KTH Datavetenskap
och kommunikation

Spektrala Transformer
Programmeringsuppgift Matlab

Inledning

I DT1130 ingår en obligatorisk programmeringsuppgift med syfte att ge erfarenhet i att självständigt lösa problem med hjälp av Matlab. Uppgiften är mindre styrd än övriga laborationer i kursen. Du kan fritt välja en av uppgifterna nedan. Om du har ett eget förslag kan det också gå bra, men kolla med kursansvarig först.

Uppgiften redovisas i gruppform, där ett par redovisar sitt med en 10-minuters presentation med stöd av bilder och en demonstration av programmet, följt av en diskussion. Innan redovisningen ska matlabkod skickas in till kursansvarig.

1 Ett belyst klot

Inom 3D-grafik simuleras belysta ytor med en serie beräkningar, där hänsyn tas till ytans orientering i förhållande till ljuskällorna, ytans färg och materialegenskaper, liksom ljuskällornas färg och intensitet.

Grunden ges av *Lamberts cosinus-lag* som säger att intensiteten av det ljus som reflekteras från en *matt* yta (den *diffusa reflektionen*) är proportionell mot cosinus för ljusets infallsvinkel (mätt i förållande till ytans normalvektor). Om man utnyttjar att *skalärprodukten mellan två vektorer av längden ett är lika med cosinus för vinkeln mellan dem* så kan man skriva sambandet som

$$I_D = \max\{\mathbf{e_l} \cdot \mathbf{e_n}, 0\}$$

där I_D är den intensiteten i en punkt $\mathbf{p}$ på ytan, $\mathbf{e_l}$ är en enhetsvektor från $\mathbf{p}$ mot ljuskällan och $\mathbf{e_n}$ är ytans normalvektor i punkten. Denna faktor multipliceras sedan med ytans färg och ljuskällans färg. För blanka ytor får man dessutom en belysningseffekt som är beroende av var betraktarens position (blänket i en motorhuv ändras när man flyttar huvudet). Denna belysningsterm, som även kallas den spekulära reflektionen, är som störst när ljusets infallsvinkel och betraktningsvinkeln är lika stora, men motsatta, och avtar olika fort beroende på hur blank ytan är. Denna term kan skrivas som

$$I_S = (\max\{\mathbf{e_s} \cdot \mathbf{e_n}, 0\})^\alpha$$

där exponenten α bestämmer hur blank ytan ska vara, och vektorn $\mathbf{e_s}$ är en enhetsvektor halvvägs mellan $\mathbf{e_l}$ (den vektor som pekar mot ljuskällan) och $\mathbf{e_v}$ - en enhetsvektor som pekar mot betraktaren. Den kan beräknas med

$$\mathbf{e_s} = \frac{\mathbf{e_l} + \mathbf{e_v}}{|\ \mathbf{e_l} + \mathbf{e_v}\ |}$$

Den spekulära termen I_S ska multipliceras med "blänkets" färg (ofta nära vitt) och med ljuskällans färg. Den slutgiltiga färg en pixel får ges av summan av de diffusa och spekulära bidragen. Finns flera ljuskällor så summeras bidragen från dessa i varje pixel.

Skriv ett program som ritar ett belyst klot enligt ovanstående modell! Färg på klot och "blänk" ska kunna varieras, liksom ljuskällans position och färg. Betraktaren och ljuskällan kan antas vara på stort avstånd från klotet i förhållande till klotets radie, så strålarna kan betraktas som parallella. Gör en liten animering där ljuskällan rör sig - använd `immovie()` för att skapa en film av matrisbilder samt `implay()` för att spela upp filmen! (den går även att spara som videofil).

2 Spektrogram

Denna uppgift går ut på att skriva ett program som ritar ett spektrogram av en godtycklig ljudfil.

