Oppgave

Manipulering av biologisk modell ved bruk av Phantom

Oppgaven går ut på å utnytte haptikk under manipulering på fysisk basert datamodell av biologisk materiale. Modellen skal kunne manipuleres ved deformering og kutting. En Phantom haptisk enhet brukes for å gi brukeren et tredimensjonalt verktøy for å kunne utføre manipulasjonen på modellen.

Modellen skal ilegges fysiske egenskaper ved å basere den på et masse-fjær-system, og den skal kunne animeres ved at den utsettes for eksterne krefter. Modellen utsettes for eksterne krefter ved bruk av en Phantom-arm som igjen yter en motkraft som brukeren kan kjenne. Dette vil gi en økt grad av realisme for brukeren da modellen både oppfører seg og føles som det naturlige objektet som etterlignes.

Sammendrag

Denne rapporten handler om hvordan en Phantom haptisk enhet kan anvendes i manipulering av geometrisk datamodell som etterligner organisk materiale. Modellen skal kunne endres ved påføring av krefter på modellens overflate. Det skal være mulig å kutte i samt å trykke på modellen slik at den deformeres. Haptisk tilbakemelding sørger før at brukeren kan føle berøringen av modellen i tillegg til at ulike teknikker benyttes for å gjøre det visuelle inntrykket realistisk. Dette systemet danner grunnlaget for simulering av kirurgiske inngrep, noe som kan være til stor nytte i planlegging av slike inngrep, samt benyttes i opplæring og trening av medisinsk personell.

Jeg tar for meg de viktigste komponentene en kirurgisk simulator må inneholde og forklarer hvordan jeg har implementert disse i programmet som er utviklet. I rapporten legges det vekt på fysisk modellering av organisk materiale, bruk av haptisk tilbakemelding, kutting i elastiske materiale og teknikker for å øke hastigheten på kollisjonssjekking mot modellen. I tillegg er det lagt vekt på hvordan det visuelle inntrykket kan forbedres.

Forord

Rapporten er et resultat av masteroppgave i faget TDT4900 i gruppen for Algoritmekonstruksjon og Visualisering ved Institutt for Datateknikk og Informasjonsvitenskap, Norges Teknologiske og Naturvitenskaplige Universitet våren 2005.

Målet for oppgaven har vært å anvende en Phantom haptisk enhet i avanserte anvendelser, nærmere bestemt i manipulering av masse-fjær basert datamodell av biologisk materiale.

Jeg vil rette en takk til Torbjørn Hallgren for god og kyndig veiledning underveis.

Trondheim, 16. juni 2005

Stig Rune Søberg

Innhold

Kapittel 1

Innledning

1.1 Motivasjon

Syn og hørsel er sanser som blir benyttet i stor sett alle dataprogrammer nå til dags, og det finnes en mengde lyd og bilde utstyr på markedet. De andre sansene som lukt, smak og berøring er det heller lite vanlig å benytte i programmer.

Haptikk er læren/vitenskapen om berøringssansene og det forskes mye på teknikker for hvordan man skal kunne benytte berøringssansene i ulike programvaresammenhenger. Ved å benytte spesielle inn/ut enheter (haptiske enheter) sammen med spesiell programvare kan man gi brukerne av programvaren mulighet til å berøre og manipulere virtuelle objekter. Tilbakemeldingen som gis brukeren er avhengig av hvilken type haptisk enhet man bruker. Det finnes flere ulike typer utstyr som kan gi en kraft tilbake til brukeren. I spillindustrien er det spesielt joystikker til bruk i flysimulatorer og ratt til bilspill som har vært populære i en tid. Dette er enheter som gir en enkel form for kraft tilbakemelding som er løst basert på den kraften man ville følt i virkeligheten. Haptiske enheter som f.eks. en Phantom Desktop gir muligheter til å gi en mye mer nøyaktig simulering av de kreftene man kan føle i virkeligheten. Figur 1.1 viser bilder av de ulike enhetene som er nevnt ovenfor.

Haptikk brukes på mange ulike områder og det kommer stadig nye bruksområder. Eksempler på bruksområder kan være:

- Kirurgiske simulatorer
- Opplæring av medisinsk personell

Figur 1.1: Kraft tilbakemeldingsenheter

- Data assistert konstruksjon (CAD)
- Spill
- Hjelpe blinde og svaksynte
- Osv...

1.2 Problemområde

Metaxas[1] diskuterer hvordan man i dag går fra 2-dimensjonale bilder i medisinsk sammenheng til 3-dimensjonale modeller som bedre og enklere viser organers struktur og funksjon. Den store utviklingen i regnekraft på maskinvare de siste årene, har banet vei for ny bildeanalyse og nye visualiseringsalgoritmer som ser ut til å endre måten medisinfaget praktiseres. Muligheten til å visualisere kroppens indre struktur og funksjon kan forbedre legers kunnskap og kanskje lede til oppdagelse av viten. Artikkelen er en introduksjon til en rekke artikler som omhandler ulike teknikker og forsøk på å forbedre medisinsk bildeanalyse og modellering fra medisinsk data.

Sanntids interaksjon med deformerbare modeller spiller en stor rolle i mange interaktive virtuellvirkelighets programmer. Simulering av kirurgiske inngrep kan benyttes i trening av medisinstudenter og i planlegging av ulike inngrep, og er en veldig interessant og lovende teknologi i slikt henseende. Simulering av kirurgiske inngrep er veldig avhengig av at modellene man jobber på gir sanntids tilbakemelding både visuelt og haptisk. Bruk av virtuellvirkelighet i medisinsk opplæring og kirurgisk simulering begynte på slutten av 80-tallet[2], og har siden da hatt en stor utvikling når det gjelder realisme og

interaktivitet. I begynnelsen var slike systemer ikke særlig realistiske eller interaktive, men med utvikling av ny maskinvare har disse systemene blitt stadig bedre.

For å lage en god kirurgisk simulator er det mange komponenter som spiller inn. Bro-Nielsen[3] angir tre hovedkomponenter en generell kirurgisk simulator bør bestå av.

- Grafikk
  Skal gi et visuelt inntrykk av hva som skjer under simuleringen av det kirurgiske inngrepet.

- Haptisk brukergrensesnitt
  Skal gi brukeren en følelse av å berøre modellen som benyttes i simuleringen. Bevegelsene og kreftene som utføres skal påvirke simuleringen og en haptisk tilbakemelding skal gies til brukeren.

- Fysisk modellering
  Det er nødvendig å benytte fysisk modellering av det virtuelle objektet for at det skal oppføre seg realistisk både visuelt og haptisk.

Denne oppgaven er en fortsettelse på et tidligere prosjekt[4] hvor jeg så på mulighetene som ligger i bruk av en Phantom Desktop haptisk enhet under medisinske anvendelser. I det prosjektet ble det ikke benyttet noen form for fysisk baserte egenskaper på modellen, men ulike teknikker for å gi slike egenskaper ble diskutert og i dette prosjektet vil et masse-fjær-system bli benyttet for å gi modellen de fysiske egenskapene som er ønskelige. I det tidligere prosjektet ble en algoritme for å kutte i modellen implementert og en lignende metode vil bli benyttet i dette prosjektet. Jeg vil også i denne oppgaven benytte en polygonbasert modell for å representere et organisk materiale og utnytte ulike teknikker for å manipulere på modellen. Manipuleringen vil bestå i deformering av, og kutting i modellen.

Kapittel 2

Tidligere arbeid

Haptisk programmering kan bli sett på som en form for gjengivelse av krefter som skal utføres av en haptisk enhet. Haptisk programmering kan inkludere vitenskap fra veldig mange grener, og kombinere grener som fysikk, robotikk, geometriske beregninger, numeriske metoder og programvareutvikling. Her kommer en oversikt over tidligere arbeid på flere av disse områdene.

2.1 Kirurgiske simulatorer

Utviklingen av kirurgiske simulatorer har som nevnt tidligere vært stor siden oppstarten på 80 tallet og stadig nye og bedre simulatorer vil komme i tiden fremover. Det eksisterer mange ulike kirurgiske simulatorer for ulike områder i den medisinske verden.

Mosegaard[5] presenterer en simulator for planlegging av inngrep på medfødt hjertesykdom. Kilden sier ikke noe om kutting i modellen, men ser bare ut til å gi muligheter for deformasjon av modellen. En lokal masse-fjær-modell benyttes for å oppnå høy oppdateringsfrekvens selv med ganske detaljerte modeller.

Zhang, Payandeh og Dill[6] presenterer en kirurgisk simulator som bruker en utvidet masse-fjær-modell for å representere menneskelig vev. Modellen deles inn i triangler og hvert punkt i modellen får tildelt en masse. Langs kantene i modellen legges det fjær, som alle initialiseres til å være strekte for å simulere spenning i vevet. Når det så kuttes i modellen vil dette medføre at kuttet åpnes automatisk. En lokal forfining av modellen ved det virtuelle verktøyet sørger for rask oppdatering av kuttet når man skjærer i modellen. Mer om kuttingen i denne artikkelen i kapittel 2.3.

Delingette og Ayache beskriver i[7] noen av hovedfasene de gikk gjennom under utviklingen av en simulator for kirurgi på en menneskelig lever. De benytter seg av CT bilder av en lever og lager så en 3 dimensjonal modell etter disse bildene. Kilden benytter elastisitetsteorien og endelige elementers metode for å gi modellen fysiske egenskaper. De forenkler etterligningen av biologisk materiale ved å anta lineær elastisitet i materialet, noe som er akseptabelt for små deformasjoner. Modellen deles inn i tetraeder med det mål om at hvert tetraeder skal være så regulært som mulig. To ulike algoritmer diskuteres, hvor den første benytter seg av en del forhåndsberegninger, men får da den begrensningen at det ikke kan kuttes i modellen. Den andre er mer allsidig men desto mer beregningstung å man får begrensninger i detaljnivå på modellen.

I [8] presenteres en simulator for å sy i vev. Som kilden nevner sys sår sammen på nesten alle slags helsestasjoner, og denne simulatoren vil kunne gi personell på disse stasjonene en mulighet til å få grunnleggende opplæring og trening i dette uten å "skade" personer. En masse-fjær-modell benyttes for å modellere vevet, og en modifisert implisitt metode benyttes for å løse systemet.

2.2 Masse-fjær-modell

Masse-fjær-systemer er en mye brukt metode i simulering av elastiske modeller. Et slikt system består av noder som er koblet sammen ved hjelp av fjær. Man diskretiserer objektet i et antall noder og fordeler massen til objektet over disse nodene. Fjærene mellom nodene modelleres vanligvis som lineært avhengige av avstanden mellom dens to koblingspunkter, men andre teknikker som delvis lineære eller ikke-lineære er også brukt.

Fjær kan i tillegg til å være avhengig av avstanden til nodene sine også ta hensyn til dempningskrefter i materialet. Dempningskrefter er avhengige av både posisjonen og hastigheten på nodene. Dette simulerer krefter som begrenser bevegelsen i materialet og dermed demper svingninger. Systemer som ikke benytter dempning vil aldri komme til ro, men svinge konstant

Som nevnt ovenfor så foretas en diskretisering av objektet, noe som gjør at de fysiske egenskapene kan modelleres i et system av differensialligninger, og deretter løses numerisk for å finne nodenes nye plasseringer. I hovedsak er det to ulike angrepspunkt for å løse slike systemer, og det er eksplisitte eller implisitte metoder. Eksplisitte metoder er en teknikk hvor man benytter

tilstanden i forrige tidssteg og så går ett lite steg i retningen av den deriverte. D.v.s. at man har posisjonen til nodene og så flytter disse et lite stykke langs fartsretningen. Hvis man benytter implisitte metoder beregner man tilstanden ved å benytte tilstanden til naboene i det samme tidssteget. Dette krever mer utregning da man må løse et ligningssystem med flere ukjente. Fordelen med denne metoden er at man kan ta større tidssteg, da implisitte metoder ikke divergerer like lett som eksplisitte metoder.

