Examensarbete, 20 poäng

Att hantera digitala ljudbibliotek

En studie i samplade trummor

Christoffer Lindmark

c02lcr@cs.umu.se

Sammanfattning

Toontrack Music AB är ett företag som sysslar med att sampla trumljud. Dessa samplingar skall sen datorer kunna spela upp för att simulera ett trumset. Mjukvara behövs som hjälper dem i hanteringen av de ljudfiler de spelat in. Rapporten behandlar hur själva samplingen av ett trumset går till men fokuserar även på hur ljud behandlas i en dator och vad en programmerare kan använda för verktyg för att spela upp digitalt ljud i sitt program. Ljudbehandling i programmeringsspråket Java ägnas ett helt kapitel åt eftersom det var detta som användes i utvecklingen av programmet. Dessutom beskrivs allt från designfasen till slutliga kommentarer om den mer färdiga produkten.

Abstract

Toontrack Music AB is a company that is in the market of sampling drum sounds. Computers will later be able to replay these samples to simulate a drum kit. Software was needed to help in managing of all the sound files that they record. The report will tell about how sampling of a drum kit is done but it will also focus on how sound is managed in a computer and what tools a sound programmer can use to enable digital audio in an application. Programming audio in Java is left a chapter of its own since it was the method used in the development of the program. Furthermore the report covers everything from the design phase to final comments about the more advanced product.

Innehållsförteckning

1 Inledning

1.1 Toontrack Music AB

Toontrack Music AB är ett av världens ledande företag inom samplade mjukvarutrummor. Deras affärsidé är att spela in trumljud från riktiga trummor och sen via en så kallad sampler, eller mjukvarusynth, spela upp realistiska trummor. Deras två kanske mest kända produkter är den lite äldre Drumkit From Hell Superior och deras senaste EZDrummer. I dessa produkter ingår en sampler som kan kopplas in i i stort sett alla moderna musikprogram och ett ljudbibliotek som samplern använder sig av. Användaren kan spela upp trummorna genom att skicka signaler till programmet, antingen från ett instrument, t.ex. ett digitalt trumset, eller från en data-fil. Mycket av musiken som hörs på radio idag innehåller inte riktiga trummor utan använder sig av just samplade sådana.

En del av Toontracks motto är att få samplade trummor att låta så likt riktiga trummor som möjligt. En del i att åstadkomma detta är att de spelar in en stor mängd av trumljud som de sedan digitaliserar. När trummorna skall spelas in kan det bli upp emot 100 000 ljudfiler som sedan behövs ordnas och döpas så att rätt ljud kan spelas upp vid rätt tillfälle. Tidigare har detta jobb gjorts manuellt vilket har varit väldigt tidskrävande. Toontrack såg därför möjligheten med att låta ett program göra mycket av jobbet automatiskt samt snabba upp processen för de delar som behövde göras manuellt.

De behövde ett verktyg som kunde visualisera och ge en överblick av alla ljudfiler, samt döpa och ordna dem precis efter användarens önskemål. Programmet skulle även kunna mixa ihop för uppspelning de ljud som senare kommer att spelas upp samtidigt men även ge möjligheten att lyssna på ljuden separat. Lättare ljudredigering skulle också vara möjligt. Utöver detta behövde programmet kunna exportera alla ljudfiler till en bestämd katalogstruktur som samplern sedan kunde använda sig av.

1.2 Bakgrund och problemmotivering

Toontracks problem var att tidigare hade allt arbete med att ordna ljudfilerna gjorts manuellt. Detta ledde (förmodligen) till många mänskliga misstag. Många av misstagen berodde troligen på trötthet och slarv, men det fanns säkert också ett behov att bättre visualisering

av vad som egentligen utförts. Att sitta och lyssna på tusentals trumslag varje dag och försöka ordna dem efter bästa förmåga förstår man är inte helt lätt.

Ett förslag på till hur detta problem kunde lösas utarbetades av Erik Phersson på Toontrack. Förslaget var ett grafiskt verktyg där alla sammanhängande ljudfiler (av samma slags trumslag och trumma) synliggjordes som punkter i en graf. Med hjälp av det verktyget skulle det vara möjligt att ordna och redigera ljudfilerna efter behag samt få en mycket bättre överblick på hur just det slaget lät i förhållande till andra slag av samma typ, eller rent av slag på andra trummor. Designförslaget återfinns i bilaga 1. Förslaget var dock inte fullständigt och många funktioner har lagts till och ändrats under arbetets gång, den viktigaste kanske möjligheten att exportera sitt arbete.

1.3 Övergripande syfte

Programmets övergripande syfte är att underlätta redigering och namngivning av de ljudfiler som skall användas av samplern. Mycket av jobbet kommer att ske automatiskt och även utan allt för stor hjälp av användaren ska programmet kunna ge ett något så när bra förslag på ordningen av ljudfilerna. Programmet ska vara skrivet i Java för att underlätta förflyttning mellan olika datorer och plattformar.

Det programmet lite mer specifikt ska klara av att utföra är:

1. Lägga till olika parametrar för var och hur programmet skall leta efter ljud.

2. Lägga till olika parametrar för var och hur programmet skall exportera ljud.

3. Kunna spara ovanstående parametrar.

4. Tillhandahålla en mixer som möjliggör lyssning på valfritt antal ljud samtidigt via så kallade "solo" och "mute"-knappar

5. Visualisera sammanhängande ljud i en graf där y-axeln representerar hårdheten på slaget.

6. Sortera ljuden automatiskt, och erbjuda möjligheten att manuellt ändra ordningen om användaren så önskar.

7. Ta bort oönskade ljud.

8. Ge användaren möjligheten att gruppvis ändra volymen för de ljud som kommer att spelas upp samtidigt.

9. Kunna placera likartade ljud i så kallade lager, som döps efter en speciell algoritm vid exportering.

10. Kunna exportera alla ljud efter en bestämd algoritm.

Förhoppningen är att verktyget kommer att hjälpa Toontrack en hel del när det gäller att ordna deras ljudbibliotek. Utifrån det designförslag som givits kommer det inte bara att gå snabbare för dem, utan graden av fel kommer säkert också att minska. Det i sin tur kommer att leda till mindre arbete både före och efter en ny produkt är släppt vilket troligtvis kommer att leda till att fler och bättre produkter kommer att kunna släppas i framtiden.

1.4 Avgränsningar

Från början var tanken att programmet förutom de punkter som tas upp i kapitel 1.3 även skulle kunna utföra redigering av enskilda ljudfiler. Detta krav ströks dock från originalspecifikationen eftersom det är ett tidsbegränsat projekt. Förhoppningsvis kommer det att ges mer tid till att även implementera detta vid ett senare tillfälle.

2 Hur ett trumset digitaliseras

I detta teoriavsnitt beskrivs hur Toontracks digitalisering av ett trumset går till. När ett analogt trumset spelas in används en uppsättning av mikrofoner. Dessa placeras runt om trumsetet och i rummet och är av olika typ beroende på vad de skall spela in för ljud. Nedan följer en bild på ett trumset. Några vanliga mikrofonplaceringar är utmärkta på bilden. Notera även följande begrepp:
Snare drum – virveltrumma
Tom – puka
Kick/Bass drum – golv/bas-trumma
OH – Over Head, överhäng

Figur 1: Trumset och mikrofonplaceringar [1].

Det som illustreras på figur 1 är namnen på några av de mikrofonplaceringar som kan användas vid inspelning av trummor. Ofta sitter en mikrofon nära just den trumma som den skall spela in. Detta gäller till exempel kick- och hi-hat-mikrofonen. Varje puka kan också ha sin egen mikrofon. Ofta döps mikrofonerna efter vad de spelar in för att de ska bli lättare att hålla reda på. Snare top spelar till exempel in ljudet som kommer från virveltrummans ovansida. Det finns dock andra mickar som inte sitter precis bredvid någon trumma eller cymbal, detta gäller bland annat för OH-mikrofonerna. Dessa mikrofoner sitter ofta en bit ovanför trumsetet och används för att spela in flera olika cymballjud och för att ge trumsetet en mer levande känsla.

När trumljud skall spelas in och digitaliseras tas varje slag upp i alla mickar, så även mickar som inte är placerade vid trumman som slogs på registrerar en ljudsignal. Alla ljuden spelas in och sparas i separata

ljudfiler för respektive kanal (mikrofon). Dessa ljud kan sedan spelas upp tillsammans för att få ett så realistiskt ljud som möjligt.

För att klara av att variera ljuden, till exempel genom att byta ut en virveltrumma mot en annan virveltrumma, har Toontrack gjort en positionsindelning av trumsetet. Positionerna representerar en plats i trumsetet och har ofta samma namn som mickarna, men ska inte blandas ihop med dessa. Som exempel finns det oändligt många cymbalpositioner (om man har oändligt antal cymbaler på sitt trumset) men oftast ingen cymbalmick. På en position kan man sätta vilken trumma man vill, givet att ljuden finns inspelade, annars kan en ljudbild som inte överensstämmer med verkligheten uppstå.

För varje position finns också ett stort antal artikulationer för själva slaget på trumman. Artikulationen innehåller saker som var på trumman och med vilken hand slaget slogs.

Slutligen finns det också ett antal verktyg som kan användas för slaget. Det kan vara vanliga trumpinnar eller kanske en filtklubba.

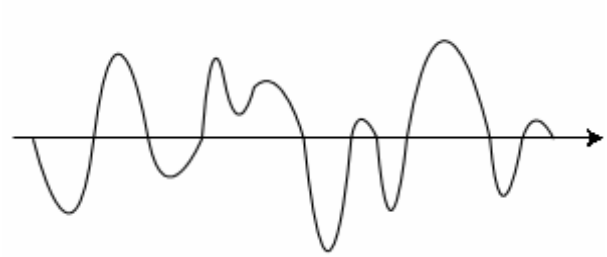
När de sedan bestämt vilka slags kombinationer av artikulationer, verktyg och trummor som skall finnas med till en ny produkt anlitas en eller flera professionella trumslagare till att fysiskt slå alla slagen samtidigt som en dator spelar in allting.

3 Ljud och datorer

3.1 Ljud i datorer

Allt ljud som lagras eller spelas upp av en dator är i digital form. Detta kommer sig av att datorer som bekant bara kan spara ettor och nollor. Ljudet behöver dock inte ha sitt ursprung i digital form utan kan vara vanliga analoga ljud såsom tal eller ett musikstycke. För att datorn ska kunna förstå något som inte är digitalt behövs någon typ av ljudhårdvara, ofta rör det sig om ett ljudkort. Ljudkortet kan via en analog/digital-omvandlare göra om en analog signal, från till exempel en mikrofon, till digital form som datorn förstår sig på. Nedan följer ett exempel på hur det kan gå till.

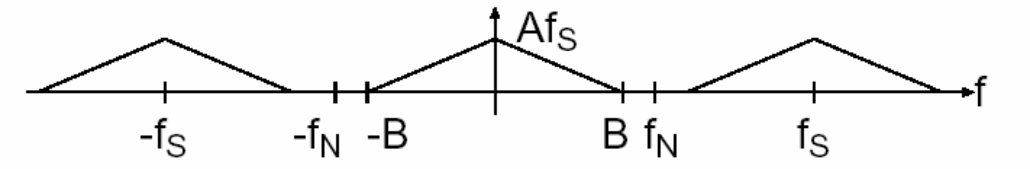
Figur 2 visar en analog signal. Den horisontella axeln representerar tiden och den vertikala axeln representerar ljudtrycket vid den tidpunkten.



Figur 2: Ljudvåg.

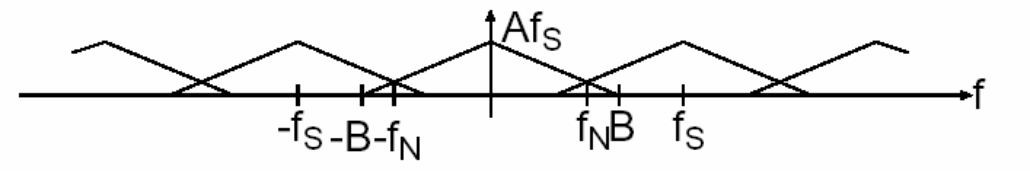
Eftersom tiden är kontinuerlig och därmed kan delas in i ett oändligt antal steg kan den inte representeras exakt i digital form. Detsamma gäller ljudtrycket som också är kontinuerligt. Signalen måste alltså diskretiseras i båda ledderna.