Ett Spektrogram är en representation av frekvensinnehållet i en signal över tid. Ofta ritas det som en bild där varje kolumn med pixlar motsvarar frekvensspektrum vid en viss tidpunkt. Detta spektrum beräknas med hjälp av DFT (FFT) på N sampelvärden $x(n)$ kring tidpunkten. Pixelintensiteten är då *proportionell mot logaritmen av FFT'ns belopp i kvadrat*, dvs $|\log(X_k^2)|$. Vidare så multipliceras ofta de N samplen i insignalen med en *fönsterfunktion* $w(n)$, för att undertrycka sidolober i spektrum. Exempel på en vanlig fönsterfunktion är ett s.k. *Hamming-fönster* som ges av ekvationen

$$w(n) = 0.53836 - 0.46164 \cos \frac{2\pi n}{N-1}$$

För varje ny kolumn flyttar man fönstret ett fixt antal sampel M, där M typiskt är ett mindre tal än N.

Ett alternativ till att rita spektrogrammet i form av en bild är att rita det i 3D, som en serie 2D-kurvor eller som en tredimensionell yta (se t.ex. matlabs `surf`). Experimentera med olika stilar för uppritning, olika fönsterlängder N och tidssteg M. Jämför gärna med spektrogram gjorda t.ex. med *WaveSurfer*.

3 Halvtoning

När en bild ska tryckas i tryckpress måste den omvandlas för att bara innehålla de färger som är tillgängliga i tryckprocessen. I en gråskalebild måste alla gråtoner representeras med bara svart och vitt. *Halvtoning* innebär att bilden delas upp i lika stora rutor. I varje sådan ruta mäter man den genomsnittliga gråskalenivån, och innehållet ersätts sedan med en "blob" som upptar lika stor del av rutan som gråskalenivån. *Dithering* eller *Error diffusion* är en mer raffinerad metod när man reducerar antalet nivåer i en bild (alltså en form av kvantisering) och därvid "sprider ut" kvantiseringsfelet på omgivande pixlar med hjälp av ett rekursivt filter. På `http://en.wikipedia.org/wiki/Error_diffusion` finns en bra beskrivning av tekniken.

Skriv ett program som gör om en gråskalebild till binär tryckbar form med hjälp av klassisk halvtoning med "blobbor" respekive *Floyd-Steinberg error diffsion*. Pröva olika storlekar på rutindelningen och jämför resultatet av de två metoderna. I mån av tid och intresse, anpassa algoritmerna för att även hantera färgbilder!

4 Videoprocessor

Gör ett program som manipulerar en videofil på olika sätt (Matlab gör det enkelt att ta in videofiler i form av flerdimensionella matriser). Programmet kan t.ex. applicera olika filter (högpass,

lågpass), ändra intensitet och kontrast (fundera på hur det går till). Man kan även byta ut vissa färger, och kanske ersätta dessa områden med annat innehåll (t.ex. från en annan videofil) - s.k. chroma key bygger på den principen. Du kan också göra operationer över tiden - att ta skillnaden mellan två på varandra följande bildrutor är ett effektivt sätt att skilja ut rörelser (t.ex. en person) från statisk bakgrund. Fantasin sätter gränserna - tanken är att experimentera fritt men kom ihåg att dokumentera era experiment och se till att ha ett körbart program i slutändan som kan göra några olika typer av manipulationer på den fil man skickar in.

Använd kommandona `ViderReader()` och `VideoWriter()` för att läsa resp. skriva videofiler.

5 Ljudeffektprocessor

Gör en *ljudeffektprocessor* för olika effekter som man vanligen tillämpar på t.ex. gitarr eller röst. Programmet ska läsa in en ljudfil, behandla den enligt angivna parametrar, och spara den behandlade filen, alternativt spela upp den. Testa på olika ljudfiler, t.ex. din egen röst, något soloinstrument, och musik. Här är några effekter du kan implementera:

