Spektrum3D

ETT PROJEKT I TNM077

Peter Asplund, petas874@student.liu.se
Anders Fjeldstad, andfj645@student.liu.se
Jonas Nilsson, jonni957@student.liu.se

Projektbeskrivning

Projektet har gått ut på att utveckla en Javaprogramvara som spelar upp ljud samtidigt som signalens frekvensinnehåll visualiseras i 3D i realtid. Anledningen till att vi valde att formulera målet på det sättet är att alla i gruppen ville programmera snarare än modellera samt att vi är mer eller mindre intresserade av ljud och musik. Med introduktionskurser till signalbehandling och transformteori i färskt minne kändes det naturligt att tillämpa kunskaperna i något praktiskt.

Hur visualiserar man frekvenser i 3D? Vår idé bygger på att vi tänker oss en frekvensaxel ortogonal mot en tidsaxel, med tredje axeln motsvarande amplitud. Utifrån detta byggde vi två varianter (se nedan) - en matris av staplar med variabel höjd och en dynamisk yta.

En stor del av arbetstiden har gått åt till att utveckla de fundamentala delarna av programmet som man inte är omedelbart medveten om när man ser resultatet. Till detta hör inläsning och tolkning av den samplade ljudsignalen, Fourieranalys, ljuduppspelning och konstruktion av scengraf med mera.

Programmets beståndsdelar

Det färdiga programmet består av flera klasser med olika funktioner. Klasserna är individuellt utbytbara vilket har gjort det lätt för oss att utveckla olika varianter av visualiseringen. I figur 1 listas de klasser som tillsammans bygger upp applikationen.

Grafiskt gränssnitt (GUI)

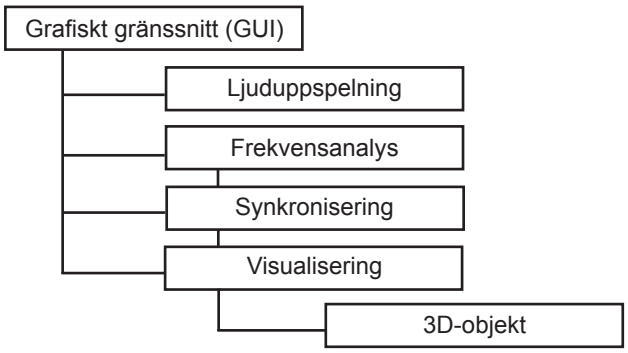
GUI:t utgör huvudklassen som knyter samman de andra klassernas funktionalitet. Här definieras även det grafiska utseendet på programmet.

Ljuduppspelning

Ljuduppspelningsklassen läser från en angiven ljudfil och spelar upp innehållet via ljudkortet. Den innehåller metoder för att starta och stoppa uppspelningen.

Frekvensanalys

Denna klass läser amplitudsampel från en given ljudfil, konverterar dem från bytes till flyttal, delar upp dem i lagom långa intervall och kör Fouriertransform på dem för att få ut frekvenserna. Längden på intervallen anpassas efter det önskade antalet bilder per sekund. Vi har inte skrivit själva Fouriertransformmetoden utan använder en färdig med tillstånd från upphovsmannen.



Figur 1. Programmets huvudklasser

Visualisering

Denna klass definierar det virutella universumet genom Java3D och bygger scengrafen. Dessutom har den en metod för att ta emot frekvensvärden och överföra denna information till operationer på de grafiska 3D-objekten som syns på skärmen. Eftersom vi har två olika varianter på visualiseringen har vi också två helt skilda versioner av denna klass.

Synkronisering

För att synkronisera analys och visualisering använder denna klass en timer som beroende på det önskade antalet bilder per sekund utför de erfordrade metodanropen. Denna lösning klarar att kompensera för tillfälliga överbelastningar som annars skulle kunnat resultera i oönskade fördröjningar.

3D-objekt

Dessa klasser är en förutsättning för att visualiseringen ska fungera. För stapelversionen är objektet en box (som vi gjort själva eftersom det gav bättre prestanda än att använda en färdig) och innehåller endast geometri och utseende. När vi implementerade 3D-ytan fick objektet förutom mer avancerade metoder för generering av geometri även en metod för uppdatera sina hörnpunkter och dess färger.

►

Problem och observationer

När vi skulle synkronisera analys, visualisering och ljud första gången och testade resultatet upplevde vi att ljudet inte låg i fas med det övriga. Samtidigt visste vi att de olika momenten startade och slutade exakt samtidigt utan någon dataförlust däremellan. Det visade sig att om vi började visa frekvensinnehållet för den aktuella 1/25-sekunden samtidigt som den började spelas upp så upplevdes det som att ljudet låg efter. Detta tror vi beror på hur människan uppfattar visuella och auditiva intryck, tillsammans med att ljudet faktiskt är långsammare än ljuset. Vi löste problemet genom att låta visualiseringen vänta 1/25 sekund innan den börjar, så att bilden egentligen alltid ligger ett intervall efter ljudet. Detta uppfattas som perfekt synkronisering.

Värt att notera är att stapelversionen av visualiseringen blir påtagligt långsam redan vid storleksordningen 400 staplar. Detta kan bero på att scengrafen innehåller många dynamiska transformgrupper (lika många som antalet staplar) som 25 gånger per sekund ska få en ny transformmatris. I lösningen med ytan är detta inte aktuellt eftersom geometrin modifieras direkt, vilket medför att vi kan köra maximal ytupplösning (lika många punkter som frekvensvärden) i full fart utan problem.

Ett kvarstående problem är att vårt program har buggar kopplade till ljuduppspelningen som vi valt att inte fördjupa oss i eftersom att projektets tyngdpunkt ligger i grafikdelen. Till exempel har vi valt att låta analysen och ljuduppspelningen läsa från samma ljudfil oberoende av varandra, eftersom de använder sig av olika buffertstorlekar. Teoretiskt är detta inte optimalt, men eftersom vi inte har några prestandaproblem vid körning på en normal persondator så har vi inte lagt ned mer energi på det.

Ett potentiellt problem är att ljuduppspelningen inte kan hantera tillfälliga överbelastningar, till skillnad från analys och visualisering. Detta innebär att om ljudet skulle hacka till så kommer det vara asynkront i förhållande till det visuella från och med den tidpunkten. Detta är något man skulle kunna lösa genom en sammanslagning av ljuduppspelning och analys, till exempel. Det skulle indirekt innebära att de på något sätt skulle använda en gemensam buffert vid läsning från ljudfilen.

Vidareutveckling

Under projektets inledande planeringsfas skrev vi abstrakta klasser för programmets huvuddelar. Detta innebär att vi i princip har "ritningar" som anger ramarna för alla tänkbara implementationer av de individuella klasserna - till exempel skulle man kunna göra en 2D-version av visualiseringen (vilket vi gjorde vid testningen av programmet) eller utöka ljuduppspelningen så att den klarar MP3-formatet och så vidare. Det är mycket enkelt att byta ut någon av programmets moduler, vilket vi haft i åtanke under utvecklandet.