Tidsindelningen, eller samplingsfrekvensen, avgör hur stort frekvensområdet kommer att vara på den digitala signalen. Delar man in tiden i till exempel steg på 0.5 sekunder, alltså 2 Hz, kommer den högsta frekvensen som kan representeras vara 1 Hz. Detta fenomen betecknas ofta *samplingsteoremet* eller *Nyquistteoremet*. Fenomenet uppstår på grund av att det vid sampling bildas ett periodiskt frekvensspektrum.



Figur 3: Periodiskt spektrum. Af_S = frekvensen 0 Hz, B = högsta frekvensen i signalen, f_S = samplingsfrekvensen, f_N = Nyquistfrekvensen.

De frekvenser som är ovanför den högsta frekvensen i signalen som samplats brukar enkelt kunna tas bort med hjälp av ett lågpassfilter som bevarar endast de lägsta frekvenserna. När en för låg samplingsfrekvens har använts går detta dock inte att göra. De periodiska spektrumen överlappar då varandra och oönskade frekvenser uppstår. Dessa "falska" frekvenser kallas vikningsdistortion.

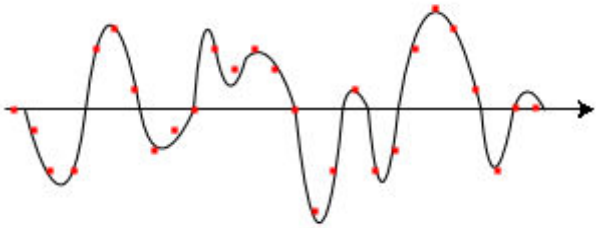


Figur 4: Överlappande spektrum vid för låg samplingsfrekvens, ger upphov till vikningsdistortion. . Af_S = frekvensen 0 Hz, B = högsta frekvensen i signalen, f_S = samplingsfrekvensen, f_N = Nyquistfrekvensen.

En människa med väldigt bra hörsel kan höra frekvenser upp till 20 kHz. På grund av detta samplas till exempel CD-skivor med 44,1 kHz för att ingen hörbar försämring av ljudet skall märkas. I andra fall är det inte lika farligt om de översta frekvenserna skulle försummas, detta gäller bland annat för telefoni.

Indelningen av ljudtrycket påverkar också kvaliteten på ljudet. Indelningen brukar kallas för bitdjup på grund av att en dator kan använda olika många bitar för att representera ljudtrycket. Används till exempel två bitar kan ljudtrycket delas in i $2^2 = 4$ olika nivåer. Som referensvärde kan nämnas att CD-kvalitet är 16 bitars indelning, vilket ger 2^{16} nivåer.

Om ljudet i figur 2 skulle delas in i diskreta punkter skulle det kunna se ut så här:



Figur 5: Diskretisering av en analog ljudvåg.

Som synes i figur 5 så passar inte punkterna perfekt in på den analoga kurvan. Detta på grund av att både samplingsfrekvensen och bitdjupet inte räcker till för att representera ljudet. Det är dock dessa punkter som datorn sparar till en ljudfil. Detta betyder så klart att en liten förändring av ljudet sker när det samplas. Därför bör en person som ska använda sig av sampling först tänka efter hur stor förändring (oftast lika med försämring i ljudkvalitet) denne är villig att tillåta. För mobiltelefoni kan den vara ganska stor medan den vid professionella musikinspelningar är väldigt liten. Vad som också bör tänkas på är att ju högre bitdjup och samplingsfrekvens desto större plats tar ljudet upp i lagringsmediet.

För att kunna spela upp ljudet på en analog förstärkare används ljudkortets digital/analog-omvandlare som via bland annat interpolering omvandlar tillbaka ljudet till en kontinuerlig analog signal som en högtalare kan spela upp.

En ljudfil kan sparas i flera olika format. Några vanliga format är WAV, AU och AIFF. Dessa format är alla helt okomprimerade och storleken på ljudfilen växer ju högre samplingsfrekvensen och bitdjupet är. För att spara på diskutrymme kan filerna även komprimeras. Några vanliga komprimeringsformat är MP3, OGG och WMA. Nackdelen med de komprimerade formaten är dock att den okomprimerade signalen aldrig går att återskapa till fullo. Vid hård komprimering är det inte heller ovanligt att ljudkvaliteten blir markant försämrad.

3.2 Programmering och ljud

Att använda sig av ljud vid programmering är något som skiljer både i svårighetsgrad och i tillvägagångssätt beroende på vilket programmeringsspråk det gäller. Detta avsnitt kommer till största delen att handla om C och C++ eftersom det är de populäraste språken, men också för att de är de språken som klarar av mest. Eftersom Java har använts i utvecklingen av programmet återfinns en mer utförlig studie i ljud för Java i kapitel 4.

3.2.1 C/C++

I programmeringsspråken C eller C++ finns det ont om riktiga standarder. Däremot finns det istället massor av lösningar vilket kan ge en programmerare stor valfrihet att välja vilken slags lösning som passar denne bäst. Det som kanske är den avgörande faktorn för vilken lösning för ljud programmeraren väljer torde vara i vilken applikation ljudet skall användas. En spelprogrammerare kan till exempel dra stor fördel av att ha färdiga 3D-funktioner för ljud, medan det för en som programmerar ip-telefoni är bättre med en lägre nivå på funktionerna där varje sample[1] går att komma åt.

En annan aspekt som bör tas hänsyn till är att C/C++ är plattformsberoende språk. Detta betyder att program inte kan köras på andra operativsystem än det de är kompilerade för utan att kompileras om. Vissa funktionsanrop med mera behöver inte heller fungera likadant på alla operativsystem. För programmeraren betyder detta ofta mycket extra jobb för att göra programmet tillgängligt på olika plattformar. Många lösningar fungerar därför bara på ett begränsat antal operativsystem.

Ett vanligt verktyg för spelutvecklare är OpenAL [3][4]. OpenAL är designat i första hand för att spela upp 3D-ljud. Det består av tre olika objekt: källa, buffert och lyssnare. Ljuden lagras in i buffertarna som buffertarna kan sedan kopplas ihop med en eller flera källor. Källorna representerar positioner i ett lyssningsrum där ljudet är tänkt skall komma från. Det tredje objektet, lyssnaren, representerar den tänkta personen som skall lyssna på ljudet. OpenAL räknar sedan ut hur ljudet skall omvandlas för att personen skall uppleva den önskade 3D-effekten. OpenAL är nära sammankopplat med grafikverktyget

[1] Ett sample är det värde som den digitala signalen har under ett samplingsintervall. En CD-skiva som är samplad med 44100 Hz har till exempel 44100 stycken samples per sekund och kanal.

OpenGL och stöds på de flesta plattformarna idag [4]. För den som vill skriva program för ljudredigering med stor åtkomst och manipulationsmöjligheter för ljuden kanske dock inte OpenAL är det bästa valet eftersom det har relativt få funktioner och agerar på en högre nivå. Möjlighet till inspelning av ljud finns inte heller.

För de som använder Microsoft Windows är den dominerande ljudstandarden idag Direct-X Audio. Detta *API* är utvecklat av Microsoft och var från början tänkt för spelutveckling i Windowsmiljö, men klarar idag av mycket mer. Det är det just nu mest kompletta ljud-API:t och passar mycket bra till så skilda användningsområden som 3D-ljud, plugin-effekter och strömmande ljud. Nackdelarna med Direct-X Audio är däremot att det är strikt bundet till Windows-plattformen och kommer troligtvis aldrig att bli något annat heller [3]. En annan nackdel är de krångliga funktionsanropen. Till exempel så är det här funktionen för att få ett ljud att sluta spela:

```
HRESULT Stop(IDirectMusicSegment* pSegment,
             IDirectMusicSegmentState* pSegmentState,
             MUSIC_TIME mtTime,
             DWORD dwFlags);
```

Ett betydligt enklare API är PortAudio [5]. Projektet PortAudio är drivet att några frivilligarbetare och är med endast 30 olika funktionsanrop ett väldigt enkelt och lättöverskådligt API [3]. Med enkelheten kommer också att programmeraren får göra det mesta själv. Själva huvudfunktionen är en callback-funktion som användaren själv skriver. Denna funktion anropas av ljudkortet några gånger i sekunden och här specificerar användaren vilka ljud som skall spelas upp för tillfället. Detta ger en stor användarfrihet, men det krävs mycket jobb även för att utföra enkla uppgifter. PortAudio har inte så många andra funktioner än de som just krävs för att vara plattformsoberoende, och därför fungerar PortAudio på de flesta kommersiella plattformarna, däribland Windows, Mac OS (8,9,X), Unix och BeOS.

Ett fjärde API är SDL_mixer som ingår i SDL (Simple DirectMedia Layer)-projektet [12]. Projektet bygger på öppen källkod och har i liknande funktionalitet som OpenAL. En fördel med SDL_mixer är dock att det är portat till väldigt många olika språk, bland annat C, C++, Java, Perl, Pascal, Python och Ada. API:t finns också tillgängligt för i stort sett alla kommersiella plattformar som används idag [3][6].

3.2.2 Andra programmeringsspråk

Förutom C/C++ har de flesta andra programmeringsspråken som används idag har ett ganska standardiserat sätt på hur ljud kan användas. Detta gäller till exempel Visual Basic, Java-Script, Flash och Perl. Att dessa språk har enklare, både i meningen att använda och i funktionalitet, utformning på funktionerna som spelar upp ljud kan bero på att de är optimerade för andra funktioner och/eller för att det skall vara så enkelt som möjligt att spela upp ett ljud.

C# och .Net har inget standardiserat API för ljud utan beror på samma sätt som C och C++ på utomstående tillverkares lösningar. Nackdelen med .Net är att det är bundet till Microsofts programvaror.

Vill man ha ett språk där det är enkelt att spela upp ljud, som är någorlunda plattformsoberoende men samtidigt ha stora möjligheter att redigera och påverka ljudet är Matlab [10] ett bra val. Matlab är designat för matematiska funktioner vilket passar alldeles utmärkt till ljudredigering. I Matlab kan användaren i princip göra vad den vill med ljudet. Många funktioner för Digital Signal Processing, *DSP,* finns också redan inbyggda i Matlab vilket gör det lätt att använda. En nackdel med Matlab kan däremot vara att det inte är designat för att göra snygga GUI:n. Däremot kan Matlab-genererade program med fördel anropas av andra språk. En annan nackdel är att Matlab är licensierat och kostar därför pengar att använda. De genererade programmen går dock i princip att använda fritt så länge användaren har licens för Matlabprogrammet [7].

4 Ljud och Java

I detta kapitel tar vi upp hur programmeringsspråket Java har implementerat hantering av digitalt ljud. Grundtanken med Java är att det skall vara ett plattformsoberoende programspråk och detta åstadkoms med en så kallad virtuell maskin som är specifik för varje plattform och som ligger som ett slags mellanlager mellan operativsystemet och javakoden.

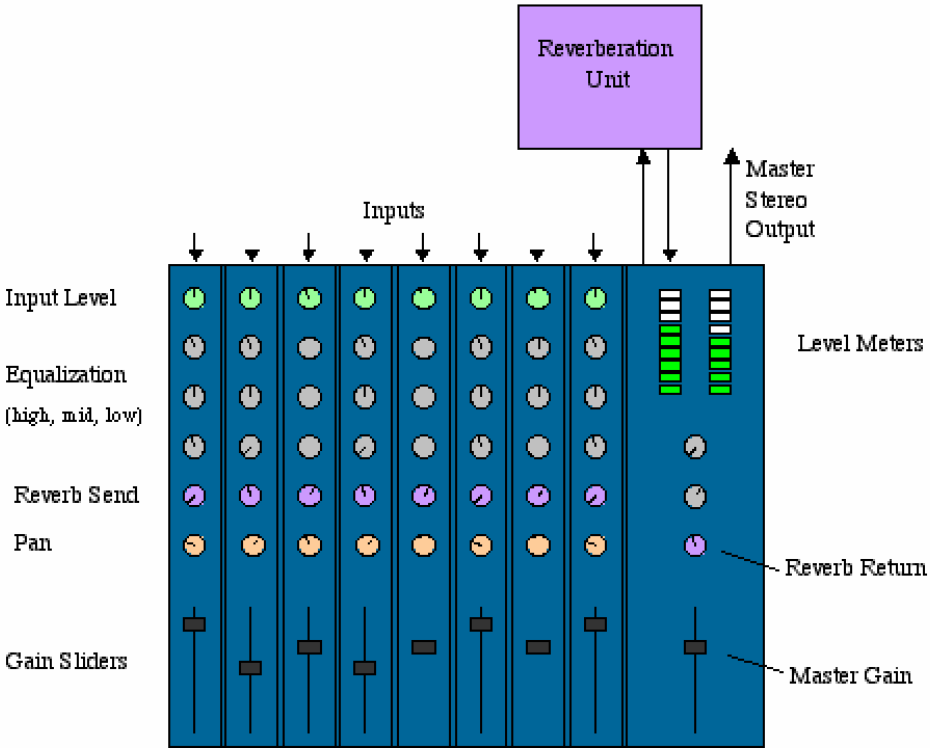
Alla javaprogram är i huvudsak en av två typer, antingen ett fristående program, eller en applet. Appleten är ett program som bara kan köras i en webbläsare och kan därför vara lite mer begränsat. Java har också två olika ljudbibliotek för de olika slags programmen. Det enklare, som bara går att köra på applets, består av bara en publik klass, *AudioClip*. Ett AudioClip representerar en ljudfil och innehåller bara de tre metoderna `start()`, `stop()` och `loop()`. Som synes så är användaren väldigt begränsad i sitt användande av ljud då varken manipulation av ljudet, val av hårdvara eller inspelning av ljud är möjlig.