Chorus/flanger/phaser

Egentligen tre olika namn på samma effekt, men med olika grundinställningar: Signalen adderas med en fördröjd variant, och fördröjningen varieras enligt en långsam sinussvängning:

$$y(n) = \alpha x(n) + \beta x(n - \tau(n))$$

där

$$\tau(n) = D(1 + \phi \sin(2\pi f n))$$

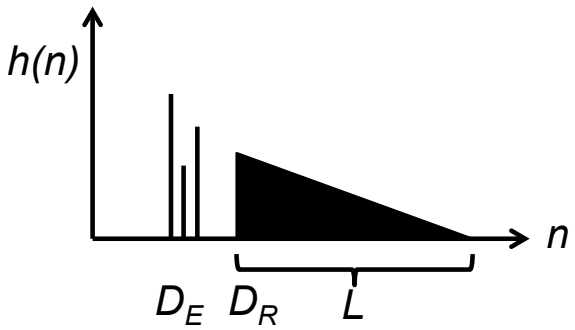
Här är D medelfördröjningen i antal sampel, f är frekvensen (i fraktioner av samplingsfrekvensen) och ϕ anger modulatiosdjupet, dvs hur mycket fördröjningen ska påverkas av sinusvågen. ϕ bör vara i intervallet 0 - 1. En *chorus*-effekt får man då fördröjningen är ca 50 ms och frekvensen är ca 0.25 Hz. För en *flanger*-effekt, pröva 10 ms . *glöm inte att räkna om fördröjningar till sampel och normera frekvenser med f_s innan du sätter in i formlerna!*. Sätter man β till ett negativt värde får man en *phaser*-effekt. Det går även att lägga till en återkopplingsterm $+\gamma y(n-1)$ vilket ger ett vassare ljud - pröva dig fram till olika inställningar.

Reverb/Eko

Rymdklang uppkommer från ljudets reflektioner mot hårda ytor. En första approximation kan fås av ett enkelt återkopplat filter:

$$y(n) = x(n) + \beta y(n - D)$$

Detta ger en enkel ekoeffekt med regelbundna avtagande ekon med D sampel mellan varje eko. Ofta kan man dock inte urskilja de enskilda ekona i en efterklang, helt enkelt för att det är så många. De första ekona (s.k. tidiga reflektioner) dröjer en bestämd tid som beror på avståndet till närmsta yta, men efter det kan ekona komma i princip hur tätt som helst och med olika intensitet. För att simulera detta kan man skapa ett implussvar som först består av några spikar (motsvarande de tidiga reflektionerna) och sedan ett ”brus” med avtagande intensitet (det får du genom att fylla en vektor med slumptal - använd funktionen `rand()` - och multiplicera med en rampfunktion, t.ex. `ramp = 1-(0:N)/N`. Det önskade impulssvaret är alltså något i den här stilen:



Reverb-effekten fås sedan enkelt genom att filtrera ljudet med impulssvaret (använd `filter()` eller `conv()`). Gör så att man kan variera D_E - tiden till den första av de tidiga reflektionerna (det får gärna vara några stycken, de bör komma med slumpmässiga korta avstånd och med olika nivå), samt starttid och längd för "bruset" (D_R och L i figuren). Försök använda dessa parmetrar för att simulera rum av olika storlek och typ. Gör t.ex. så att man kan ange rumsstorlek/avstånd till närmsta yta i meter och räkna ut D_E därefter.

Ett annat effektivt sätt att få olika reverb-effekter är att helt enkelt ladda ner impulssvar uppmätta från verkliga platser - allt från akvedukter till konserthallar. På `https://www.propellerheads.se/blog/f`

finns länkar till ett antal sajter där du kan hämta olika impulssvar.

Distortion

Att låta signalen passera en olinjäritet är ett effektivt sätt att öka övertonsinnehållet i en signal, och utnyttjas typiskt för t.ex. elgitarr. Undersök vilken effekt på ljud och frekvensspektrum som fås av olika olinjäriteter. Pröva t.ex. *hard* respektive *soft clipper* på denna sida:

`http://ccrma.stanford.edu/~jos/pasp/Nonlinear_Distortion.html`

Rita upp förstärkningskurvan, dvs utsignal som funktion av insignal för olika effekter.

