LjudVis – 3D Datorgrafik

Juni 1, 2017

1 Inledning

Elias Elmquist	eliel503
Emil Gustafsson	emigu039
Fredrik Jäderland	freja328
Samuel Rising	samri821
Simon Forsberg	simfo596

Vår grupp har haft som mål att jobba med ”low-level programing” där slutresultatet skall vara en robust visualiseringsmotor med stor inverkan av ljud. Visualiseringsmotorn skall kunna skapa program som ljudvisualiserare där objekt påverkas av ljudet.

2 Arbetsprocess

Vi valde att jobba i C++ med IDE:n Microsoft Visual Studios vilket var helt nytt för vissa utav oss. Tidigt i projektet kom vi fram till att vi ville att slutresultatet skulle vara i realtid. D.v.s. att visualiseringsmotorn måste vara väldigt effektiv. En grundfunktion som behövdes var att kunna läsa av ljud från en inspelningsenhet i realtid. Några variabler vi ville manipulera med frekvensdata var färger, kameraörelser och modellernas position.

Själva arbetsprocessen i sig var väldigt flexibel, ett möte i veckan arrangerades för att stämma av och ange nya uppgifter som skulle utföras. Den plan vi hade fallerade snart in i projektet, då det dök upp fler problem samt att en del saker var svårare att tackla än förväntat. Eftersom vi jobbade med ”low-level programing” kände vi att det var bäst att jobba i grupper om två. Vi valde även att använda Visual Studios integrerade versionshanterare Team service.

Eftersom det är ”low-level programming” vi arbetade med kände vi att varje moment skulle vara ordentligt utfört innan vi gick vidare med nästa, vilket i sin tur ledde till att vi hela tiden stötte på problem som var svåra att uppskatta tidsåtgången. Själva strulet var aldrig småsaker utan mer vilket som var det bästa konceptet samt hur man implementerar det på det mest effektiva sättet. Detta ledde till att det var svårt att uppskatta tidsåtgången för varje delmoment.

3 Dokumentation

C++
http://www.cplusplus.com/
OpenGl
https://www.khronos.org/registry/OpenGL-Refpages/gl4/
GLFW
http://www.glfw.org/documentation.html
GLM
http://glm.g-truc.net/0.9.8/glm-0.9.8.pdf
PowerPoint om optimering
https://www.dropbox.com/sh/l8ubvayw70y39dn/AABJOwBnfV2v-kKD_QOCzkFma?dl=0&preview=parallel+game+engine+design.pdf
BASS Audio Library
https://www.un4seen.com/doc/

De dokumentationer som användes fyllde sitt syfte väl. Ibland räcker dock inte dessa råa källor och ett forum kan vara mycket mer hjälpsamt, till exempel Stack Overflow som har svar på alla tänkbara frågor om både c++ och OpenGl.

4 3D-motorn

Motorn använder OpenGl för rendering med wrapper biblioteket GLFW som underlättar användningen av OpenGl, skapandet av fönster och hantering av inmatningsenheter.

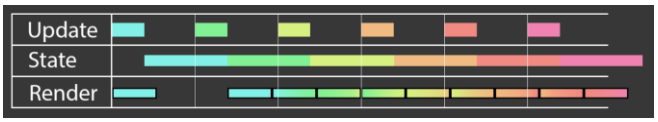
4.1 Indirect draw calls

Ett ”draw call” är ett meddelande som skickas via OpenGl till GPU:n. Det finns många olika sätt att konstruera dessa meddelanden, den primära typen som vi använder oss av i detta projekt är så kallade “Indirect draw calls”. Denna typ använder sig av en buffer, GL_DRAW_INDIRECT_BUFFER, som innehåller information om modellens vertexar och hur många instanser som ska renderas av den. När modellerna ska renderas så används en eller flera av dessa buffrade kommandon via en enda rad kod, “glMultiDrawElementsIndirect” eller “glMultiDrawArraysIndirect”. Detta resulterar i extremt snabb kod då nästan alla uträkningar försvinner från CPU:n och istället görs på GPU:n.

