Klangerzeugung durch additive Synthese

Fabian Eberts

26. Mai 2021

Zusammenfassung

Es wird gezeigt, wie durch additive Synthese und Hüllkurven ein künstlicher Klavierklang erzeugt werden kann. Zunächst werden die mathematischen Grundlagen geschaffen. Dann wird die Implementierung mithilfe der Mozilla Web Audio API erläutert.

Inhaltsverzeichnis

1. Synthetische Klangerzeugung

Dieses Kapitel erörtert zwei Varianten der synthetischen Klangerzeugung. Anschließend werden die mathematischen Grundlagen der additiven Synthese beschrieben.

1.1. Arten der Klangerzeugung

Die synthetische Klangerzeugung bietet die Möglichkeit, einen Ton rein algorithmisch zu erzeugen und dabei den spezifischen Klangcharakter eines Instruments zu modelliert. Zwei der verschiedenen Formen der Klangsynthese sollen nachfolgend kurz erläutert werden [Gui16, S. 54 f.][Wik18]:

- Bei der *subtraktiven Synthese* werden aus einem obertonreichen Rohmaterial bestimmte Frequenzspektren herausgefiltert. Am Ende bleibt dann das gewünschte Audiosignal übrig.
- Genau umgekehrt funktioniert die *additive Synthese*. Hier wird durch Kombination zahlreicher Sinusschwingungen der gewünschte Klang „zusammengebaut“. Ein Grundton wird durch das Hinzufügen anderer Sinusschwingungen um Obertöne angereichert. Zusätzlich kann der Amplitudenverlauf des Signals durch Hüllkurven geformt werden.

Bei der ersten Variante ist noch immer ein Ausgangsmaterial nötig, während die zweite Variante ganz ohne Input von Außen funktioniert. Mit der additiven Synthese kann clientseitig rein algorithmisch ein Klavierklang modelliert werden.

1.2. Grundlagen der additiven Synthese

Grundlage eines Tones ist immer eine Sinusschwingung. Sie hat die Frequenz der entsprechenden Note. Darauf aufbauend kommen Obertöne hinzu, welche in einem bestimmten Verhältnis zur Grundfrequenz stehen. Diese Schwingungen sind leiser als die Grundschwingung. Hinzu kommt ein instrumentenspezifischer Amplitudenverlauf, welcher mithilfe von Hüllkurven erreicht wird. So wird der Ton eines Klaviers beispielsweise nach dem Anschlagen einer Taste kontinuierlich leiser. Nachfolgend wird gezeigt, wie dieser Sachverhalt mathematisch modelliert werden kann.

1.2.1. Grundton und Amplitudenverlauf

Zunächst benötigt man eine reine Sinusschwingung. Abbildung 1 zeigt den Funktionsgraphen der Sinusfunktion $s(t) = \sin(\omega t)$ mit der Kreisfrequenz $\omega = 2\pi f$ mit $f = 2\,\text{Hz} = 2 \cdot \frac{1}{\text{s}}$ [Gui16, S. 4 f.]. Alle weiteren Veränderungen des Klangs bauen auf dieser Funktion auf.

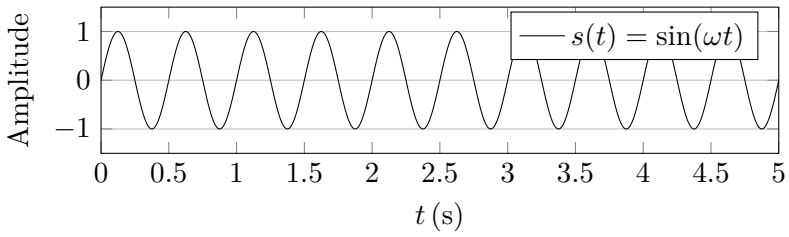

Abbildung 1: Eine gewöhnliche Sinusfunktion mit der Kreisfrequenz $\omega = 2\pi f$ mit $f = 2\,\text{Hz} = 2 \cdot \frac{1}{\text{s}}$.

Zunächst soll der Amplitudenverlauf geformt werden. Nach dem Tastenanschlag wird der Klang immer leiser (Decay). Das kann mit einer Exponentialfunktion nachgebildet werden. Die e-Funktion $d(t)$ hat einen stetig fallenden Verlauf, wie in Abbildung 2 zu sehen ist. Direkt nach dem Tastenanschlag, wenn der Hammer gegen die Saite schlägt, entsteht allerdings kurzzeitig ein hoher Lautstärkepegel (Attack). Das bildet die e-Funktion $a(t)$ nach. Kombiniert man beide Funktionen, ist das Ergebnis $f_{da}(t)$ der charakteristische Amplitudenverlauf eines Klaviers.

