ICC-3D

"I SEE COLOR 3D"

"Visualisering av 3D fargerom"

Gruppemedlemmer:

- **Anders Rindal**
- **Arne Magnus Bakke**
- **Ståle Kopperud**

3DP: data - programmering

Hovedprosjekt ved Høgskolen i Gjøvik,
avdeling for teknologi,
våren 2002

Forord

Høsten 2001 begynte Anders Rindal, Arne Magnus Bakke og Ståle Kopperud med prosessen som skulle ende i et hovedprosjekt den etterfølgende våren. Alle studenter ved Høgskolen i Gjøvik (HiG) må siste semester gjennomføre et prosjekt på tilsammen 6 vekttall for å bli uteksaminert som høgskoleingeniører i data.

Etter grundige drøftinger kom vi fram til at det kun var en av oppgavene som appellerte til oss. Etter konsultasjoner med daværende kontaktperson Dr. Ivar Farup, ble det besluttet at vi skulle satse på dette prosjektet. Problemstillingen gikk i korthet ut på å visualisere fargerom i tre dimensjoner, noe som vi oppfattet som interessant og utfordrende. Vi fikk etterhvert større kontakt med Dr. Jon Yngve Hardeberg, vår oppdragsgiver. Resultatet av samarbeidet ble en applikasjon som dekker de problemene vi ble forespeilet, og denne rapporten beskriver dette arbeidet.

Vi ønsker å takke følgende personer

- Dr. Ivar Farup, som fulgte oss gjennom hele perioden som kontaktperson og veileder. Han var alltid tilgjengelig for spørsmål vedrørende praktiske problemer, og var en medvirkende årsak til at gruppemedlemmene beholdt inspirasjonen og entusiasmen.
- Dr. Jon Yngve Hardeberg, som var villig til å spille en aktiv rolle som oppdragsgiver. Hans fagkunnskap var avgjørende for at prosjektet i det hele tatt skulle komme i havn.

Gjøvik, 21 mai 2002

Anders Johan Rindal, Arne Magnus Bakke og Bo Ståle Erland Kopperud

Signaturer

Hovedprosjekt Visualisering av 3D-fargerom

ved Høgskolen i Gjøvik

_____	_____	_____
Anders Rindal	Arne Magnus Bakke	Ståle Kopperud

Innhold

1 Innledning **1**

1.1 Oppgavedefinisjon / problem / avgrensning 2

1.2 Prosjektorganisering 2

1.3 Målgruppe, rapport og oppgave / produkt 3

1.4 Formål / hvorfor dette emnet 4

1.5 Egen bakgrunn og kompetanse. Hva må læres 4

1.6 Utviklingsmodell 5

1.7 Arbeidsformer 7

1.8 Organisering av rapporten 8

1.9 Rapportlayout 9

2 Analyse / Kravspesifikasjon **11**

2.1 Kravspesifikasjon og forberedelser 11

2.2 Java3D vs OpenGL - Utdypning av analysefasen 14

2.3 Kildekode / kommentering 15

2.3.1 Versjonshåndtering 16

2.3.2 Web 16

2.4 Plattformuavhengighet 17

3 Design **18**

3.1 Sessjonshåndtering 18

3.2 Layout, style, fonter 18

3.2.1 Generelt 18

3.2.2 Vinduene 18

3.2.3 Ikoner 21

3.2.4 Fonter 21

3.3 Fargerom . . . 22
3.3.1 Struktur . . . 22
3.3.2 Aksesystem . . . 23
3.4 2D . . . 23
3.5 Dynamisk klasseloading og modularitet . . . 24
3.5.1 Generelt . . . 24
3.5.2 Virkemåte . . . 24
3.5.3 Struktur . . . 26
3.6 Interaksjoner . . . 27
3.6.1 Globale transformasjoner . . . 27
3.7 Visualiseringer . . . 28
3.8 Visualiseringer og interaksjoner - en studie i samhandling . . . 28
3.9 Begrensninger . . . 30
3.10 Utstyrsgamut . . . 31

4 Implementasjon / koding / produksjon 32
4.1 Grensesnitt / API . . . 32
4.1.1 Java . . . 32
4.1.2 Java3D . . . 33
4.1.3 JAI - Java Advanced Imaging . . . 36
4.1.4 JMF - Java Media Framework . . . 37
4.2 Plattformer . . . 37
4.2.1 Microsoft Windows . . . 37
4.2.2 Linux . . . 37
4.2.3 Apple Macintosh . . . 38
4.2.4 Foretrukket system . . . 38
4.3 Fargerom . . . 39
4.3.1 Generelt . . . 39
4.3.2 Utvidelse . . . 39
4.3.3 Konvertering av fargeverdier . . . 40
4.3.4 Formler / kode for konvertering . . . 41
4.3.4.1 sRGB $\rightarrow$ sRGB . . . 41
4.3.4.2 sRGB $\rightarrow$ CIEXYZ . . . 41
4.3.4.3 CIEXYZ $\rightarrow$ sRGB . . . 42