Det eksisterer flere ulike eksplisitte metoder for å løse slike differensialligninger. Witkin m.fl. tar i [9] for seg flere av disse, hvor Euler metoden er den enkleste. Dersom startverdien (startposisjonen) til node x kalles $x_0 = x(t_0)$ og vår beregning av x på et senere tidspunkt $t_0 + h$ kalles $x(t_0 + h)$, hvor h er tidssteget, kan man med Eulers metode beregne $x(t_0 + h)$ ved å gå ett steg i den derivertes retning,

$$x(t_0 + h) = x_0 + h \cdot \dot{x}(t_0).$$

Eksplisitte metoder har som sagt ulempen med at tidsstegene må være små for å hindre at systemet divergerer. Får å øke nøyaktigheten er det mulig å beregne den deriverte flere ganger i ett tidssteg. I Midpoint metoden beregner man den deriverte to ganger for hvert tidssteg. Runge-Kutta er en metode hvor den deriverte beregnes fire ganger.

Xavier Provot[10] beskriver en masse-fjær simulering av tekstiler. Elastiske egenskaper simuleres, men det tas hensyn til de uelastiske egenskapene til vevde tekstiler. Tekstilobjektet diskretiseres i ett sett av punkt med masse og ulike fjær mellom disse punktene. Områder med høyt stress får i slike simuleringer et unaturlig utseende da de strekkes alt for mye. En vanlig løsning på dette er å øke stivhetskonstantene få fjær som strekkes mye, men dette kan få uønskede resultater og øker beregningskostnadene. Denne artikkelen presenterer en ny løsning på dette problemet. Dersom en fjær strekkes mer enn en bestemt faktor f.eks. 10% korrigeres ganske enkelt dette ved å minske lengden på fjæren til maks utstrekking. Dersom massepunktene i begge endene på fjæren som skal korrigeres er løse (ikke låst til et fast punkt), flyttes begge punkt mot midten på fjæren slik at lengden blir lik maks lengde. Dersom det ene massepunktet er låst flyttes det løse punktet mot det låste inntil maks lengde oppnås.

I [11] beskriver Vassilev og Spanlang en masse-fjær-modell for sanntids simulering av lukkede objekter med volumbevaring. De introduserer en ny type fjær, kalt støttefjær, som simulerer materialet som er inne i modellen.

Dette sørger for at modellen bevarer sitt volum uten noe behov for eksplisitte beregninger av volumet i modellen, noe som er vanlig i konvensjonelle metoder. For å oppnå dette legger de disse nye fjærene fra alle punkt i modellen til sentrum av modellen. Kraften som virker fra en slik fjær er avhengig av lengden på alle de andre støttefjærene. Dersom summen av alle støttefjærene på et tidspunkt i simuleringen er mindre enn ved oppstart, vil dette føre til at de strekkes ut og volumet økes, og dersom summen av alle støttefjærene er større enn ved oppstart vil de trekkes sammen.

2.3 Kutting

Et viktig aspekt i kirurgiske simulatorer er metoder for å kunne gjøre kutt i modeller som benyttes. Det er mange ulike angrepsvinkler for hvordan kuttingen utføres. Bruyns m.fl.[12] har oversikt over ulike kutteteknikker og grupperer disse etter hvordan de håndterer ulike oppgaver. Her følger en rask oppsummering av de ulike teknikkene.

- Definisjon av kuttevei.
  Den enkleste metoden er å finne punktene som er nærmest start og ende punkt på kuttet og så åpne kantene mellom disse. Mer avanserte metoder vil være å følge det virtuelle verktøyet og kutte fortløpende i de flatene man passerer. Valg av metode avhenger av hvor realistisk simuleringen ønskes.

- Åpning av kuttet.
  Hvordan kuttet skal åpnes er også avhengig av hvor realistisk simuleringen ønskes. En enkel metode er å fjerne alle flater som berøres av kuttet, men dette gir ikke særlig realistiske kutt. Andre muligheter er å endre modellen ved å dele flater langs det virtuelle verktøyets bane.

- Antall nye primitiver.
  Ved kutting må det genereres nye primitiver og det må bestemmes hvor mange nye primitiver som skal genereres. Ved å flytte punkter slik at kanter legges langs kuttretningen genereres et minimum av nye primitiver. Det genereres da ingen nye flater og bare ett nytt punkt genereres for hvert punkt som åpnes. Dersom kuttet går gjennom k punkt genereres $k - 2$ nye punkter. Ved å tillate generering av nye flater og punkter kan kvaliteten på og utseende til modellen forbedres da ingen eksisterende punkt flyttes og detaljene langs kuttet øker som følge av nye og mindre flater.

- Når skal nye primitiver genereres.
  Generering av nye primitiver kan utføres ved ulike tidspunkt. Man kan vente til kuttingen er ferdig, fortløpende under kuttingen eller når kutteretningen endres. Generering av primitiver er enklest utført dersom dette utføres når kuttingen er ferdig, men hvis primitiver genereres fortløpende gir dette en mye mer realistisk og visuelt tilfredsstillende resultat.

- Representasjon av virtuelt kutteverktøy.
  En vanlig representasjon er ett enkelt punkt som det sjekkes kollisjon mot. Andre representasjoner kan være linjer, triangler eller objekt bestående av flere primitiver.

Lim og De presenterer i [13] en metode hvor kutting oppnås uten at nye flater må genereres. Dette oppnås ved at eksisterende punkt flyttes slik at kantene mellom disse blir liggende langs kuttet. Ved start på et kutt befinner det virtuelle verktøyet seg på en flate i modellen og punktet i flaten som er nærmest kollisjonspunktet med verktøyet flyttes til kollisjonspunktet. Samme metode brukes idet man avslutter et kutt. Mellom start og slutt punktet skjærer man over flere flater og idet man passerer en kant mellom to flater flyttes punktet på kanten som er nærmest skjæringspunktet til skjæringspunktet. Ett nytt punkt må genereres for alle punkt langs kuttet bortsett fra start og sluttpunktet for at kuttet skal kunne åpnes. Flater på den ene siden tilordnes så det originale punktet, mens flatene på den andre siden tilordnes det nye punktet. Det foretas også en generering av polygon nede i kuttet for å gi overflatemodellen en illusjon volum. En lokal forfining benyttes for å øke den visuelle kvaliteten i nærheten av det virtuelle vektøyet. For å etterligne fysiske egenskaper brukers en *meshfree* metode for å beregne deformasjons felt og interaksjons krefter.

Zhang, Payandeh og Dill viser i[14] en kutteteknikk som tillater generering av nye flater. De skiller her mellom ulike stadier i kuttingen og har forskjellige algoritmer for oppdeling av flater på start/slutt og mellomveis flater. Dersom det virtuelle verktøyet kun beveges over en kant i flaten er denne flaten enten start- eller sluttflate og en algoritme for oppdeling av disse kjøres. Men dersom to kanter i en flate berøres kjøres en annen algoritme for å dele denne. Også i denne artikkelen benyttes generering av flater nede i kuttet for å gi en illusjon av volum. Artikkelen diskuterer også en metode for hvordan man kan forbinde to kutt.

De samme forfatterne presenterer i[6] en kutteteknikk som er ganske lik den ovenfor, men her benytter de seg av en midlertidig lokal forfining av flater som

kuttes. Når en flate er kuttet helt gjennom og man befinner seg i en annen flate fjærenes den lokale forfiningen og det endelige kuttet gjennom flaten beregnes. Dette gjør at brukeren vil oppleve at modellen åpnes fortløpende under kutting, noe som øker realismen betraktelig. Andre teknikker hvor et kutt ikke blir synlig før man har kuttet helt gjennom flaten er noe enklere å implementere, men lider som sagt av en forsinkelse i åpning av kuttet, noe som vil virke litt urealistisk for brukeren av systemet. Denne forfiningen består i at når kutting i en flate startes, fjernes denne flaten fra modellen og erstattes av en "lapp" bestående av tre mindre flater. Disse flatene deles så opp i mindre flater avhengig av hvordan det virtuelle verktøyet beveges på denne lappen. Når verktøyet befinner seg i en ny flate på modellen erstattes lappen av flater som legges til modellen, og som viser det endelige kuttet. I denne artikkelen diskuteres også hvordan en modell kan deles i to eller flere modeller når man kutter helt rundt/gjennom modellen.

Bruyns og Montgomery[15] presenterer et system for sanntids kutting i objekter med ulik topologi. Det beskrives hvordan kutting, i enkel og lagvis overflate samt en blanding av disse og en volummodell implementeres. I tillegg viser kilden hvordan denne kuttingen kan utføres med virtuelle verktøy som består av en eller flere kutteflater. Kutting foretas mot verktøyenes skarpe kanter, som representeres som kanter. Når verktøyet beveges legges en grenseramme(bounding box) rundt disse skarpe kantene og kollisjonsrespons kalkuleres mot de delene av modellen som befinner seg innenfor denne grenserammen. Valget om ett primitiv skal kuttes eller ikke er avhengig av primitivets tilstand. Tilstanden til primitivene lagres i primitivet selv og brukes når primitivet skal oppdateres. Dersom primitivet ikke er gjennombrutt fra før men blir brutt, lagres gjennombrytningen og primitivet settes i tilstanden *start*. Dersom primitivet i neste iterasjon fortsatt er i kollisjon med verktøyet settes det til tilstanden *oppdater*, hvor ingen kutting foregår, og bare kollisjonsposisjoner oppdateres. Hvis i en senere iterasjon primitivet ikke er i kollisjon med verktøyet settes tilstanden til *flytt*, og primitivet blir kuttet ved hjelp av primitivets konfigurasjon og kantkollisjon med det nye primitivet. Modellen kuttes dermed hvis det virtuelle verktøyet beveges i riktig retning og fra en flate til en annen. Modellen kuttes fortløpende når man drar verktøyet videre gjennom nye flater.

Kapittel 3

Løsning

I dette kapitlet beskriver jeg min løsning på problemet. Jeg har benyttet et masse-fjær-system for å modellere det organiske materialet, og benytter OpenHaptics api'et fra Sensable Technologies Incorporated[16] for gi den haptiske koblingen mot modellen. For å tegne modellen på skjermen benyttes OpenGL som er et api utviklet av SGI[17]. Jeg tar her for meg de forskjellige delene jeg har brukt for å implementere et system hvor man kan kutte og deformere den virtuelle modellen med en haptisk enhet.

3.1 Haptisk grensesnitt

Den haptiske enheten som benyttes er en Phantom Haptisk enhet. Denne produseres av Sensable Technologies Incorparated. Phantomen har 6 frihetsgrader. Den kan beveges fritt i et tredimensjonalt rom og roteres rundt alle akser. Den kan gi haptisk tilbakemelding i de 3 bevegelsesfrihetsgradene, mens rotasjonen alltid er fri.

For programmeringen mot Phantom enheten benyttes OpenHaptics, som er et api levert av Sensable. Dette api'et er i hovedsak delt i to ulike deler. Den ene delen kalles Haptic Device API (HDAPI) og denne gir programmereren lavnivå tilgang til den haptiske enheten. HDAPI gir programmereren mulighet til å rendre krefter direkte til enheten, samt å kontrollere kjøretidsoppførselen til driveren. Den andre delen heter Haptic Library API (HLAPI). HLAPI gir programmereren høynivå tilgang og er designet slik at det ligner på OpenGL og man kan legge inn haptikk i programmet på nesten identisk måte som man tegner primitiver i OpenGL. Dette gjør at man kan gjenbruke eksisterende kode, og ikke trenger å tenke på synkronisering mellom haptikken og grafikken. Her følger et eksempel som

viser hvordan man enkelt kan legge til haptisk tilbakemelding på en firkant som tegnes i OpenGL.

