Virtual Reality & Physically-Based Simulation
Sound Rendering

G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Why/Where is Sound/Audio Important?

https://www.menti.com/7wucj265rw

Sound in VR

- Reminder: complete immersion = *all* human senses are being stimulated *consistently*
  - (Counter-example: watch a thriller or action movie 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*)

Physical Factors of Sound Propagation

- Differences between light and sound:
  - Velocity of propagation (sound = 343 m/s in air)
  - Wavelength
- Makes rendering of sound so difficult
- Physical effects that need to be simulated (ideally):
  1. Reflection,
  2. Refraction (Brechung),
  3. Scattering (Streuung),
  4. Diffraction (Beugung),
  5. Interference.

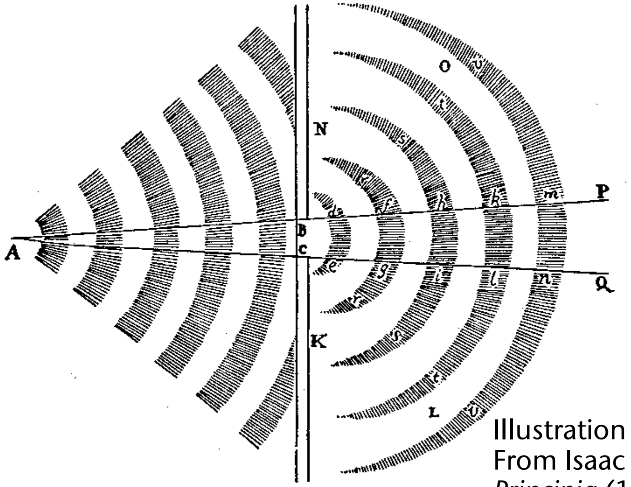
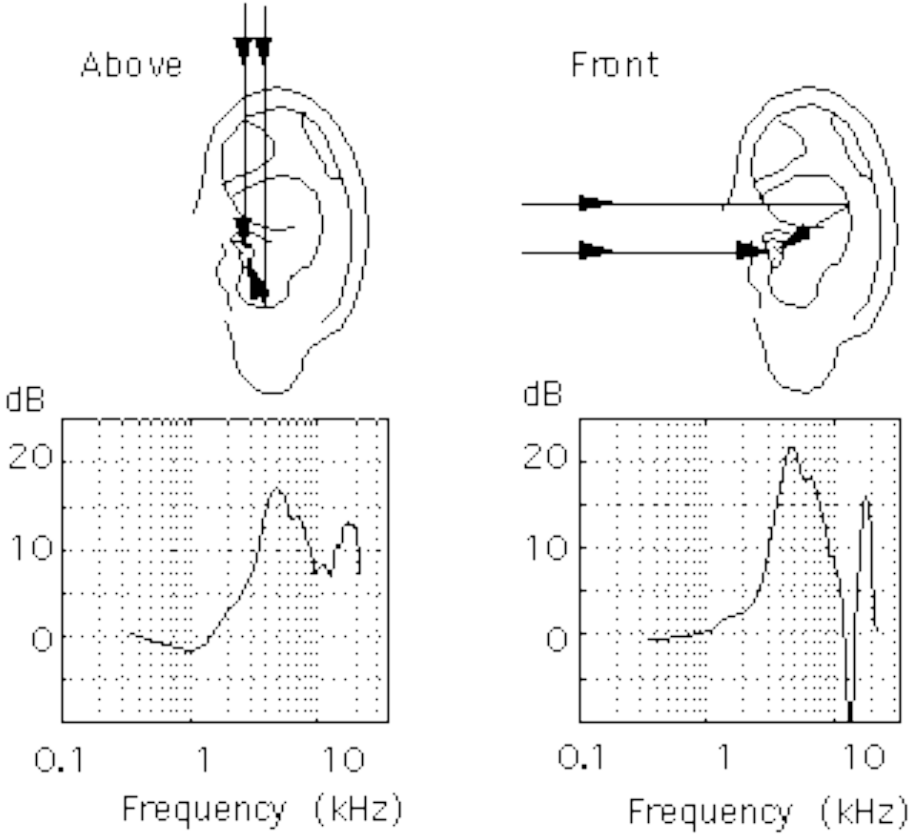


Illustration for diffraction. From Isaac Newton's *Principia* (1686)

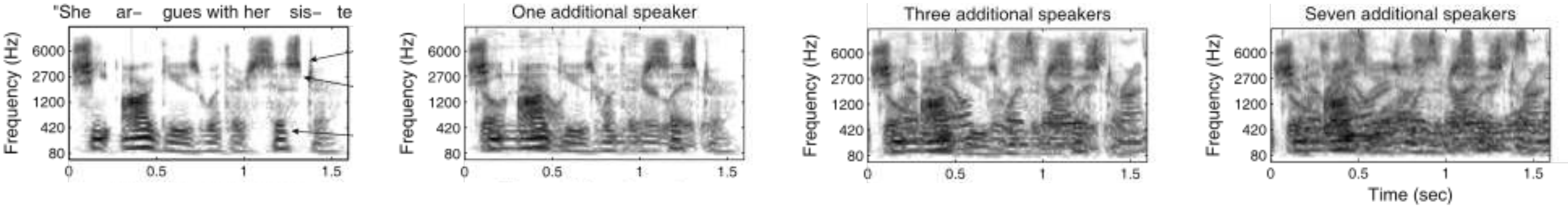
Human Factors

- Our ears (including cochlea and brain) are wonder-sensors regarding localization of sound sources
- Because we have two ears, the brain can compute:
  - Difference in amplitude
  - Difference in time of arrival
- Changes in the spectrum caused by the auricle (Ohrmuschel) and head inform the brain about the direction of the sound source



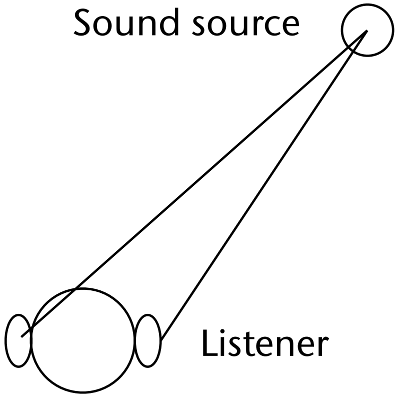
Digression: the Cocktail Party Effect

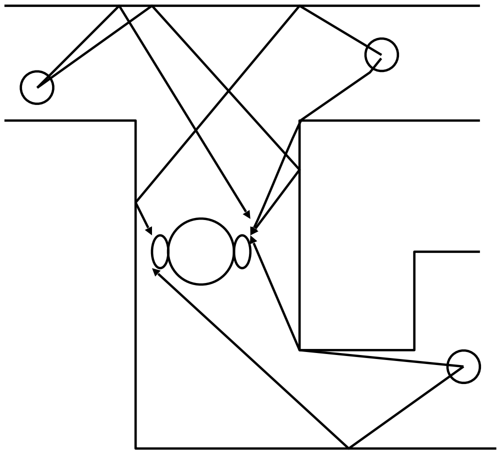
- Describes the ability of humans to extract a single sound source from a complex auditory environment (e.g., cocktail party)
- Challenges:
  - Sound segregation
  - Directing attention to sound source of interest