4.2 Multithreading

När fysikaliska beräkningar ska utföras i kontinuerlig ordning under en lång tid så är det bra om två olika maskiner som kör samma kod får samma resultat av samma inputs. Ett viktigt element för att uppnå detta är ett statiskt uppdateringsintervall. Det uppstår dock ett problem om renderaren körs i samma tråd som uppdateringen av objekt, renderaren är låst till den uppdateringsintervall som används.

För att lösa detta problem måste renderaren och uppdateringen separeras i två olika trådar där renderaren interpolerar mellan de två senaste uppdateringarna, detta skapar dock ytterligare problem. Nu måste de två trådarnas data synkroniseras, renderaren måste vid varje loop se till att den är uppdaterad med de senaste hårda förändringarna (add och delete) och sedan interpolera mellan de två senaste buffrade uppdateringarna. Under uppdateringen finns ett annat problem, renderaren måste ha hunnit hämta informationen från den äldsta buffern innan det skrivs över. Detta betyder att uppdateringen kan behöva pausas för att renderaren inte ska hamna ur synk med uppdateringarna.



Uppdateringstråden och renderings tråden pratar via en StateBuffer och “std::atomic” vilket är det snabbaste sättet att kommunicera mellan trådar.

5 BassWrapper

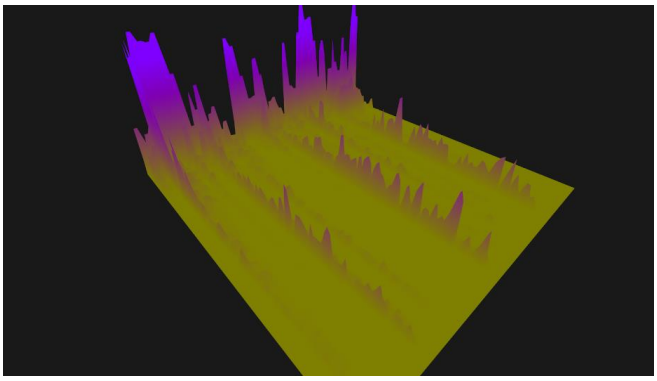
För att extrahera frekvensdata ur ljud så användes ett externt ljudbibliotek kallad BASS. Med hjälp av BASS så skapade vi en egen klass vid namn BassWrapper som hanterar och manipulerar all ljuddata. BASS innehåller flera användbara funktioner som vi använde oss av, exempelvis BASS_ChannelGetData och BASS_ChannelGetLevel. Dessa funktioner extraherar frekvensdata från datorns standard-inspelningsenhet genom FFT (Fast Fourier Transform). Inspelningsenheten kan vara en intern eller extern mikrofon, men det kan också vara det ljud som datorn skickar ut i högtalarna vilket ges tillgängligt i inspelningsenheten “Stereo mix”. Vi skapade egna funktioner med hjälp av BASS som vi samlade i en egen klass kallad BassWrapper.

6 Resultat

Nedan visas och förklaras de visualiserare som gruppen har skapat.

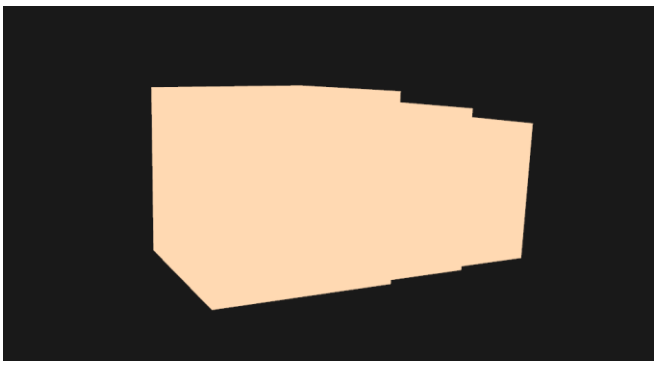
6.1 Frekvensvåg

Denna shader använder sig av den data som BassWrapper genererar och placerar varje frekvens intensitet på ett rutnät i form av en mesh. Intensiteten driver både höjden på varje vertex och färgen. Effekten som detta försöker uppnå är en våg av frekvensförändringar som animeras i realtid.



