Simulation der Ausbreitung von Schallwellen in einem dreidimensionalen Raum

Frank Röder, Julius Plehn

Inhaltsverzeichnis

.

Zusammenfassung

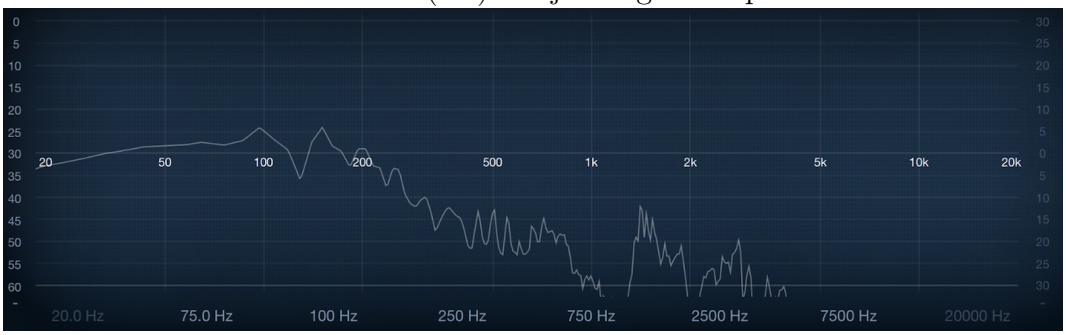
Da es sich in der Realität als sehr umständlich erweist wenn man Probleme durch Probieren einer Lösung annähert, haben wir ein Programm entwickelt, welches eine Simulation von Audioschall in einem Raum zur Verfügung stellt. In diesem Raum können wir nun an beliebiger Stelle Audioschall und Hindernisse erzeugen, um das Verhalten bei beliebigem Setup zu beobachten. Die Erkenntnisse sollen uns dabei helfen, eine gute Positionierung von Schallquellen zu bestimmen.

Kapitel 1

Problemstellung

Da es sich in der Realität als sehr unpraktisch erweist, wenn man Lautsprecher in einem großen Raum mehrmals umplatzieren muss, durch wiederholtes ausprobieren, hinhören und erneutes umstellen nicht gerade schnell zu einem guten Ergebnis kommt, haben wir uns als Ziel gesetzt ein Modell zur Simulation von Schall in einem 3-dimensionalen Raum zu entwicklen.
Die wichtigen Eigenschaften, wie die Interferenz, Reflexion und das Auslöschen haben wir uns zur Aufgabe gemacht. Wir wollen Schallwellen darstellen, indem wir Frequenzteilchen wellenartig sich im Raum bewegen lassen. Als mögliches Szenario hat man nun 2 Quellen, von denen Schall ausgeht. Mögliche Hindernisse und Wände sollen sich auf das Verhalten des Schalls auswirken.Dieses soll zu unterschiedlichsten Erkenntnissen führen. Durch die angestrebte Visualisierung könnte man nun die kritischen Stellen, bei denen sich besonders viel Schall auslöscht erkennen, das gegebene Szenario modifizieren und das Ereignis erneut durchlaufen. Nach langer Überlegung und Recherche haben wir uns schlussendlich auf eine sehr einfache abstrakte Weise geeinigt, die uns nicht zu sehr von unseren eigentlichen Parallelisierungszielen abbringt und eine Idee von der Realität gibt. Jedoch wird nicht das vollkommene physikalische Wesen des Schalls widerspiegelt. Da man sich dabei auf hohe Mathematik besinnen kann, beschränken wir uns auf die genannte Modellierung und gehen auf eine möglichst genaue Auflösung ein, bei der auch ein hoher Rechneraufwand entsteht. Ein weiteres Problem war die Umsetzung der Idee. Die Frage nach Datenstrukturen zur Speicherung und zur Speicherung der Frequenzteilchen. Des Weiteren stehen einem auch noch sehr viele physikalische Formeln zu verfügung, die das Verhalten viel genauer beschreiben, jedoch mehr physikalischen Verständnis bedürfen und dann auch noch korrekt in ein Programm integriert werden müssten. Von Gesetzten und der Eigenschaft des Mediums Luft, kreisten die Gedanken um den Zeitplan, um die neue Programmiersprache und die möglichen Arbeitsaufteilungen für die Pro-

zesse. Folgende Grafik soll die Idee der Teilchen beschreiben, die pro Zeiteinheit eine bestimmte Lautstärke(dB) des jeweiligen Frequenzbereiches haben.