"She ar– gues with her sis– te
One additional speaker
Three additional speakers
Seven additional speakers
Frequency (Hz)
6000
2700
1200
420
80
0
0.5
1
1.5
Time (sec)

Naïve Sound Propagation Methods

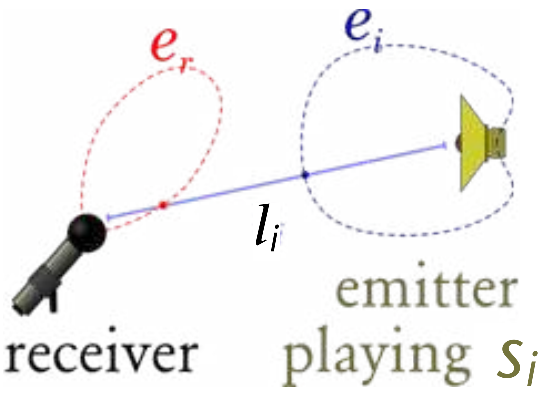
- Sound in games, e.g., noise from motors or guns
  - Sound = function of speed, kind of car/gun, centrifugal force, orientation of plane, etc.
  - Sounds have been sampled in advance
- Simple "3D" sound:
  - Compute difference in volume of sound that reaches either ear
  - Possibly add some reverberation (Nachhall)
- But how to render sound *realistically* and in *real-time*?
- What we need to simulate, is this:





Mixing Sound Sources

- Given: *n* sound sources in different directions, each one sends signal $s_i(t)$
- Attenuation (Dämpfung), a_i , of each source s_i is

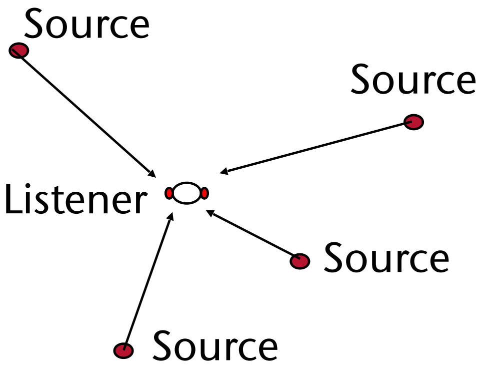


$$a_i = \frac{e_i(\phi_i, \omega_i) e_r(\phi_i, \omega_i) v_i}{l_i}$$

$$\begin{aligned} e_i(\phi, \omega) &= \text{sound source radiation characteristic,} \\ &\quad \text{e.g., } e_i = \tfrac{1}{2}(1 + \cos(\phi)^\alpha) \\ e_r &= \text{receiver characteristic} \\ v_i &= \text{visibility} \in [0,1] \quad \text{(includes diffraction)} \\ l_i &= \text{distance} \end{aligned}$$

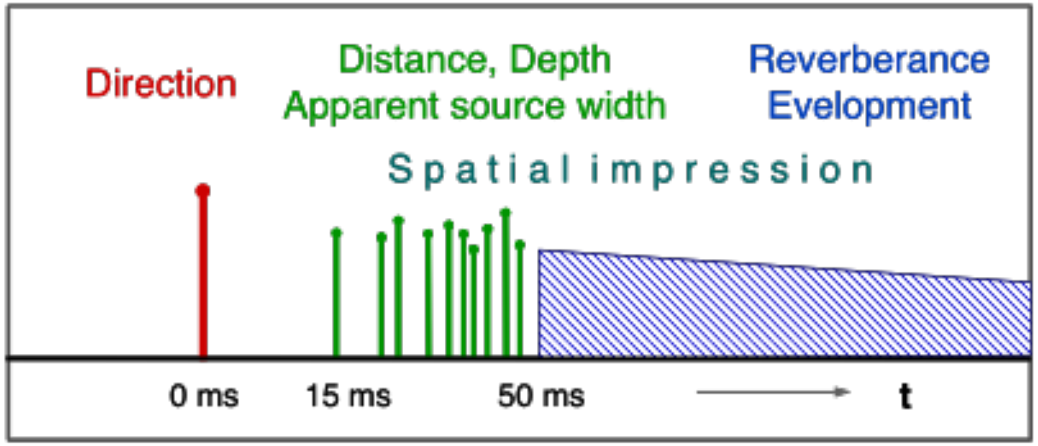
- Accumulation yields $s(t) = \sum_i a_i s_i(t - \tau_i)$

  with $\tau_i = l_i / c$, c = speed of sound



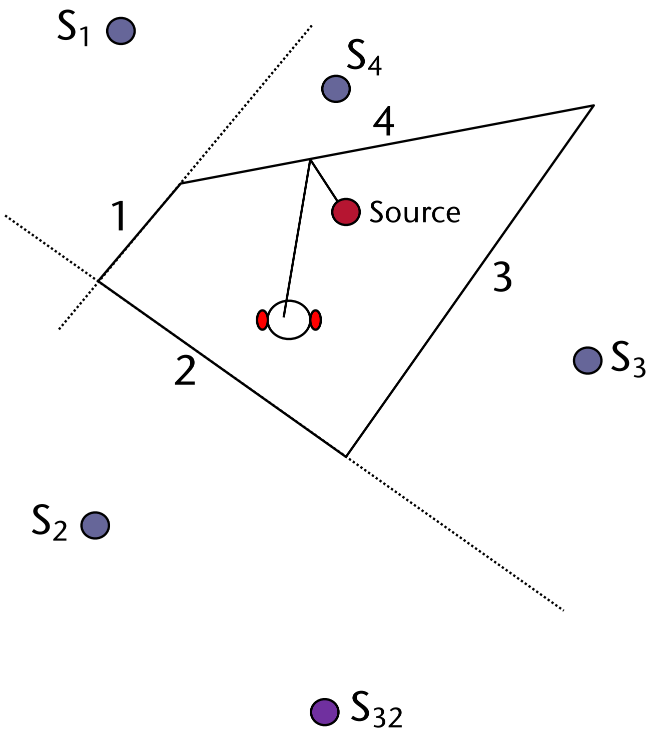
The Spatial Impression of Rooms (Psychoacoustics)

- Different sound paths carry different information:
  - Direct path → yields information about the direction of the sound source
  - Early reflections (i.e., only a few reflections on the path) → information about distance and "width" of the sound source
  - Late reflections (= reverberations, i.e., lots of reflections on the path) → informationen about the room as a whole
- Result: a simulation needs to compute a large number of sound paths!