Original OpenGL kode	OpenGL kode med haptisk tilbakemelding
	hlBeginShape(SHAPE TYPE);
glBegin(GL_POLYGON) glVertex3f(0,0,0) glVertex3f(1,0,0) glVertex3f(1,2,0) glVertex3f(0,2,0) glEnd()	glBegin(GL_POLYGON) glVertex3f(0,0,0) glVertex3f(1,0,0) glVertex3f(1,2,0) glVertex3f(0,2,0) glEnd()
	hlEndShape()

Dokumentasjonen til OpenHaptics sier ikke noe om akkurat hvordan kollisjonssjekkingen foregår, men det er to ulike metoder som kan velges mellom og disse er forklart i 3.3. Når kollisjon er oppdaget beregnes kraften Phantomen skal utøve ved hjelp av proxymetoden som er forklart i 3.5.2.

3.2 Virtuell Modell

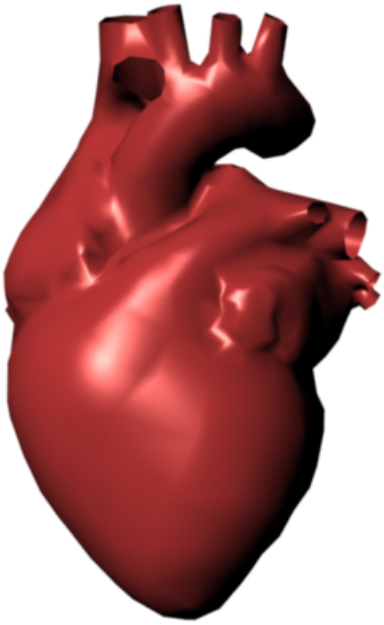
Prosjektet retter seg som sagt mot manipulering på en virtuell modell av organisk materiale. Det var derfor ønskelig med en modell av et organ fra menneskekroppen, og en polygonbasert modell av et hjerte ble benyttet til dette formålet. Denne modellen ble hentet fra det fritt tilgjengelige biblioteket med modeller på 3DCafe[18] og er en modell som viser den geometriske overflatestrukturen og de største blodårene inn og ut av hjertet. Det sies ikke noe om hvordan denne modellen er bygd opp, eller hva slags referansemateriale den er modellert etter. Modellen var lagret i formatet .3ds som er format benyttet av 3D Studio og senere 3D Studio Max[19], så jeg kan bare anta at modellen er modellert i et av disse programmene, men det finnes plugins til stort sett alle modelleringsverktøy som kan skrive filer til dette formatet. Modellen ble importert til Maya[20] hvor en del geometriske feil ble rettet. Dette var feil som f.eks. hull i modellen på steder som ikke skulle ha hull, samt punkter som åpenbart var feilplasserte. Modellen ble også triangulert for å sikre at alle flatene var triangler da dette er den flatetypen kirurgiprogrammet krever. Alle punktene må ligge innenfor en enhetsboks med sentrum i origo for å komme på riktig sted og med

riktig størrelse i programmet. Modellen ble derfor skalert til riktig størrelse før den endelige versjonen av modellen ble lagret. For å bruke modellen i programmet ble det utviklet et eget filformat som er tilpasset det formålet. En maya modelleksportør ble laget for å konvertere modellen fra maya's filformat (.ma eller .mb) til det nye formatet. Data som skrives til det nye formatet er punktene i modellen, flatenes sammensetning, normalvektorer og tekstur koordinater. Data lagres som et indeksert flatesett ved at hver flate inneholder indekser til sine tre hjørner. Dette er hensiktsmessig da flere flater kan ha samme punkt som ett av hjørnene sine og man trenger da ikke å lagre hjørnekoordinatene flere ganger. Det er også hendig med en slik organisering da man kan oppdatere posisjonen til kun ett punkt og flatene som deler dette punktet blir automatisk påvirket (man trenger ikke oppdatere hver flates punkt). Koordinatene til alle punktene i modellen ligger i en egen liste, og den fysiske simuleringen som skal gjøres mot modellen jobber kun mot denne listen av punkter. I tillegg til data som skrives automatisk når man konverterer/eksporterer filen må det manuelt legges inn hvilken massen modellen har, samt stivhet- og dempningskoeffisienter på struktur- og posisjoneringsfjærene (se 3.5.1 for mer info om disse). Figur 3.1 viser hvordan hjertemodellen ser ut. En nærmere beskrivelse av filformatet finnes i tillegg B.

3.3 Kollisjonssjekking

Visuelt er en oppdateringshastighet 20 - 30 Hz regnet som et akseptabelt nivå, men den haptiske tilbakemeldingen må ha en oppdateringshastighet som er mye høyere. De taktile sensorene i huden bør kjenne en oppdateringsfrekvens på 200-300Hz eller høyere[21] for at det skal føles virkelig. OpenHaptics benytter en egen tråd for den taktile tilbakemeldingen som bør operere på omkring 1000Hz. Dersom hastigheten på tilbakemeldingen faller mye under 1000Hz vil brukeren merke risting p.g.a. at den haptiske enheten flytter seg for langt inn i modellen og når kraften som skal trekke Phantom posisjonen ut av objektet beregnes (mer om dette i 3.5.2), blir denne for stor og det blir en overkorrigering av posisjonen.

Kollisjonssjekking foretas ved å legge inn de flatene man ønsker å sjekke kollisjon imot slik som beskrevet tidligere. OpenHaptics tar seg av selve kollisjonssjekkingen og benytter et punkt som sjekkes for kollisjon mot flatene. Metoden OpenHaptics benytter i kollisjonssjekkingen bestemmes ved hjelp av innparameter til metoden `hlBeginShape`. Det er to mulige valg av kollisjonssjekking og det er HL_SHAPE_DEPTH_BUFFER og



Figur 3.1: Modell av menneskelig hjerte som er benyttet i simulatoren

HL_SHAPE_FEEDBACK_BUFFER. HL_SHAPE_DEPTH_BUFFER er en metode hvor OpenHaptis bruker dybde bufferet til OpenGL og sjekker for kollisjon mot dette. Når `hlEndShape` kalles henter OpenHaptics et bilde fra dybde bufferet og sammenligner posisjonen på den haptiske enheten mot fargen i bildet, (som sier hvor i rommet en flate er). En ulempe med denne metoden er at man bare kan kjenne på flater som er synlige fra det visuelle synspunktet. Man kan ikke kjenne på f.eks. baksiden av et objekt, inne i objektet eller andre deler av objektet som ikke er synlige. Den andre metoden, HL_SHAPE_FEEDBACK_BUFFER, benytter seg av OpenGL's feedback buffer for å fange geometriske primitiver til den haptiske tegningen. I dette bufferet ligger informasjon om primitivene lagret og kollisjonssjekkingen foretas mot denne dataen.

I programmet benyttes den siste metoden da brukeren skal ha mulighet for å kjenne på modellen på steder som ikke er synlige som f.eks. baksiden på modellen.

Den haptiske tråden som tar seg av kollisjonssjekkingen og tilbakemeldingen til den haptiske enheten bør som sagt ha høy oppdateringshastighet. Får å

oppnå høy hastighet bør det sjekkes for kollisjon mot så få flater som mulig. To ulike metoder er brukt for å oppnå dette i implementsjonen. Når man ikke berører modellen sjekker man i neste steg kun for kollisjon mot flater som er i nærheten av den haptiske enheten, og dersom man berører modellen sjekker man for kollisjon mot den berørte flaten og dens naboer i neste steg.

3.3.1 Binærtre

For å sjekke for kollisjon mot flater i nærheten av den haptiske enheten benyttes binærtre. Dette er en metode hvor man suksessivt deler hver del av modellen i to nye deler. Det finnes ulike teknikker for å bestemme hvor man skal dele modellen. Noen baserer seg på å skape et balansert tre da det har den fordelen at man får en konstant søketid uansett hvor i modellen man befinner seg. Andre teknikker partisjonerer modellen ved å dele hver del på midten. Man får da ikke nødvendigvis et balansert tre, men et slikt tre er raskere å lage da man ikke trenger ulike beregninger for å bestemme hvor man må dele for å få balanse. Det er denne løsningen som er valgt i implementasjonen fordi oppdatering av binærtreet må være rask når man kutter i modellen og søketiden for et ubalansert tre ikke blir så veldig høy uansett.

Binærtreet lages ved at modellens punkter og flater sendes inn i en metode som genererer treet. Metoden deler så rommet punktene okkuperer i 2 deler på midten langs yz-planet og flatene legges til den delen av rommet de hører hjemme. Dersom en flate ligger på begge sider av planet legges den til i begge delene. Så foretas den samme inndelingen av de 2 delene, men da langs xz-planet. Og den tredje inndelingen blir langs xy-planet. Slik forstetter inndelingen inntil delene inneholder færre flater enn en gitt terskelgrense. Når det skal sjekkes kollisjon mot modellen kalles en metode, med den haptiske posisjonen som argument, som så traverserer binærtreet og returnerer hvilke flater som er i den delen av treet. Disse flatene legges inn i kollisjonssjekkingen.

3.3.2 Naboer

Når man berører modellen skal man som forklart tidligere sjekke kollisjon mot berørt flate og dens naboer. Dette betyr at det må beregnes hvilken flate man berører og legge denne inn i kollisjonssjekken. For å beregne hvilken flate man befinner seg på benytter jeg en enkel metode som sjekker sum av vinklene fra kollisjonspunktet til flatens tre hjørner. Denne metoden benyttes fordi kollisjonspunktet ikke nødvendigvis ligger akkurat på flaten, og man

slipper da med denne metoden å flytte kollisjonspunktet slik at det ligge i planet til flaten før man sjekker kollisjon. Følgende formel viser hvordan denne vinkelsummen beregnes, hvor **a**, **b** og **c** er hjørnene i flaten, og **p** er kollisjonspunktet.

$$\begin{aligned}
\overrightarrow{pa} &= \frac{p-a}{|p-a|} \\
\overrightarrow{pb} &= \frac{p-b}{|p-b|} \\
\overrightarrow{pc} &= \frac{p-c}{|p-c|} \\
angle1 &= \overrightarrow{pa} \cdot \overrightarrow{pb} \\
angle2 &= \overrightarrow{pb} \cdot \overrightarrow{pc} \\
angle3 &= \overrightarrow{pc} \cdot \overrightarrow{pa} \\
vinkelsum &= \arccos(angle1) + \arccos(angle2) + \arccos(angle3)
\end{aligned}$$

Den flaten som gir størst vinkelsum er kollisjonsflaten. Denne flatens naboer legges også til kollisjonssjekken slik at det virtuelle verktøyet ikke faller gjennom modellen når det beveges over kanten på den berørte flaten og inn i en ny flate. Får å slippe å konstant beregne hvilke flater som er naboer til den berørte flaten, beregnes det hvilke flater som er naboer med hverandre en gang i oppstart av programmet og alle flatene kjenner i ettertid sine naboer. Idet kutting i modellen begynner må "naboskap" på de involverte flatene oppdateres etter hvert. Mer om dette i 3.6.

3.4 Verktøy

For å manipulere på modellen finnes det to ulike verktøy, skalpell og tang. Verktøyene ble modellert i Maya, og er polygonmodeller som i likhet med hjertemodellen må bestå av bare triangler for å kunne brukes. Verktøyene er ikke modellert etter eksakt dimensjon og form fra de virkelige verktøyene, men løst basert på bilder av dem. De måtte også dimensjoneres i forhold til hjertemodellen for å få et sannsynlig størrelsesforhold. Verktøyene har ulike egenskaper som følge av deres form og funksjon. Med skalpellen kan man kutte i modellen dersom man trykker hardt nok mot den, mer om dette i 3.6, mens man med tangen kan trykke og skyve på modellen. Disse verktøyene eksporteres fra maya på samme måte som den virtuelle modellen. Etter eksporteringen må de få en grenseverdi som sier noe om hvor hardt de må trykkes mot modellen for at de skal kunne kutte i den. Disse verdiene kan

dermed justeres for å gi verktøyet den ønskede egenskapen. Skalpellen gis en ganske lav grenseverdi slik at kutting starter ved forholdsvis lavt påtrykk, mens tangen gis en veldig høy grenseverdi slik at kutting aldri startes når den brukes. Figur 3.2 viser hvordan de virtuelle verktøyene ser ut.

