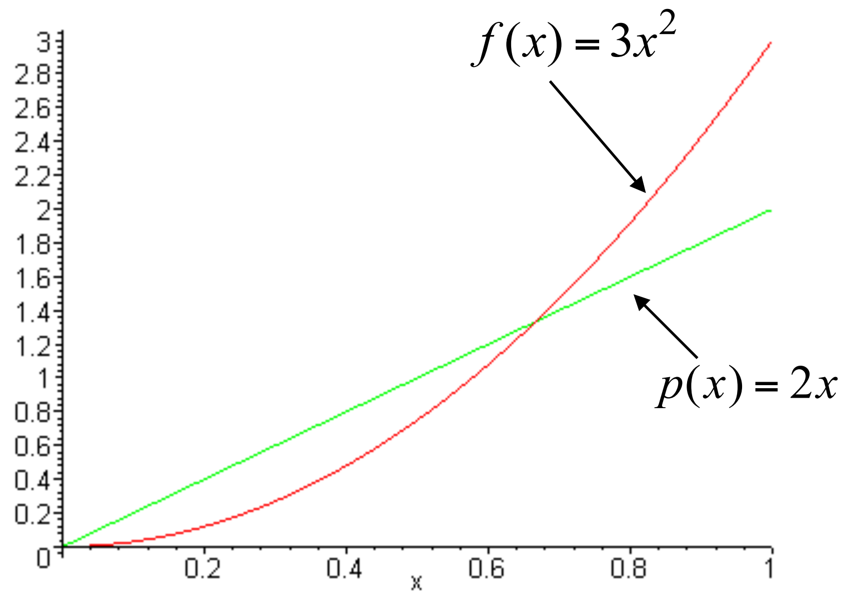
- Weitere Probleme:
  der Beitrag manchener / vieler Sample-Punkte x_i zu $\sum_{i=1}^{N} \frac{1}{N} f(x_i)$ kann sehr gering sein
- Ziel: versuche nur dort zu sampeln, wo *f* groß ist

→ Importance Sampling

- Beispiel: $$E = \int_0^1 3x^2 \, dx$$
- Der größte Beitrag kommt von x-Werten nahe 1
- Wähle Sample-Stellen nicht uniform, sondern gemäß Dichtefunktion $p(x)$, z.B.

$$p(x) = 2x$$

- Frage: wie muß das Integral geschrieben werden, um den *Bias* (hier: "Vorliebe" für bestimmte Sample-Region) zu kompensieren?



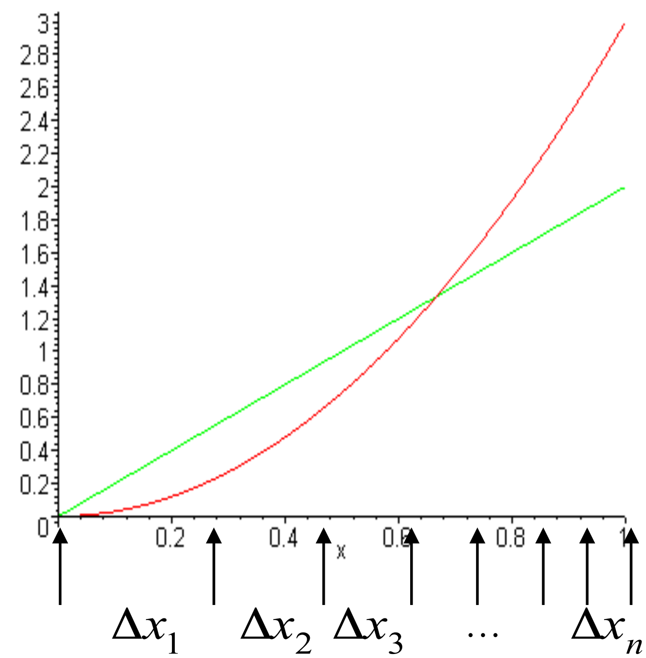
- Eine (eindimensionale) Plausibilitätsbetrachtung:
  - Betrachte numerische Integration mit Rechteck-Regel:

$$E_N = \sum_{i=1}^{N} \Delta x_i \cdot f(x_i)$$

  - Ein Δx_i = Kehrwert der lokalen Dichte der Sample-Punkte:

$$\Delta x_i = \frac{b-a}{N} \cdot \frac{1}{p(x_i)}$$

Größeres p
→ mehr Sample-Punkte
→ kleineres Δx_i



- MC-Integration mit Importance-Sampling:

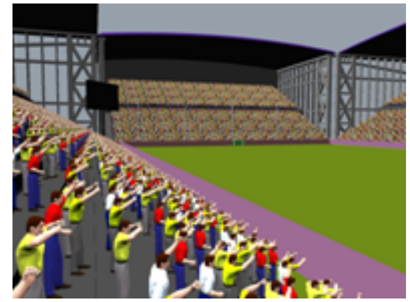
$$E_N = \frac{b-a}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

