Einleitung

Wenn ein akustisches Instrument einen Ton erzeugt, dann verläuft die Lautstärke der Zeit gegenüber von Instrument zu Instrument verschieden. Dies wird in der Klangsynthese Mithilfe eines „Envelopers“ (Hüllkurve) simuliert. Dabei wird in vier Phasen eingeteilt: Attack, Decay, Release, Sustain. Beim Anschlag des Tons bestimmt der Attack wie schnell die höchste Lautstärke erreicht wird. Anschließend fällt die Lautstärke in der Decay-Phase bis zum „Sustain-Level“. Dieser Pegel wird je nach Anschlagstärke des Tons gesetzt. Der Ton wird solange in dieser Phase gehalten bis die Taste losgelassen wird. Dann beginnt die Release-Phase in der die Lautstärke vom Sustain-Level bis 0 fällt.

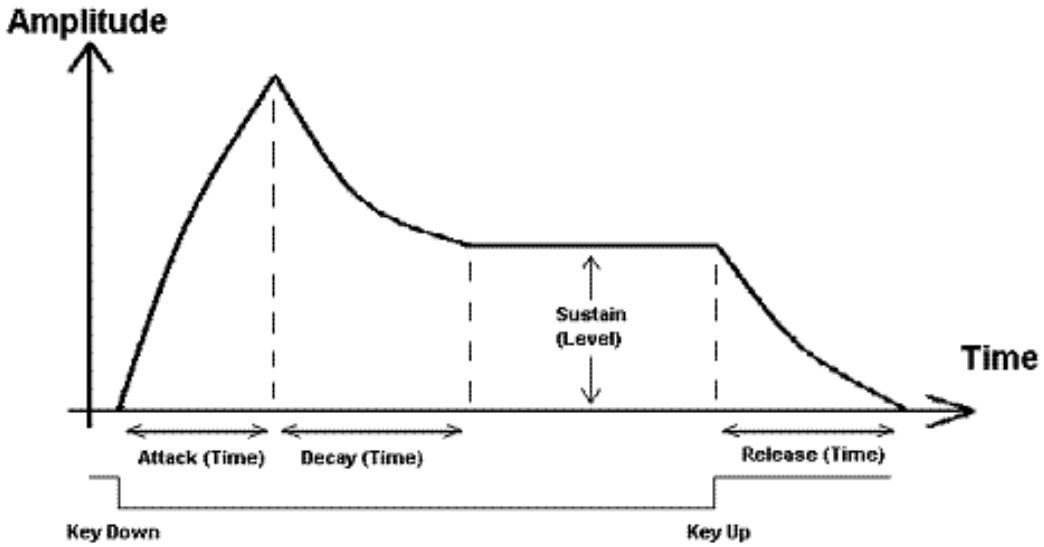
Klassen: ADSREnvelope.h und ADSREnvelope.cpp

Funktionen: nextSample(), calculateMultiplier(), enterStage()

Der verwendete Algorithmus für den Multiplier ist von **Christian Schonebeck** (http://www.musicdsp.org/showone.php?id=189).

In unserem Envelope Generator gibt es noch den Zustand: „OFF“, also aus. Attack, Decay und Release sind Zeitwerte, während Sustain ein Pegelwert ist. Daher springt der Generator von selbst aus den Attack, Decay und Release Phasen. In unserem Envelope wird dann die Funktion enterStage() aufgerufen, um in die nächste Phase zu springen. Aus den Phasen OFF und Sustain springt der Generator nur wenn enterStage() aufgerufen wird.

1



Hauptteil

Zuerst muss der ADSR-Envelope wissen in welcher Phase er sich gerade befindet. Dafür gibt es die Variable "**currentStage"**. In der Funktion "**nextSample()**" wird geprüft ob der Generator sich nicht in der "Off" oder "Sustain" Phase befindet, da nur in der "Attack", "Decay" und "Release" Phase Veränderungen an der Kurve stattfinden. Nach einer vom Benutzer festgelegten Zeit verlässt der Generator eine Phase und geht in die nächste über, dazu werden "currentSampleIndex" und "nextStageSampleIndex" miteinander verglichen. Sind die beiden gleich, dann wird "newStage" die nächste Phase zugewiesen, anschließend wird diese der Funktion "**enterStage()**" übergeben.

```
double ADSREnvelope::nextSample()
{
if (currentStage != ENVELOPE_STAGE_OFF &&
currentStage != ENVELOPE_STAGE_SUSTAIN)
{
if (currentSampleIndex == nextStageSampleIndex)
{
EnvelopeStage newStage = static_cast<EnvelopeStage>((currentStage + 1) % kNumEnvelopeStages);
enterStage(newStage);
}
currentLevel *= multiplier;
currentSampleIndex++;
}
return currentLevel;
}
```


Nima Chizari 2147812, Andreas Höpcke 2156953

Die Funktion **"enterStage()"** greift auf die Funktion **"calculateMultiplier()"** zu, um in der "Attack", "Decay" und "Release" Phase zu berechnen, wann diese auslaufen. Da unser Gehör Lautstärke logarithmisch wahrnimmt, wird dort der diskrete Laustärke Wert für das Sample so berechnet, das eine exponentielle Kurve entsteht. Der Variable "currentStage" wird die aktuelle Phase zugewiesen und der aktuelle Index wieder auf 0 gesetzt, dann folgt eine Abfrage. Hier ist der else-Teil interessant, denn dort bekommt der Index für das nächste Sample einen Wert. Im Switch wird dann schließlich der Multiplier für die jeweilige Phase berechnet, welcher in **"nextSample()"** zu "currentLevel" hinzuaddiert wird. Der aktuelle Index wird ebenfalls hochgezählt um den Zeitablauf im Auge zu behalten. So wird die Funktion "**nextSample()**" für jedes neue Sample aufgerufen.

```
void ADSREnvelope::enterStage(EnvelopeStage newStage) {
if (currentStage == newStage) return;
currentStage = newStage;
currentSampleIndex = 0;
if (currentStage == ENVELOPE_STAGE_OFF ||
currentStage == ENVELOPE_STAGE_SUSTAIN) {
nextStageSampleIndex = 0;
}
else {
nextStageSampleIndex = stageValue[currentStage] * sampleRate;
}
switch (newStage) {
case ENVELOPE_STAGE_OFF:
currentLevel = 0.0;
multiplier = 1.0;
break;
[...]
```


Nima Chizari 2147812, Andreas Höpcke 2156953