Figur 3.2: Verktøy

3.5 Deformasjon av overflate

For å gjøre modellen dynamisk er et masse-fjær-system benyttet, og som sagt i 2.2 så trenger man da en diskretisering av en modell. Modellen, som er en polygon modell, består av en rekke punkter som er koblet sammen til trekanter. Dette betyr at modellen er ferdig diskretisert og man kan dermed benytte punktene i modellen som noder i masse-fjær-systemet. Massen til modellen, som må legges inn manuelt i filen som inneholder modellen (se. 3.2), fordeles over alle nodene i modellen. Foruten sin egen masse kjenner nodene til sin posisjon, hvilken hastighet og retning de beveger seg, og hvor stor kraft som virker på dem. I dette systemet er det ikke lagt inn noen gravitasjon, så alle kreftene som virker på modellen kommer fra enten fjær eller den haptiske enheten.

3.5.1 Fjær

Det legges struktur fjær langs alle kantene i modellen, og kreftene som virker på partikkel **a** og **b** som er i endene av en slik fjær beregnes som følgende,

$$\mathbf{f_a} = \left[k_s(|\mathbf{l}| - r) + k_d \frac{\dot{\mathbf{l}} \cdot \mathbf{l}}{|\mathbf{l}|} \right] \frac{\mathbf{l}}{|\mathbf{l}|}, \mathbf{f_b} = -\mathbf{f_a}$$

hvor $\mathbf{f_a}$ og $\mathbf{f_b}$ er kreftene som virker på henholdsvis **a** og **b**. $\mathbf{l}$ er lengden mellom **a** og **b**, r er hvilelengden, k_s er stivheten på fjæren og k_d er dempningen. $\dot{\mathbf{l}}$

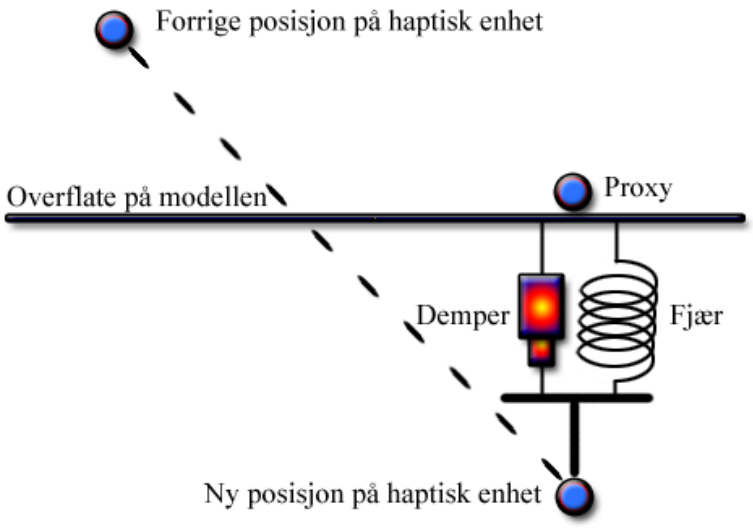
er den tidsderiverte av $\mathbf{l}$ som igjen bare er differansen i hastighet mellom partikkel **a** og **b**.

I tillegg til strukturfjær er det også blitt lagt inn "posisjoneringsfjær". Dette er fjær fra den opprinnelige posisjonen til en partikkel til den posisjonen den har på det aktuelle tidspunktet. Disse fjærene sørger for å holde modellen på plass slik at hvis man f.eks. trykker på modellen på en side så "driver" den ikke avsted. Kreftene som disse fjærene utøver beregnes på nesten samme måte som strukturfjærene, men her er hvilelengden null så man må ta spesielle hensyn for å hindre divisjon med null. Dersom $|\mathbf{l}|$ og $\mathbf{i}$ er under en terskelverdi gjøres ingen beregninger og partikkelen settes ganske enkelt tilbake til sin opprinnelige posisjon.

For å få en god haptisk tilbakemelding kreves det at den haptiske tråden har høy oppdateringsfrekvens. Siden den haptiske oppdateringshastigheten må være høy bør også simuleringshastigheten være høy, og som forklart i 2.2 fungerer eksplisitte metoder bra dersom tidsstegene ikke er for store. Derfor benyttes eksplisitte metoder for å løse systemet og de tre eksplisitte metodene: Euler, Midpoint og Runge-Kutta av grad 4 er alle implementert, men Runge-Kutta av grad 4 benyttes for å gi størst mulig nøyaktighet.

3.5.2 Påvirking av krefter (proxy metoden)

Når man "berører" modellen med Phantom enheten skal modellen "bule" inn på det stedet hvor man berører den og kuttes åpen dersom man trykker hardt. Dette betyr at nodene i nærheten av den haptiske posisjonen må påvirkes av krefter som er avhengige av hvor hardt man trykker. Denne kraften beregnes ved hjelp av proxy metoden. Figur 3.3 viser en illustrasjon av hvordan denne metoden fungerer. Proxyen er et punkt som følger den faktiske posisjonen til den haptiske enheten, men som alltid holdes på utsiden av modellen. Idet man berører modellen er som oftest den faktiske posisjonen inne i modellen mens proxy som sagt er på utsiden. Man trekker da en virtuell fjær med dempning mellom disse to posisjonene for å trekke den faktiske posisjonen ut av modellen. Det er denne kraften som gir den haptiske tilbakemeldingen (Man kjenner at man berører noe). Kraften denne fjæren utøver brukes i tillegg til å gi haptisk tilbakemelding også for å påvirke modellen. Kraften hentes ut og sendes (vektet) til punktene i den flaten man berører.



Figur 3.3: Proxy metoden

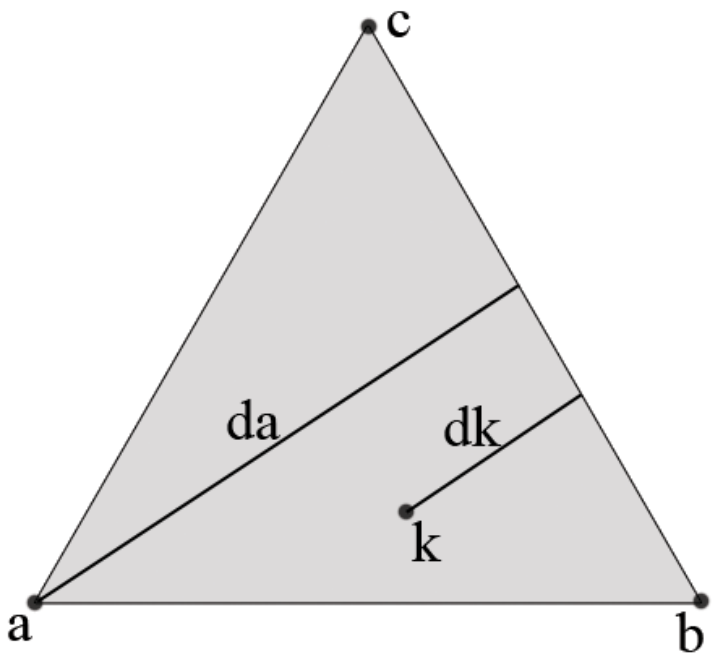
3.5.3 Vekting

Som sagt ovenfor så skal kraften som sendes til modellen vektes mellom punktene i berørt flate. Vekten til et punkt beregnes ved å se på lengden kollisjonspunktet har fra det aktuelle punktets motsatte kant i forhold til det aktuelle punktets avstand til den samme kanten. Formelen nedenfor sammen med figur 3.4 viser hvordan dette forholdet **x** blir beregnet for punkt **a** i et triangel. **k** er kollisjonspunktet i trianglet og **da** og **dk** er avstanden fra henholdsvis **a** og **k** til den motstående kanten til **a**.

$$x = \frac{dk}{da} = \frac{\left(\frac{|\overrightarrow{bc} \times \overrightarrow{kb}|}{|\overrightarrow{bc}|}\right)}{\left(\frac{|\overrightarrow{bc} \times \overrightarrow{ab}|}{|\overrightarrow{bc}|}\right)} = \left(\frac{|\overrightarrow{bc} \times \overrightarrow{kb}|}{|\overrightarrow{bc} \times \overrightarrow{ab}|}\right)$$

Dersom lengden er lik maks lengde, d.v.s. like lang som lengden fra det aktuelle punktet til den motstående kanten ($dk = da$), blir vekten lik 1. Dersom lengden er 0 ($\frac{0}{da}$) blir vekten 0. Vektingen mellom disse ytterpunktene kan kalkuleres ved å benytte ulike funksjoner. Enkleste løsning er å benytte lineær interpolasjon som fås direkte fra forholdstallet x, men dette gjør at man kjenner overgangen mellom flater godt, så andre løsninger som eksponentiell eller en interpolasjon basert på en cosinus funksjon er bedre. I programmet har jeg valgt å benytte eksponentiell interpolasjon da dette gir

den glatteste overgangen (vekten = x^2).



Figur 3.4: Vekting av krefter

3.6 Kutting

Dersom man trykker "hardt" mot modellen med skalpellen skal man kunne kutte i den. Modellen skal da åpnes langs den retningen man skjærer. Som nevnt i 2.3 er det mange ulike teknikker for hvordan kutting i modellen utføres. I dette prosjektet ble det valgt en løsning hvor generering av nye primitiver er lovlig. Kuttet åpnes fortløpende, d.v.s. at idet en ny flate møtes og et nytt kuttepunkt settes inn så åpnes det forrige kuttepunktet. Får å representere verktøyet benyttes ett enkelt punkt, og all kollisjonssjekking foregår mot dette. Flere ulike steg er nødvendige for å implementere den ønskede kutteteknikken. Det må utvikles metoder for å sette inn nye punkt i flater som beskjæres og nye punkt på kanter som kuttes over. I tillegg til dette må metoder som åpner kuttene, d.v.s. at punktene på hver side av et kutt flyttes fra hverandre, implementeres.