Frequenzgang in einem Equalizer

Kapitel 2

Lösungsansatz

Um der Problemstellung her zu werden, haben wir uns überlegt eine einfache 3-D Matrix in C zu nehmen, in der wir pro Eintrag einen Pointer (auf ein Item), welcher auf ein struct zeigt, ablegen. Das struct wiederum besteht in Falle eines Sounds mit der id = 0 aus 10 Frequenzbereichen mit einer vorgegeben Ausbreitungsrichtung. Zudem machen wir auch mit der id = 1 kenntlich, wenn es sich um ein Hindernis handelt. Da wir hierbei auf das Problem stießen, bei mehrfachen Einträgen weitere Matrizen zur Speicherung der Überlagerung zu reservieren, haben wir uns auf Doppelt-Verkettete-Listen geeinigt, die nun in jedem Matrixeintrag i,j,k bestehen. In einer solchen verketteten Liste kommen nicht überdurchschnittlich viele Eintr¨ge vor, da wir nach jeder Verschiebung sofort die Überlagerung verrechnen und die Liste wieder entlasten. Schlussendlich arbeiten wir mit 2 identischen Matrizen. In der ersten stehen die vorhanden Elementen, in der 2. schreiben wir nun die Teilchen nach ihrer Bewegung hinein, somit dient sie als Puffer für die Verschiebung. Man kann Sounditems und Hindernisse an beliebiger Position erstellen und eine Richtung zuweisen in die sich der Sound bewegen soll. Bei der Bewegung erstellen wir durch eine Funktion Nachbarelemente, welche eine Welle formen sollen. Zur Feststellung ob ein Teilchen verschoben werden soll, gehen wir in einer 3-fach verschachtelten For-Schleife den Raum je Programmablauf i-mal durch und überprüfen jeden Eintrag auf nötige Operationen. Dieses wiederholen wir hunderte Male, je nachdem wie viele Schritte die Visualisierung von uns verlangt, um ein schönes Ergebnis zu erhalten.

Kapitel 3

Parallelisierungsschema

Um Objekte in einem dreidimensionalen Raum zu paralllelisiern, bietet es sich an, den Raum in kleinere, möglichst unabhängige, Bereiche einzuteilen. Die Unabhängigkeit der einzelnen Bereiche reduziert die Kommunikation der Prozesse und vereinfacht die Programmierung enorm. Im folgenden ein kleines Beispiel: Angenommen wir unterteilen den Raum an der X-Achse entlang, dann haben wir einzelne Abschnitte, die alle die gleichen Y & Z Ausmaße besitzen. Lediglich die X Ausmaße unterscheiden sich (maximal um 1). Diese Verteilung übernimmt der Prozess 1 (MPI Task 0). Nach dieser Aufteilung wird den Simulationsprozessen eine Konfiguration geschickt. Diese beinhaltet für alle die gleichen Y und Z Grenzen, die eben berechneten X Grenze und einen Offset für die Visualisierung. Des weiteren verteilt der Master nicht nur die Aufgaben, sondern er fungiert auch als Taktgeber. Diesen brauchen wir desshalb, da die Prozesse, die ihre aufgeteilten Räume berechnen, in unterschiedlich Stadien wären. Das würde die ganze Simulation sinnlos machen, da das Resultat nicht einer unparallelisierten Berechnung entsprechen würde, sondern vollkommen verzerrt wäre. Neben diesem Masterprozess gibt es noch einen extra Prozess für die Visualisierung. Dieser wird ebenfalls von dem Masterprozess synchronisiert, sodass er in jedem Durchlauf die zu visualisierenden Daten von den anderen Prozessen erwartet. Die Simulationsprozesse senden nach jedem Durchlauf ihre Feldbelegungen an die Visualisierung. Dazu addiert jede Simulation ihre X Grenze mit dem in der Startkonfigurierung mitgeteilten X Offset. Hieraus ergibt sich wieder die Koordinate in dem Gesamtraum, sodass wieder ein großes Gebilde entsteht. Der Visualisierungsprozess speichert diese Informationen in einer JSON Datei, die mit dem Javascript Visualizer abgebildet werden kann.