4.3.4.4 Lab→sRGB 44
4.3.4.5 Lab→CIEXYZ 44
4.3.4.6 CIEXYZ→Lab 44
4.3.4.7 CIEXYZ→sRGB 45
4.3.4.8 XYZ→CIEXYZ 45
4.3.4.9 CIEXYZ→XYZ 45
4.4 2D 46
4.5 Oppbygning av 3D struktur 46
4.5.1 Generelt 46
4.5.2 Struktur 47
4.5.3 Implementasjon 47
4.6 Parsere 48
4.6.1 Generelt 48
4.6.2 Filformater 49
4.6.2.1 IT8.7/2 - Parser.Java 49
4.6.2.2 Regulært - Parser2.Java 50
4.6.2.3 ICC profiler - IccReader.Java 51
4.6.3 Eksportering av data 52
4.7 Forandring av posisjon og zoom i 3D 53
4.8 Visning av aksesystemer i 3D 54
4.9 Visualiseringer 55
4.9.1 Standard 55
4.9.2 Kvantisert 56
4.9.3 Segment maxima 57
4.9.3.1 Algoritmen 58
4.9.3.2 Unøyaktighet 59
4.9.3.3 Problemer 59
4.9.3.4 Interpolering 60
4.9.3.5 Implementasjon 60
4.9.4 Convex Hull (3D) 61
4.9.5 Alpha Shapes 65
4.9.5.1 Delaunaytriangulering i 3D (ved flipmetoden) . . . 65
4.9.5.2 Delaunaytriangulering i 3D (via convex hull i 4D) . 67
4.9.5.3 Alpha Shapes fortsetter 68

4.10 Generell posisjon 70
4.11 Interaksjoner 70
4.11.1 Interaksjoner generelt. 70
4.11.2 Standard interaksjon 71
4.11.3 Segmentinteraksjon 71
4.11.4 Tetraederinterpolasjon 72
4.12 Globale transformasjoner 72
4.13 Begrensninger 74
4.14 Capture 75
4.15 Generell optimalisering av kode 77
4.16 Installasjon 80
4.16.1 Windows 81
4.16.2 Linux (x86) 81
4.16.3 InstallAnywhere 83

5 Testing / kvalitetssikring 85
5.1 XP testing 85
5.2 Backup 86

6 Sluttdiskusjoner 88
6.1 Drøftinger og diskusjoner underveis 88
6.2 Kritikk av oppgaven 89
6.3 Videre arbeid, nye (hoved)prosjekter 89
6.4 Evaluering av gruppens arbeid 89
6.4.1 Innledning 89
6.4.2 Organisering 90
6.4.3 Fordeling av arbeidet 90
6.4.4 Prosjekt som arbeidsform 90
6.4.5 Subjektiv opplevelse av hovedprosjektet 91

7 Konklusjon 92

Figurer

1.1 Utviklingsmodellen XP, eXtreme Programming. 5

3.1 ICC3D og alle dets vinduer. 20
3.2 Tittel i “2D Image” vinduet. 21
3.3 Tittel i hovedvinduet til ICC3D. 21
3.4 Dynamisk lasting av plugins 25

4.1 Eksempel på et BranchGroup subtre. 35
4.2 Eksempel på et “virtuelt univers”. 36
4.3 Kontrollpanel og valg tilknyttet fargerom. 39
4.4 Konvertering av fra sRGB til CIEXYZ fargerom. 42
4.5 Konvertering fra CIEXYZ til sRGB fargerom. 43
4.6 Arvehierarki for visualiseringer. 47
4.7 Kontrollpanel og valg tilknyttet gamut for IT7.8/2 formatet. 50
4.8 Kontrollpanel og valg tilknyttet regulært filformat. 50
4.9 Kontrollpanel og valg tilknyttet ICC profiler. 51
4.10 Organiseringen i en ICC profil. 52
4.11 Kontrollpanel og valg tilknyttet synsvinkel. 54
4.12 Kontrollpanel og valg tilknyttet visualiseringer. 55
4.13 Standard visualisering. 55
4.14 Kvantisert visualisering. 57
4.15 Segment maxima oppdeling 58
4.16 Segment maxima visualisering. 59
4.17 Convex hull visualisering. 62
4.18 Alpha shapes visualisering. Øverst: stor alphaverdi. Nederst: liten alphaverdi. 66
4.19 Kontrollpanel og valg tilknyttet constraints/begrensninger. 74

