BACHELOR-ARBEIT

vorgelegt an der
Technischen Hochschule Würzburg-Schweinfurt
in der Fakultät Informatik und Wirtschaftsinformatik
zum Abschluss eines Studiums im Studiengang Informatik

Simulating Room Acoustics Using Ray Tracing

Angefertigt an der Fakultät für Informatik und Wirtschaftsinformatik der Technischen Hochschule Würzburg-Schweinfurt

Erstprüfer: Prof. Dr.-Ing. Frank Deinzer

Zweitprüfer: Prof. Dr.-Ing. Arndt Balzer

Abgabetermin: 2. 04. 2024

Eingereicht von: Christina Reichel aus Würzburg

Hiermit versichere ich, dass ich die vorgelegte Bachelorarbeit/Masterarbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Würzburg, den

(Unterschrift)

Hiermit willige ich ein, dass zum Zwecke der Überprüfung auf Plagiate meine vorgelegte Arbeit in digitaler Form an PlagScan übermittelt und diese vorübergehend (max. 5 Jahre) in der von PlagScan geführten Datenbank gespeichert wird, sowie persönliche Daten, die Teil dieser Arbeit sind, dort hinterlegt werden.

Die Einwilligung ist freiwillig. Ohne diese Einwilligung kann unter Entfernung aller persönlichen Angaben und Wahrung der urheberrechtlichen Vorgaben die Plagiatsüberprüfung nicht verhindert werden. Die Einwilligung zur Speicherung und Verwendung der persönlichen Daten kann jederzeit durch Erklärung gegenüber der Fakultät widerrufen werden.

Würzburg, den

(ggf. Unterschrift)

Übersicht

TEXT DEUTSCH

Abstract

TEXT ENGLISCH

Inhaltsverzeichnis

1 Introduction **1**

1.1 Motivation 1

1.2 Scope 2

2 Interpolating Intersection Checks **5**

2.1 Intersection Checks for Spheres 5

2.2 Intersection Checks for Surfaces 5

3 Time-Based Chunks **7**

3.1 Chunks vs. Bounding Volumes 7

3.2 Data Structure 8

3.3 Calculating Chunks 9

3.4 Traversing Chunks 9

4 Time-Based Impulse Responses **11**

Literatur **13**

Abbildungsverzeichnis **15**

Tabellenverzeichnis **17**

Kapitel 1

Introduction

1.1 Motivation

While there has been plenty of research into simulating sound propagation using both geometric and numerical methods, nearly all of it only concerns itself with static scenes, where neither objects within the scene nor the sound emitter nor receiver move with time.
Simulation of dynamic or moving scenes is mostly unexplored. This is mostly because it is only really relevant for scenarios where objects or receivers can move at speeds within one or two orders of magnitude from the ray's travelling speed. If this condition is not fulfilled, the error introduced from the snapshot method described below becomes minimal enough to be disregarded. Since this condition does not apply to light (the highest speed a human has ever moved at, 39,937.7 km/h, does not come anywhere near the speed of light), errors introduced to computer graphics by the snapshot method are especially small. Only a comparatively small part of the research into ray tracing comes from outside of computer graphics, and even in other fields, such as acoustics simulation, the error introduced by the snapshot method is only relevant in edge cases. Despite this, some research into simulating moving or dynamic scenes' acoustics has been done: Raghuvanshi et al. [9] explore a numerical approach to handle dynamically moving emitters and receivers, but do not look into moving objects within the scene. Chandak et al. [3] attempt to simulate dynamic scenes in real-time, with support for dynamically moving scenes but sacrificing accuracy. This is less exploring how to handle moving scenes and more exploring methods to run acoustics simulation fast enough to re-calculate acoustics at a similar rate to that at which objects move by significant amounts. Similarly, EAR, an acoustics simulation tool based on the 3d modelling software blender, allows for moving scenes through blender's keyframe system, but is inaccurate for the same reason as Chandak's real-time approach:
Both EAR and Chandak et al. approach the simulation of moving scenes by taking a static ver-