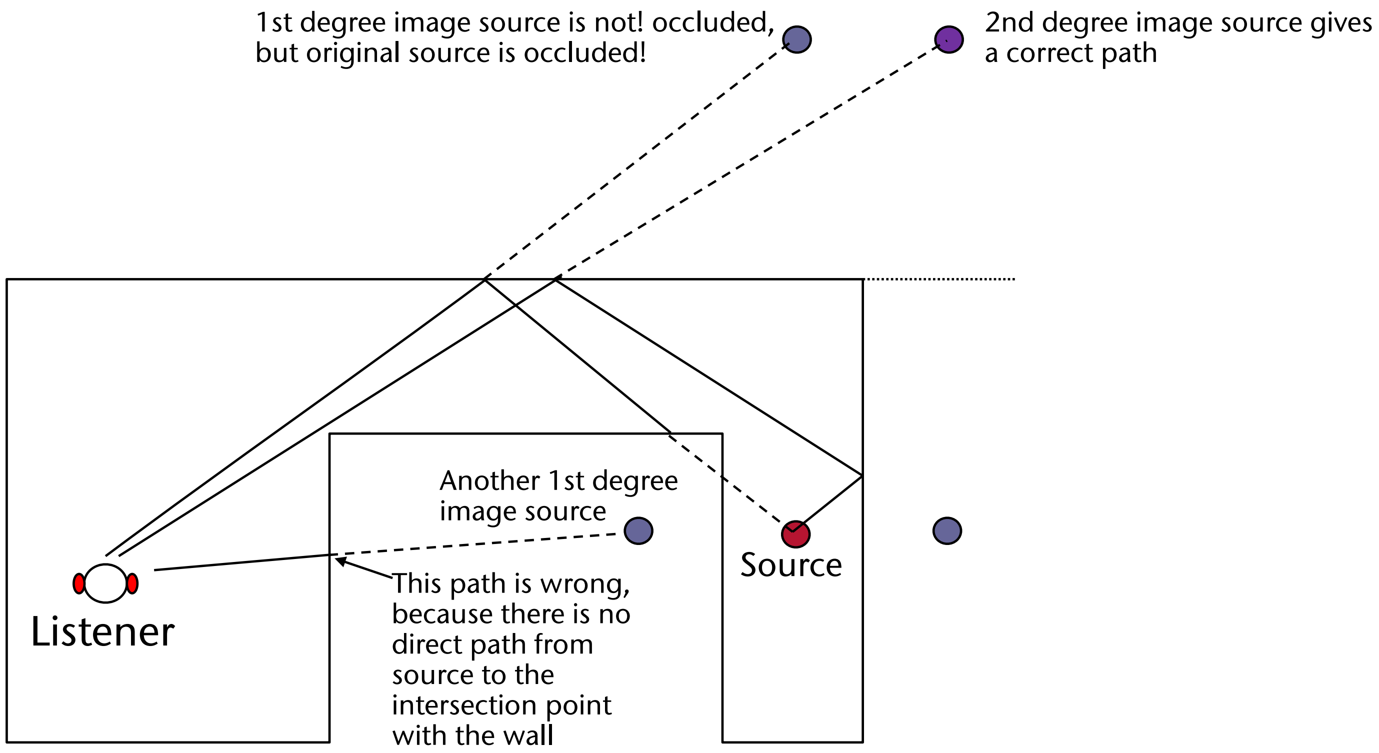


The Image Source Method for Sound Propagation

- Fast method for creating the sound paths
- Idea: for each wall (and each source), create a "virtual" sound source → image source
- For each image source: generate line listener↔image-source, then reflect path as often as degree of image source
- Then: length of path ~ travel time ~ phase shift; #bounces ~ attenuation
- Complexity, considering 1 reflection only: O($n \cdot r$) , n = # sources, r = # walls
- Problems and limitations:
  - Need to recompute all images sources, if the sound source moves
  - Can model only reflection



Example showing the method (and some problems with it)



1st degree image source is not! occluded, but original source is occluded!

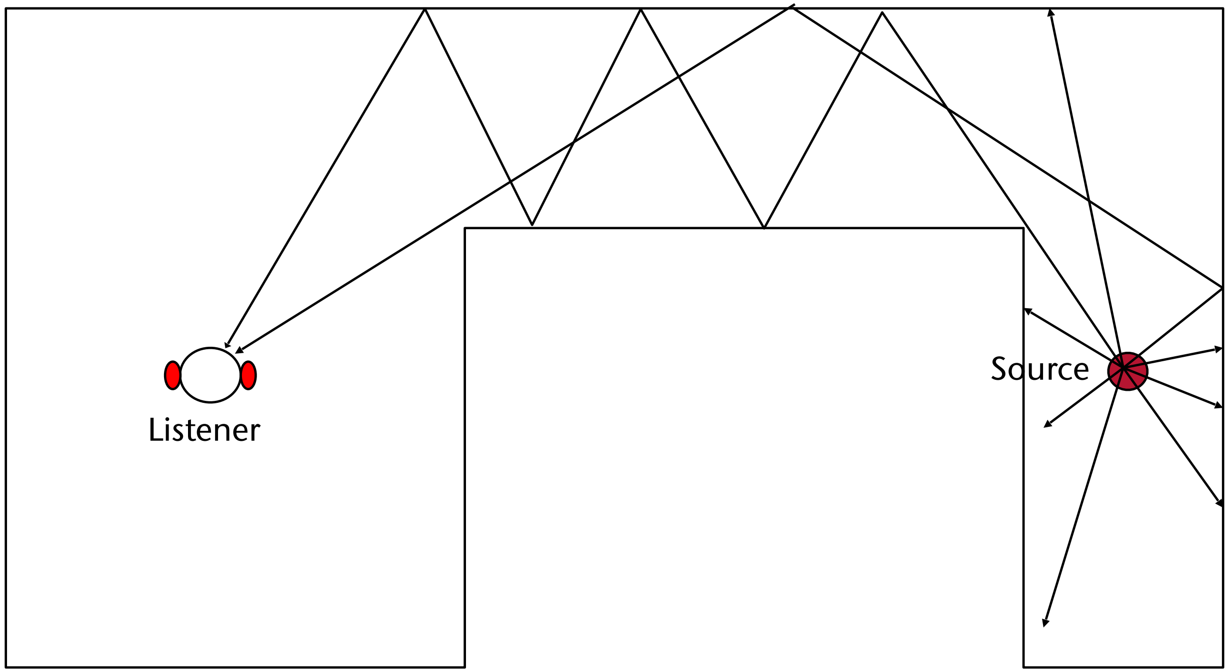
2nd degree image source gives a correct path

Another 1st degree image source

This path is wrong, because there is no direct path from source to the intersection point with the wall

Problems of Simple Ray Tracing Methods

- Shoot many rays from source (or from listener)
- Problems:
  - Most rays do not reach the listener, i.e., have been shot in vain
  - Actually, the listener consists of two very small points in space
  - How can you make sure to find the main paths?

