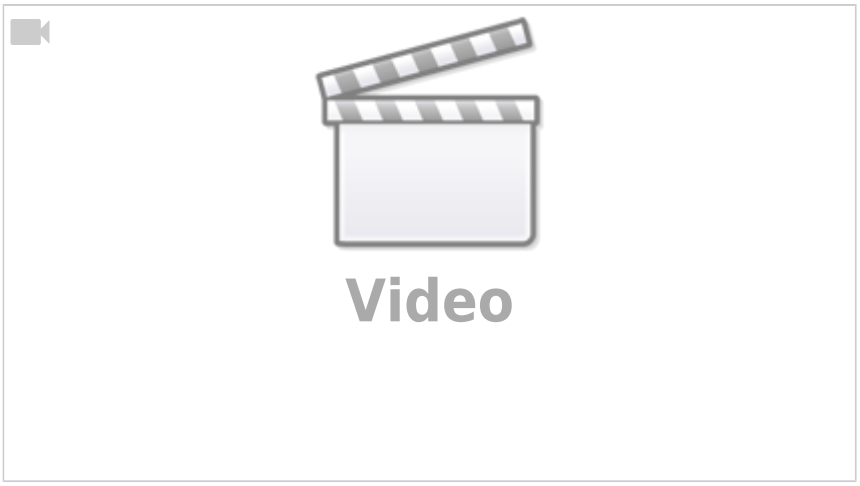
Cladni Light Installation Art (CLIA)

In unserem Projekt werden Cladni'sche Klangfiguren digital simuliert und in Form einer interaktiven Light-Audio-Installation präsentiert. Dies soll eine einfache und spaßige Methode sein die Physik in ihrer Ästhetik Menschen näher zu bringen, ohne diese mit komplexen Formeln und/oder Ähnlichen zu überfordern. Im fertigen Endprodukt soll eine einfache Tastatur den Besuchern erlauben bestimmte Töne abzuspielen und simultan das simulierte Muster einer digitalen Metallplatte auf einer Leinwand projiziert zu sehen. Durch diese künstlerische, audio-visuelle Licht-Installation wird versucht bei den Besuchern das Interesse an der Physik zu wecken.

Physikalische Grundlagen

Ernst Florens Friedrich Chladni war ein deutscher Physiker und Astronom. Er beschrieb 1787 die nach ihm benannten Klangfiguren. Sie waren einer der ersten Ansätze Ton visuell sichtbar zu machen. Im Folgenden Abschnitt werden erstmal die physikalischen Grundlagen physischer Chladni-Figuren erklärt. Darauf folgt die notwendige Theorie und Annahmen die hier gemacht worden sind, um die Figuren Digital darstellen zu können.

Hinweis: Dieser Text versucht das Gleichgewicht zwischen wissenschaftlicher Genauigkeit und Verständlichkeit für Menschen die nicht vom Fach sind zu wahren. Um das Thema besser verstehen zu können empfehlen wir die im text verlinkten Quellen und YoutTube Videos anzuschauen. Wörter die in **Bold** geschrieben sind werden im Nachhinein weiter erläutert. Wenn du, geehrte/r Leeser*in diese nicht sofort verstehst so keine Angst. Lesen Dieses Textes erfolgt auf eigene Gefahr.



Video

Grundprinzip? Die Chladni-Klangfiguren entstehen wenn Oberflächen durch mechanische Anregung zum Schwingen/Vibrieren (umgangsprachlich wackeln) gebracht werden. Für Chladni-Figuren werden meist Metallplatten verwendet. Vibration der Platten bei bestimmten Frequenzen (genannt **Eigenfrequenzen**) erzeugen sogenannte **stehende Wellen** bei denen auf bestimmten Linien die Platte gar nicht schwingt. Dies lässt sich durch das streuen von Salz (o.ä.) visualisieren. Die durch das Salz veranschaulichten Linien ergeben immer Muster. Die Form der Mustere, Frequenzen mit der die Platte schwingt, die Dimensionen der Platte und die Material Eigenschaften