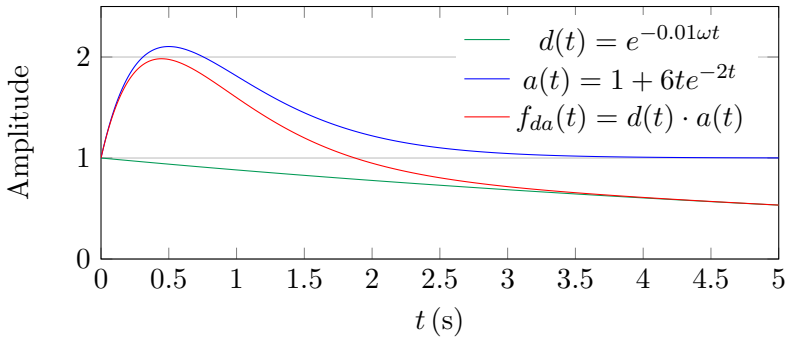

Abbildung 2: Erzeugen des gewünschten Amplitudenverlaufs $f_{da}(t)$ durch Multiplikation zweier Exponentialfunktionen.

Wendet man die so entstandene Funktion $f_{da}(t)$ auf die Sinusfunktion $s(t)$ an, sieht das Ergebnis aus wie die Funktion $f_{sda}(t)$ in Abbildung 3.

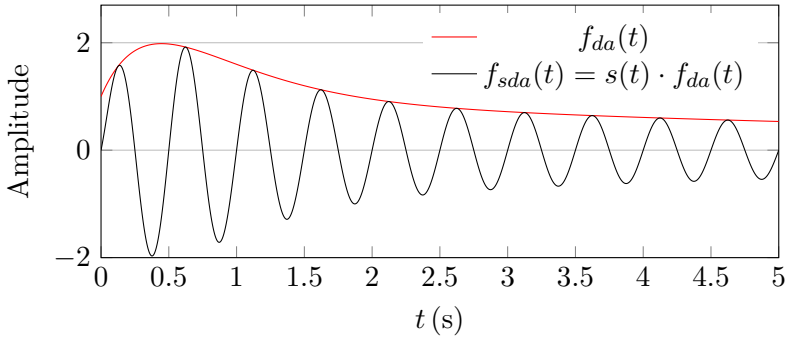

Abbildung 3: Formung des Amplitudenverlaufs einer Sinusfunktion.

1.2.2. Obertöne

Da eine Sinusschwingung jedoch sehr unnatürlich klingt, muss sie mit Obertönen „angereichert“ werden. Dazu werden weitere (leisere) Sinusschwingungen hinzugefügt. Die Frequenzen dieser Obertöne sind häufig ein Vielfaches der Grundfrequenz. In Abbildung 4 ist die schon bekannte Sinusfunktion und eine weitere Funktion $o(t)$ mit vierfacher Frequenz und halber Amplitude zu sehen.

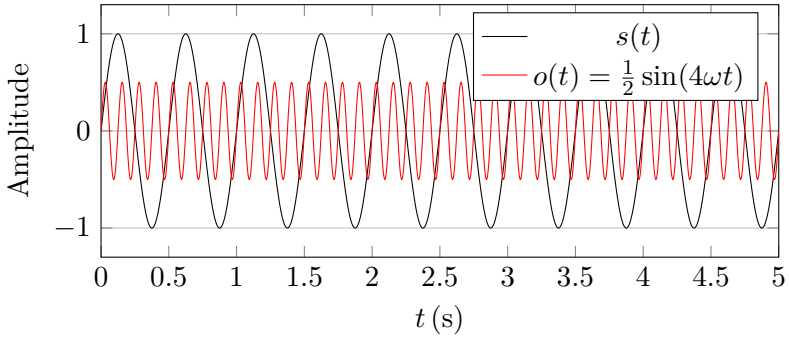

Abbildung 4: Zwei Sinusfunktionen mit unterschiedlicher Frequenz und Amplitude.