3.6.1 Kutting i flate

Når man trykker så hardt mot modellen at en terskelverdi overskrides starter kuttingen. Idet kuttingen startes må et punkt plasseres på det stedet man trykker, og punktet registreres som første punkt i kuttingen. Dersom man trykker nært et eksisterende punkt i modellen bruker man dette punktet som startpunkt, men dersom man ikke er i nærheten av et eksisterende punkt må et nytt punkt settes inn i den flaten man berører, og flaten blir dermed delt i tre mindre flater. Modellen er deformert når man kutter i den (buler inn), og eksakt kollisjonspunkt beregnes mot den opprinnelige udeformerte modellen. Eksakt kollisjonspunkt $\mathbf{k}'$ beregnes på følgende måte:

$$\begin{aligned} \overrightarrow{R} &= k - a \\ d &= \overrightarrow{R} \cdot \overrightarrow{N} \\ k' &= k - (\overrightarrow{N} \cdot d) \end{aligned}$$

hvor **R** er en vektor fra et hjørne i flaten **a** til kollisjonspunktet **k**, **N** er normalvektoren til den udeformerte flaten. Ved innsetting av ett nytt punkt må det også legges strukturfjær fra punktet til dets nabopunkt, samt posisjoneringsfjær på punktet selv. Det er for at disse fjærene skal få riktig lengde man må beregne eksakt kollisjonspunkt i den uderformerte flaten som forklart ovenfor. Massen til det nye punktet beregnes ved at man summerer massen til de tre punktene i flaten som kuttes i for så å fordele denne massen over de tre punktene og det nye punktet. Modellen benytter seg av teksturer for å bedre den visuelle kvaliteten, se 3.7 for mer om dette, og derfor må det nye punktet få tekstur koordinater. Det nye punktet får teksturkoordinater som beregnes fra kollisjonspunkt samt koordinatene på punktene i kollisjonsflaten. Flaten som kuttes projiseres ned i det planet hvor flaten får størst areal (det planet hvor normalvektoren peker). Figur 3.5 viser en flate som er projisert ned i dette planet. Aksene døpes om til s og t for å forhindre forvirrelse da dette i virkeligheten er xy, yz eller xz planet. Teksturkoordinatene beregnes på følgende måte:

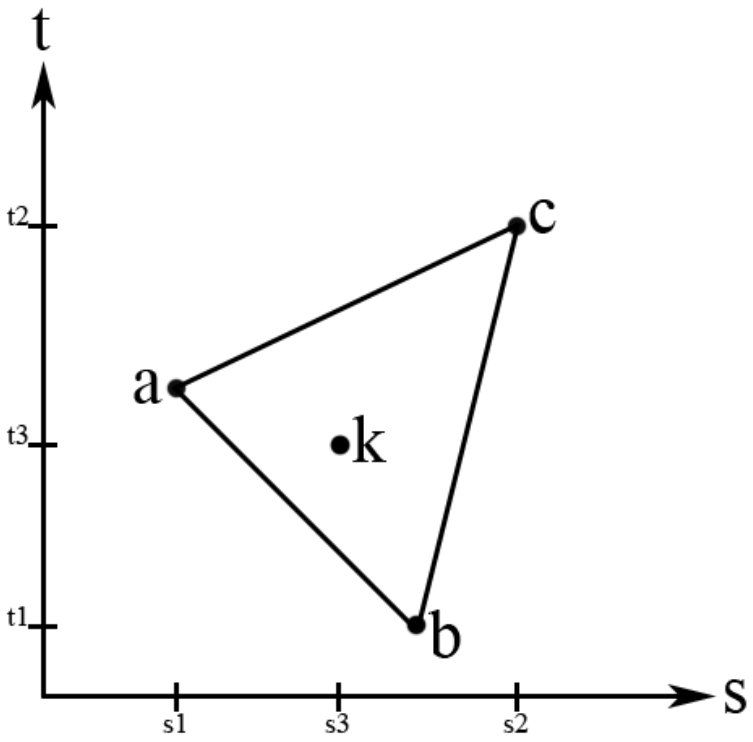
$$\begin{aligned} ks &= \frac{s3 - s1}{s2 - s1} \\ kt &= \frac{t3 - t1}{t2 - t1} \end{aligned}$$

hvor **s1** er den minste **s** verdien blant de tre punktene **a**, **b** og **c** langs **s** aksen. **s2** er den største **s** verdien og **s3** er **s** verdien for kollisjonspunktet. **t** verdiene er likens, men da for **t** aksen. **ks** vil da inneholde relativ

posisjon for kollisjonspunktet langs **s** aksen, mens **kt** inneholder relativ posisjon langs **t** aksen. Disse relative posisjonene brukes så slik for å beregne teksturkoordinatene **u** og **v** til det nye punktet:

$$\begin{aligned} u &= u1 + ks(u2 - u1) \\ v &= v1 + kt(v2 - v1) \end{aligned}$$

u1 og **v1** er minste verdiene for teksturkoordinatene for de tre punktene, mens **u2** og **v2** er de største. Etter at teksturkoordinatene er beregnet lages de nye flatene og naboflatene deres og naboflatene omkring den opprinnelige flaten får beregnet sitt nye naboskap. Det siste som gjøres er å ta vare på naboflatene til kuttepunktet.



Figur 3.5: Beregning av teksturkoordinater

3.6.2 Kutting av kant mellom to flater

Så lenge man berører modellen søkes det etter hvilken flate man berører, og dersom kutting er startet og man berører en flate som ikke er nabo til forrige kuttepunktet betyr det at man skal kutte en kant. Denne metoden er også slik at man benytter eksisterende punkt dersom man kutter nært et eksisterende punkt. Dersom dette ikke er tilfellet settes et nytt punkt inn på kanten. Siden man ikke kan hente ut posisjon akkurat idet man er på kanten, men henter posisjon idet man har kommet inn i en ny flate, må punktet hvor kanten ble kuttet beregnes. For å beregne kutte punkt $\mathbf{p'}$ som er det punktet på kanten mellom punkt **a** og **b** som er nærmest nåværende posisjon **p** benyttes følgende formel:

$$\begin{aligned} u &= \frac{\overrightarrow{ab} \cdot \overrightarrow{ap}}{\overrightarrow{ab} \cdot \overrightarrow{ab}} \\ p' &= a + u \cdot \overrightarrow{ab} \end{aligned}$$

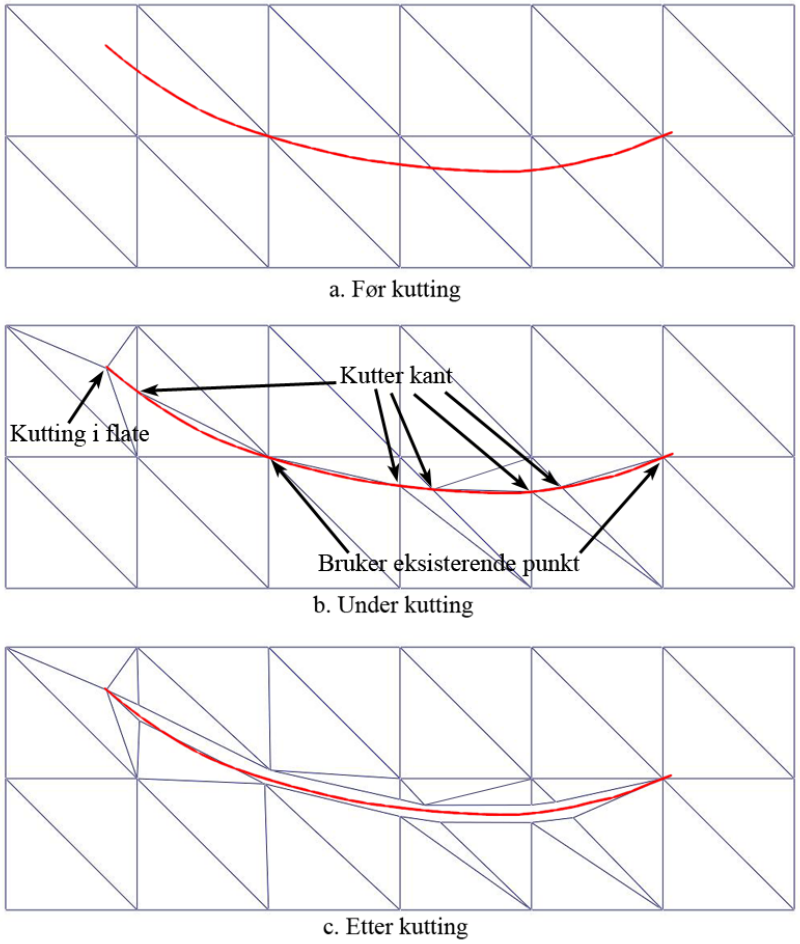
u er en verdi som ligger mellom 0 og 1 og den sier hvor langt på linjen mellom **a** og **b** det nye punktet skal befinne seg. Strukturfjæren som ligger på kanten som kuttes fjærens og nye strukturfjær fra det nye punktet til dets naboer settes inn. Punktet blir også koblet til en posisjoneringsfjær. Når ett nytt punkt settes inn på en kant må man også beregne teksturkoordinatene til dette punktet. Dette gjøres ganske enkelt ved å benytte den samme **u** verdien man beregner for å finne kollisjonspunktet ovenfor. Den nye teksturkoordinaten blir da teksturkoordinaten til punkt **a** pluss en del (**u**) av "teksturvektoren" fra **a** til **b**.

3.6.3 Åpning av kuttet

Når minst tre punkt er kuttet gjennom skal kuttet åpnes ved at flatene på hver side av kuttet flyttes fra hverandre. Alle flater som ligger på venstre side av kuttet (i forhold til kutteretningen), og har et av hjørnene sine i det nest siste kuttepunktet skal tilordens ett nytt punkt i stedet for kuttepunktet. Det nye punktet plasseres på samme sted som kuttepunktet og posisjoneringsfjær kobles til det. Massen til kuttepunktet blir delt med det nye punktet. Strukturfjær som ligger langs kantene på flatene på venstre side og er koblet til kuttepunktet kobles i stedet til det nye punktet. Etter dette strammes alle fjær som ligger fra kuttepunktet til andre punkt, og fra det nye punktet til andre punkt, bortsett fra fjær som ligger langs kuttekantene. Oppstrammingen består i at hvilelengden på fjærene forkortes med en 10%. Dette sørger for at modellen åpnes fordi fjærene trekker seg sammen etter

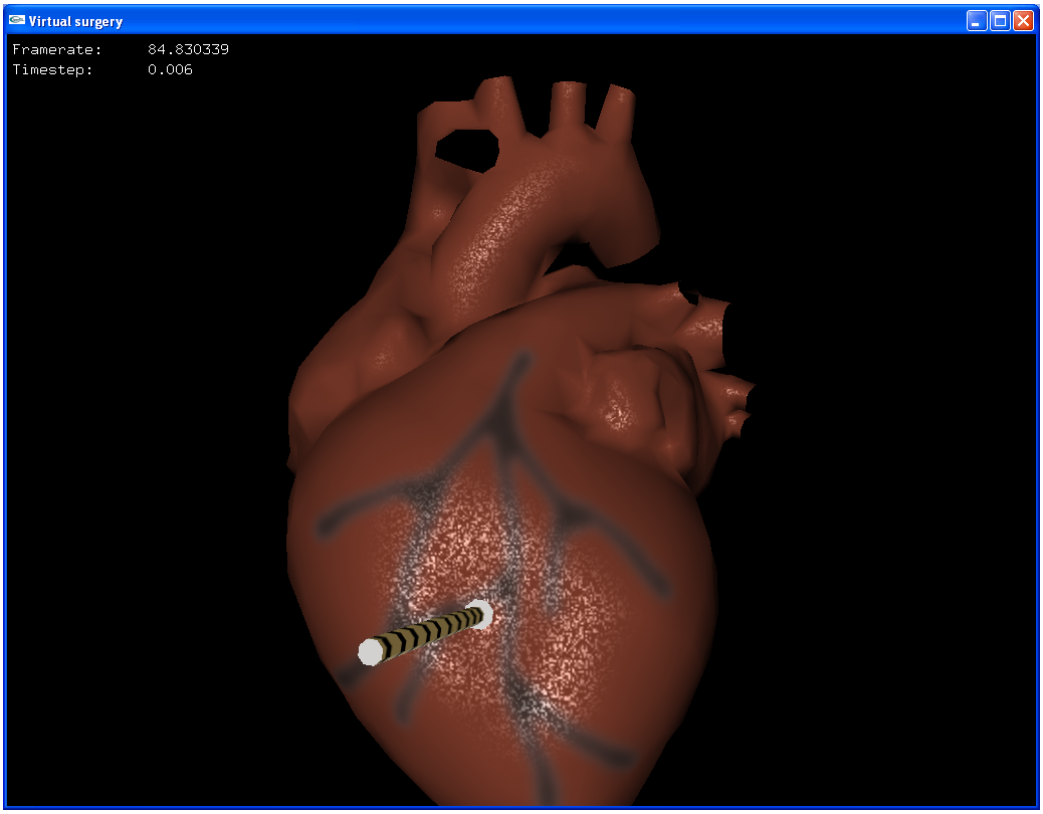
hvert som simuleringen fortsetter, og kuttet blir synlig. Kuttingen fortsetter fortløpende etter man har kuttet minst tre punkt, hvor det hele tiden er det nest siste punktet som skal åpnes.