Det andra biblioteket är betydligt mer invecklat och komplext. Här har meningen varit att ge användaren så stora möjligheter som det bara går med att manipulera ljudet, eller som Sun[2] uttrycker det i sin manual: "Den centrala uppgiften för Java Sound API är att förflytta bytes bestående av formaterad ljuddata in och ut ur systemet" [9]. I princip all slags ljudredigering är möjlig och det enda som sätter gränserna är tidsaspekten. Lyckligtvis verkar det som att den gamla uppfattningen om att Java skulle vara ett långsamt språk håller på att raseras och kanske en dag så kan mycket väl Java bli lika snabbt på att hantera ljud som till exempel C++ [8].

4.1 Uppbyggd som en ljudmixer

Ljudsystemet består i huvudsak av tre viktiga typer: format, mixer och port. Formatet bestämmer vilket format det är på ljudet som hanteras. Allt ljud kommer via någon slags port in i mixern där det hanteras och sedan skickas vidare till någon annan port. Själva lösningen är uppbyggd efter en modell av en fysisk icke-digital mixer.

[2] Sun Microsystems skapade Java och står för merparten av utvecklingen av Java.



Figur 6: En klasisk mixer [9].

Figur 6 föreställer en mixer som skulle kunna användas vid mixning vid till exempel en konsert. De åtta kolumnerna till vänster representerar varsin ljudkanal där ett instrument eller en mikrofon kan kopplas in. Varje ljudkanal har en input och ett antal möjliga effekter. Alla kanaler mixas sedan ihop till ett antal olika outputs, i fallet ovan, två stycken, som vanligtvis är kopplade till något slags högtalarsystem.

I Javas ljudbibliotek representerar klassen *Mixer* den fysiska mixern. För att användning av ljudbiblioteket skall vara möjligt måste användaren välja en mixer. Denna mixer kan antingen vara Javas inbyggda mjukvarumixer som använder sig av datorns standardljudinställningar eller om ljudkortstillverkaren försett sin produkt med ett sånt; ett interface till ljudkortets mixer. En mixer kan synkronisera flera ljud så att de startar och stoppar på samma gång. Den har även funktioner, eller så kallade kontroller, för att lägga på effekter. Dessa effekter kan vara både enkla effekter såsom volym eller panorering och mer avancerade saker som reverb (rumsklang) och chorus (dubblering).

Både input och output representeras av klassen *Port*. *Port* kan sedan delas in i *TargetDataLine* som representerar en inputport och *SourceDataLine* och *Clip* som representerar outputportar. En *TargetDataLine* kopplas alltid ihop med en inputport på själva ljudkortet, till exempel mikrofoningången. Innan användaren kan börja läsa data från inputporten måste denne specificera vilket format datat skall få. När det är klart kan sedan data läsas till en buffert i minnet specificerad av användaren. Att datat skrivs till en buffert betyder att såväl uppspelning och inspelning av ljudet i realtid är möjligt.

4.2 Uppspelning av ljud

Vilken av de två olika klasserna *SourceDataLine* och *Clip* som skall användas vid uppspelning av ljud beror på vart ljudet kommer ifrån och hur snabbt det skall spelas upp. Den stora skillnaden är att *Clip* inte är buffrad medan *SourceDataLine* är det.

Ett *Clip* laddar in det valda ljudet direkt till datorns internminne innan uppspelning. Detta medför att det kan ta ett tag innan ljudet spelas upp, men samtidigt också att det går väldigt snabbt om ljudet skall spelas upp flera gånger. *Clip* har funktioner för att loopas, även mellan valfria positioner i ljudet, och kan lätt spolas tillbaka eller starta exakt där användaren så önskar. En begränsning med *Clip* är dock att det inte stödjer allt för stora data så vill användaren spela upp långa ljud är det nog inte det bästa valet.

För strömmande eller längre ljud är *SourceDataLine* ett mycket bra val. En *SourceDataLine* fungerar på liknande sätt som en *TargetDataLine* men med skillnaden att den spelar upp ljud (eg. skickar ljuddata ut till en port på ljudkortet) istället för att läsa in ljud. *SourceDataLine* använder alltså en buffert där den läser ifrån och som programmet måste förse med ny data hela tiden för att uppspelningen ska ske utan hack eller klick i ljudet.

4.3 Service Provider Interface

Javabiblioteket för ljud består av tre delar, Audio, MIDI och SPI. SPI står för Service Provider Interface och är en mycket smart lösning. SPI har inga konkreta klasser utan är bara ett interface där utomstående funktioner kan implementeras. De klasser som går att implementera är läsare av ljudfiler, konverterare mellan olika format, mixers och olika in- och outputs. Poängen med SPI-interfacet är att inga nya okända funktioner behöver anropas för att använda sig av det nya biblioteket. Försöker till exempel användaren läsa in en MP3-fil utan att ha en

konverterare installerad får denne ett undantag tillbaka när denne försöker läsa in audiodata. Finns däremot en SPI-implemenatation för att läsa in MP3-format installerad på korrekt sätt fungerar inläsningen av MP3-filen utan problem. En annan fördel är att program inte behöver ges ut i en ny version för att stödja ett nytt filformat eller hårdvara. Implementeringen av vissa vanliga filformatsomkodningar behövs inte heller göras om flera gånger. Det enda som behövs göras är att tala om för programmet var de implementerade SPI-klasserna ligger.

4.4 Egna kommentarer

Jag tycker att Sun har gjort ett bra jobb med hur de lagt upp ljudbiblioteket för Java. Det mesta faller sig ganska logiskt och många bra funktioner finns. Nackdelen ligger dock i Javas grundidé, att vara plattformsoberoende. Detta betyder nämligen att alla plattformstillverkare själva måste göra uppdateringar den virtuella maskinen som är specifik för varje operativsystem när en ny version av java API:t kommer ut. Detta verkar inte vara den högsta prioriteten hos många av plattformstillverkarna; inte ens Sun själva har implementerat alla funktioner. Till exempel så fungerar inte synkronisering av strömmar på någon plattform och det samma gäller för funktionen getLevel() som skall göra något så enkelt som att undersöka den nuvarande volymen på ljudet som spelas upp. Det finns så klart lösningar på dessa problem. De flesta bygger på att programmeraren har kunskaper i digital signalhantering. Det kan till exempel vara nödvändigt att sample för sample lägga ihop två eller flera ljud för att få dem att spela exakt samtidigt. En väldigt informativ hemsida är [11] där en av författarna är ljudprogrammerare på Sun.

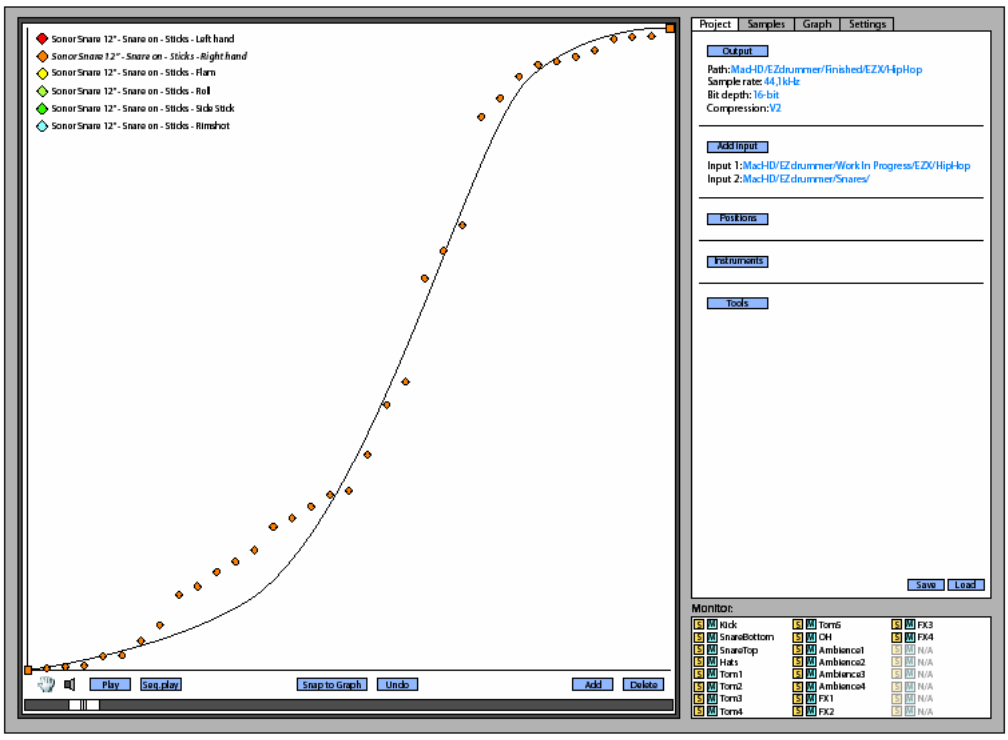
Den operativsystemstillverkare som ligger längst efter i utvecklingen av Javas virtuella maskin när det gäller ljud är i min mening Apple. Ända tills förra releasen av Java API:T (JSE 5.0) var till exempel inte inspelning av ljud tillåten på Mac OS X. Detta beror på att Apple själva är ansvariga för all implementering av Java för Mac. Java är tänkt att vara en i högsta grad integrerad del av operativsystemet. All uppdatering skall ske automatiskt och finns inte att ladda hem från Sun's hemsida där alla andra operativsystems uppdateringar finns. Med den senaste uppdateringen går det nu att spela in ljud med Java på en Mac men det finns fortfarande stora begränsningar. En stor sådan är till exempel att det inte går att spela upp ljud som har högre samplingsfrekvens än 44.1 kHz och högre bitdjup än 16-bitar. Jag tycker det hela är väldigt konstigt eftersom Apple har stor fokus på media och Java, men tydligen inte på kombinationen av de två.

5 Systemdesign

Detta kapitel kommer att ta upp hur designen av programmet gick till, både grafiskt och implementationsmässigt.

5.1 Grafikdesign

Till den grafiska utvecklingen av programmet var ett designförslag från Toontrack till stor hjälp. Eftersom verktyget är i väldigt stor utsträckning ett internt verktyg och Toontrack hade designat layouten precis efter deras egna önskemål kändes ytterligare designförslag och/eller utvärdering av förslaget inte nödvändigt. Layouten i förslaget har en klassisk look med diverse inställningar till höger och ett grafiskt fönster till vänster. Monitorn för att välja vilka kanaler som ska spelas upp befinner sig längst ner till höger. Storlek och placering av de olika komponenterna känns helt naturlig. Det grafiska fönstret där det mesta av arbetet sker tar upp störst plats och ligger direkt i blickfånget.



Figur 7: Grafiskt förslag från Toontrack, huvudfönstret.

Monitorn som troligtvis kommer att användas minst har fått en lite mer diskret placering. Det slutgiltiga programmet utgick till stor del från

detta förslag, med enbart mindre grafiska förändringar. För utförligare information om designförslaget, se bilaga 1.

5.2 Tillvägagångssätt

Arbetet påbörjades genom att strukturera upp de datatyper som skulle behövas. Typerna position, instrument, artikulation och verktyg kändes relativt självklara som egna klasser. Det samma gällde för input som innehåller information om en mapp med ljudfiler som skall läsas in av programmet.