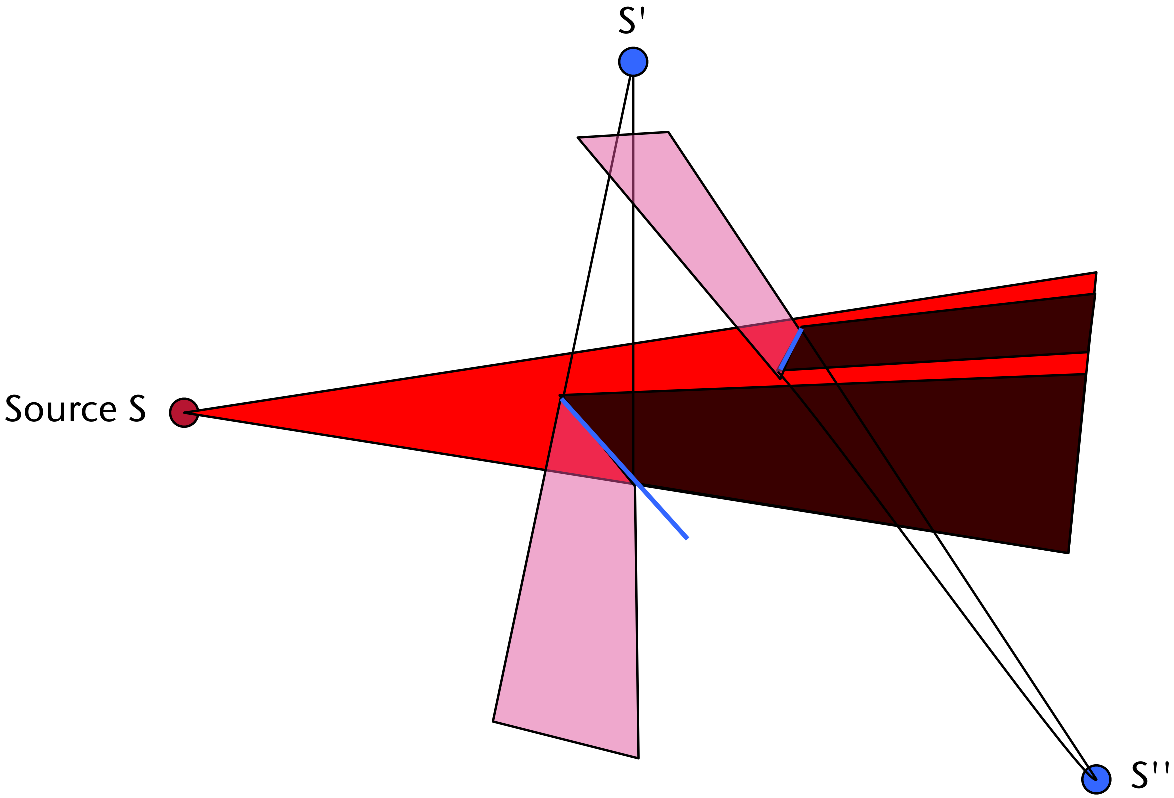


Beam Tracing

- Beam = "thick" ray, like a generalized frustum with arbitrary polygon as cross section
- Used here to precompute a lot for backwards tracing of sound paths later
- Beam tracing:

1. Sort polygons along the beam,
2. For each polygon that intersects the beam:
   split beam by the polygon (results in two, sometimes more, beams),
   for each reflected beam: construct an image source
3. Recursion into the new beams

Example

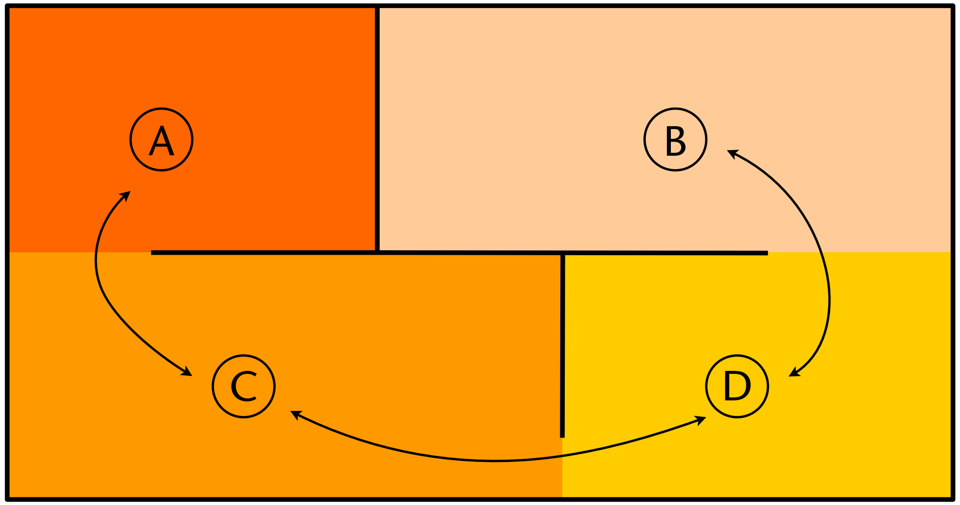


Sound Propagation with Beams

- Compute all the beams (up to some recursion level)
  → beam tree, one for each original sound source
- With each beam store:
  - associated source / image source
  - polygon from which it was reflected
  - parent beam
  - number of reflections since the source (i.e., depth in beam tree)
  - original sound source (i.e., root of beam tree)
- At runtime, for a given listener:
  1. Determine all beams that contain the listener
  2. Follow sound path backwards towards the source (i.e., up in the beam tree), like in the image source method
  3. Accumulate all signals (using correct attenuation and time shift)

Improvements of the Performance of Beam Tracing

- Idea for complex environments consisting of lots of walls: partition the scene into convex cells
  - Note: any point inside a convex polyhedron "sees" all walls
- Definition of a cell: a volume of the universe that
  1. is convex
  2. does not contain other polygons inside
- With each cell store:
  - neighboring cells → cell adjacency graph
  - polygons on the border of the cell (= real walls)