Figur 3.6 viser en illustrasjon av hvordan kuttingen i overflaten foregår. 3.6a viser overflaten før kuttingen starter. Overflaten kuttes langs den røde linjen fra venstre mot høyre. 3.6b viser overflaten etter kutting, men uten at kuttet er åpnet. Dette bildet er ikke helt riktig da kuttet åpnes fortløpende under kutting, men det viser hvordan nye punkt har blitt satt inn i flater, kanter og gjenbruk av eksisterende punkt ved kutting nært disse. Til slutt i fig. 3.6c vises overflaten etter at kuttet er åpnet.



Figur 3.6: Kutting i overflate

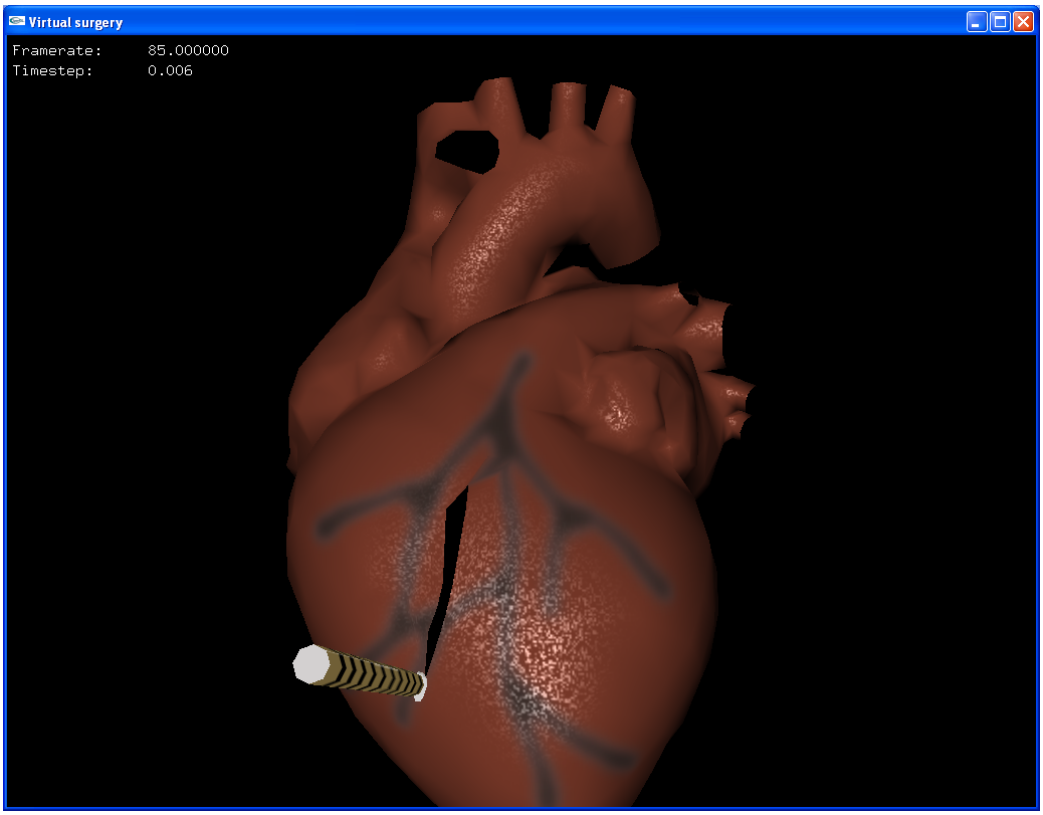
Følgende to figurer viser hvordan modellen påvirkes når den deformeres og kuttes. Figur 3.7 viser modellen når man trykker på den, mens figur 3.8 viser modellen etter litt kutting i den.



Figur 3.7: Deformering av overflate

3.7 Visualisering

For å tegne modellen til skjermen er det flere ulike teknikker som er nødvendige og ønskelige. Som nevnt tidligere benyttes OpenGL for å tegne modellen på skjermen, men det finnes ulike teknikker som kan benyttes for å øke det visuelle inntrykket. Her følger en nærmere forklaring av teknikken som er benyttet for forbedre utseendet på modellen samt stereo teknikken som er benyttet for å gi en bedre dybdefølelse i bildet.



Figur 3.8: Kutting i overflate

3.7.1 Forbedre utseende

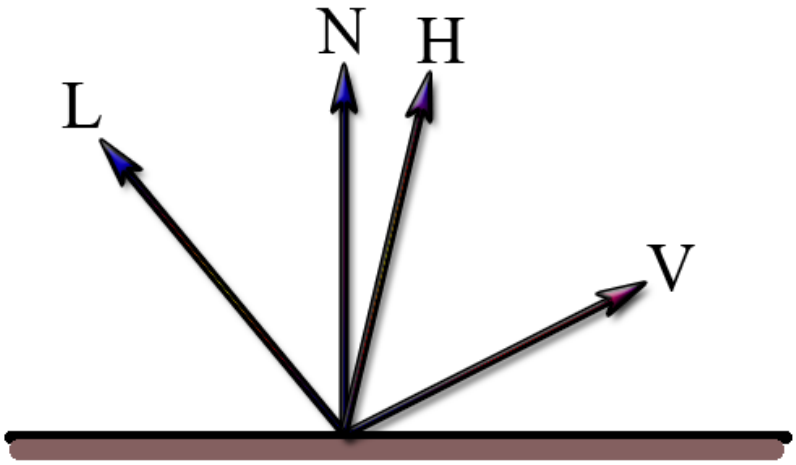
Nyere skjermkort har i tillegg til å bli stadig raskere fått muligheter som gjør at man kan skrive kode som kjøres direkte på skjermkortet. Dette gjør at man kan benytte OpenGL til translering, rotering, rastrering, osv. men i tillegg til dette kan det skrives en egen kode som erstatter den bestemte belysningsmodellen i OpenGL. Cg(C for graphics)/HLSL(High Level Shading Language) er programmeringsspråk som gjør at man enkelt kan skrive kode som kjøres av skjermkortet. Cg/HLSL er utviklet av Nvidia[22] og Microsoft[23]. HLSL er Microsoft's implementasjon av språket og er en del av deres DirectX API. HLSL kan kun brukes sammen med DirectX mens Cg er laget for å være plattformuavhengig og kan brukes sammen med både DirectX og OpenGL. I dette prosjektet som benytter OpenGL for grafikk er det derfor Cg som er brukt som programmeringsspråk mot skjermkortet.

For å forbedre det visuelle bildet ble en enkel belysningsmodell (basert på Blinn's modell) implementert i Cg. I tillegg ble det benyttet to ulike teksturer hvor det ene kobles til diffus fargen og den andre kobles til den glinsende fargen til materialet på modellen. Figur 3.9 viser de ulike vektorene Cg programmet må ha for å beregne fargen på et punkt i modellen. **L** er en vektor som peker mot lyskilden, **V** peker mot kamera, **N** er normalvektoren på det gitte punktet og **H** er en midtveis vektor mellom **L** og **V**. I tillegg til disse sendes det inn en verdi som sier noe om hvor glinsende materialet er. Når programmet kjenner disse vektorene beregnes farge på punktet på følgende måte:

$$\begin{aligned} I &= I_{diff} + I_{spec} \\ I &= I_l(k_d(\mathbf{N} \cdot \mathbf{L}) + k_s(\mathbf{N} \cdot \mathbf{H})^{shininess}) \end{aligned}$$

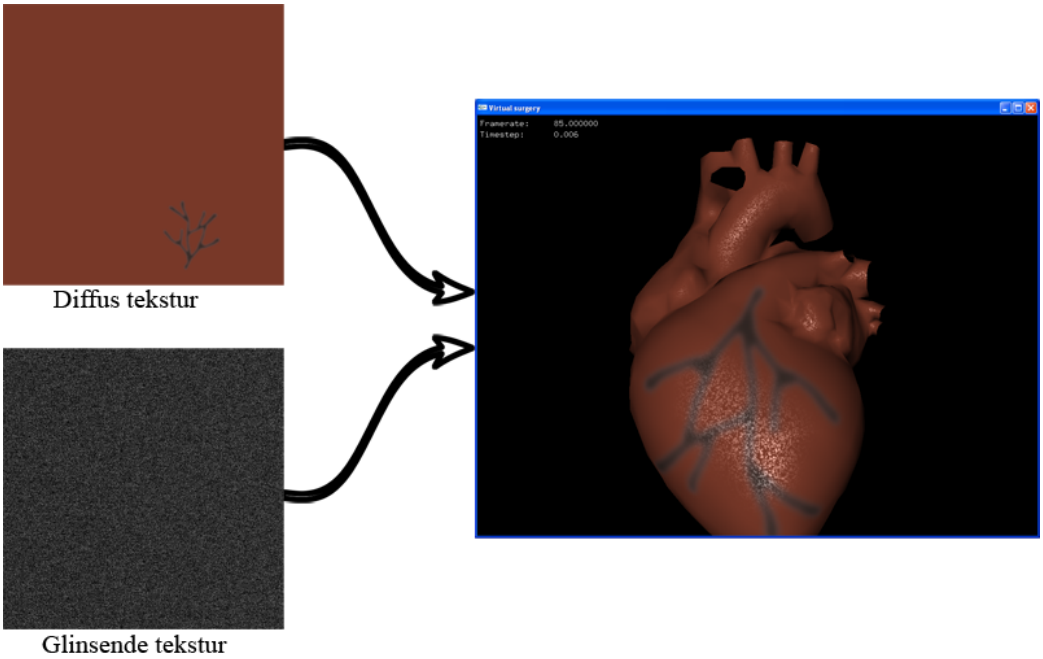
hvor I er resultat farge på punktet, I_l er fargen på lyset, k_d og k_s er henholdsvis diffus og glinsende farge som hentes fra hver sin tekstur ved hjelp av tekstur koordinater og *shininess* sier noe om hvor glinsende overflaten er.

Som diffus farge benyttes en enkel rød farge men noen blodårer inntegnet for å etterlikne fargen på ett hjerte. Denne teksturen kunne gjerne blitt erstattet med en tekstur basert på ett virkelig hjerte som viste blodårer og lignende for økt realisme. Den glinsende teksturen er ett svarthvitt bilde hvor "fargen" sier hvilken farge som reflekteres på det gitte punktet. Dersom fargen er sort blir det ingen refleksjon og materialet får bare det diffuse bidraget, men dersom fargen er hvit vil det gi et hvitt høylys dersom vinkelen mellom normalvektoren **N** og halvveisvektoren **H** ikke er for stor. En tekstur med



Figur 3.9: Lysmodell

støy ble benyttet for å etterligne et vått materiale. Figur 3.10 viser disse to teksturene og hvordan resultatet blir når man bruker de på hjerte modellen.