Programmet ska inte bara kunna läsa in och förstå ljudfilerna den läst, det är ju meningen att det skulle kunna spela upp och ordna ljuden också. För detta behövs ett grafobjekt där användaren kan klicka för uppspelning och dra och släppa för att ordna. Till uppspelningen behövs också ett monitorobjekt för att kontrollera vilka kanaler som ska spelas upp samtidigt. I grafen ska det rymmas olika lager av olika typ av trumslag och instrument. Dessa lager fick också bli egna objekt. I varje lager måste det finnas ett objekt för varje kanal, så att alla ljudfiler som tillhör en specifik kanal kan stängas av när monitorn signalerar det. Slutligen måste varje kanal innehålla ett objekt för varje ljud. Detta objekt innehåller själva ljudinformationen. Om lager önskas användas så är lagerobjektet ett arv från ljudobjektet med den stora skillnaden att det innehåller fler än ett ljud.

Efter att ha strukturerat upp denna underliggande struktur övergick arbetet till Grafic User Interface (*GUI*)-programmering. Eclipses alternativ till Suns swingbibliotek kallat Standard Widget Toolkit (*SWT*) [2] användes för layouten. Enligt Eclipses hemsida [2] skulle SWT använda sig mer direkt av användargränssnittet för operativsystemet och är open-source vilket det finns många fördelar med. Målet var att efterlikna designen i förslaget från Toontrack men samtidigt göra det så smidigt, snabbt och minnessnålt som möjligt.

Under arbetets gång stöttes en hel del hinder på. Det mest var relaterat till att Java beter sig olika på Windows och Mac OS X. Förutom rena grafiska aspekter ställde också bland annat ljuduppspelning och trådhantering till bekymmer. Som nämndes i kapitel 4.4 går det för närvarande inte att spela upp ljud som har högre samplingsfrekvens än 44100 Hz via Java på en Mac OS X-maskin. Detta ledde till att omvandling av ljudet är nödvändig före uppspelning vilket både ger sämre kvalitet och en liten fördröjning innan ljudet spelas upp. Trådhanteringen fungerade ibland smärtfritt och ibland inte alls på Mac OS X så några funktioner som använder sig av trådar fick tyvärr strykas.

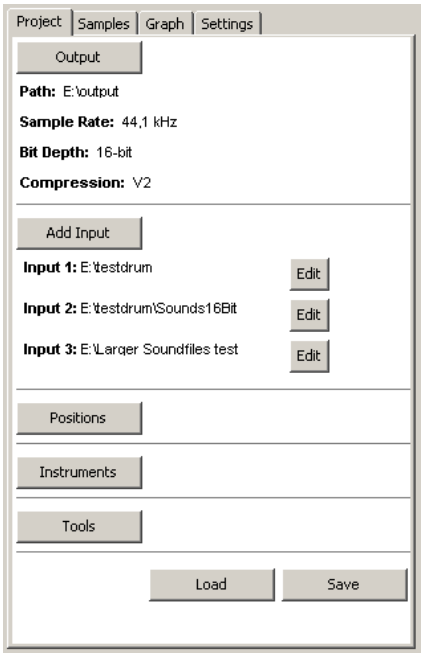
6 Analys av programmet

I det här kapitlet kommer jag att syna programmet del för del. Jag kommer även att kommentera vad som fattas, skulle kunna byggas på eller förbättras

Programmet har utvecklats i stort sett som jag hade förväntat mig även om många nya funktioner som inte var med i orginalspecifikationen har lagts till efter att det tagits i bruk.

6.1 Projektmenyn

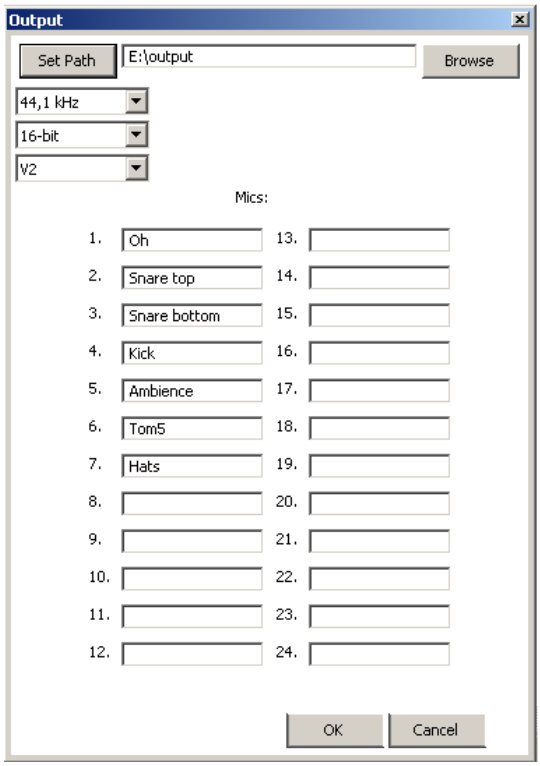
Till höger i huvudfönstret finns en projektmeny. Här väljer användaren olika undermenyer för att göra inställningar i programmet. Projektmenyn är i princip klar och kommer nog inte att förändras så mycket i framtida versioner.



Figur 8: Projektmenyn.

6.2 Output

Här bestämmer användaren var programmet skall exportera de slutliga ljudfilerna. För tillfället går det inte att välja att exportera ljudfilerna i annat format än de skickades in i. I framtiden kommer att det att vara möjligt att göra en omvandling i till exempel bitdjup eller rent av att komprimera ljudet för användning i olika slags applikationer. Användaren får också välja vilka mickar som skall finnas med vid exporteringen. Dessa mickar är de samma som kommer att finnas med på monitorn som kontrollerar vilka ljud som skall spelas upp.



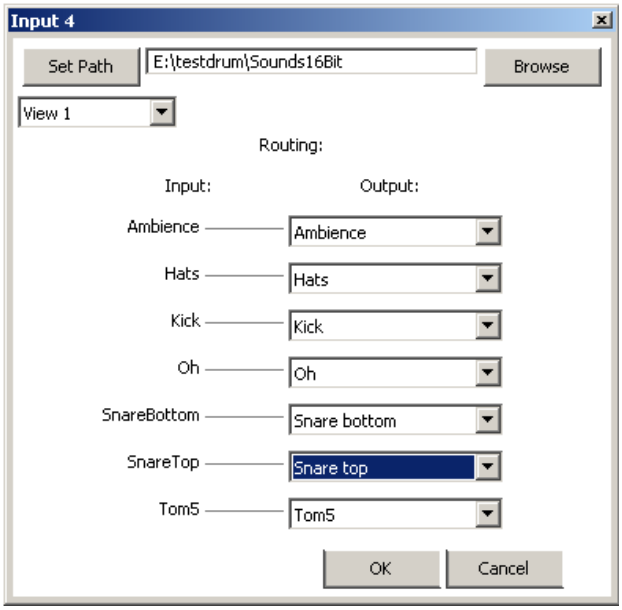
Figur 9: Outputdialogen.

6.3 Input

En input till programmet är en katalog i filsytemet. Katalogen måste ha en eller flera underkataloger i vilka det ligger ljud. När användaren sätter upp en input väljer denne också vilka inputkataloger som skall korrespondera, routas, till vilka mickar i output:en. Det är vanligt att både input- och outputmickarna har samma namn så en automatisk sortering görs där namnen överensstämmer.

Ett projekt kan ha flera inputs om användaren till exempel vill hämta en virveltrumma från något annat ställe i filsytemet. Routingschemat måste alltså innehålla för varje output alla kataloger i alla inputs där den skall

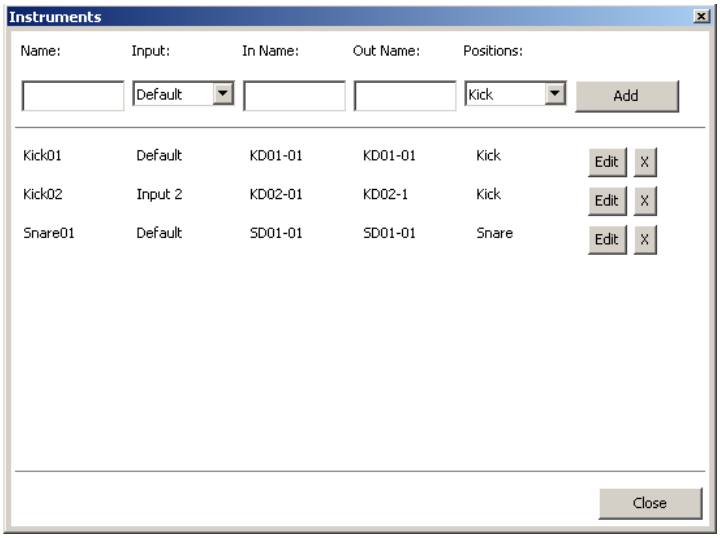
läsa ljudfiler från. En nackdel med den nuvarande versionen är att om någon outputmickarna tas bort eller läggs till förstör detta hela routingschemat och alla inputs tas bort. Detta är dock bara en temporär lösning och kommer att åtgärdas i framtiden.



Figur 10: Inputdialogen.

6.4 Positions, Instruments, Tools

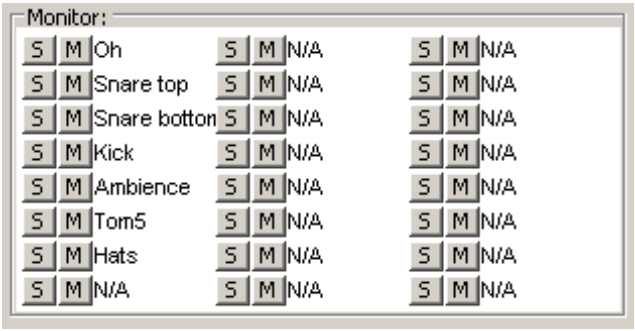
Programmet innehåller dialoger för att lägga till positioner, instrument och verktyg. Här har användaren möjlighet att både lägga till, ändra och ta bort befintliga inställningar. Dessa dialoger är troligen så som de kommer se ut i framtiden också.



Figur 11: Instrumentdialogen, där användaren lägger till nya instrument.

6.5 Monitor

Monitorn har som uppgift att bestämma vilka kanaler i ett trumslag som skall låta. En kanal är det samma som ljudet från en trummick. Framför varje kanal finns det två knappar. Den ena knappen är märkt "M" vilket står för "mute" vilket tystar ner den aktuella kanalen. Den andra knappen är märkt "S" vilket står för "solo" och gör så att kanalen spelar själv utan att några andra ljud hörs. När en ny grupp av slag, ett ljudlager, laddas in får den samma inställningar på sina kanaler som alla de andra ljudlagren. Ett alternativ vore att varje ljudlager hade sin egen inställning.



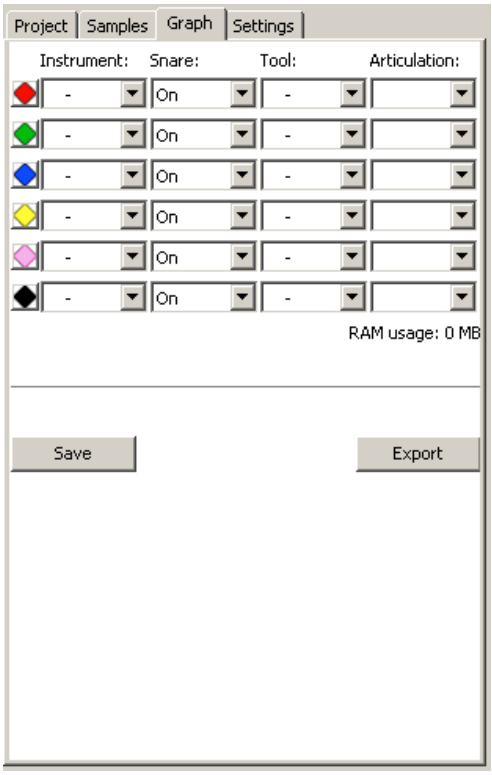
Figur 12: Monitorn, som bestämmer vilka kanaler som skall låta.

6.6 Grafmeny

I grafmenyn väljer användaren vilka grupper av slag som skall laddas in i minnet och visas på grafen. Varje ljudlager har en unik färg så att de kan skiljas åt. Proceduren för att välja ett ljud görs genom dropdown-boxar. Först väljer användaren ett instrument, sedan om virvelrasslet under virveltrumman ska vara spänt eller inte, sedan vilken slags artikulation på slaget det skall vara och till sist vilket slags verktyg slaget skall utföras med. När användaren är nöjd trycker denne på den färgade knappen till vänster om dropdown-boxarna och kanalen laddas in i minnet och visas på graffönstret till vänster.