Stehende Wellen Diese Schwingungen sind erstmal nichts anderes als transversale Schallwellen, welche durch sich durch den Körper bewegen. Die Geschwindigkeit, mit der sich diese durch den Körper bewegen darf nicht mit der Schallgeschwindigkeit der logitudinalen

Mathe für Wer's wissen will

Der Plan

Simulation der Klangfiguren

Da wir die chladnischen Klangfiguren digital samt Übergängen darstellen wollen, ist es notwendig ein Programm zu schreiben, dass dies tut. Unsere Anforderung an das Programm war, dass die Übergänge zwischen den verschiedenen Figuren so nah an der Realität wie möglich sind. Aus diesem Grund haben wir uns für eine Partikelsimulation entschieden, wo jeder Partikel sozusagen einem Sandkorn entspricht. Die einzelnen Partikel haben keine Kollisionen untereinander, da dies das Programm sehr viel langsamer machen würde.

Mathematische Grundlagen der Simulation

Die Differenzialgleichungen, um eine Chladni-Platte korrekt zu simulieren, gehören zwar mittlerweile zum Allgemeingut von Wikipedia [1)], sind jedoch immer noch recht kompliziert und würden den Rahmen dieses Teil des Projektes sprengen. Glücklicherweise gibt es viele Menschen, die sich mit diesem physikalischen Expermiment und den zugehörigen Differenzialgleichungen befasst haben. Einer dieser Menschen hat netterweise die Lösungen dieser Differenzialgleichungen für den Fall, dass die Platte unendlich dünn ist, auf einer Webseite gepostet [2)]. Im Nachfolgenden verwenden wir die Lösung, die dort zur Verfügung gestellt wird. Der Sand einer Chladni-Platte sammelt sich in Bereichen, wo die Platte nicht schwingt. Diese Bereiche werden durch die Nullstellen der Funktion $s(x,y) = a \sin(\pi n x) \sin(\pi m y) + b \sin(\pi m x) \sin(\pi n y)$.

Nun sollen sich alle Partikel irgendwie zu denn Nullstellen bewegen. Das ist das gleiche wie eine Nullstellenberechnung mit beliebigen Startpunkt. Hierfür würde sich zum Beispiel das Newton-Verfahren anbieten. Leider würden die einzelnen Partikel dann aber recht wild springen und wahrscheinlich nicht zur nächstgelegenen Nullstelle konvergieren. Ein besserer Kandidat ist das Gradientenverfahren, da wir dort die Schrittweite sehr einfach kontorllieren können. Das Gradientenverfahren findet aber ein Minimum. Die Minima unserer Funktion sind jedoch nicht die Nullstellen. Das können wir aber durch quadrierung der Funktion ganz schnell ändern.

Der nächste Schritt ist nun also den Gradienten der Funktion s' = s^2 zu bestimmen. Wolfram-Alpha ist hierbei eine große Hilfe und gibt uns direkt das gewünschte Ergebnis aus.

Betrachten wir nun einen Partikel, so wird dieser an einem zufälligen Punkt starten. Berechnen wir dann den Gradienten an diesem Punkt, so erhalten wir die Richtung des größten Anstiegs der Funktion. Ziehen wir diesen Vektor nun (mit einem gut gewählten Skalierungsfaktor) von der aktuellen Position ab. Es ist wohl bekannt, dass wir mit diesem Verfahren (mit richtig gewählten Parametern) nach und nach an einem Minimum und in unserem Fall an einer Nullstelle angelangen. Unser Ziel ist es nun, dieses simple Verfahren für viele Partikel gleichzeitig auszuführen, um die Übergänge der Muster zu simulieren.