sion of the scene at the time rays are emitted to create an impulse response, then bouncing rays through this static snapshot. This approach will be called the snapshot method in this thesis.
The snapshot method comes with a few advantages: As the snapshot is just a static scene, the same well-explored and -optimised bouncing logic used for static scenes can be copied without changes. Also, crucially, knowledge of how the scene will move over the time the ray spends bouncing around it is not required. All data necessary to simulate the bouncing is available at the time the ray is emitted, without a need for information on how the scene will continue to move. This is especially helpful for real-time simulation of dynamic scenes, as data about how the scene will continue to move is not fully known at runtime.
The downside of this snapshot approach is that it tends to introduce errors when objects or receivers move at high speeds. As a simple example case, take the scene described in (IMAGE): A receiver starts 343 meters away from an emitter and moves towards it at 1/9th the speed of sound, roughly 38 meters per second (137.2 kilometers per hour, a speed most modern cars can reach without problems). Using the snapshot approach, a ray traveling directly from emitter to receiver would arrive after travelling the full 343 meters, taking 1 second for it to arrive at the receiver. In actuality, in the time the ray takes to travel the first 90% of that distance, the receiver has already travelled the remaining 10%, making for a response time of 0.9 seconds rather than 1 second.
While Bilibashi et al. [2] have attempted to solve this issue, they only aimed to simulate waves bouncing between a few set points, namely cars, rather than simulating full room acoustics. Their vector-based approach cannot be used for a full scene simulation.
To accurately simulate both edge cases occurring in the real world (such as the example above) and hypothetical situations such as the test cases described below, a new method needs to be developed.

1.2 Scope

This thesis proposes a method to simulate rays bouncing through arbitrary scenes with moving receivers and/or objects, assuming all movement within the scene is known at time of calculation. An improved way of checking for intersections between rays and objects is developed, accommodating for this new requirement. Additionally, a method is developed to to losslessly and efficiently store the multiple impulse responses created by re-calculating the impulse responses for different points in time. The goal of this research is to simulate effects such as the situation described in (IMAGE) without errors introduced by the snapshot method as well as accurately recreate the acoustics of a hypothetical, rapidly rotating room. Three test cases are

developed for this and compared to an implementation of the snapshot method: An empty scene with the sound receiver approaching the sound emitter at 1/3 the speed of sound, a square room rapidly rotating and a large, L-shaped room also rapidly rotating around one of its ends, with the receiver and emitter both sitting in said end.
Side effects of moving scenes, such as sounds emitted by moving objects, are discarded as they are irrelevant to the changed intersection logic. A note-worthy side effect that gets ignored is mass inertia: The example case where this would become relevant is the inside of a linearly moving enclosed room, such as a driving car. Due to mass inertia, sound waves travelling inside this moving rome behave the same as if the car stood still. Since this effect is only relevant in a niche scenario and it can be simulated using a method that ignores movement entirely, it can be ignored for this research.
Real-time applications cannot use this proposed method as it requires knowledge of objects' future movements ahead of time. Further research is required to develop an alternative method for real-time or dynamic simulations. A real-time approach could work by not calculating the rays' entire movements at emission time, but instead keeping track of all moving rays and incrementally continuing their journey through the now updated scene at recurring intervals.

Kapitel 2

Interpolating Intersection Checks

2.1 Intersection Checks for Spheres

2.2 Intersection Checks for Surfaces

Kapitel 3

Time-Based Chunks

3.1 Chunks vs. Bounding Volumes

One common optimisation for ray tracing systems is to limit the amount of intersection calculations by eliminating objects the ray cannot intersect with in a simpler way. There are two general sets of methods used for this:
Bounding Volume Hierarchies (BVHs), first proposed by Clarke [4], work by enclosing each object in the scene within a volume containing it. This bounding volume uses a simpler geometric primitive that allows for faster intersection checks than the object itself, usually quadric surfaces or spheres. These bounding volumes are then grouped into bigger bounding volumes into a hierarchical tree structure. Rays then walk down the tree structure, checking for intersections with the corresponding bounding volumes. If it does not intersect with a branch's bounding volume, any objects within that branch can be ignored for further intersection checks.
Chunks, first proposed as a Three Dimensional Digital Differential Analyzer by Fujimoto and Iwata [6], instead divide a scene into separate cells (chunks), with each chunk keeping a list of which objects are inside it. Rays can then traverse from chunk to chunk along their trajectory and only check for intersections with the objects contained in the chunk they're currently in.
Since objects can move around the scene, using one of these methods without changes becomes inefficient. If, for example, a receiver moves from one end of the scene to the other over the course of ten seconds, its bounding volume would extend over all of that distance for the entirety for the scene, despite it not touching the majority of it for the most part. Similarly, it would be kept in its starting position's chunk for the entirety of the scene despite leaving that area very early, making for needless intersection checks.
For this use case, chunks become a lot more efficient than BVHs: When taking movement over time into account, each object would need separate bounding volumes for separate segments of

time, forcing a ray to not just check one bounding volume, but multiple per object. This also means that in order to be able to create meaningful bounding volume hierarchies, each object's bounding volumes would need to be separated at the same points in time, which can lead to redundancies if objects move at different times. Calculating a useful BVH becomes impossible. The amount of chunks, in turn, does not change: They can be adapted simply by storing not just which objects are inside them, but also when each object enters and exits the chunk. If chunk contents are calculated correctly, this means that no intersection checks take place for objects that aren't inside the given chunk at the given time.