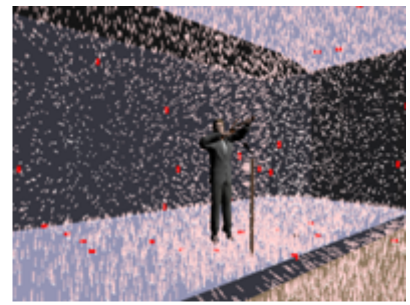
  wobei die x_i zufällig gemäß der Wahrscheinlichkeitsdichtefunktion $p(x)$ gewählt werden

- Mehrdimensionale funktioniert es ganz analog:

$$E_N = \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

Stochastisches Sound-Rendering





- Szenen mit vielen Schallquellen (z.B. 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 Abstrahlcharakteristik ist nicht kugelförmig)
  - Occluder oder Reflektoren bewegen sich
  - Die a_i ändern sich, damit auch die p_i

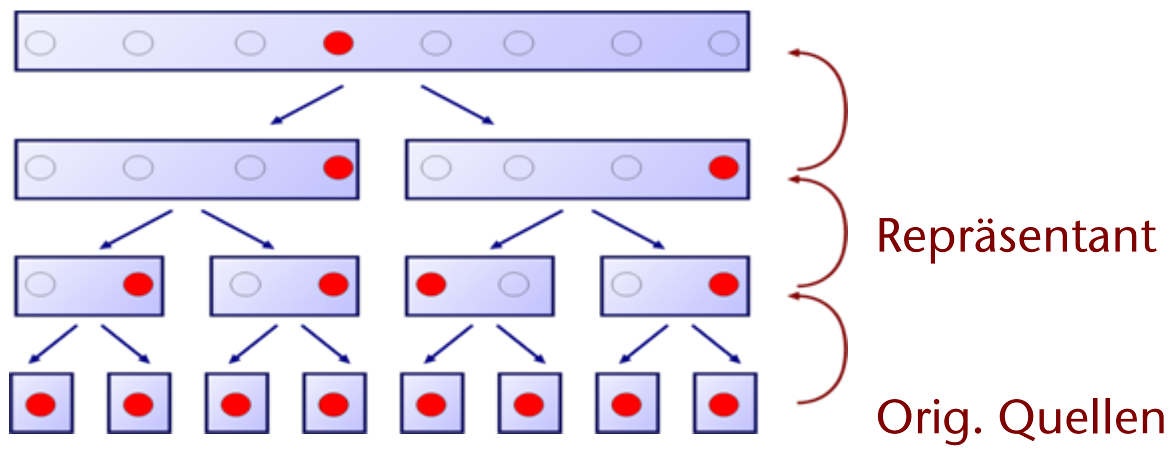
Dynamisches Importance Sampling

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:



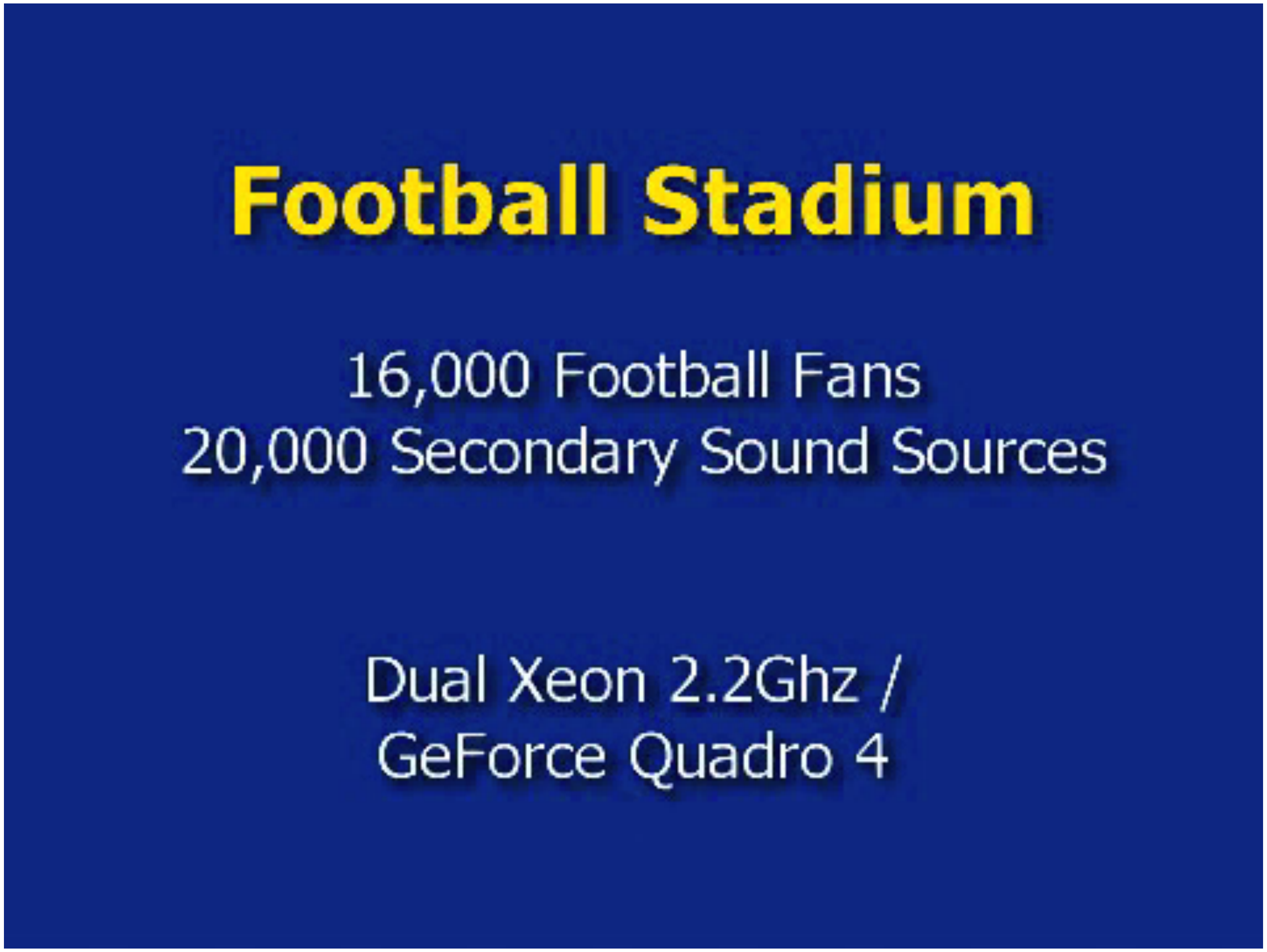
- Repräsentant = einer der Repräsentanten der 8 Kinder (Wahrscheinlichkeit ~ Volumen)
- Volumen des Repräsentanten = Summe der Volumen der Kinder