Durch Addition dieser beiden Sinusfunktionen erhält man die Funktion $f_{so}(t)$, bei der die ursprüngliche Periodenlänge der Grundschwingung noch zu erkennen ist (Abbildung 5). Zusätzlich enthält diese nun aber einen Oberton. Auf dieselbe Weise können weitere Obertöne hinzugefügt werden.

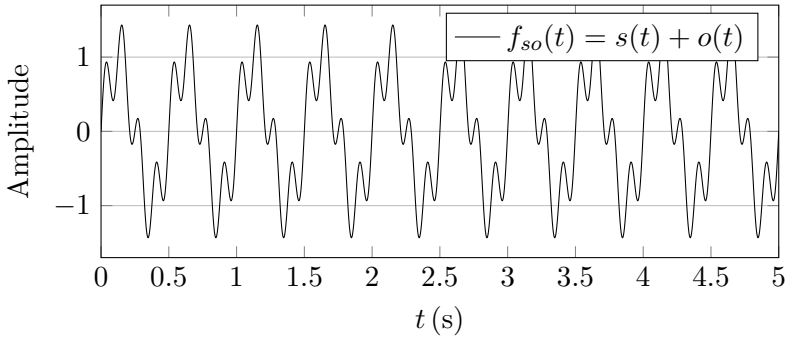

Abbildung 5: Das Additionsergebnis zweier Sinusfunktionen.

Zuletzt wird der auf diese Weise veränderte Sinuston noch mit dem zuvor modellierten Amplitudenverlauf kombiniert (Abbildung 6). Das Ergebnis ist ein primitiver Klavierklang.[1]

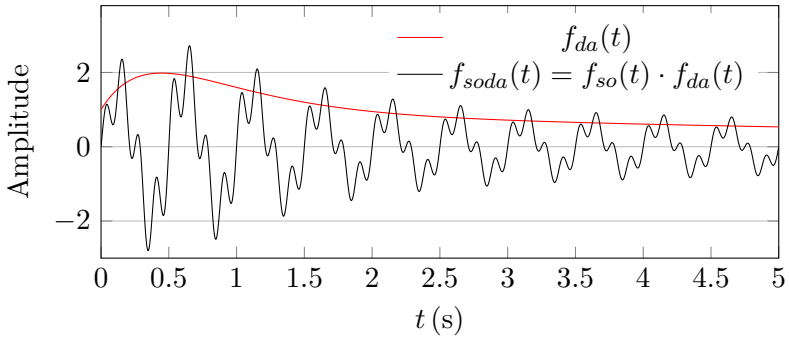

Abbildung 6: Synthetisch erzeugter Klavierklang mit typischem Amplitudenverlauf und einem Oberton.

Im nächsten Kapitel wird die Implementierung der soeben geschilderten Methode der Klangsynthese erläutert.

[1] Die hier gewählten Parameter weichen von denen der Implementierung ab. Sie wurden hier zur Vereinfachung der Funktionsgraphen angepasst.

2. Implementierung

Dieses Kapitel schildert die Implementierung der Klangerzeugung durch Anwendung der zuvor beschriebenen Methode.

2.1. Web Audio API

Für die Audioausgabe wurde die *Web Audio API* verwendet, welche Teil der Mozilla Web API ist. Dabei handelt es sich um eine JavaScript-API. Sie arbeitet mit sogenannten *Audio-Nodes*. Diese werden innerhalb eines *Audio-Contextes* zu einem *Audio-Routing-Graph* verkettet. Jeder Node kann das Audiosignal verändern. So könnte ein Source-Node mithilfe eines Oszillators eine Sinusschwingung erzeugen, welche dann von den folgenden Nodes weiterverarbeitet werden kann. Alternativ können Samples generiert und in Arrays geschrieben werden, welche dann an einen Ausgabepuffer weitergereicht werden, der das Sample-Array abspielt [Moz21b].

Der Vorteil der letzteren Methode ist, dass sie die größte Flexibilität bei der Gestaltung des Klangs erlaubt. Ein potenzieller Nachteil ist, dass das Sample vor dem Abspielen bereits fertig generiert worden sein muss. Der größte Vorteil ist aber, dass das Befüllen des Arrays mit Samples durch rein mathematische Methoden geschehen kann.

Im Folgenden wird die Implementierung vorgestellt.