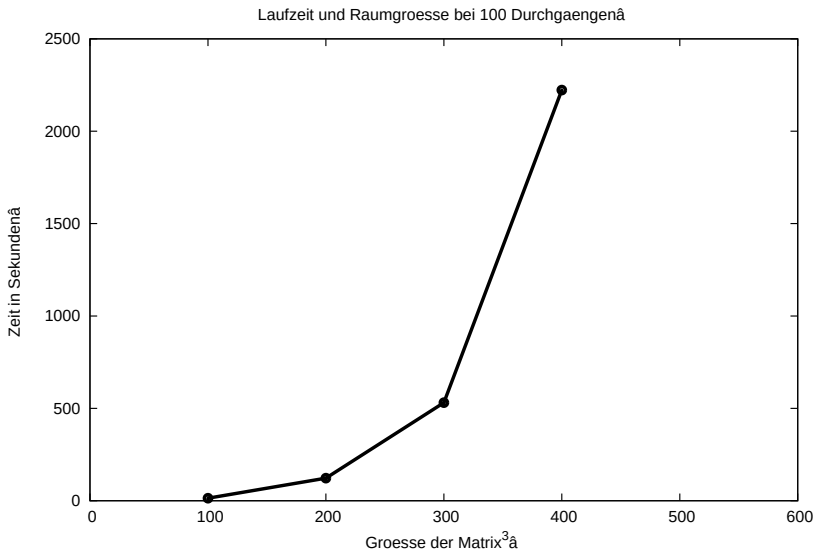
Da die einzelnen Tonelemente wandern, kann es auch vorkommen, dass Elemente ihren Prozess verlassen müssen. Wie mit diesen Elementen verfahren

werden muss unterscheidet sich durch ihre Richtung. Wenn Elemente also über ihre Y oder Z Grenzen treten (und hierbei nicht auf eine Wand stoßen), wird das jeweilige Element entfernt. Sollte ein Element jedoch ihre X Grenze übertreten, so bedeutet das, dass das Element an einen Nachbarprozess übergeben werden muss. Hierbei muss beachtet werden, dass der 1. Simulationsprozess nur an der rechten Seite einen Nachbarn hat und das der letzte Prozess nur Links einen Nachbarn hat. Alle anderen Prozesse besitzen zwei Nachbarn. Diese Spezialfälle werden also gesondert bearbeitet. Ab der zweiten Iteration, die durch den Masterprozess veranlasst wird, wird von den Nachbarn ein Array mit neuen Tonelementen erwartet. Mit diesen erhaltenen Informationen wird dann ein neues Element mit den alten Spezifikationen generiert. Dem entsprechend wird am Ende jeder Iteration, also nachdem alle Elemente verschoben und behandelt wurden, ein Array mit allen zu sendenen Elementen an die Nachbarn weitergegeben. Diese empfangen das Array am Anfang der nächsten Iteration. Da es linke und rechte Nachbarn gibt werden auch zwei unterschiedliche Arrays für die Kommunikation erzeugt.

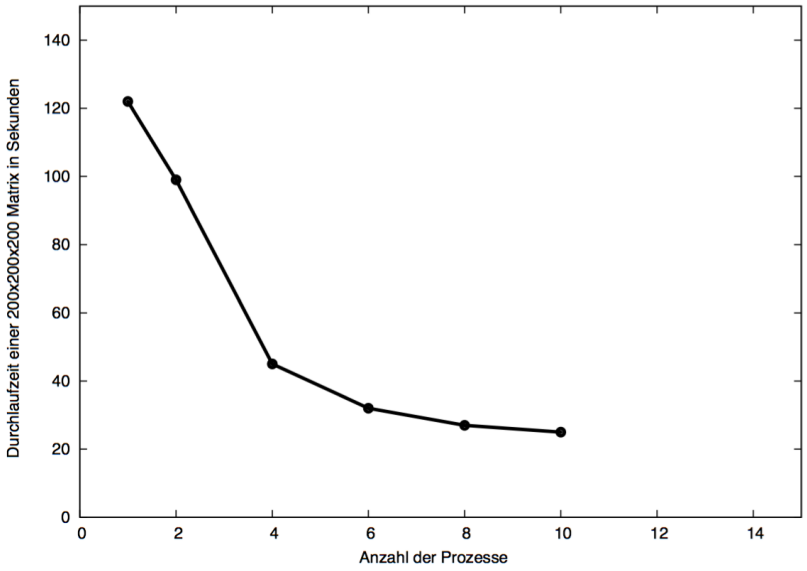
Kapitel 4

Laufzeitmessung

Durch Messungen des sequenziellen Programms konnten wir ein sinnvolles Maß der Arbeit , die ein Prozess verrichten soll, ermitteln. Diese Erkenntnisse sollen uns später bei der Parallelsierung das Verhältnis von Prozessen und Raumgröße helfen.