Prioritätsgesteuerte Traversierung

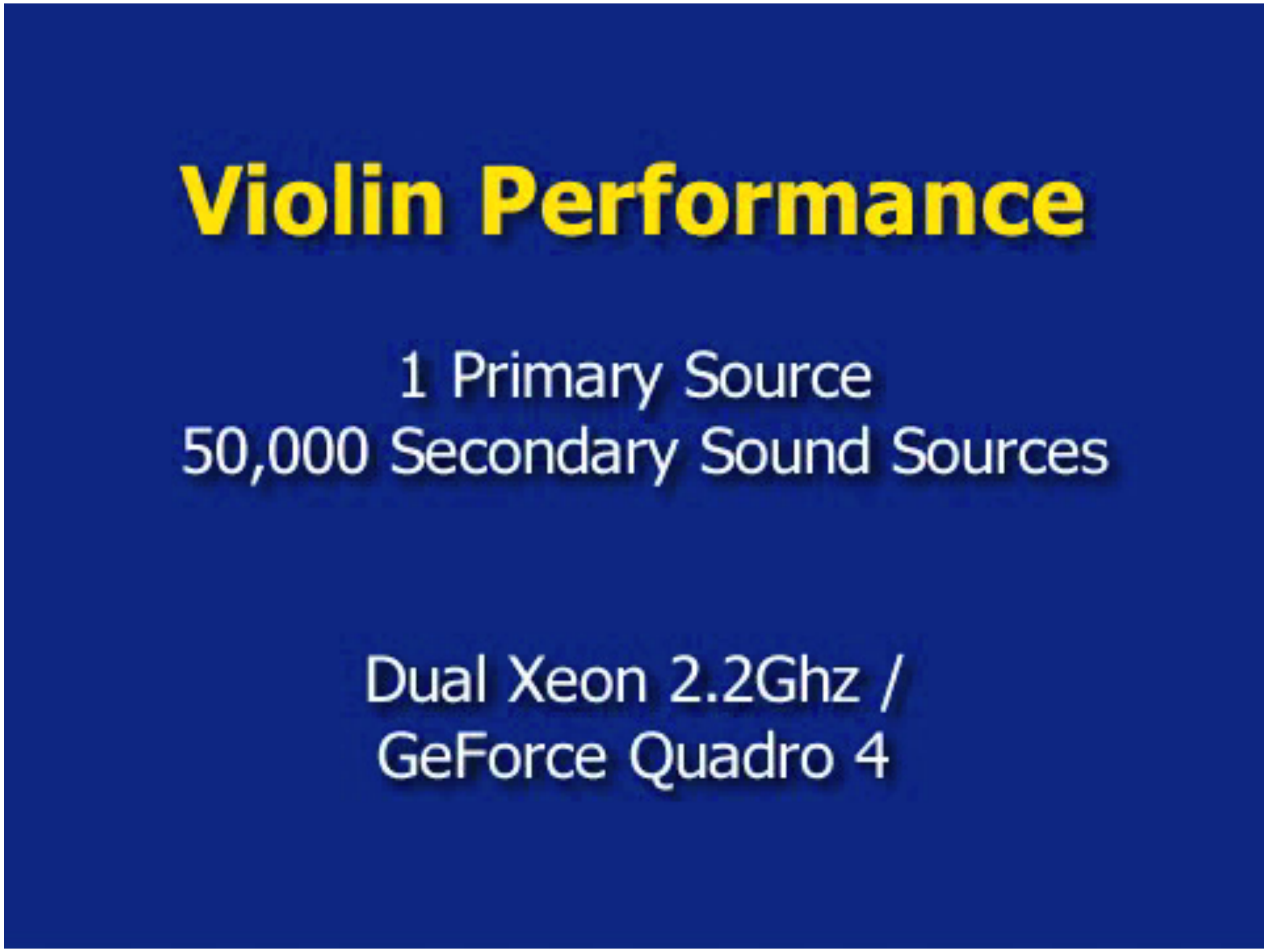
- Priorität = Lautstärke des Repräsentanten x Dämpfung
- Algorithmus:

```
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*



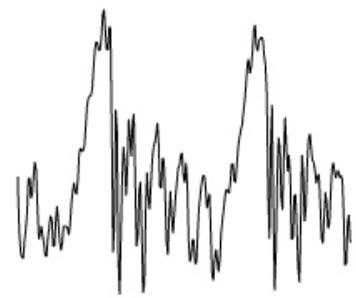
13 versch. *Soundtracks*, zufällige Phasenverschiebung, 16 bit, 44.1 kHz, Mono;
Mixer erzeugt Stereo.
Sekundärquellen durch *Photon-Tracing* erzeugt (*diffuse Reflexion*)

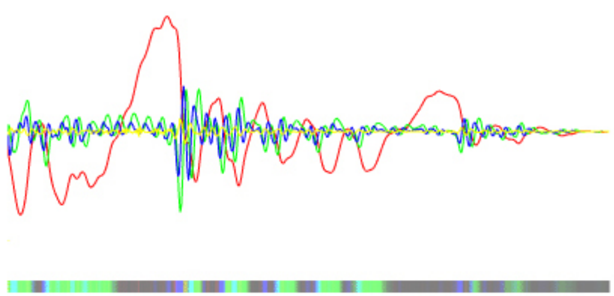


Sekundärquellen durch Spiegelquellenmethode

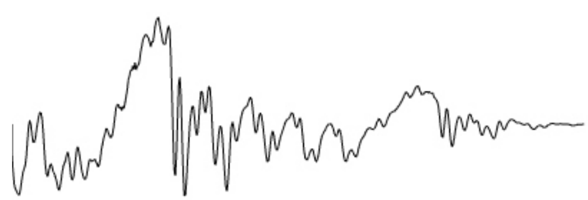
Audio-Processing on the GPU

- Aufgaben des 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

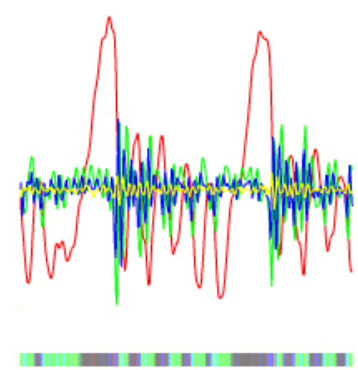




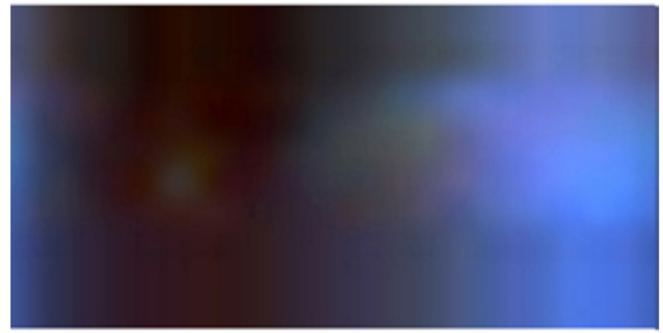
Gestreckte Textur



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