Die Simulation

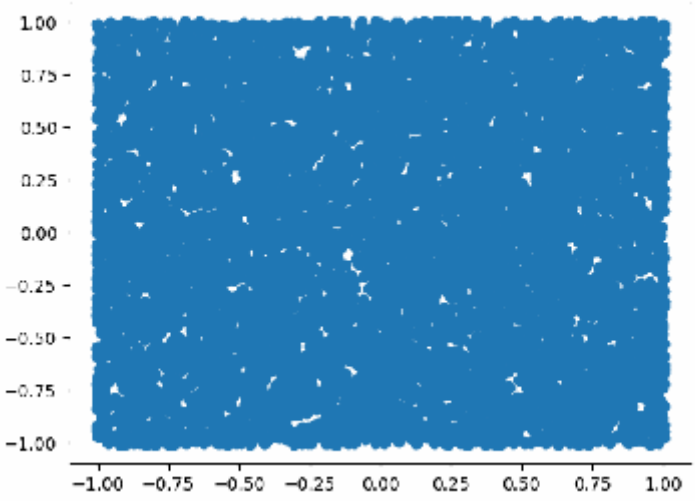
Mithilfe von Angaben eines Aufgabenblattes [3)] des Klett-Verlags können wir berechnen, dass man in einem studierendenüblichen Shotglas (4 cl) bereits ungefähr 100 000 Sandkörner unterbringen kann. Wenn man diese Menge an Sankörnern nun auf eine große (1 x 1 m) Chladni-Platte verteilt, so wird man wahrscheinlich schon schöne Muster erkennen können. Der Unterschied zur Simulation ist hier jedoch (unter vielen anderen), dass Sandkörner in der echten Welt kollidieren und somit nicht am

selben Ort sein können. Um also dennoch schöne Muster in der Simulation zu erhlaten, müssen wir deutlich mehr Sandköner simulieren. Wir haben uns für 1024 x 1024 entschieden (also ca 10 studierendenübliche Shotgläser).

Die Frage ist nun, wie man ca 1 000 000 Partikel gleichzeitig in Echtzeit simuliert. Auf einer CPU wäre dies wohl schwierig, da diese zwar sehr komplizierte Operationen ausführen kann, dafür aber nur verhältnismäßig wenige dieser. Die Lösung hierbei ist GPU-Computing (also das Rechnen auf einer Grafikkarte). Grafikkarten sind dafür optimiert, sehr viele einfache Operation parallel auszuführen - also genau das, was wir für dieses Projekt brauchen. Tatsächlich hat es auch einen Namen, die Grafikkarte zur Berechnung von Dingen zu verwenden, die nichts mit der Darstellung auf einem Bildschirm zu tun haben: GPGPU (General Purpose GPU).

Versuch 1 - GPGPU durch NVIDIA Warp

Zunächst mussten wir ausprobieren, ob die ganze Theorie überhaupt funktioniert und ob das Projekt realistisch ist. Dafür eignet sich eine einfach Python-Implementierung sehr gut. Glücklicherweise stellt NVIDIA eine Library [4] bereit, mit der es sehr einfach ist über Python auf die Grafikkarte zuzugreifen. Ein erster Test hat gezeigt, das unsere Ansätze korrekt sind und gut funktionieren. Die Darstellung der Partikel hat an dieser Stelle noch matplotlib übernommen, wobei dies ein großen Bottleneck darstellte, das diese Library nicht für diesen Anwendungsfall gedacht ist.