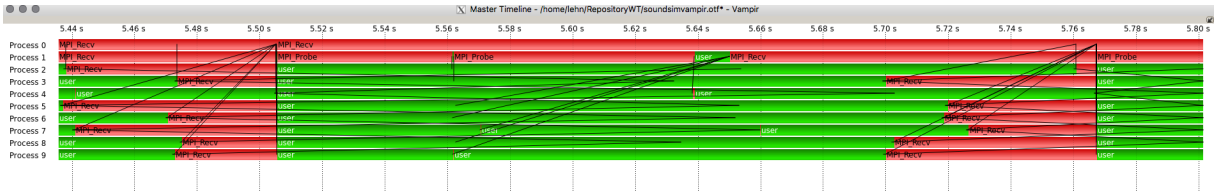
Die folgende Grafik zeigt die Laufzeit bei gegebener Anzahl an Prozessen um den Raum zu durchlaufen und jede Zelle auf ihren Zustand zu überpüfen.



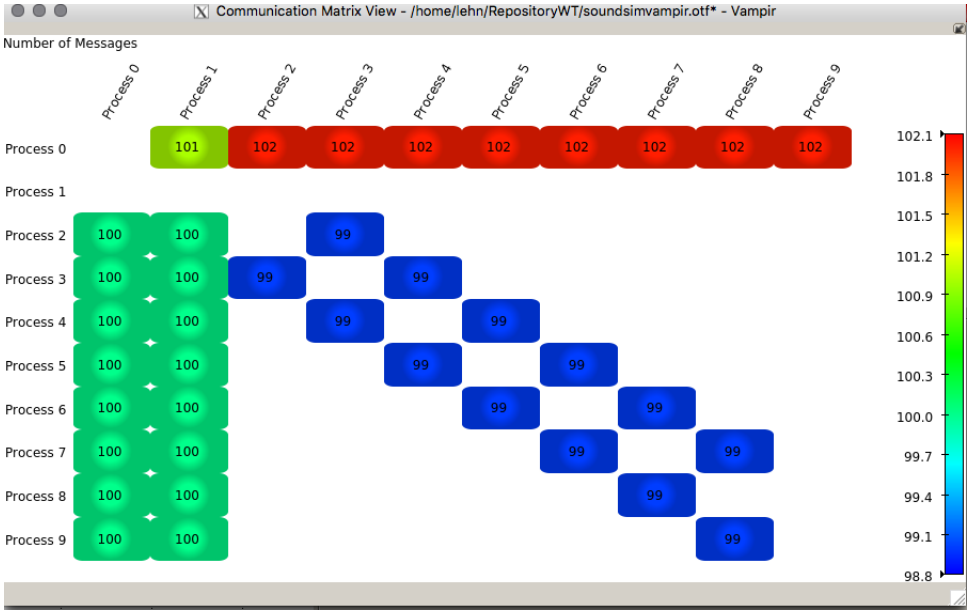
Kapitel 5

Leistungsanalyse

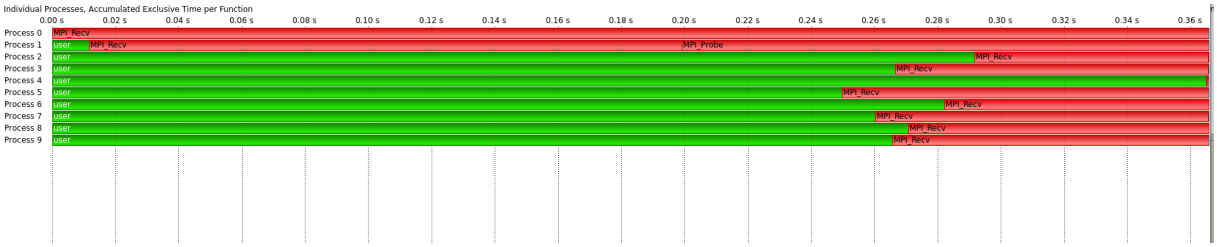
Hierbei handelt es vom Trade-Off zwischen Größe des Raumes und Anzahl an Prozessen, die in Richtung der X-Achse kommunizieren müssen. In der folgenden Abbild zeigt uns Vampir, dass schon im Kapitel Parallelisieren erklärte Kommunikationsschema.



Da wir unseren Raum an der X-Achse teilen, hat jeder Prozess nur maximal 2 Nachbarn mit denen er zu kommunizieren hat.



Hier noch eine weitere Grafik zur verbrachten Zeit der Prozessoren mit bestimmter Arbeit. Deutlich ist hier auch die Aufgabe des Prozesses 1 zu erkennen, welcher für die Visualisierung zuständig ist.