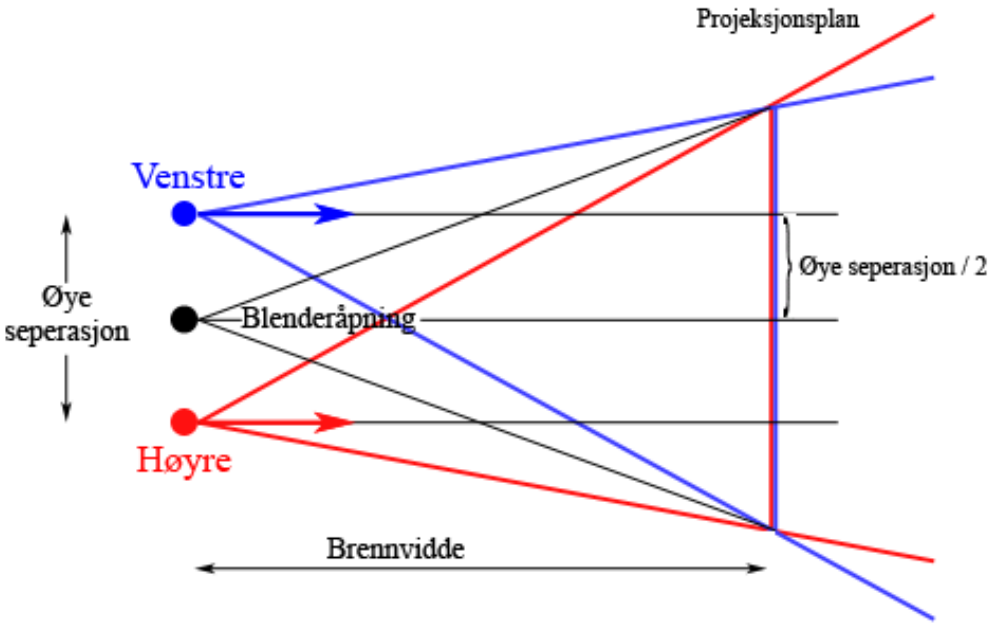


Figur 3.10: Bilde av hjerte med diffus og speilende tekstur

3.7.2 Stereo

For å gi brukeren en bedre dybdefølelse har programmet mulighet for å kjøre i stereo modus. For at stereomodus skal virke må maskinvaren støtte dette og man må ha stereobriller med tilhørende sender. Ved oppstart av programmet får brukeren to valg, hvor det første er om programmet skal kjøres i stereo eller ikke, og det andre er om programmet skal kjøres i fullskjerm eller i et vindu.

En vanlig feil ved bruk av stereo er å benytte "toe-in" metoden. Det er en metode hvor de to kameraene (venstre og høyre) flyttes litt fra hverandre, men de ser mot samme punkt. Denne metoden er geometrisk feil og fører til vertikal skjevinnstilling mellom stereopar som igjen kan føre til at det blir ukomfortabelt å se på scenen. En bedre teknikk som også er geometrisk riktig er en stereoteknikk som kalles "parallell akse usymmetrisk frustum perspektiv projeksjon"[24]. Denne teknikken blir benyttet i programmet og dette er en teknikk hvor de to bildene dannes ved å flytte sentret av perspektivprojeksjonen (kameraet) relativt i forhold til den andre perspektivprojeksjonen. Projeksjons aksene er parallelle, mens frustum gjøres usymmetrisk slik at kameraets endelige projeksjon viser mer av scenenbildet på den ene siden sin akse enn den andre. Venstre kamera viser mer av høyre side av scenen, mens høyre kamera viser mer av venstre side. Dette gjør at projeksjonsplanene faller sammen og det blir ingen skjevinnstilling mellom stereoparene. Figur 3.11 viser en illustrasjon av denne teknikken.



Figur 3.11: Parallell akse usymmetrisk frustum perspektiv projeksjon

Kapittel 4

Testing

For å teste programmet har jeg benyttet en Intel Pentium 4 Xeon 3,2GHz maskin med 2,0 GB minne. Maskinen har et Nvidia Quadro FX 3400 skjermkort med 256 MB minne og stereo muligheter. Her følger noen resultater fra ulike tester som er utført.

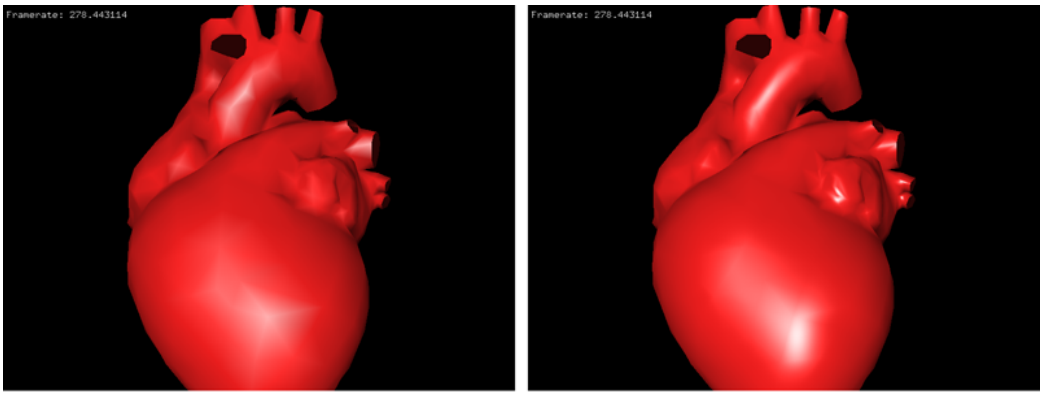
4.1 Oppdateringshastighet

For å teste programmets oppdateringshastighet ble to ulike modeller av hjertet benyttet. Oppløsningen ble variert og programmet ble testet i både stereo modus og vanlig modus i både vindu og fullskjerm. Under følger en tabell som viser resultatene av testen.

		Antall bilder i sekundet			
		Uten stereo		Med stereo	
Oppløsning	Antall triangler	Vindu	Fullskjerm	Vindu	Fullskjerm
800x600	1619	255	255	210	210
1024x768	1619	255	255	206	208
1280x1024	1619	255	255	125	126
1600x1200	1619	230	220	95	93
800x600	9714	52	53	44	44
1024x768	9714	52	53	44	46
1280x1024	9714	53	53	43	43
1600x1200	9714	35	50	28	39

4.2 Visualisering

Figur 4.1 vises en sammenlikning av hvordan utseende blir forandret ved å endre beregningen av belysningsmodellen fra per punkt i modellen (per-vertex) til hvert punkt i resultatbildet(per-pixel). Som kan sees av figuren får man mye bedre resultat når belysningen beregnes for hvert punkt i resultatbildet.



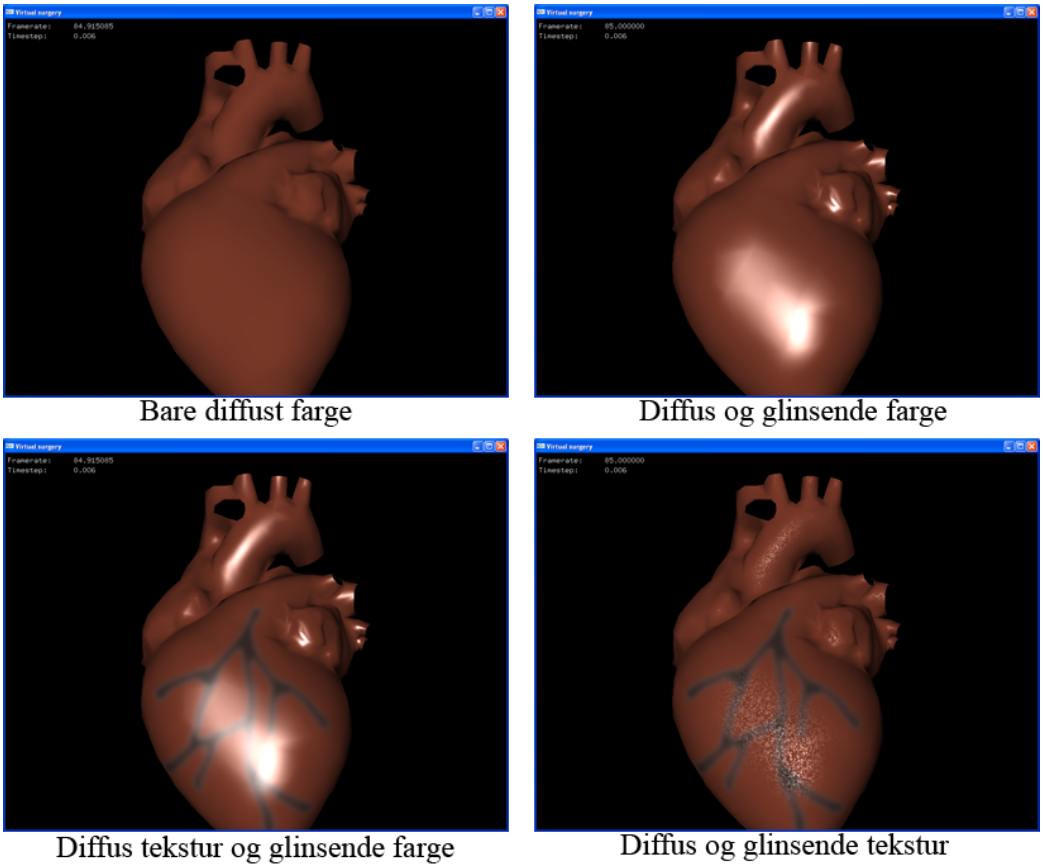
a. Per-Vertex b. Per-Pixel

Figur 4.1: Beregning av belysningen på modellen

Figur 4.2 viser hvordan modellen blir seende ut ved ulike kombinasjoner av diffus og glinsende farge og tekstur.

4.3 Numerisk løser

Tre ulike numeriske løsere er implementert, men Runge-Kutta av grad 4 benyttes da den gir det mest nøyaktige resultatet. Alle tre fungerer og ved å teste de på hjertemodellen er det ingen synlig forskjell på resultatet de produserer. Oppdateringshastigheten på programmet påvirkes derimot ganske sterkt av hvilken løser man benytter. Tabellen under viser hvordan antall bilder i sekundet påvirkes ved å benytte de ulike løserne. Oppløsningen i denne testen er 1024x768 og programmet kjøres på modellen med 1619 triangler. Det brukes ikke stereo, og programmet kjøres i et vindu.



Figur 4.2: Ulike kombinasjoner av diffus og glinsende farge og tekstur

Løser	Antall bilder i sekundet
Euler	463
Midpoint	372
Runge-Kutta	255

Som kan sees ut fra resultatene er oppdateringshastigheten over 80 % høyere ved bruk av Euler i forhold til Runge-Kutta, men siden det ved bruk av Runge-Kutta likevel er såpass høy oppdateringshastighet benyttes som sagt denne.

Kapittel 5

Oppsummering og konklusjon

Jeg har i dette prosjektet tatt for meg manipulering av datamodell som etterligner et organisk materiale. Modellen er bygget rundt et masse-fjær-system for å gi den fysiske egenskaper, og manipulering av modellen gjøres ved hjelp av en Phantom haptisk enhet. En del av arbeidet har vært å se på tidligere arbeider innenfor samme område, men da dette var hovedfokus ved forprosjektet er mye av arbeidet i denne oppgaven ligget i implementering av de ulike teknikkene.

Prototypen som er implementert laster inn modeller lagret i et egetutviklet filformat. Dette filformatet inneholder all informasjon om den geometriske oppbygningen av modellen samt konstanter som beskriver de fysiske egenskapene til modellen. All informasjon om de virtuelle verktøyene er også lagret i det samme filformatet. Et program for å konvertere modeller fra 3D modellerings/animerings verktøyet Maya til mitt eget format er utviklet.

Prototypen benytter Cg for å kjøre kode direkte på skjermkortet, og en enkel belysningsmodell basert på Blinns belysningsmodell er implementert. Diffus og glinsende teksturer er benyttet for å øke realismen på visualiseringen.

Arbeidet med prosjektet har foregått forholdsvis smertefritt. Noe tid har gått med til å sette seg inn i de ulike API som er benyttet under prosjektet, (Maya API, OpenHaptics og OpenGL). Samt å sette seg inn i programmeringsspråket Cg og hvordan man benytter dette i OpenGL programmer.