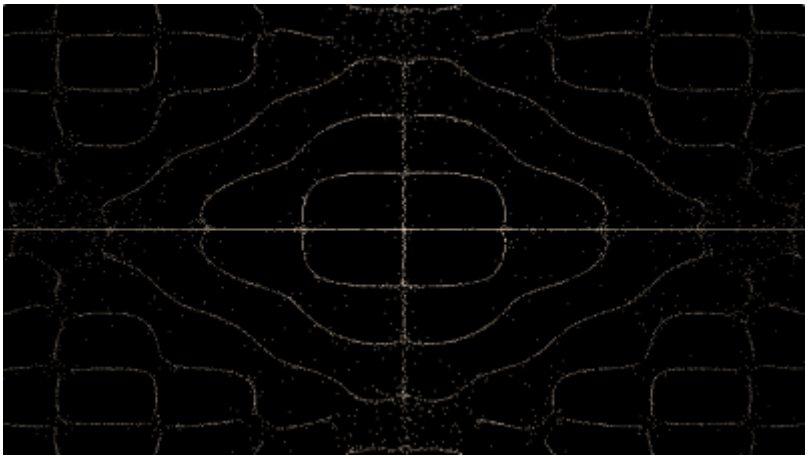


Die Zeit, um alle Partikel einen Schritt weiter zu setzen war sehr gering (< 0.1 ms). Leider konnte matplotlib da nicht mithalten und hat für das updaten der daten ca 10 ms und für die Darstellung dessen bedeutend länger gebraucht, sodass eine flüssige Echtzeitsimulation mit dieser Methode nicht möglich war.

Versuch 2 - Shader

Shader sind eine wunderbare Möglichkeit, auf die GPU zuzugreifen. Genauer haben wir uns hier mit einem Fragment-Shader beschäftigt. Mit so einem Shader kann man angeben, was für jeden Pixel auf dem Bildschirm dargestellt wird. Hierbei hat man aber keinen Zugriff auf die Berechnung der anderen Pixel, da alle parallel ausgeführt werden können sollen und Mutual Exclusion diesen Prozess deutlich verlangsamen würde. Wir haben also einen Shader [5] geschreiben, der eine Chladni-Platte mit Sand simuliert. Die Art der Simulation hat jedoch einen großen Nachteil. Wenn mehrere Partikel in einer

Iteration zum gleichen Pixel wollen, so werden alle bis auf einen ausgelöscht. Dies führt dazu, dass man immer neue Partikel erschaffen muss, was die Simulation im Endeffekt etwas unschöner erscheinen lässt. Zudem wird die Simulation nicht quadratisch dargestellt, obwohl die Platte in Theorie quadratisch ist.