4.20 Kontrollpanel og valg tilknyttet capture. Stillbilde og animasjon. . . . 75
4.21 Utseende på installasjonsprogrammet 80
4.22 Installasjonsprogram laget av InstallAnywhere 83

Tabeller

3.2 Arvehierarki for fargerom. 23
3.4 Arvehierarki for klasser som lastes dynamisk 26

4.3 Funksjonsoppbygning av 3d struktur. 48
4.5 Arvehierarki for transformasjoner 73
4.7 Arvehierarki for begrensninger. 75

Kapittel 1

Innledning

Best mulig gjengivelse av farger på ulike enheter (skjermer/skrivere etc.) er et komplekst problem. Siden det er såpass store variasjoner i kvalitet og karakteristikk på de enkelte enhetene, ikke bare fra fabrikat til fabrikat men også fra monitorer av samme modell, er det vanskelig å opprettholde konsistent fremvisning av farger. Det å kunne gjengi et bilde likt på to forskjellige enheter er et stort problem. Grunnen til dette er bla. fordi en farge spesifisert på et visst format vil være utstyrsavhengig. Dvs. hvilke farger som faktisk kan gjengis av den enkelte enheten.

Det finnes flere metoder for å unngå dette problemet. En av dem er å kalibrere alle enheter til den samme standarden. Men problemer med lysstyrke- og kontrast-dynamikk (gamma) mellom for eksempel Mac og PC monitorer vil fortsatt være tilstede. En slik standard er ICC Color Profiles. Denne standarden er utviklet for å tilby et multiplattform format for å oversette fargedata fra en enhets fargerom til et annet. Algoritmer for å gjøre tilpasninger mellom ulike enheters fargerom finnes (color gamut mapping), men den automatiske tilpasningen blir ikke alltid like god. Til profesjonelle anvendelser er det derfor ofte behov for manuelle justeringer. Dette medfører et behov for å kunne visualisere et fargerom. Siden en farge ofte beskrives som et punkt i et tredimensjonalt rom, dreier det seg her altså om 3D-grafikk.

1.1 Oppgavedefinisjon / problem / avgrensning

Problemstillingen vår i denne sammenhengen er å forsøke å lage en løsning som gjør det mulig å representere bilder i forskjellige tredimensjonale fargerom, samt foreta justeringer mens man sammenligner bildets farger med det en eller flere enheter i form av for eksempel printere er i stand til å vise.

Et viktig aspekt ved oppgaven er at det skal utvikles et generelt og modulært rammeverk som skal være lett utvidbart og fortrinnsvis mest mulig plattformuavhengig. Dette gir sterke føringer på valg av utviklingsmiljø, samtidig som det er med på å prege den interne strukturingen av applikasjonens oppbygning. Det er viktig at gruppen klarer å holde fokus på hovedmålene underveis i prosjektet.

Vi konsentrerer oss i første omgang om å utvikle et stabilt, tilpasningsdyktig system som enklest mulig kan deles inn i mest mulig uavhengige deler. Kravene som skal tilfredsstilles med hensyn til funksjonalitet, blir ikke fastsatt før senere i prosjektet på grunn av prosjektets forskningsrelaterte natur. Det er imidlertid klart at programmet skal støtte flere ulike typer visning av bilder og enheters mulige fargesammensetning. I tråd med problemstillingen, må programmet også inkludere flere muligheter for at brukeren skal kunne flytte på objektene i 3D, slik at disse endringene også blir utført på bildet i 2D. Fordi operasjoner kan være enklere å utføre i andre fargerom enn det bildene benytter, må det være mulig å konvertere mellom forskjellige fargerom.

1.2 Prosjektorganisering

Oppdragsgiver: HiG v/ Jon Yngve Hardeberg

Veileder: Ivar Farup

Prosjektleder og webansvarlig: Anders Rindal

Øvrige gruppemedlemmer: Arne M. Bakke og Ståle Kopperud

Oppdragsgiver

Vår oppdragsgiver er HiG, i dette tilfellet representert ved Dr Jon Yngve Hardeberg.

Jon Yngve Hardeberg mottok sin sivilingeniørtittel i 1995 fra Norges Tekniske Høgskole i Trondheim. Der hadde han spesialisert seg innen signalprossesering. Han fullførte så sitt doktorgradsstudium ved Ecole Nationale Superieure des Telecommunications i Paris, Frankrike i februar 1999. Dette studiet omhandlet uthenting og reproduksjon av fargebilder fra programmer. Han jobbet deretter ved Conexant Systems Inc. som systemutvikler og fargespesialist.