Här visas också en mätare på hur mycket minne som ljuden just nu förbrukar. En kanske bättre mätare vore att veta hur mycket minne själva programmet förbrukar.

Det finns också två stora knappar märkta "Save" och "Export". Den första knappen används till att spara de inställningar som gjorts i grafen och den andra används till att exportera projektet.



Figur 13: Grafmenyn, där användaren laddar in ljud.

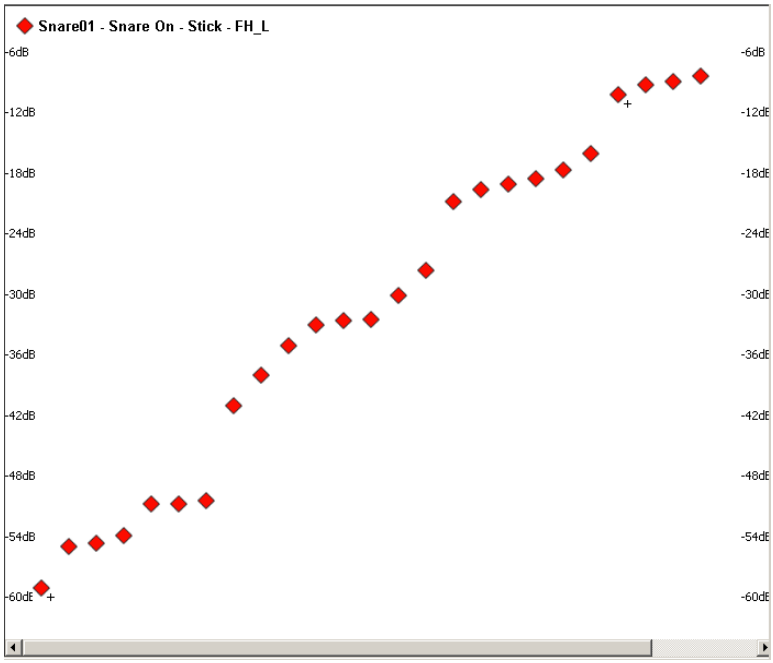
I framtiden kommer det troligtvis att finnas en funktion här där användaren kan rita kurvor i grafen som sedan slagen anpassas i volym efter.

6.7 Graf över ljuden

När ett ljudlager har laddats in i minnet visas det med en gång i graffönstret. Y-axeln, som är graderad med en decibelskala, symboliserar hur hårt slaget bedöms vara. Ljuden sorteras också från löst till hårt från vänster till höger. Användaren kan här lyssna på alla ljuden för att se om sorteringen blivit korrekt utförd. Bedöms något ljud vara felplacerat kan användaren dra det till önskad position med hjälp av musen.

Om användaren tycker att vissa ljud är väldigt likartade kan denne lägga dessa ljud i ett så kallat lager. Lagret visualiseras genom ett litet +-tecken bredvid märket för slaget. Alla ljud som ingår i lagret kan visualiseras genom ett högerklick på lagret. Vill användaren ta bort ett ljud från lagret håller denne in shift och vänsterklickar på ljudet. Detta kanske inte är den mest logiska lösningen, men den är åtminstone snabb och troligtvis lätt att lära sig.

Vill användaren ta bort ett ljud markerar denne ljudet och trycker sedan på delete eller backspace på tangentbordet. Ljud som tillhör ett lager går inte att ta bort direkt utan måste först tas bort från lagret, men även detta kommer att åtgärdas i senare versioner. Hela lager går dock bra att ta bort.



Figur 14: Graf över de olika trumslagen.

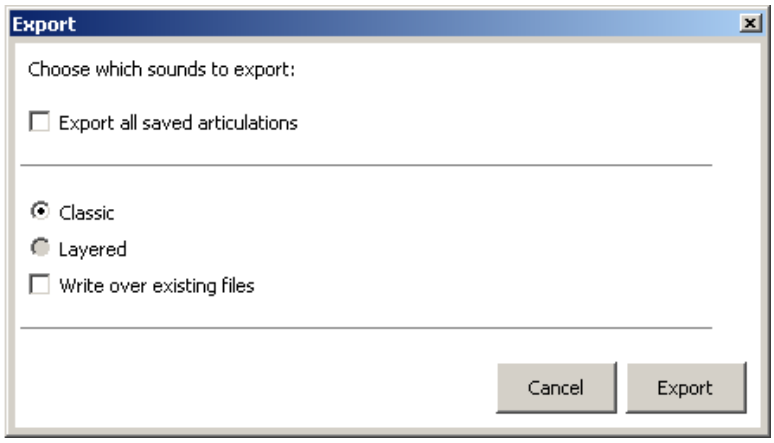
6.8 Kontrollpanel för graf

Under grafen finns en liten panel med diverse knappar och en rad som visar programmets status. Går något fel med inladdning eller uppspelning så visas det här. De knappar som ingår i panelen är knappar för att välja om man ska flytta eller spela upp ljud, en knapp för att spela upp alla ljuden i ett ljudlager sekventiellt och en undoknapp. I panelen finns också en lista på alla ljud som är borttagna. Här kan användaren lyssna på de borttagna ljudet och lägga tillbaka det till de andra ljuden om så önskas.

Figur 15: Panel för grafen.

6.9 Exportering

När alla ljudlagren är ordnade till användarens belåtenhet kan de exporteras. Alla ljud som sparats exporteras här. Detta för att slippa ladda in alla ljuden i minnet varje gång de ska exporteras. I framtiden kommer användaren att kunna välja exakt vilka av de sparade ljuden som skall exporteras, men för tillfället är det inte möjligt. Programmet skapar alla kataloger och ljudfiler som användaren specificerat. Allt visualiseras med hjälp av en mätare och processen kan när som helst avbrytas. De skapade filerna och katalogerna ligger dock kvar i filsystemet även om processen avbröts. En sak som kommer att implementeras i framtiden är att användaren själv kan specificera formatet på ljuden som skrivs till disk.



Figur 16: Exportdialogen.

Figur 17: Mätare för hur långt exporteringen kommit.

6.10 Minneshantering

Minneshanteringen i Java visade sig vara ett problem eftersom programmet kraschar när Javamaskinens heap-space blev full. För att öka heap-storleken till önskad storlek, säg kanske 512 MB, behövs en speciell option ges till den virtuella Javamaskinen när programmet skall startas. Minnet behöver också rensas upp så fort ett ljudlager ska avallokeras. Att låta Javas automatiska skräphantering göra detta skulle ta flera minuter. Därför var det nödvändigt att göra ett explicit anrop till skräphanteraren vid sådana tillfällen.

7 Avslutande kommentarer

Det har varit mycket lärorikt att designa och utveckla programmet. Den största utmaningen har helt klart varit ljudet. Det är också därför jag har valt att fokusera den här rapporten kring ljudhantering i Java. Det har också varit lärorikt att se hur olika ett program kan se ut och uppträda i olika operativsystem. Till exempel gick det åt fyra veckor att få ett program som fungerade smärtfritt på en Windowsmaskin att ens vara användbart på en Macintosh. Återigen var det ljudet som ställde till de mesta besvären. Att omvandla ljud både i samplingshastighet och bitdjup i realtid var något som jag både tyckte kändes onödigt och som försämrade prestandan på såväl uppspelningskvalitet som snabbhet.

Utöver själva programmeringen har det också varit en intressant att slutföra programmet så att andra kan använda det. Steget från fungerande program i utvecklingsmiljö till en enda fristående körbar fil har varit längre än jag trodde. Att ha andra som utvärderar och testkör ens produkt är heller inget jag varit med om tidigare. Från det att Toontrack för första gången använde programmet i december 2006 har nya funktioner och buggfixar tagit mycket av min tid i cirka två månader framöver.

Som det har visat sig efter att programmet tagits i bruk har det reducerat utvecklingstiden för en ny Toontrack-produkt väsentligt. Jobb som förut kunde ta flera månader att slutföra har nu kortats ner till runt 75% av tiden. Troligtvis kommer också slutresultatet att kanske bli bättre eftersom uppgiften att sortera alla filer kommer att bli enklare och kanske till och med roligare vilket leder till färre mänskliga misstag. Möjligheten att lättare gå tillbaks och hitta exakt vilka slag som kan tänkas ställa till problem torde göra att uppdateringar av produkter också kan göras enklare och snabbare.

Min förhoppning är också att fler studenter vid Umeå universitet kan få upp intresset för ljud i Java eftersom jag har visat att det finns stora möjligheter däri. Java är inte bara ett programmeringsspråk för enkla och snabba lösningar utan kan även användas till seriösa applikationer med stora krav på prestanda.

Källförteckning

1. EZDrummer manual, 2006, Toontrack.
Packeterad med EZDrummer från Toontrack, 2006

2. The Eclipse Foundation, SWT: The Standard Widget Toolkit, hämtad februari 2007 från
http://www.eclipse.org/swt

3. Eklund, Jakob (2005), Användarstudie av OpenAL, Examensarbete KTH avdelningen för numerisk analys och datalogi. Finns även på
http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2005/rapporter05/eklund_jakob_05177.pdf

4. OpenAL, Cross Platform 3D-Audio,
hämtad december 2006 från
http://www.*openal*.org/

5. Ross Bencina and Phil Burk , PortAudio - an Open-Source Cross-Platform Audio API, hämtad december 2006 från
http://www.portaudio.com/

6. Lantinga, Sam, Peter, Stephane och Gordon, Ryan, SDL mixer 1.2
hämtad december 2006 från
http://www.libsdl.org/projects/SDL_mixer/

7. The MathWorks, Inc. (2006) Software License Agreement,
hämtad februari 2007 från
http://www.mathworks.com/company/aboutus/policies_statements/agreement.pdf

8. Lewis, J.P. och Neumann, Ulrich (2004), Performance of Java versus C++, Computer Graphics and Immersive Technology Lab, University of Southern California,
hämtad december 2006 från
http://www.idiom.com/~zilla/Computer/javaCbenchmark.html

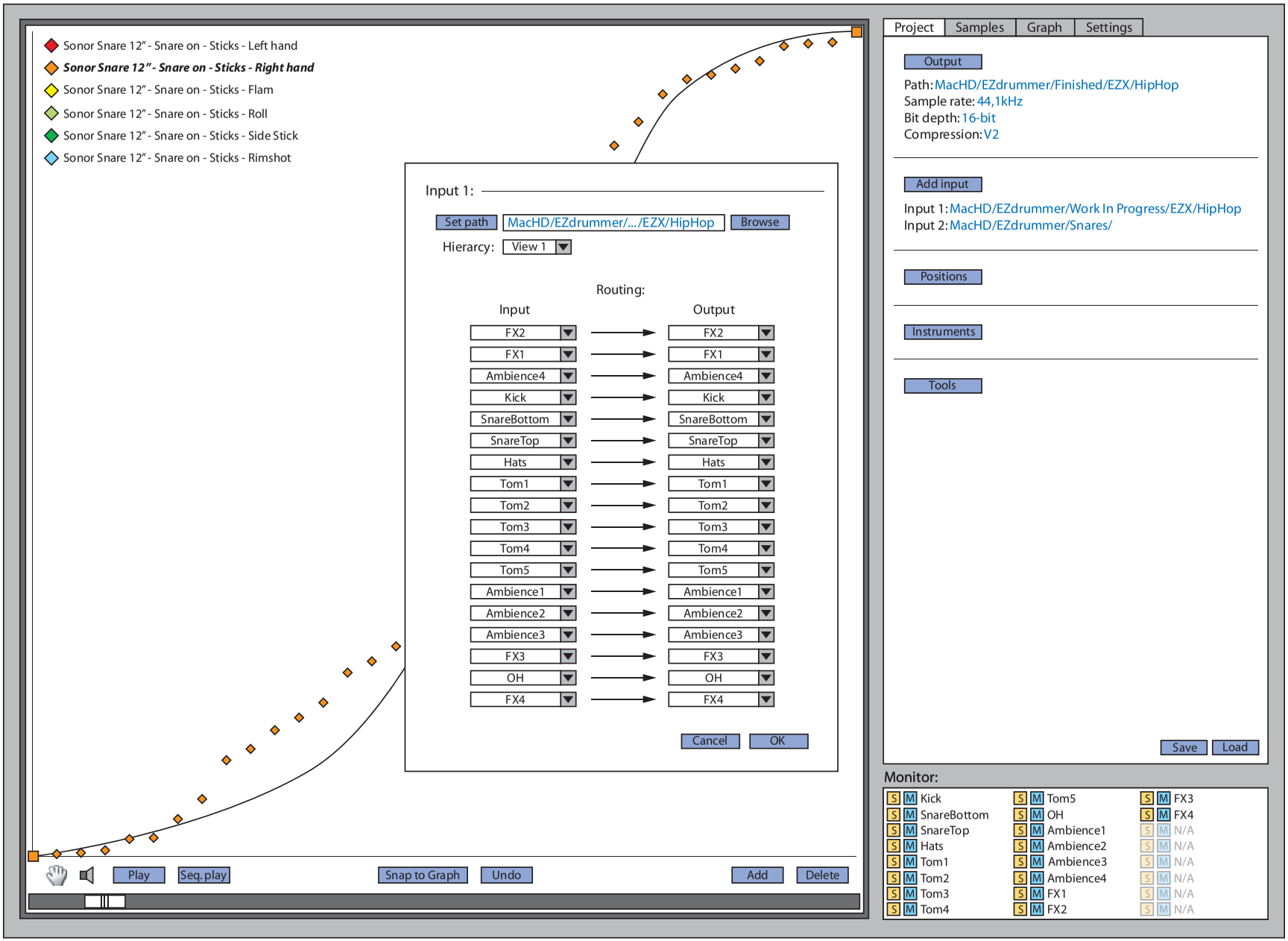
9. Sun Microsystems (2002), Java Sound Programmer Guide.
Hämtad oktober 2006 från
http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/contents.html