3.2 Data Structure

In a simple system, a chunk stores a list where each entry represents an object inside it. To accommodate for objects moving in and out of chunks, entries can instead contain three fields: One containing the index of the object in question, one containing the time at which the object enters the chunk and one containing the time at which the object leaves the chunk. Since the latter two fields might both be optional if the scene starts/ends with the object inside the scene, this can be nicely represented using a sum type such as Rust's Enumerators or C's union types with different states:

```
// Object stays within chunk for the whole scene
// only store the index
Static(object)
// Object enters and exits chunk at the given times
Dynamic(object, time_entry, time_exit)
// Object enters chunk at the given time
// and stays until the end
Final(object, time_entry)
```

As the scene's start time is known and the state of objects before it is irrelevant, a state containing only an exit time is not necessary as it can be modelled using the `Dynamic` state with a `time_entry` matching the scene's starting time. Using a more common product type system, chunk entries can instead be represented as a struct or class where the entry and exit times are optional or nullable fields.
A ray traversing this scene can now simply check when it enters and exits a chunk and pick out the objects to check for intersections with accordingly. When using sum types, the space requirements for static objects only increase by one byte denoting the type's variant (with even

that potentially getting left out, as Herzog showed [8]). For moving objects, only up to two additional fields plus the variant field are required, with the timestamp fields' size depending on the implementation. Compared to the performance gains from avoiding needless intersection checks, this additional space requirement is comparatively minimal.

3.3 Calculating Chunks

3.4 Traversing Chunks

Kapitel 4

Time-Based Impulse Responses

Literatur

[Bro08] Ilja N. Bronstein, K. A. Semendjajew, Gerhard Musiol und Heiner Muehlig. *Taschenbuch der Mathematik*. Harri Deutsch, 2008.

[BVD20] D. Bilibashi, E. M. Vitucci und V. Degli-Esposti. „Dynamic Ray Tracing: Introduction and Concept“. In: *2020 14th European Conference on Antennas and Propagation (EuCAP)*. 2020, S. 1–5. DOI: 10.23919/EuCAP48036.2020.9135577.

[Cha08] Anish Chandak, Christian Lauterbach, Micah Taylor, Zhimin Ren und Dinesh Manocha. „AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation“. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), S. 1707–1722. DOI: 10.1109/TVCG.2008.111.

[Cla76] James H. Clark. „Hierarchical geometric models for visible surface algorithms“. In: *Commun. ACM* 19.10 (Okt. 1976), S. 547–554. ISSN: 0001-0782. DOI: 10.1145/360349.360354. URL: https://doi.org/10.1145/360349.360354.

[CW88] John G. Cleary und Geoff Wyvill. In: *The Visual Computer* 4 (1988), S. 65–83. DOI: 10.1007/BF01905559.

[FI85] Akira Fujimoto und Kansei Iwata. „Accelerated Ray Tracing“. In: *Computer Graphics*. Hrsg. von Tosiyasu L. Kunii. Tokyo: Springer Japan, 1985, S. 41–65. ISBN: 978-4-431-68030-7.

[Hav00] Vlastimil Havran. „Heuristic Ray Shooting Algorithms“. Diss. Czech Technical University, Prague, Nov. 2000.

[Her] Ben Herzog. *Rust Binary Analysis, Feature By Feature*. https://research.checkpoint.com/2023/rust-binary-analysis-feature-by-feature/. Accessed: 2024-02-16.

[Rag10] Nikunj Raghuvanshi, John Snyder, Ravish Mehra, Ming Lin und Naga Govindaraju. „Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes“. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH ’10. Los Angeles, California: Association for Computing Machinery, 2010. ISBN: 9781450302104. DOI: 10.1145/1833349.1778805. URL: https://doi.org/10.1145/1833349.1778805.

[Sec] Mr. Secret. *Online Article mit extrem langer URL*. http://www.sueddeutsche.de/panorama/japan/atomkatastrophe/neue/explosion/im/akw/fukushima/strahlung/tritt/aus/1.1072079.

[SS15] Lauri Savioja und U. Peter Svensson. „Overview of geometrical room acoustic modeling techniques“. In: *The Journal of the Acoustical Society of America* 138.2 (Aug. 2015), S. 708–730. ISSN: 0001-4966. DOI: 10.1121/1.4926438. eprint: https://pubs.aip.org/asa/jasa/article-pdf/138/2/708/13242401/708_1_online.pdf. URL: https://doi.org/10.1121/1.4926438.

Abbildungsverzeichnis

Tabellenverzeichnis