6.2 Mask

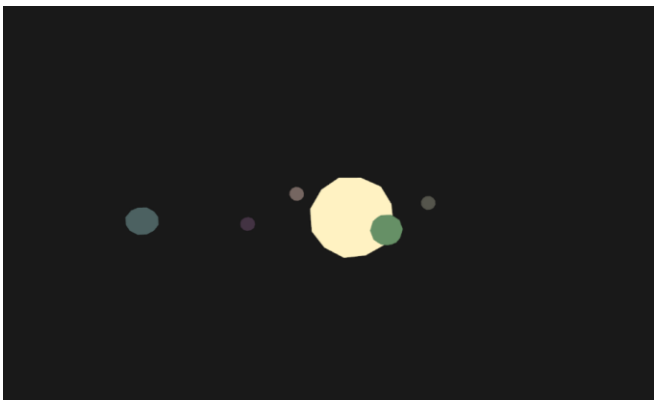
Masken består av tre kuber som rör sig fram och tillbaks med hjälp av en sinusfunktion. Maskens hastighet påverkas av intensiteten hos lägre frekvenser, masken rör sig fortare ju högre intensitet det är. Dess färg beror på medelvärdet av alla frekvensers intensitet, den är beige vid noll ljud och blir rödare ju mer ljud det är.



6.3 Solsystem

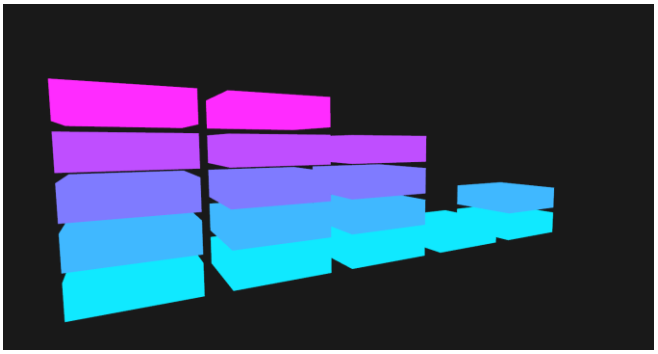
Solsystemet består av fem planeter som roterar med en kontinuerlig fart runt solen i centrum. Planeterna ökar i storlek och ändrar färg då intensiteten blir högre i vissa frekvenser. Ju längre bort planeterna är från solen desto högre frekvenser påverkas de av. Solen å andra sidan påverkas av medelvärdet av intensiteten i alla frekvenser

precis som masken. Solen ändrar inte storlek utan ändrar bara färg från att vara vit till att bli gulare ju mer ljud det är.



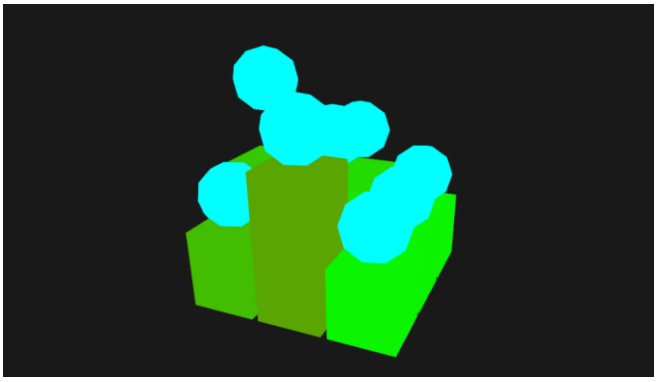
6.4 Staplar

De fem staplarna består av fem kvadratiska block vardera. Staplarnas höjd beror på intensiteten i utvalda frekvenser. Stapeln till vänster beror på de lägsta frekvenserna och ju längre bort åt höger staplarna är desto högre frekvenser beror de på.



6.5 Rutnät med flygande bollar

Detta är ett ett 3x3 rutnät med bollar som studsar. Gridsystemet är uppbyggt utav staplar som skalas om beroende på vårt frekvensspektrum. Bollarna flyger med stapeln upp men har en konstant fallande translation.



7 Slutsats

Vi kom fram till att det var svårt att uppskatta tiden för varje delmoment, mycket tid gick åt till att förstå konceptet och implementera det på ett så effektivt sätt som möjligt. Själva 3D-motorn var det som tog längst tid att implementera. Resultatet valde vi att ha lite mer fritt för att se vilka möjligheter som fanns samt att det blev tidsbrist för att göra en riktigt välarbetat 3D-scen med objekt som reagerade på ljud. Vilket förmodligen är det vi vill förbättra om vi hade haft mer tid.