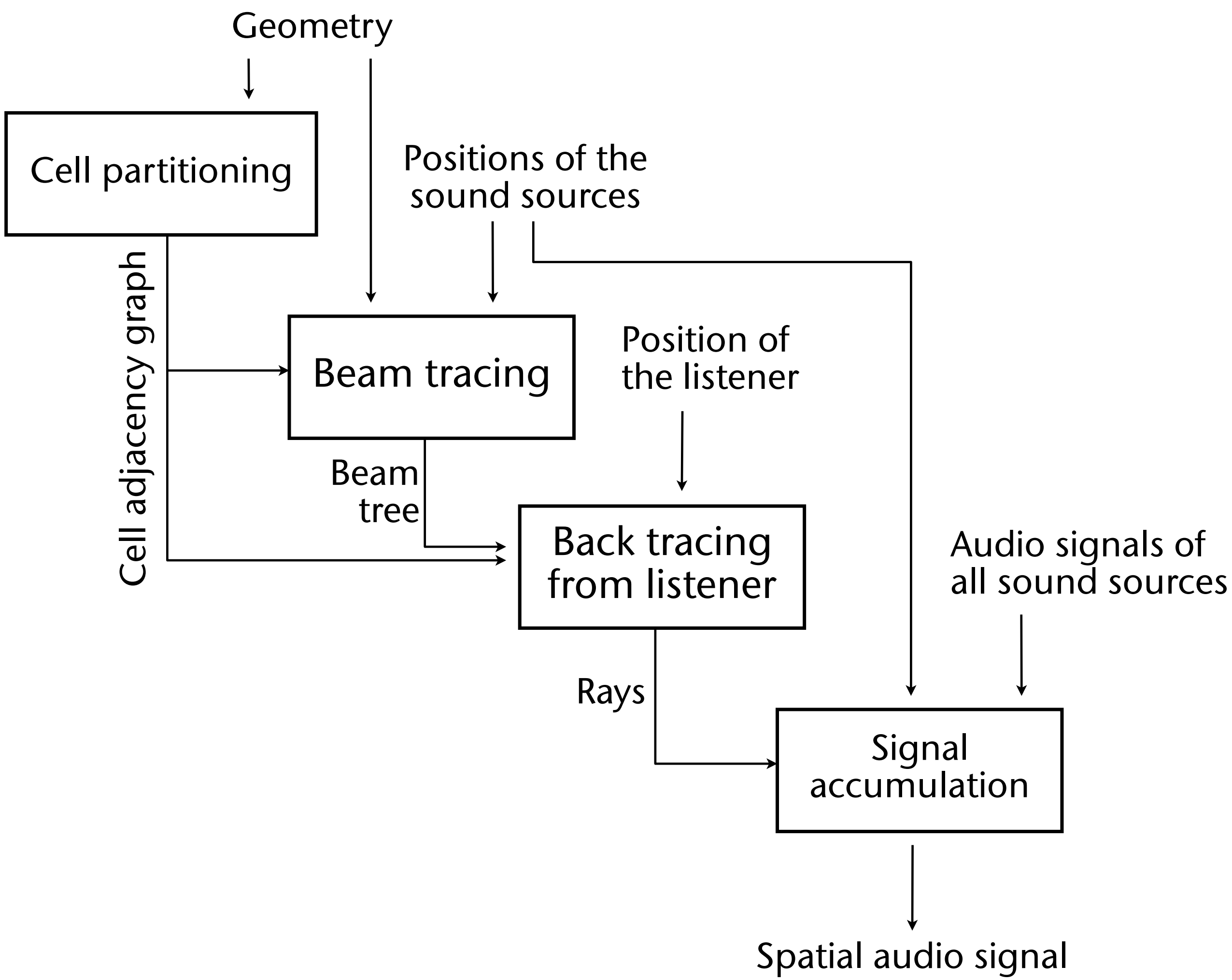


Beam Tracing with Cell Partitioning

- Given a sound source and the cell adjacency graph

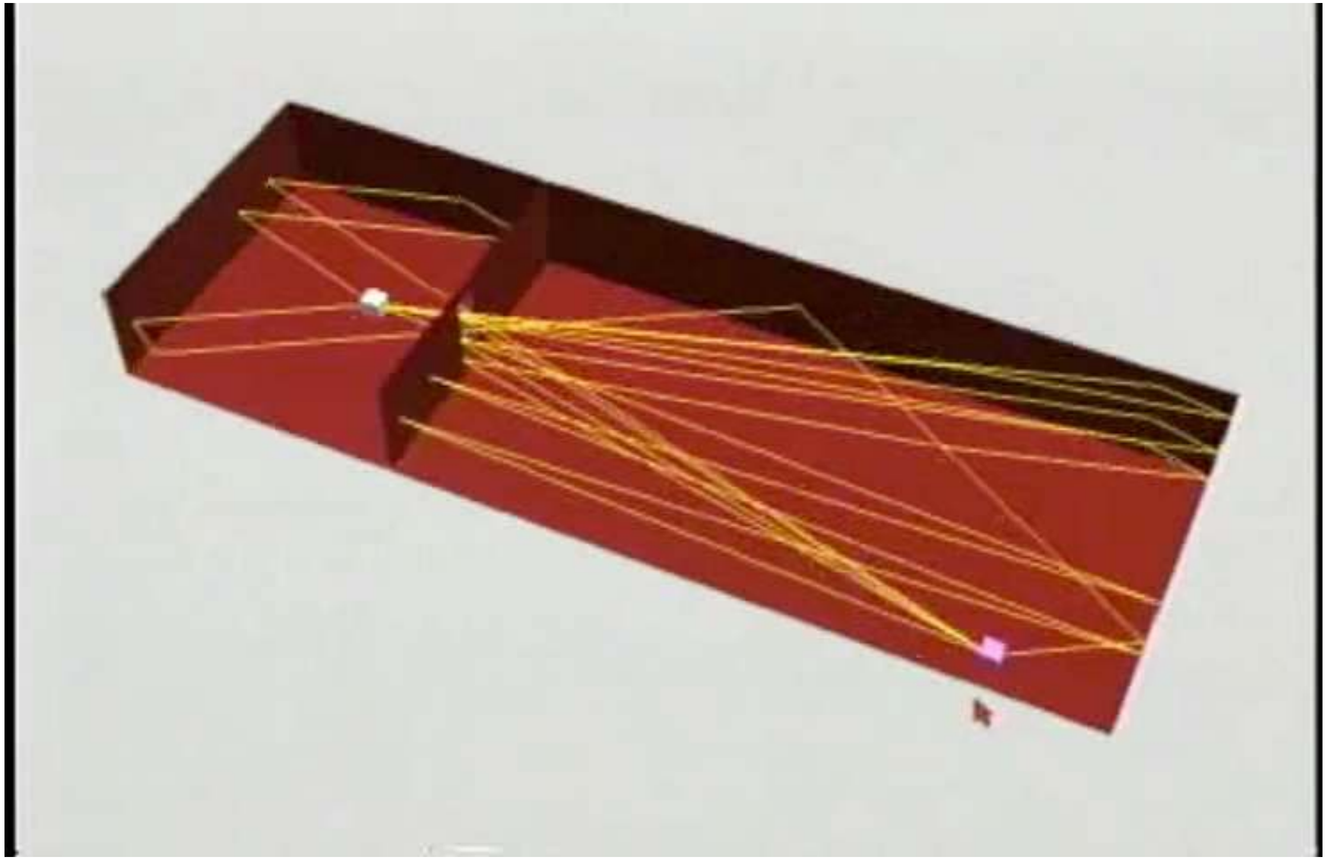
1. Determine the cell containing the source (beam starts here)
2. For each polygon on the cell's border: generate mirrored beam, image source, and clip the original beam
3. If cross section of the *original* (but clipped) beam is not empty, then traverse into neighboring cells and repeat
4. Recursion with the the mirrored beams