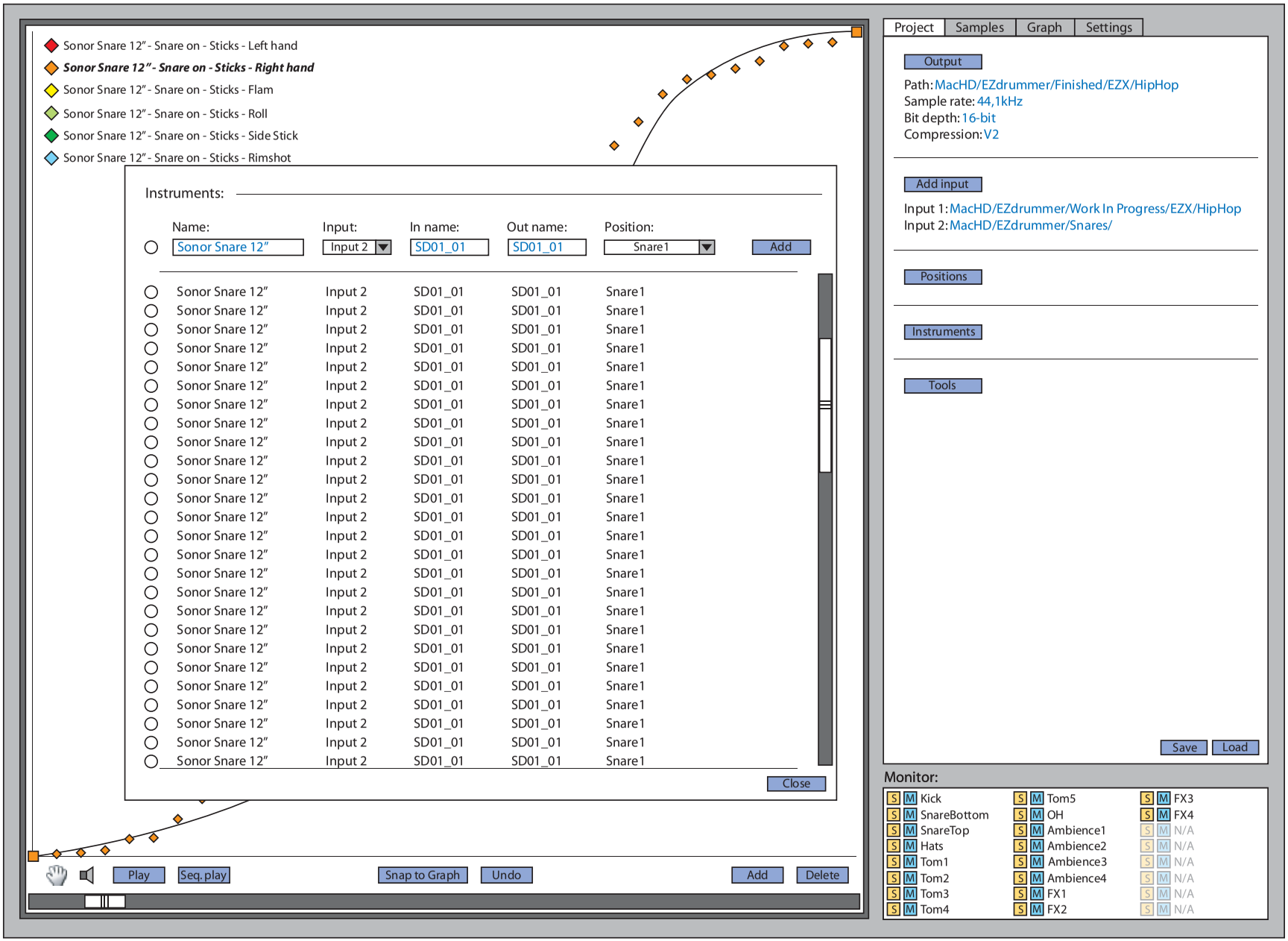
10. The Mathworks Inc.
Hämtad mars 2007 från
http://www.mathworks.com/

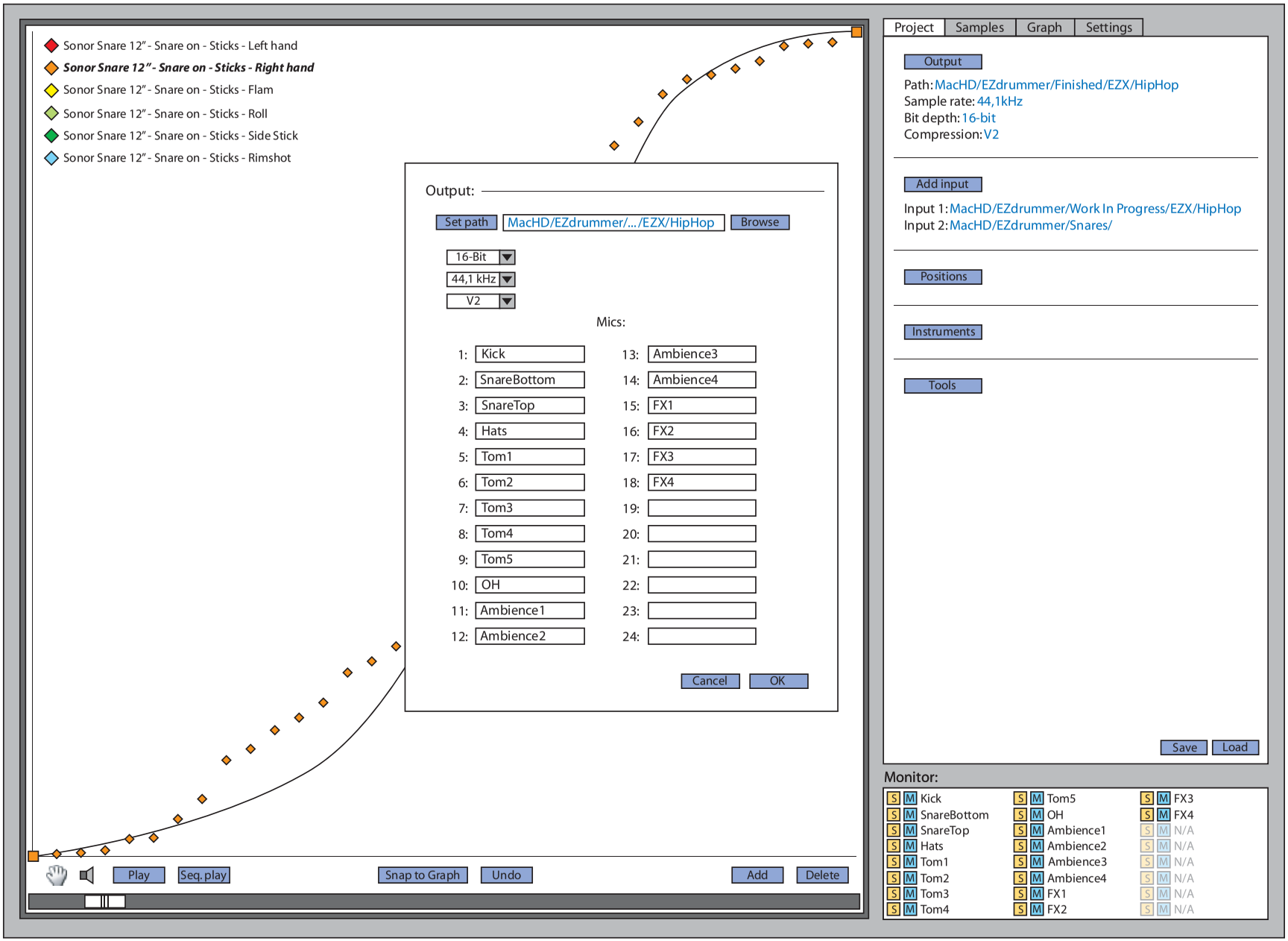
11. Bomers, Florian och Pfisterer, Matthias, Java Sound Resources
Hämtad under hela hösten 2006 från
www.jsresources.org

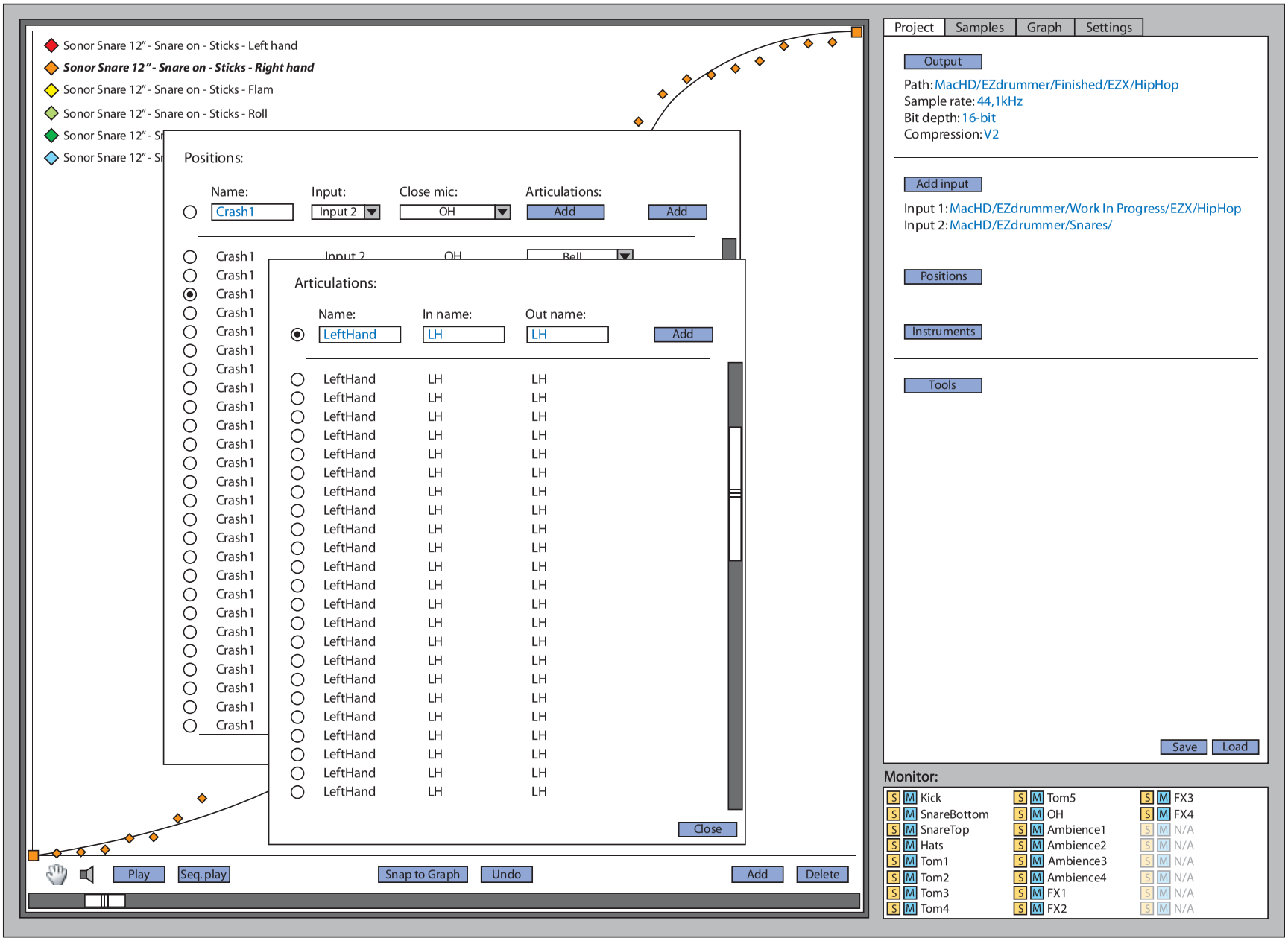
12. Simple Direct Media Layer.
Hämtat december 2006 från
www.libsdl.org