Han har igjennom de senere årene utgitt mange publikasjoner innen fargereproduksjon og nærliggende emner.

For tiden er han ansatt ved HiG som førsteamanuensis, hvor han foreslo den problemstillingen som vi baserer vårt hovedprosjekt på.

For mer informasjon, se hjemmesiden på `http://www.hig.no/~jonh`.

Ansvarsforhold

Anders Rindal er gitt ansvaret som prosjektleder. Han vil være ansvarlig for innkalling til møter, virke som en kontaktperson mot gruppa i forhold til oppdragsgiver/veileder samt å holde gruppen oppdatert og motivert. Sistnevnte vil generelt gjelde for hele gruppa.

Alle deltakerene er innforstått med hvilket ansvar den enkelte har både for seg selv og for resten av gruppa i prosjektets løp. Hver enkelt vil være ansvarlig for å gjennomføre tildelte oppgaver innen avtalte tidsfrister. Dersom dette ikke er mulig, skal den enkelte gi beskjed om dette i relativt god tid slik at det vil være mulig for gruppen å løse problemet i fellesskap.

Det vil også bli brukt en logg for å dokumentere prosjektets fremgang/forløp. Ansvaret for loggføringen vil her ligge på den/de som har deltatt i aktiviteten. I tillegg til dette vil møtereferater inngå som en del av loggen.

Backup vil bli ivaretatt av Ståle Kopperud.

1.3 Målgruppe, rapport og oppgave / produkt

Dette prosjektet er av en relativt teknisk art, noe som vil ha konsekvenser for målguppen rapporten og produktet vil være egnet for.

Målgruppe

Produktet som dette prosjektet har resultert i, vil først og fremst være rettet mot personer som har kjennskap til og kunnskaper innenfor fagområdet farger og nærliggende emner. Grunnen til dette er fordi det vil være mange tekniske ord og uttrykk som en bruker bør kjenne til dersom han/hun skal få mest mulig utnyttelse av produktet. Dette gjør at ikke alle vil ha like stor glede av de mulighetene som ICC3D tilbyr.

Produktet tilbyr en rekke funksjoner som vil være høyst relevante for personer som jobber innenfor den grafiske bransjen. Personer som jobber med bildeproduksjon/reproduksjon og gjengivelse/testing av diverse enheters fargeområde. Sistnevnte vil være svært nyttig for å kunne avgjøre kvaliteten til en evt. skriver, printer eller monitor vha. en 3 dimensjonal visualisering av f.eks. gamutområdet. Manipulering av bilder for tilpasning til en bestemt profil vil også være en meget aktuell problemstilling som vil kunne løses vha. ICC3D.

Rapport

Rapporten vil også være av en mer teknisk art. Dette pga. de mange tekniske anliggendene som er knyttet til oppgaven. Det vil dog være fult mulig å få en grei oversikt da vi har forklart hva som menes med forskjellige aspekter rundt aktuelle felt/problemstillinger.

Oppgave

Problemstillingen i denne oppgaven vil være mer knyttet til et forskningsprosjekt i motsetning til mange andre typer prosjekter. Dels fordi det ikke finnes en klar og tydelig definisjon eller avgrensning av oppgaven, og fordi aktuelt tema innenfor fagområdet er relativt lite utforsket. Dette er også med på å avgrense målgruppen som vil ha mest nytte av produktet og dets funksjonalitet.

1.4 Formål / hvorfor dette emnet

Det å kunne reprodusere farger likt på flere forskjellige medier har vist seg å være en vanskelig oppgave. Det kan være kritisk for et bilde at informasjonen forandres når det vises på annet utstyr. Algoritmer som skal justere dette automatisk har vist seg å ikke fungere godt nok, og det er derfor nødvendig med et program der brukeren kan styre dett manuelt.

Å få ting til å se forholdsvis likt ut på utstyr med forskjellige utstyrsgamut kan være en tidkrevende og vanskelig oppgave. Man kan løse dette ved prøv-og-feil metoden, noe som ikke er optimalt. Ved å tilby brukeren et arbeidsmiljø i tre dimensjoner der både gamut og fargerommet vises samtidig, vil det kunne lette arbeidsmengden betraktelig.