Overview of the Whole Method

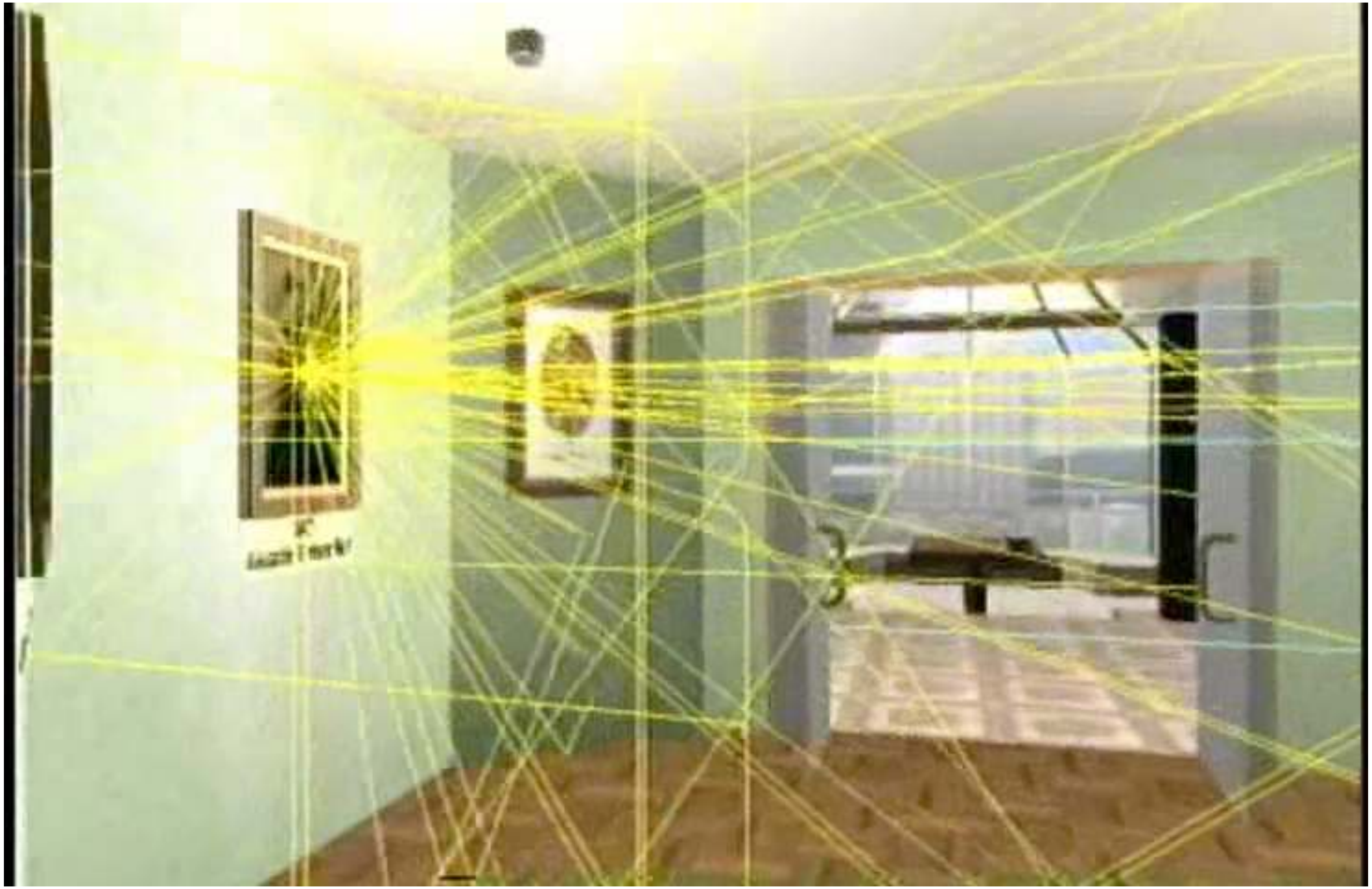


Properties of the Method

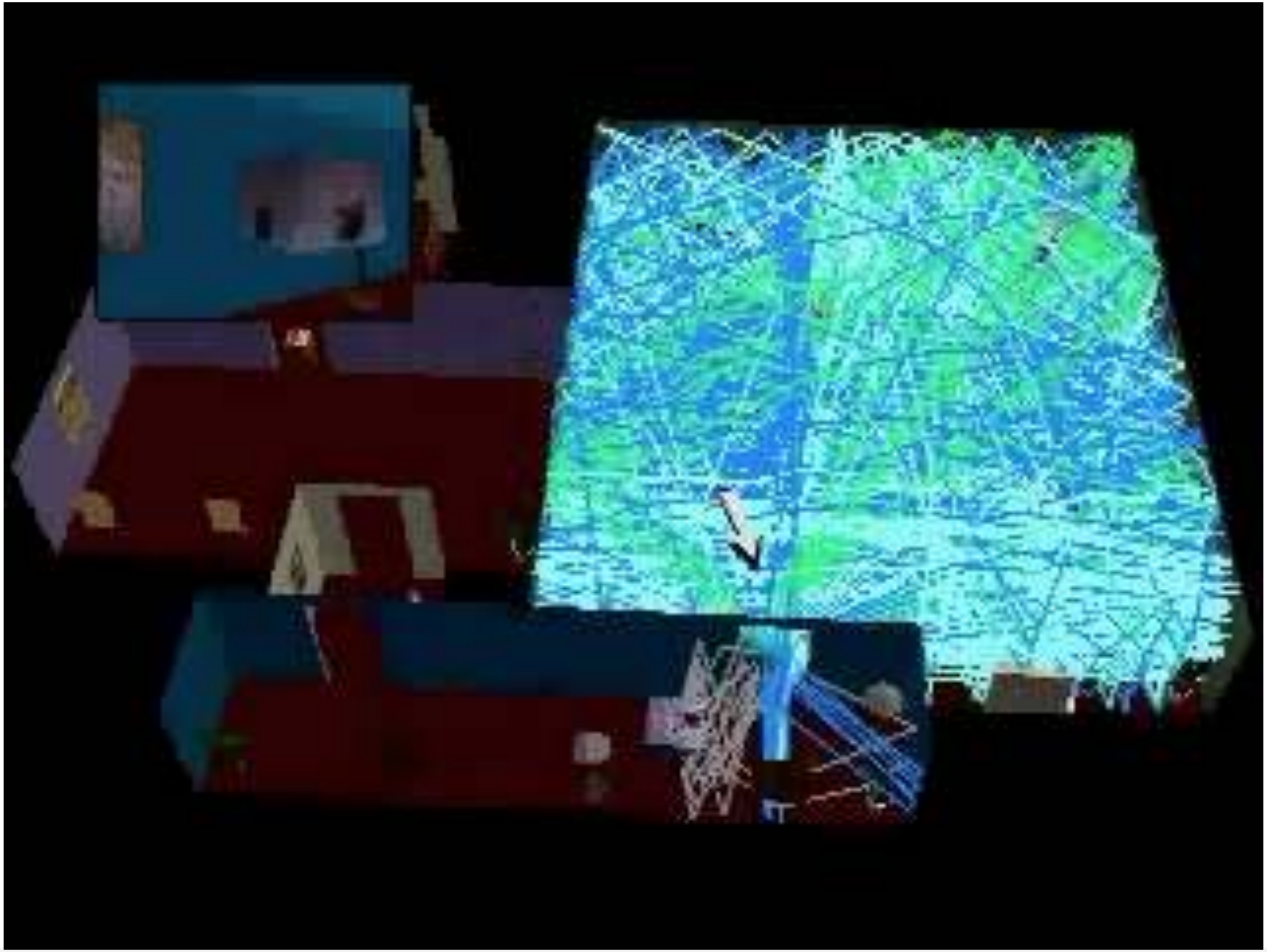
- Very fast, at runtime
- Can also handle diffraction (to some degree)
- Can be parallelized (maybe not massively)