3D-ljud (auralisering)

Människor är ganska bra på att bedöma varifrån ett ljud kommer. Denna förmåga bygger på det faktum att ljudet som når höger resp vänster öra skiljer sig avseende tidsfördröjning, energinivå och frekvensinnehåll. Ett ljud som kommer rakt bakifrån får t.ex. ett annat frekvensinnehåll än ett ljud som kommer rakt framifrån pga ytterörats utformning. Ett ljud som kommer från höger kommer nå högerörat innan vänsterörat, och nivån kommer även vara högre på höger sida.

Alla dessa aspekter (alltså fördröjning, nivå och frekvensinnehåll) går att baka ihop till ett filter, vilket gör att man ganska enkelt kan simulera 3D-ljud digitalt: allt man behöver är ett filter för varje öra. Detta kallas för en Head Related Transfer Function (HRTF). I praktiken behöver man en uppsättning impulssvar för höger resp. vänster öra som motsvarar olika riktningar i rummet.

Det finns färdiga uppsättningar impulssvar för HRTF-filter som man kan använda för att bygga sin egen 3D-ljudsimulering, t.ex. här:

`http://sound.media.mit.edu/resources/KEMAR.html`.

Denna databas innehåller impulssvar motsvarande 710 olika riktningar erhållna genom att mätningar på ett "dummyhuvud" där man satt mikrofoner i örongångarna, och spelat in ett känt ljud från ett antal olika riktningar i ett ekofritt rum.

Den här uppgiften går ut på att skriva ett program som kan ta ett monoljud och - med hjälp av ovanstående resurs - göra om signalen till ett stereoljud som låter som om det kom från viss plats i rummet (angiven relativt huvudet i sfäriska koordinater). Gör även så att ljudet kan "vandra

runt" huvudet. För att effekten ska bli bra måste man lyssna i hörlurar och gärna använda ett så "torrt" ljud som möjligt t.ex. en röst inspelad med närmikrofon (t.ex. mobiltelefon).

Gör även ett litet lyssningsförsök där ett antal olika försökspersoner får lyssna på ljuden och bedöma riktningen, och gör statstik på hur bra det stämmer med förväntan.

6 Simulering av svängande sträng

En svängande sträng kan simuleras som en serie av N punktmassor med massan M samanbundna av fjädrar av längden L och fjäderkonstanten K. När systemet är i vila kommer punktmassorna ligga på en rät linje längs x-axeln (vi bortser från gravitation). Vi nöjer oss med att betrakta punktmassornas rörelser vinkelrät mot strängen, alltså i y-led, den sk transversella svängningen. Vi betecknar punktmassans k avstånd från vilolinjen vinkelrät denna som $y(k)$. Den kommer då påverkas av krafter från massorna vid $k-1$ och $k+1$, och vi räknar med att de tar ut varandra i x-led, men kan ge ett nettobidrag i y-led. Denna nettokraft påverkar massan enligt newtons rörelselag $F = ma$, vilket leder till en ändring i punktens hastighet, som vi kan beteckna $v(k)$. Dessutom finns en dämpfaktor D, som ger upphov till en kraft som är proportionell mot hastigheten, men motriktad: $F = -Dv(k)$.

Skriv ett program som simulerar strängens rörelse! Börja med att initialisera alla punkters lägen $y(k)$ och hastigheter $v(k)$ - det motsvarar att slå an strängen, och kan göras på hur många sätt som helst - och räkna sedan fram en ny position och hastighet för varje massa med hjälp av fjäderekvationen, newtons rörelselag, och dämpningsformeln. Stega sedan fram tiden ett steg, och gör om beräkningarna!

Pröva dig fram med simuleringen, försök med olika antal och värde på massor, fjäder- och dämpningskonstanter och anslagsvarianter. Spara y-värdet vid en viss position över tid i en vektor, och spela upp denna som ett ljud! Låter det naturligt?

Matlab-tips: för att visualisera strängen som en animation vartefter som beräkningarna pågår kan man använda matlab-kommandot `drawnow` efter varje plot-anrop.

Om du vill och har tid kan du försöka spela melodier med strängen genom att ändra t.ex. fjäderonstanten eller massan!