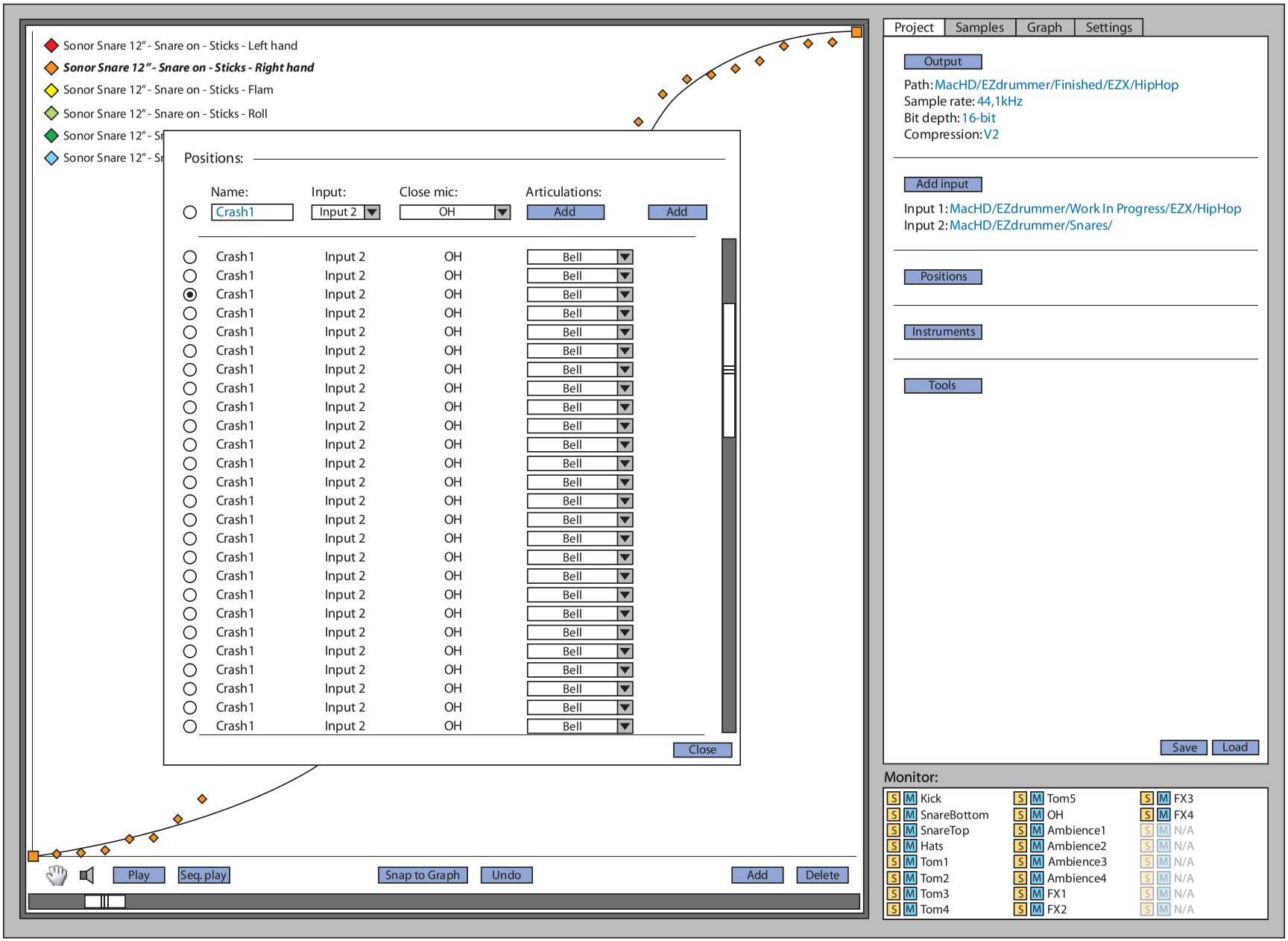
Bilaga A: Designförslag från Toontrack

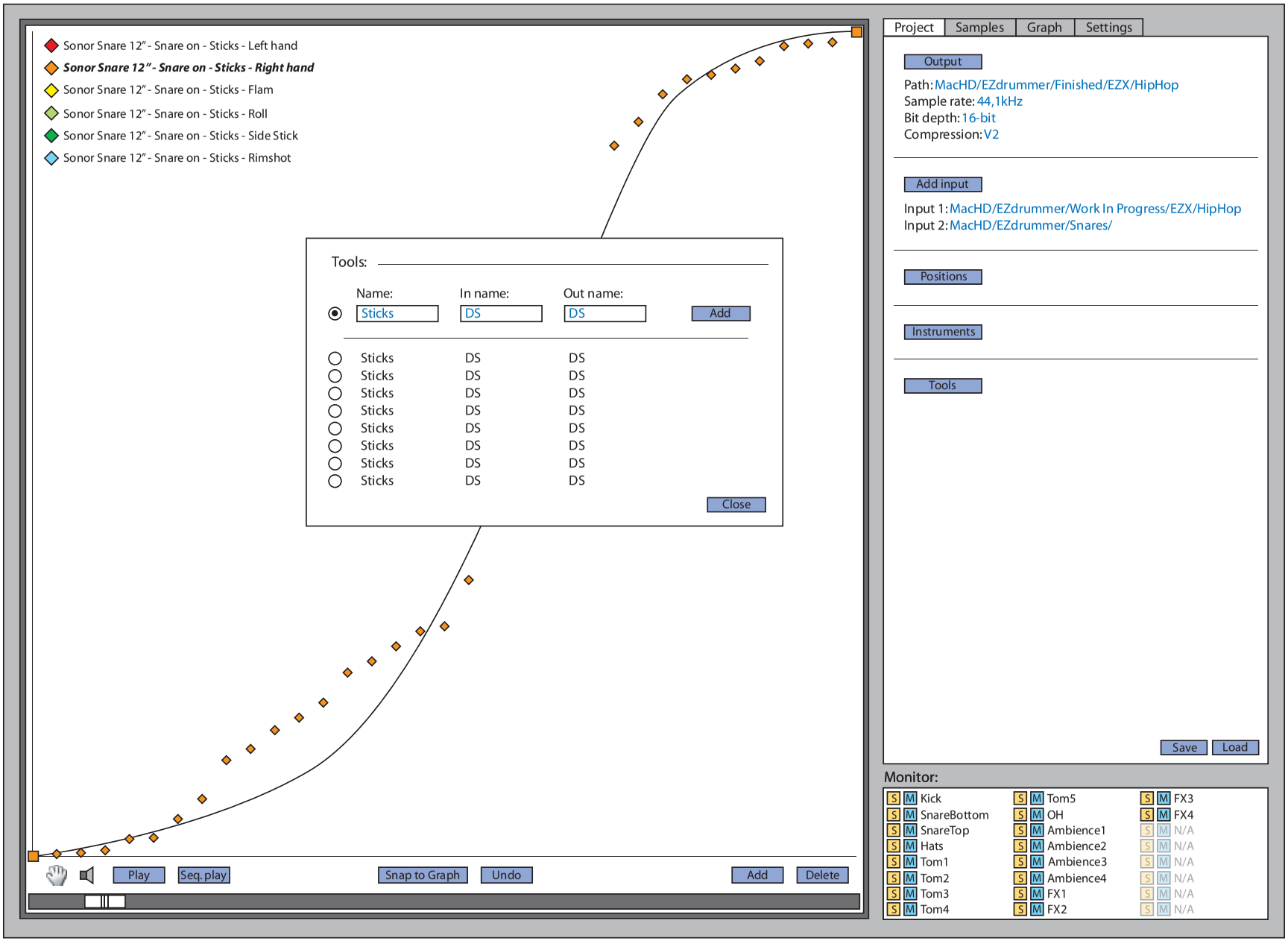

Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Input 1:
Set path
MacHD/EZdrummer/.../EZX/HipHop
Browse
Hierarcy:
View 1
Routing:
Input
Output
FX2
FX1
Ambience4
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
Ambience1
Ambience2
Ambience3
FX3
OH
FX4
Cancel
OK
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
S M Kick
S M SnareBottom
S M SnareTop
S M Hats
S M Tom1
S M Tom2
S M Tom3
S M Tom4
S M Tom5
S M OH
S M Ambience1
S M Ambience2
S M Ambience3
S M Ambience4
S M FX1
S M FX2
S M FX3
S M FX4
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A

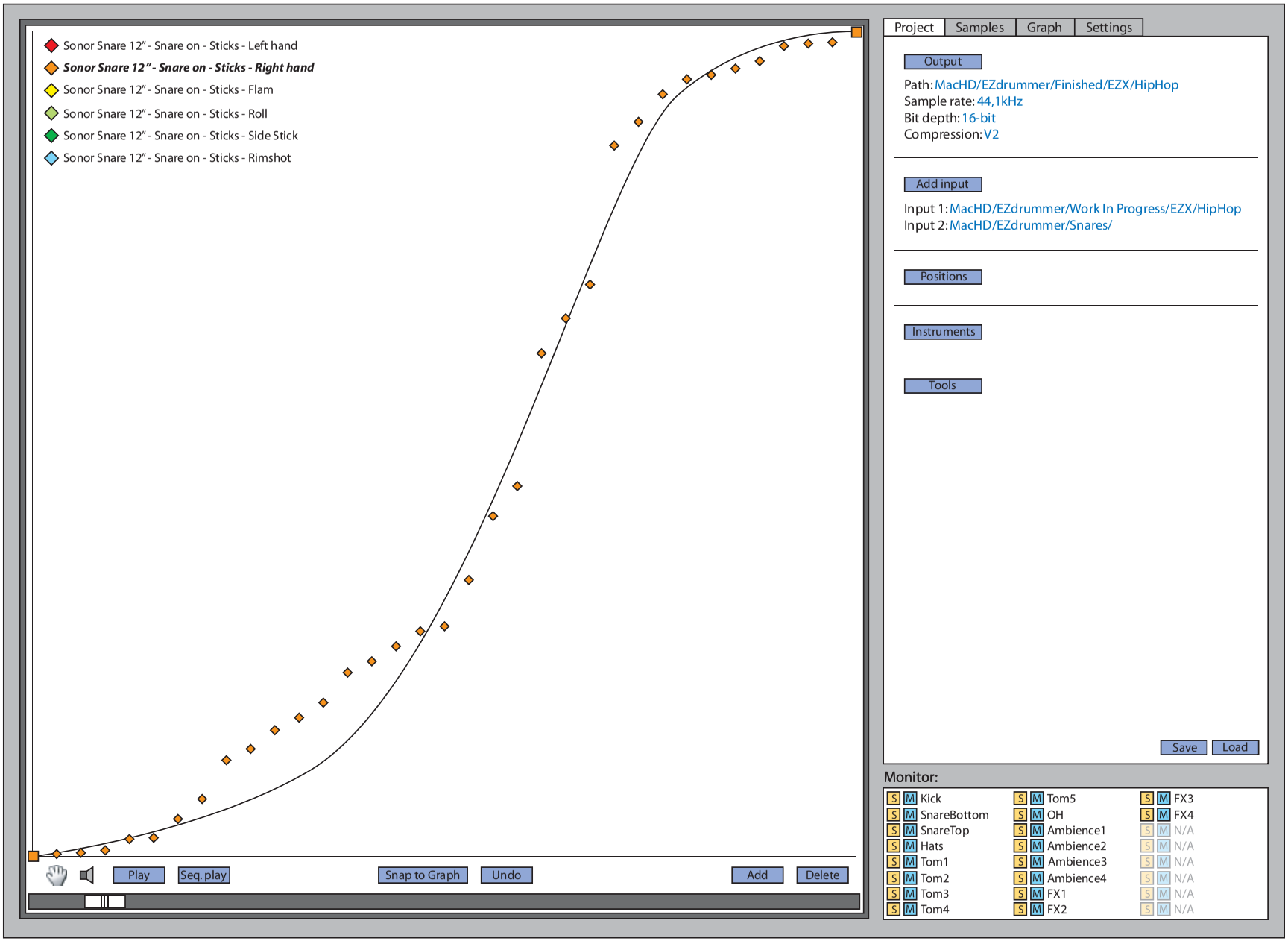

Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Instruments:
Name:
Input:
In name:
Out name:
Position:
Sonor Snare 12"
Input 2
SD01_01
SD01_01
Snare1
Add
Sonor Snare 12" Input 2 SD01_01 SD01_01 Snare1
Close
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
S M Kick
S M SnareBottom
S M SnareTop
S M Hats
S M Tom1
S M Tom2
S M Tom3
S M Tom4
S M Tom5
S M OH
S M Ambience1
S M Ambience2
S M Ambience3
S M Ambience4
S M FX1
S M FX2
S M FX3
S M FX4
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A