Kapitel 6

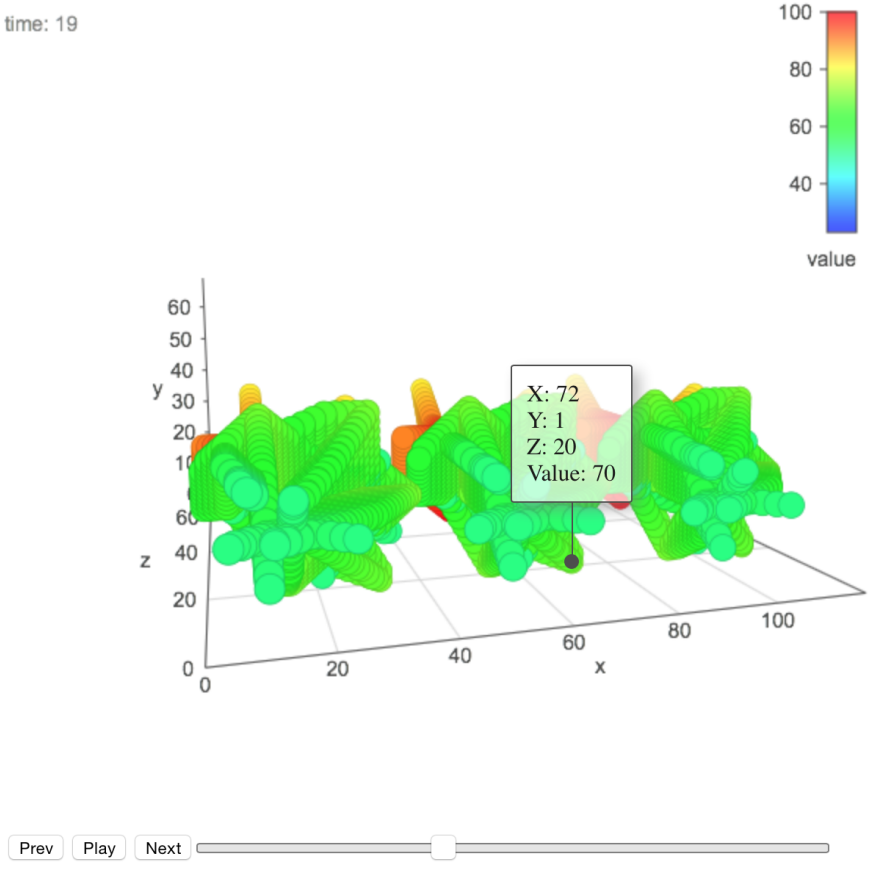
Skalierbarkeit

Vom Prinzip her kann man einen belieb großen Raum für die Simulation nutzen, da jeder Prozess für sich einen kleinen Raum bearbeitet und durch die Aufteilung der X-Achse maximal 2 Nachbarn hat. Jedoch kommt es vor, dass sich besonders viel Sound an bestimmten Stellen bündelt und 2 Prozesse dann überdurchschnittlich viel miteinander kommunizieren müssen. Zudem haben die anderen Prozesse wenig zu tun und kostbare Rechenkraft wird durch wenige überlastete Prozesse nicht optimal ausgenutzt.

Kapitel 7

Programmausgabe

Die Programmausgabe erfolgt als Darstellung der Frequenzelemente auf ihrer Koordinate mit Färbung im Bezug auf die dB. Die Informationen werden dafür im json-Format gespeichert und über eine von uns selbst erstellte html-Datei visualisiert.



Kapitel 8

Fazit und Zukunft

Das Praktikum erwies sich als eines der lehrreichsten Module, die wir je belegt haben. Wir haben sehr viel Spaß gehabt und gleichzeitig viel gelernt, denn aus eigenen Fehlern und aus der Erfahrung die man durch Erkunden der neuen Umgebung macht, erntet man große Früchte an Wissen. Auch die Möglichkeit an der Universität sich an einem Projekt auszuprobieren ist wohl der letzte Zeitpunkt in dem Fehler noch tolerierbar sind. Oft haben wir uns mit nicht zu erwartenden Fehlern der Speicherreservierung in C herumgeschlagen. Dieses nahm nicht nur sehr viel Zeit in Anspruch, sondern forderte auch das Denken über das Zeitmanagement. Haben wir uns zu lange mit einer Sache aufgehalten, gab es glücklicherweise noch einen Plan B, der uns fürs erste wieder vernünftig an dem Projekt weiter arbeiten lies. Das was fehlte war noch ein Plan C, denn wenn einem die Problemlösungen ausgehen, dann wird auch der Zeitdruck höher möglichst schnell die Lösung zu finden. Umso mehr haben wir uns gefreut, wenn ein solches Hindernis nicht wiederkehrend beseitigt wurde. Insgesamt haben wir sehr viel Zeit benötigt um uns immer wieder in Vorgehen einzulesen und Erfahrung darin zu sammeln, wie man die Meldungen der Debugger interpretiert.