Movies with Visualization



Ahuja, Gatlin, Surendran



Funkhouser, Min, Carlbom. Siggraph 99

Limitations of the (Simple) Beam Tracing Method

https://www.menti.com/7wucj265rw

Interludium: Monte-Carlo Integration

- Task: compute the integral

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

  (where *f* is too "complicated" for computing the integral analytically)

- Trivial numeric approximation: evaluate *f* at equidistant positions

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

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

- If $N \to \infty$, then $E_N \to E$
- Alternative formulation: $E_N = (b-a) \sum_{i=1}^{N} \frac{1}{N} f(x_i) \quad , x_i = a + i \cdot h$

  i.e., each $f(x_i)$ gets *weighted* by $\frac{1}{N}$

- Analogous for multi-dimensional integrals, i.e., for $f : \mathbb{R}^d \to \mathbb{R}$ compute

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

  where x_i are equidistant positions on a regular grid within Ω (in total N pos.)

- Problems:
  - Computational effort increases exponentially with dimension d, since N ~ Vol(Ω) = (length of grid-side)^d
  - Sampling of f is non-trivial, if Ω has a non-canonical shape (not box, sphere, etc.)
- Solution: Monte-Carlo integration = sample f at *random, uniformly distributed* positions x_i, and still use the equation $E_N = (b-a)\sum_{i=1}^{N} \frac{1}{N} f(x_i)$ as before

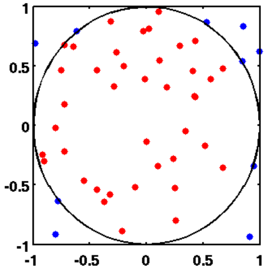
Managing Uncertainty

- What is the probability, that E_N is "close" to E?
- Observation: if the $\{x_i\}$ are generated randomly, then E_N is a random variable, too; consequently, it must have a specific probability density function (PDF)
- *Central limit theorem (Zentraler Grenzwertsatz*) to the rescue:
  when N becomes large, then the PDF of E_N approaches the Gaussian normal distribution, with mean E, and standard deviation

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

- So $E_N \in [E\text{-}\sigma_N, E+\sigma_N]$ with probability 0.67
- I.e., σ_N tells us something about the uncertainty of our estimation of E

- Advantages of Monte-Carlo integration:
  - N can be chosen arbitrarily
  - You can sample any domain very easily, you just need a test for $x \in \Omega$



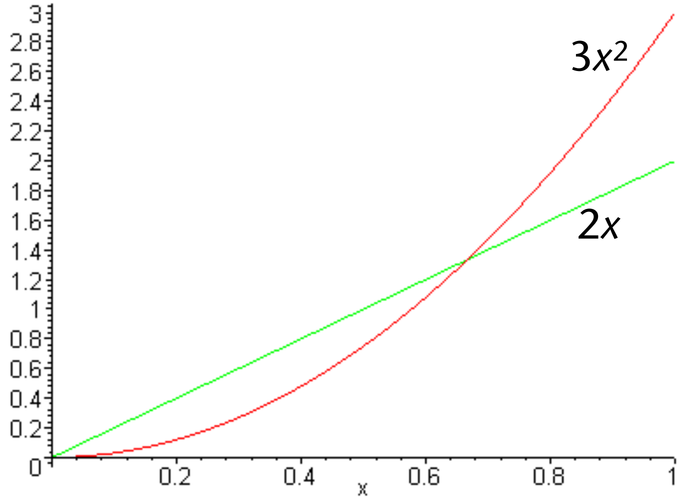
- Remaining problem: the contribution of sample points x_i to E_N can be very small (depending on $f(x_i)$)
- Approach: place sample points x_i where the value of f is big

→ Idea of Importance Sampling

- Example: $E = \int_0^1 3x^2 \, \mathrm{d}x$
- Obviously, the biggest contribution to *E* is made by samples near 1
- Choose sampling positions according to an importance PDF, $p(x)$, e.g.

$$p(x) = 2x$$

- Still to solve: how to rewrite the integral such that the sampling bias is compensated for?

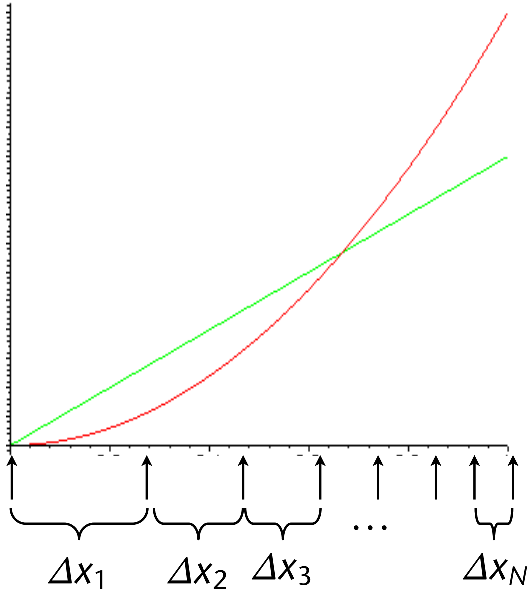


- A one-dimensional plausibility argument:
  - Consider the numerical integration with rectangles:

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

  - Since the x_i are to be chosen according to $p(x)$, it follows that $\Delta x_i \sim \frac{1}{p(x_i)}$, more precisely:

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



- Together, MC integration with importance sampling:

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

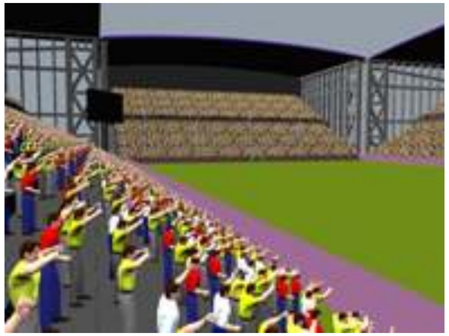
where the x_i are generated randomly according to the PDF $p(x)$

- Multi-dimensionally, it works exactly the same:

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

Stochastic Sound-Rendering of Large Crowds

- Given a VE with a huge number of sound sources (e.g. 100k), for instance, stadiums, concerts, etc.
- Mixing, i.e., computing the final sum $s(t) = \sum_i a_i s_i(t - \tau_i)$, is no longer feasible (at 40 kHz)
- Approach:
  - Like Monte Carlo integration, choose *k* samples out of *n*:
    $$\pi : \{1, \cdots, k\} \rightarrow \{1, \cdots, n\}$$
  - Then, estimate the total signal *s*(*t*) by
    $$\tilde{s}(t) = \frac{n}{k} \sum_{i=1}^{k} \frac{a_{\pi(i)} s_{\pi(i)}(t - \tau_i)}{p_{\pi(i)}}$$
    where $p_{\pi(i)}$ = probability that sounds source *i* gets chosen



How to do the Importance Sampling

- Amounts to define a good PDF $p(x)$:
  - Optimal would be: $p_i = a_i s_i(t)$
  - Simplification: $p_i = \frac{a_i}{\sum_{i=1}^{n} a_i}$
- What about dynamic changes in the scene, such as
  - Moving listener
  - Moving sound sources (or sound sources rotate and they have non-spherical radiance characteristics)
  - All of these amount to changing attenuation a_i, hence changes in p_i

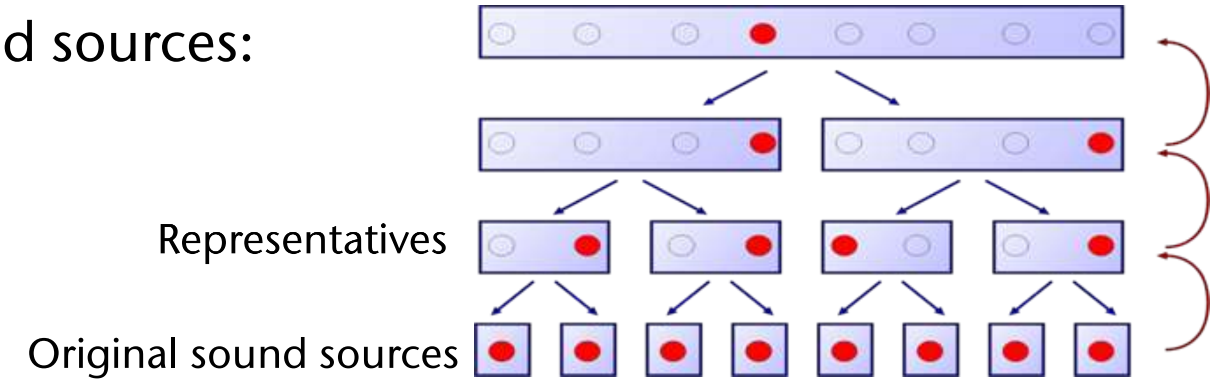
Dynamic Importance Sampling

1. Do a complete resampling:
   - Problem: since $\tilde{s}$ is not exact, the switch to a different sample set will be audible
2. Stochastic diffusion:
   - Only replace *some* of the samples in the sample set at any one time
   - Then, *fade out* the old sound source, fade in the new one
   - Better, but still audible
3. Adaptive diffusion:
   - Adapt the number of replaced samples to the change in the a_i

- Problem: computational complexity is still in $O(n)$, since finding new samples still requires some kind of sampling all of Ω

Hierarchical Sampling

- Suitable for static scenes
- Build an *octree* over all sound sources:



- Representative = one out of its 8 children, chosen with probability proportional to their respective volume (loudness)
- Volume (loudness) = sum of the volumes of the children

Traversing the Octree by Priority

- Maintain a p-queue where priority = loudness of a representative × attenuation
- The algorithm:

```
initialize p-queue with top of octree
repeat
  r ← get front of p-queue
  add r to set of samples
  add children of r to the p-queue
until k samples have been found
```

- Properties:
  - Running time: $O(k \log k)$
  - Does actually realize *importance sampling*
  - Also performs kind of a stratification ("more uniform sampling than just random")

- Limitations:
  - Attenuation can be achieved only by distance (not reflections)
  - Visibility of sound source is ignored
  - Source characteristics (*e*) cannot be implemented (at least not straight-forward)
- One possible solution: combine hierarchical with dynamic sampling
  - Choose 10,000 sample candidates by hierarchical sampling
  - From those, choose 100-1000 by dynamic sampling

Visualization of Source Selection
(150 Sound Sources)

16,000 people
20,000 secondary sound sources

13 different sound tracks, random phase shift, 16 bit, 44.1 kHz, mono. Mixer creates stereo effect. Secondary sound sources have been created by *photon tracing* (*diffuse reflection*)

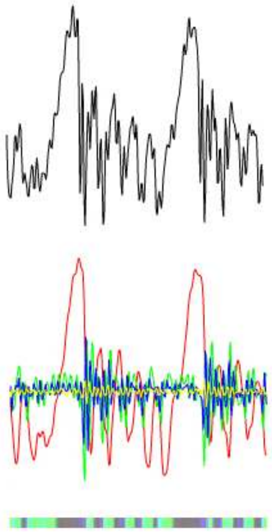
[Michael Wand, 2004]

Diffuse Reflection,
150 Sound Sources (Realtime)

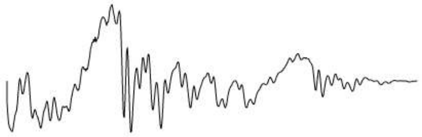
1 primary sound source.
50,000 secondary sound sources.

Secondary sound sources
have been created by
image source method

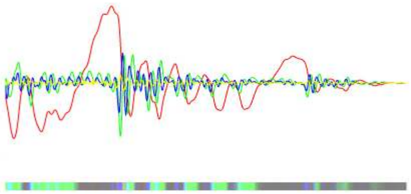
Audio-Processing on the GPU



- 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

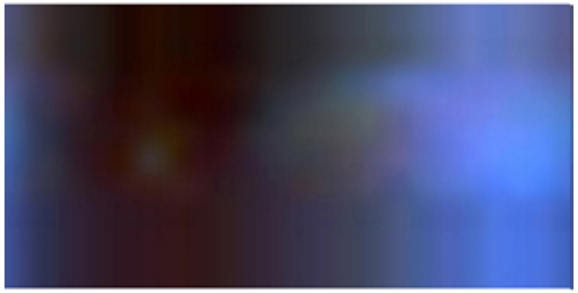


Neues Signal (Dopplerv.)



Gestreckte Textur

- 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