2.2. Berechnung der Abtastwerte

In einem der Tutorials [Moz21a] zur Mozilla Web Audio API wird die Implementierung eines virtuellen Klaviers geschildert. Die dortige Umsetzung hat allerdings einige Nachteile: Es wird ein Array verwendet, welches die Klaviertasten auf festcodierte (!) Frequenzen abbildet. Das Array enthält dementsprechend mehrere Dutzend Einträge. Außerdem wird nur ein reiner Sinuston erzeugt.

Statt eines langen Arrays werden die konkreten Frequenzen in der Implementierung der hier entwickelten Anwendung dynamisch berechnet. Dazu wird folgende Funktion verwendet:

$$f(i) = f_0 \cdot \sqrt[12]{2^i} = f_0 \cdot 2^{i/12}$$

Von Oktave zu Oktave verdoppelt sich die Frequenz eines Tons. Es handelt sich also um ein exponentielles Wachstum. Unter den vielen – teils historischen – Stimmungssystemen ist heute die *gleichstufige Stimmung* am weitesten verbreitet. Bei dieser Stimmung wird die Oktave in zwölf gleich große Halbtonschritte unterteilt [Wik20]. Die obige Funktion

berechnet die Frequenz eines Tons relativ zu einer Referenzfrequenz f_0, die in der folgenden Implementierung auf 440 Hz festgelegt wurde:

```
function noteToFrequency(note) {
    return 440 * Math.pow(2, note / 12)
}
```

Eine Parameterprüfung kann entfallen, da negative Werte für `note` bedeuten, dass der gewünschte Ton um entsprechend viele Halbtonschritte *tiefer* erklingt als der Referenzton. Da die Frequenz einer Exponentialfunktion folgt, kann sich der Funktionswert 0 Hz nur annähern, aber nie erreichen.

Wenn die Frequenz des zu spielenden Tons berechnet wurde, kann aufbauend auf der daraus resultierenden Sinusschwingung ein Klavierklang erzeugt werden:

```
1  function waveFunction(sampleNumber, frequency) {
2      const f = frequency / audioContext.sampleRate
3      const w = 2 * Math.PI * f
4      const decayFunction = Math.exp(-0.001 * w * sampleNumber)
5
6      var sample = Math.sin(1 * w * sampleNumber) * decayFunction
7      sample += Math.sin(2 * w * sampleNumber) * decayFunction / 2
8      sample += Math.sin(3 * w * sampleNumber) * decayFunction / 4
9      sample += Math.sin(4 * w * sampleNumber) * decayFunction / 8
10     sample += Math.sin(5 * w * sampleNumber) * decayFunction / 16
11     sample += Math.sin(6 * w * sampleNumber) * decayFunction / 32
12     sample += sample * sample * sample
13     sample *= 1 + 16 * sampleNumber * Math.exp(-6 * sampleNumber)
14
15     return sample
16 }
```

Diese Funktion wendet das beschriebene mathematische Vorgehen an. Während rein physikalisch eine Schallwelle in Abhängigkeit der Zeit verläuft, muss hier die *Abtastrate* (englisch Samplerate) berücksichtigt werden. Dabei wird eine Sekunde in eine bestimmte Anzahl von Samples unterteilt – das Signal wird quantisiert. Die tatsächliche Samplerate kann mithilfe der Web Audio API ausgelesen werden. In Zeile 2 wird die Frequenz entsprechend umgerechnet. Der Rest der Funktion folgt dem bereits beschriebenen Schema: In Zeile 4 wird die mit der Zeit sinkende Amplitude berechnet (Decay). In Zeile 6 folgt die grundlegende Sinusschwingung. Danach werden die Obertöne mit steigender Frequenz (`2` bis `6`) und sinkender Lautstärke (`2` bis `32`) hinzuaddiert. In Zeile 12 wird der Klang noch „verdichtet“, sodass er etwas voller klingt. Zuletzt folgt in Zeile 13 die Anhebung der Amplitude beim Klaviertastenanschlag (Attack).