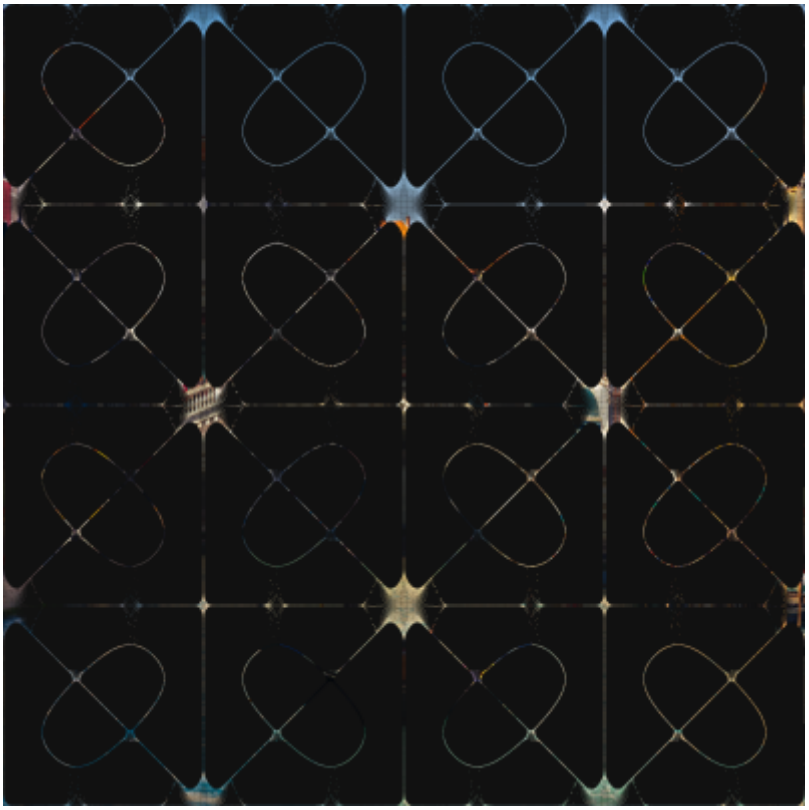


Das Endprodukt dieses Exkurses hat jedoch einen Lichtblick gegeben. Viele Partikel konnten problemlos in Echtzeit und mit hoher Framerate simuliert werden. Das Problem war nun, dass wir MIDI-Keyboard Inputs einbinden wollten. An sich ist es kein Problem, die Variablen des Shaders zur Laufzeit von Außen zu verändern, leider biete die Seite Sahdertoy keine gute Möglichkeit für Keyboard oder MIDI-Inputs. Dies führe dazu, dass wir uns mit WebGL befasst haben nach dem Motto: „Wenn Sahdertoy im Browser läuft, muss man es ja auch irgendwie selbst schaffen, was im Browser laufen zu lassen.“

Versuch 3 - Das WebGL Abenteuer

Es gibt überraschend viele Menschen, die auf die Idee kommen, sich in Ihrer Freizeit mit WebGL zu beschäftigen. Das führt dazu, dass sehr gute Tutorial Seiten entstehen, wie WebGL2 Fundamentals [6]. WebGL2 ist deutlich mächtiger als in diesem Projekt benötigt, aber auch die einfachste Möglichkeit, lokal einen Shader gut zum laufen zu bringen. Eigentlich war der Plan, den Shader aus Versuch 2 irgendwie lokal auszuführen und fertig zu sein. Jedoch gibt es auf der genannten Tutorial-Seite ein Beispiel für GPGPU Partikel [7], als könne der Autor unsere Gedanken lesen. Mit etwas Fantasie war das Projekt nun also schon fertig. Nach ein paar Stunden Copy-Pasten, googlen und sich in WebGL2 einlesen, hatten wir eine laufende Partikelsimulation, für die Sandkörner auf einer Chladni-Platte, wobei sich die einzelnen Partikel nicht gegenseitig ausgelöscht haben.

Tatsächlich hatten wir alles zunächst in WebGL implementiert (genauer WebGL1). Jedoch ist uns dann, als wir fast fertig waren, aufgefallen, dass WebGL2 deutlich bessere Features bietet. Zum Beispiel ist man bei der Array-Größe nicht gebunden durch die Größen von Texturen. Des Weiteren ist es möglich Bilder als Texturen zu verwenden, die als Pixelmaße keine Zweierpotenzen sind. Auch ist unser Problem in WebGL2 etwas effizienter berechnenbar, da man bestimmte Features bei Bedarf ein- und ausschalten kann. Wir ersparen dem Leser hier nun weitere Details und verweisen noch einmal auf die oben genannte Turtorial-Seite und präsentieren stattdessen ein fast konvergiertes Chladni-Muster mit Venedig als Hintergrundbild.



Da die Implementierung nun in JavaScript ist, konnten wir auch recht einfach ein paar Spielereien einbauen, die in der nächsten Sektion beschrieben werden.

1)

Chladnische Klanfigur https://de.wikipedia.org/wiki/Chladnische_Klangfigur#Mathematisches_Modell

2)

Creating Digital Chladni Patterns https://thelig.ht/chladni/

3)

Wie viel ist viel? https://www2.klett.de/sixcms/media.php/229/700371_0101.pdf

4)

NVIDIA Warp - Preview Release https://developer.nvidia.com/warp-python

5)

Chladni-Shader https://www.shadertoy.com/view/cssfRr

6)

WebGL2 Fundamentals https://webgl2fundamentals.org/

7)

GPGPU Partikel https://webgl2fundamentals.org/webgl/webgl-gpgpu-particles-transformfeedback.html

From:
http://www.labprepare.tu-berlin.de/wiki/ - **Project Sci.Com Wiki**

Permanent link:
http://www.labprepare.tu-berlin.de/wiki/doku.php?id=clia&rev=1696419460

Last update: **2023/10/04 13:37**