Arbeidet med prosjektet har vært krevende, men gitt meg mye god kunnskap om hva som kreves for å lage en god kirurgisk simulator. Både når det gjelder teknikker for å gi god haptisk tilbakemelding og visuell kvalitet. Prototypen

har muligheter for både kutting og manipulering av den virtuelle modellen. Brukeren har to ulike verktøy som gir ulik interaksjon når man jobber på modellen.

Kapittel 6

Videre arbeid

Jeg har i dette prosjektet hatt fokus på å utvikle en kirurgisk simulator hvor man ved hjelp av en haptisk enhet skal kunne manipulere på organiske modeller. Prototypen er i stand til å utføre slik manipulasjon på modellen. Brukeren kan deformere modellen ved å trykke eller kutte i den, og den haptiske og visuelle tilbakemeldingen er på plass. Det er likevel flere mulige forbedringer både på den haptiske og den visuelle delen. Jeg tar her for meg noen mulige forbedringer som vil øke realismen i simulatoren.

6.1 Haptisk rendring

Den haptiske enheten følger i programmet overflaten på modellen, noe som både ser og føles naturlig når man trykker på modellen, men når man skjærer i modellen hadde det vært bedre om den haptiske enheten hadde vært nede/inne i modellen. Dette er ikke mulig når man benytter høynivådelen av OpenHaptics da denne benytter proxy metoden 3.5.2, så en overgang til å bruke lavnivådelen kunne vær ønskelig for å få bedre kontroll over den haptiske enheten. Ved en overgang til lavnivå må man selv beregne all kollisjon mot modellen og har dermed en mye større fleksibilitet til å beregne krefter på og fra armen i simuleringen.

6.2 Deformering av modellen

Som man kan se av resultatene i 4 går hastigheten på programmet drastisk ned ved bruk av en detaljert modell. En lokal simulering vil være en mulig teknikk som kan forbedre dette. Som nevnt i 2.1 benytter Mosegaard[5] seg av en slik teknikk.

6.3 Visualisering

For å bedre den visuelle kvaliteten er det flere teknikker som kunne vært interessante.

Som nevnt tidligere i rapporten hadde det vært ønskelig med en tekstur som hadde vært basert på et bilde av et virkelig hjerte. En slik tekstur kunne inneholdt blodårer og lignende detaljer som finnes på et hjerte for å gi økt realisme.

Normalavbildning (normalmapping) er en teknikk hvor en tekstur som beskriver normalretningene på modellen benyttes for å øke detaljene i visualiseringen uten å måtte øke detaljnivået på modellen. Teksturen legges på modellen ved hjelp av teksturkoordinater på samme måte som en vanlig tekstur. Teksturen tegnes ikke til skjermen, men brukes til å beskrive hvilken vei normalen peker. Normalretningen som hentes fra normalteksturen brukes sammen med lysretningen for å beregne farge/belysning på modellen.

En annen interessant teknikk er å bruke omgivelsesavbildning (environmentmapping), en teknikk som gjør at objekter på en enkel måte kan reflektere omgivelsene i det området de befinner seg. En omgivelsesavbildning kan bestå av 6 teksturer som settes sammen i en kube, og under tegning av objektet beregnes refleksjonsvektoren på et punkt på modellen ved hjelp av synsvinkelen og normalretningen på dette punktet. Ved hjelp av refleksjonsvinkelen finnes punktet i kuben som skjæres og fargen på teksturen på det gitte stedet brukes som reflekterende farge på modellen. Dette er en teknikk som kunne vært interessant å bruke på de virtuelle verktøyene da disse er ganske reflekterende i virkeligheten.

Tillegg A

Programmet

Prototypen som er implementert kan kjøres ved å starte filen VS.exe som ligger på under mappen *Programmering/VS* på den vedlagte cd'n. For at programmet skal kunne kjøres må OpenHaptics være installert på maskinen og maskinen må ha et skjermkort med programmeringsmuligheter da Cg krever dette.

Programmet bruker hjertemodellen som ligger i filen *heart.txt* på mappen *Models* på cd'n. Verktøyene lastes inn fra filene *scalpel.txt* og *poke.txt* som ligger på samme mappe som hjertemodellen. Når programmet starter er *poke* valgt som verktøy, men brukeren kan bytte mellom dette verktøyet og *scalpel* med tastene *1* og *2*. Ved å tykke på *w* kan det velges mellom wireframe eller shaded tegning av modellen. Tastene 4, 5, 6 og 7 kan benyttes når programmet kjøres i stereomodus for å endre henholdsvis øyeseperasjon og brennvidde.

Hvilken modell som brukes, og hvilke verktøy som skal være tilgjengelige er skrevet direkte i koden (main.cpp), så for å endre dette må koden endres og kompileres/bygges på nytt. Microsoft Visual C++ 6.0 er benyttet under utviklingen.

Tillegg B

Filformat

Programmet som er utvikler henter den virtuelle modellen og de virtuelle verktøyene fra filer. Filformatet på disse filene er et egetutviklet format som inneholder all informasjon programmet trenger. Her følger en beskrivelse av formatet, samt forskjellene mellom filformatet til verktøyene og modellen.

Filformat for virtuell modell.

Det første som kommer i filen skal være teksten physics. Denne linjen må legges inn manuelt etter at en modell er konvertert fra Maya's filformat til dette filformatet. Under denne linjen skal modellens fysiske egenskaper beskrives med fem ulike desimaltall. Det første tallet er massen på modellen, de to neste er stivhet- og dempningkonstantene til strukturfjærene. De to siste er stivhet- og dempningkonstantene til posisjoneringsfjærene. All data etter dette skrives til filen når den konverteres. Først kommer linjen vertices med et påfølgende heltall. Dette tallet sier hvor mange punkt det er i modellen, og de neste linjene inneholder koordinatene til disse punktene. Etter alle punkt koordinatene følger linjen faces med påfølgende tall som sier hvor mange flater det er i modellen. Under denne ligger data om hvordan flatene er bygd opp. De tre første tallene er indekser til de tre hjørnepunktene som definerer flaten. Så følger seks tall hvor to og to tall er teksturkoordinater til hjørnene i flaten. Etter flate dataen følger normaler til flatene på liknende måte. Til slutt kommer kant data med indekser til punktene kantene består av. Under følger et lite utsnitt av filen til hjertemodellen som benyttes i programmet.

```
physics
200.000000 20.000000 2.000000 5.000000 1.000000
vertices 860
0.213016 0.040133 0.436831
```

```
...
faces 1619
2 1 0
0.429887 0.918213 0.419531 0.909077 0.425015 0.899331
...
normals 1619
-0.841427 0.136404 0.522872
...
edges 2498
2 0
...
```

Filformat for virtuelt verktøy.

Filformatet for de virtuelle verktøyene er ganske likt filformatet for modellene. Den eneste forskjellen er at linjen physics, med påfølgende data er byttet ut med en linje threshold med påfølgende desimaltall. Dette tallet sier noe om hvor hardt man må trykke mot modellen før kutting oppstår. Under følger et lite utsnitt av filen til skalpellen som benyttes i programmet.

```
threshold 4.000000
vertices 80
-0.000000 0.001170 0.172313
...
faces 156
1 2 0
0.278121 0.488404 0.252469 0.477779 0.278121 0.469953
...
normals 156
-0.000000 -0.000000 -1.000000
...
edges 234
0 1
...
```

Tillegg C

Maya eksportør

For å kunne bruke modeller laget i Maya må man bruke et lite program jeg har utviklet for å konvertere fra maya's filformat til mitt eget format. Programmet hetet MayaToVS.exe og er utviklet for Maya 6.0. Dersom en annen versjon av Maya benyttes må kildekoden kompileres på nytt til den aktuelle versjonen av Maya.

Programmet brukes ganske enkelt fra kommandolinjen og krever to argumenter. Det første argumentet skal være Maya filen som skal leses, og det andre argumentet er filnavnet hvor den konverterte filen lagres.

Eks:
MayaToVS heart.mb heart.txt
Her leses hjertemodellen fra filen *heart.mb* og lagres i det nye filformatet i filen *heart.txt*.

Tillegg D

CD

På cd'n ligger all kildekode og programmer som er utviklet til prosjektet. Både maya eksportøren og selve kirurgi simulatoren ligger på cd'n. Begge programmene er utviklet i Microsoft Visual C++ 6.0. Her følger en kort beskrivelse av innholdet på cd'n.

Mappe	Beskrivelse
Models	Inneholder hjertemodellen og de virtuelle verktøyene, både i maya og eget format, samt teksturene som benyttes
Programmering	Inneholder Kirurgisimulatoren (ligger i undermappen VS) og Maya eksportøren (ligger i undermappen MayaToVS)
Rapport	Inneholder denne rapporten

Bibliografi

[1] Dimitris Metaxas. Medical Image Modeling Tools and Applications. *Communications of the ACM, Februar*, 2005.

[2] Richard M. Satava. Current and Future Application of Virtual Reality for Medicine. *Proceedings of the IEEE, vol. 86*, pages 484–489, March 1998.

[3] Morten Bro-Nielsen. Finite Element Modeling in Surgery Simulation. *Proceedings of the IEEE*, 86(3):490–503, 1998.

[4] Geir Fagerholt and Stig Rune Søberg. Manipulering av Biologisk Modell ved hjelp av Phantom. *Forprosjekt høsten 2004, NTNU*, 2004.

[5] Jesper Mosegaard. LR-Spring Mass model for Cardiac Surgical Simulation. In *Proceedings of Medicine Meets Virtual Reality 12*, 2004.

[6] Hui Zhang, Shahram Payandeh, and John Dill. On Cutting and Dissection of Virtual Deformable Objects. *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference*, pages 3908–3913, April 2004.

[7] Herve Delingette and Nicholas Ayache. Hepatic Surgery Simulation. *Communications of the ACM, Februar*, 2005.

[8] Roger W. Webster, Dean I. Zimmerman, Betty J. Mohler, Michael G. Melkonian, and Randy S. Haluck. A Prototype Haptic Suturing Simulator. *Medicine Meets Virtual Reality*, 2001.

[9] David Baraff and Andrew Witkin. Physically Based Modeling. *Proceedings of ACM SIGGRAPH 2001*, 2001.

[10] Xavier Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour. In *Proceedings of Graphics Interface*, page 141, 1995.

[11] Tzvetomir Vassilev and Bernhard Spanlang. A Mass-Spring Model for Real Time Deformable Solids. *East-West-Vision*, September 2002.

[12] Cynthis D. Bruyns, Steven Senger, Anil Menon, Kevin Montgomery, Simon Wildermuth, and Richard Boyle. A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *Proceedings of Journal of Visualization and Computer Animation v13*, pages 21–42, 2002.

[13] Yi-Je Lim and Suvranu De. On the Use of Meshfree Methods and a Geometry Based Surgical Cutting Algorithm in Multimodal Medical Simulations. *Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2004.

[14] Hui Zhang, Shahram Payandeh, and John Dill. Simulation of Progressive Cutting on Surface Mesh Model. *DRAFT6-08*, September 2002.

[15] Cynthia D. Bruyns and Kevin Montgomery. Generalized Interactions Using Virtual Tools within the Spring Framework: Cutting. *Medicine Meets Virtual Reality (MMVR02)*, January 2002.

[16] Sensable. http://www.sensable.com.

[17] SGI. http://www.sgi.com/.

[18] 3DCafe. http://www.3dcafe.com.

[19] 3D Studio Max. http://www4.discreet.com/3dsmax/.

[20] Maya. http://www.alias.com.

[21] Grigore C. Burdea. Haptic Issues in Virtual Environments. *Proceedings of Computer International*, 2000.

[22] Nvidia. http://www.nvidia.com/.

[23] Microsoft. http://www.microsoft.com/.

[24] Bob Akka. Writing Stereoscopic Software for StereoGraphics System Using Microsoft Windows OpenGL. http://www.stereographics.com/support/developers/pcsdk.htm.