2.3. Audioausgabe

Die so berechneten Samples werden in ein Array geschrieben, welches dann an einen Ausgabepuffer weitergereicht wird:

```
1  window.AudioContext = window.AudioContext || window.webkitAudioContext
2  var audioContext = new AudioContext()
3
4  var sampleArray = []
5  const volume = 0.25
6  var noteDuration = 1.5 // Abspieldauer in s
7  const nSamplesFirstNote = audioContext.sampleRate * noteDuration
8  var frequency = noteToFrequency(this.currentRootNote)
9
10 for(var sampleNumber = 0; sampleNumber < nSamplesFirstNote; sampleNumber++)
11     sampleArray[sampleNumber] = waveFunction(sampleNumber, frequency) * volume
12
13 // dasselbe noch mal fuer den Intervallton
14
15 playSamplesArray(sampleArray)
```

Zunächst wird ein `AudioContext` erzeugt. Danach wird das `sampleArray` mithilfe der weiter oben vorgestellten `waveFunction` in einer Schleife befüllt. Beide Töne[2] werden nacheinander in das Array geschrieben, welches anschließend an `playSamplesArray()` übergeben wird:

```
1  function playSamplesArray(sampleArray) {
2      var tempAudioBuffer = new Float32Array(sampleArray.length)
3
4      for(var sampleNumber = 0; sampleNumber < sampleArray.length; sampleNumber++)
5          tempAudioBuffer[sampleNumber] = sampleArray[sampleNumber]
6
7      var audioBuffer = audioContext.createBuffer(1, tempAudioBuffer.length,
8                                                  audioContext.sampleRate)
9      audioBuffer.copyToChannel(tempAudioBuffer, 0)
10
11     if(this.bufferSource != null)
12         this.bufferSource.stop(0)
13
14     this.bufferSource = audioContext.createBufferSource()
15     this.bufferSource.buffer = audioBuffer
16     this.bufferSource.connect(audioContext.destination)
17     this.bufferSource.start(0)
18 }
```

[2] Bei der Demo-Applikation handelt es sich um einen Gehörtrainer für Intervalle. Daher werden hier zwei Töne erzeugt.

In dieser Funktion wird das Array an den Ausgabepuffer `bufferSource` übergeben. Mit `this.bufferSource.start(0)` wird die Audioausgabe veranlasst. Vorher wird in den Zeilen 11 und 12 überprüft, ob momentan schon ein Signal abgespielt wird. Würde man diese Überprüfung unterlassen, könnte man durch wiederholtes Betätigen des Buttons zum erneuten Abspielen mehrere Audioausgaben gleichzeitig starten, welche sich dann akustisch überlagern und zu unerwünschten Ergebnissen führen würden. Damit diese Überprüfung überhaupt stattfinden kann, war es nötig, `bufferSource` als Datenattribut in die Vue-Instanz aufzunehmen.

Autoplay-Regeln

Bei einem Test im Google-Chrome-Webbrowser fiel auf, dass nach dem Laden der Anwendung keine Audioausgabe erfolgte. Stattdessen wurde in der Entwicklerkonsole die Meldung `"The AudioContext was not allowed to start. It must be resumed (or created) after a user gesture on the page."` ausgegeben. Ursächlich dafür sind die relativ neuen Autoplay-Regeln von Google. Das automatische Abspielen von Medieninhalten ist erst nach einer Benutzerinteraktion erlaubt [Bea21]. Eine mögliche Lösung wäre, beim Laden der Anwendung ein Popup oder einen Start-Button anzuzeigen, welcher vom Benutzer betätigt werden muss.

Literaturverzeichnis

[Bea21] François Beaufort. *Autoplay Policy Changes*. Google. 25. Feb. 2021. URL: https://developers.google.com/web/updates/2017/09/autoplay-policy-changes (besucht am 02. 03. 2021).

[Gui16] Dieter Guicking. *Schwingungen. Theorie und Anwendungen in Mechanik, Akustik, Elektrik und Optik*. 1. Aufl. Wiesbaden: Springer Vieweg, 2016.

[Moz21a] Mozilla. *Example and tutorial: Simple synth keyboard*. Mozilla Foundation. 19. Feb. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Simple_synth (besucht am 02. 03. 2021).

[Moz21b] Mozilla. *Web Audio API*. Mozilla Foundation. 19. Feb. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (besucht am 02. 03. 2021).

[Wik18] Wikipedia. *Additive Synthese*. 28. Okt. 2018. URL: https://de.wikipedia.org/w/index.php?title=Additive_Synthese&oldid=182227583 (besucht am 02. 03. 2021).

[Wik20] Wikipedia. *Gleichstufige Stimmung*. 8. Okt. 2020. URL: https://de.wikipedia.org/w/index.php?title=Gleichstufige_Stimmung&oldid=204374287 (besucht am 02. 03. 2021).