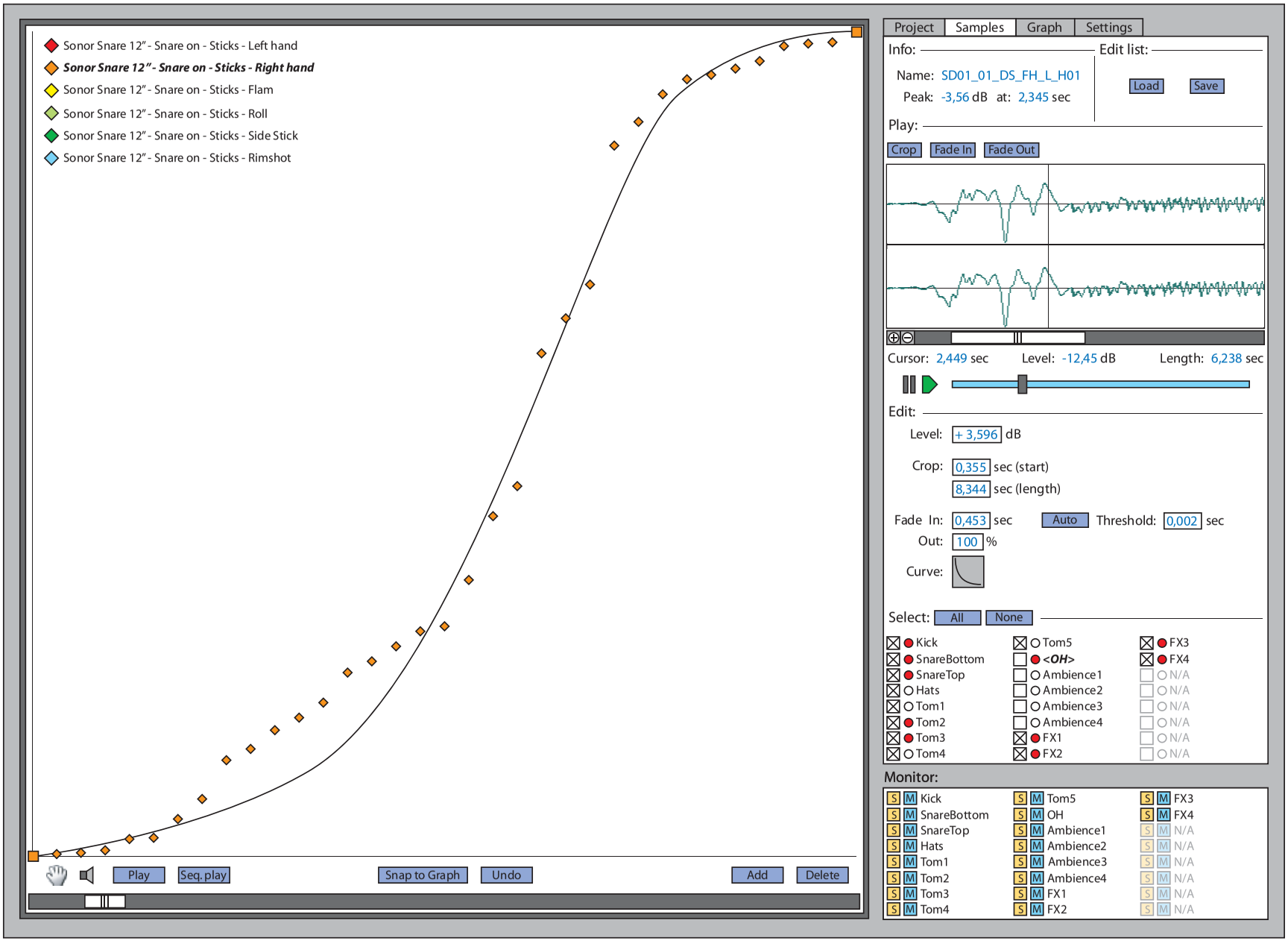

Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Output:
Set path
MacHD/EZdrummer/.../EZX/HipHop
Browse
16-Bit
44,1 kHz
V2
Mics:
1: Kick
2: SnareBottom
3: SnareTop
4: Hats
5: Tom1
6: Tom2
7: Tom3
8: Tom4
9: Tom5
10: OH
11: Ambience1
12: Ambience2
13: Ambience3
14: Ambience4
15: FX1
16: FX2
17: FX3
18: FX4
19:
20:
21:
22:
23:
24:
Cancel
OK
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
S M Kick
S M SnareBottom
S M SnareTop
S M Hats
S M Tom1
S M Tom2
S M Tom3
S M Tom4
S M Tom5
S M OH
S M Ambience1
S M Ambience2
S M Ambience3
S M Ambience4
S M FX1
S M FX2
S M FX3
S M FX4
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A

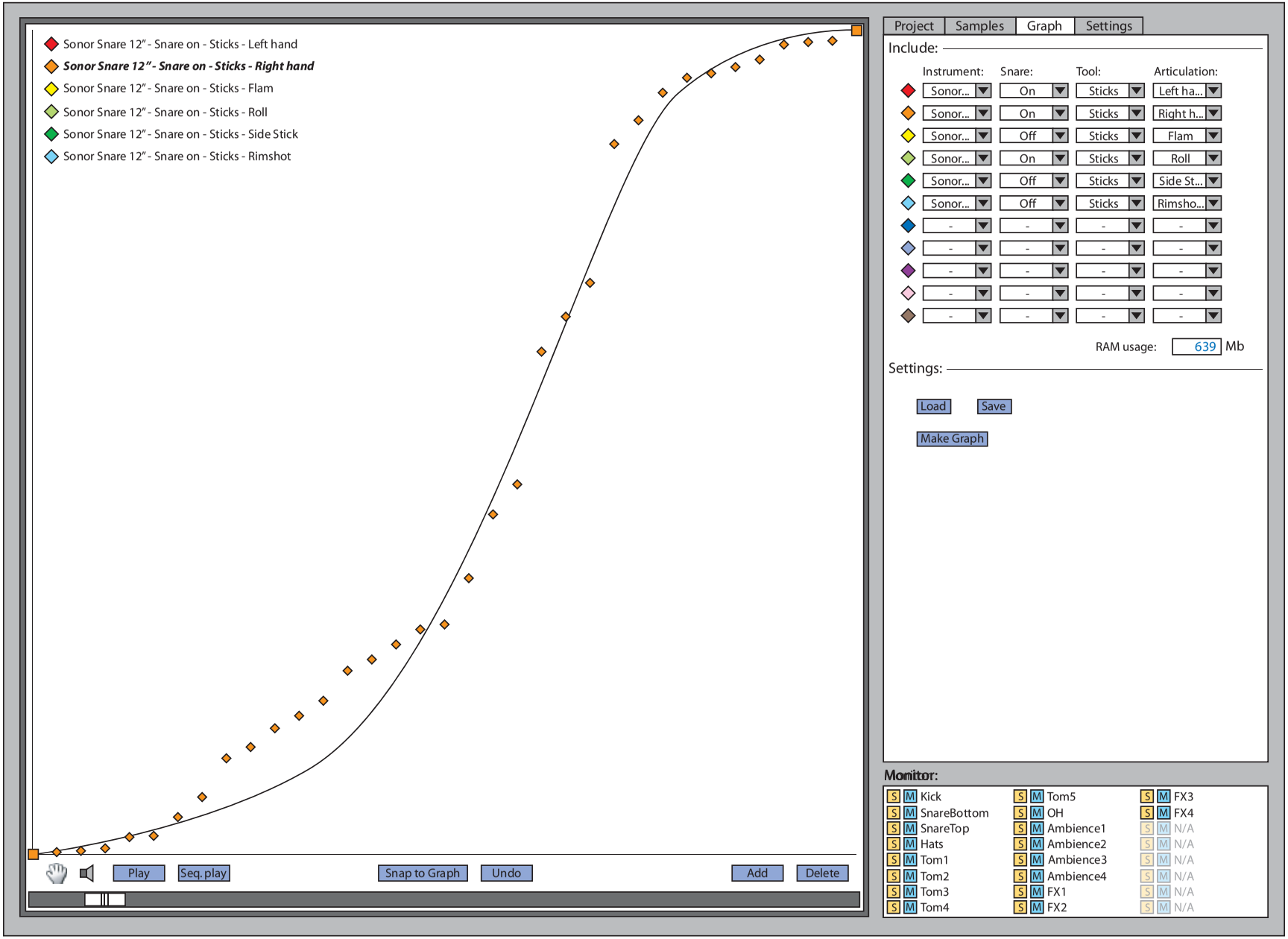

Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Positions:
Name:
Input:
Close mic:
Articulations:
Crash1
Input 2
OH
Add
Add
Articulations:
Name:
In name:
Out name:
LeftHand
LH
LH
Add
Close
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
OH
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A


Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Positions:
Name:
Input:
Close mic:
Articulations:
Crash1
Input 2
OH
Add
Add
Bell
Close
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
OH
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A


Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Tools:
Name:
In name:
Out name:
Sticks
DS
DS
Add
Sticks DS DS
Close
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
OH
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A


Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Output
Path: MacHD/EZdrummer/Finished/EZX/HipHop
Sample rate: 44,1kHz
Bit depth: 16-bit
Compression: V2
Add input
Input 1: MacHD/EZdrummer/Work In Progress/EZX/HipHop
Input 2: MacHD/EZdrummer/Snares/
Positions
Instruments
Tools
Save
Load
Monitor:
S M Kick
S M SnareBottom
S M SnareTop
S M Hats
S M Tom1
S M Tom2
S M Tom3
S M Tom4
S M Tom5
S M OH
S M Ambience1
S M Ambience2
S M Ambience3
S M Ambience4
S M FX1
S M FX2
S M FX3
S M FX4
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A
S M N/A


Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Info:
Name: SD01_01_DS_FH_L_H01
Peak: -3,56 dB at: 2,345 sec
Edit list:
Load
Save
Play:
Crop
Fade In
Fade Out
Cursor: 2,449 sec
Level: -12,45 dB
Length: 6,238 sec
Edit:
Level: + 3,596 dB
Crop: 0,355 sec (start)
8,344 sec (length)
Fade In: 0,453 sec
Auto
Threshold: 0,002 sec
Out: 100 %
Curve:
Select:
All
None
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
<OH>
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A
Monitor:
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
OH
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A


Sonor Snare 12" - Snare on - Sticks - Left hand
Sonor Snare 12" - Snare on - Sticks - Right hand
Sonor Snare 12" - Snare on - Sticks - Flam
Sonor Snare 12" - Snare on - Sticks - Roll
Sonor Snare 12" - Snare on - Sticks - Side Stick
Sonor Snare 12" - Snare on - Sticks - Rimshot
Play
Seq. play
Snap to Graph
Undo
Add
Delete
Project
Samples
Graph
Settings
Include:
Instrument:
Snare:
Tool:
Articulation:
Sonor...
On
Sticks
Left ha...
Sonor...
On
Sticks
Right h...
Sonor...
Off
Sticks
Flam
Sonor...
On
Sticks
Roll
Sonor...
Off
Sticks
Side St...
Sonor...
Off
Sticks
Rimsho...
RAM usage:
639
Mb
Settings:
Load
Save
Make Graph
Monitor:
Kick
SnareBottom
SnareTop
Hats
Tom1
Tom2
Tom3
Tom4
Tom5
OH
Ambience1
Ambience2
Ambience3
Ambience4
FX1
FX2
FX3
FX4
N/A