1.5 Egen bakgrunn og kompetanse. Hva må læres

Vi er en gruppe bestående av tre studenter på dataingeniørlinjen på Høgskolen i Gjøvik. Alle tre har tatt fordypning innen programutvikling, og er interessert i programmering. Vi har også tidligere utviklet diverse små programmer som baserer seg på 3D i form av OpenGL, både i skolesammenheng og i fritiden. Dette vil være gunstig for prosjektet, da vi allerede kan en god del om 3D baserte strukturer og algoritmer, selv om vi i dette tilfellet ikke kommer til å benytte OpenGL. Det må også bemerkes at dette prosjektet er en stor oppgave, og det vil teste gruppens evne til å beholde oversikten og strukturere både data og arbeid.

Java er et programmeringsspråk vi har benyttet en god del før, først og fremst i skolesammenheng, men også på fritiden. Muligheten til å arbeide med et velkjent språk vil gi oss muligheten til å konsentrere oss om andre utfordringer. Dette vil likevel være et prosjekt hvor hoveddelen av tiden vil gå med til forarbeid og undersøkelser, på grunn av den meget forskningsrelaterte problemstillingen. Vi ser for oss mange vanskelige oppgaver og uforutsette problemer, og et prosjekt som vil preges av den praktiske siden av systemutvikling.

Java3D er relativt nytt, og med unntak av ett gruppemedlem har vi ingen erfaring med dette programmeringsbiblioteket. Siden det vil stå sentralt i den løsningen vi har tenkt å utvikle, vil det derfor være avgjørende for prosjektets resultat at alle setter seg inn i hvordan Java3D fungerer raskt. Forundersøkelser har antydet at dette ikke burde bli et problem, da det virker som om Java3D har visse likhetstrekk med OpenGL, selv om det skjuler en god del av den underliggende arkitekturen fra brukeren. Forhåpentligvis vil eksperimentering med verktøyet gi oss en pekepinn på de sterke sidene som kan

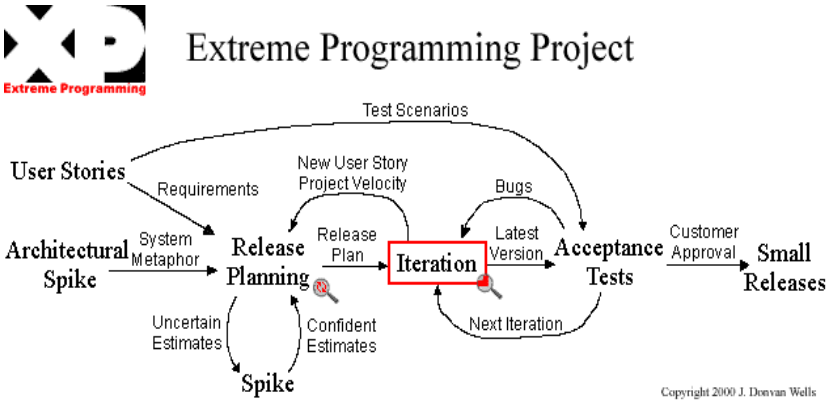
hjelpe oss med å traversere utviklingsfasen, i tillegg til de svakhetene som må unngås dersom dette overhodet er mulig.

Med hensyn til det endelige produktets arbeidsområde, er dette noe som er helt nytt for gruppen. Vi har ingen tidligere erfaring med denne typen bruk av bilde- eller utstyrsgamut, og vil derfor være avhengig av et nært samarbeid med veileder og oppdragsgiver for å forstå de oppgavene som skal utføres. Dette vil bli essensielt for prosjektets suksess, og er også med på å påvirke valget av den utviklingsmodellen vi kommer til å følge.

Anders Rindal, en av gruppens medlemmer, vil stå sentralt i arbeidet med utformingen av de nødvendige grafiske elementer i brukergrensesnittet. Hans tidligere erfaringer med 3D Studio Max, Photoshop og andre liknende verktøy vil være til stor nytte for gruppen. Ellers må det bemerkes at gruppens sammensetning (tre studenter fra programutviklingslinjen) er i samsvar med kontaktpersonens vurderinger før prosjektstart, som indikerte at prosjektet ville være utfordrende, med vekt på den praktiske siden av systemutvikling. Alle gruppemedlemmene ser frem til å benytte og utvide sine kunnskaper innen systemutvikling.

1.6 Utviklingsmodell

Som utviklingsmodell har vi valgt å bruke eXtreme Programming, XP.



Figur 1.1: Utviklingsmodellen XP, eXtreme Programming.

Årsakene til dette er at:

- prosjektet tilfredsstiller kravene til XP.
  - Liten prosjektgruppe
  - Engasjert kunde (oppdragsgiver) som deltar aktivt i prosjektet
  - Det er mulig (og hensiktsmessig) å følge de aller fleste av XPs 12 regler. (se underliggende punkt)
- prosjektet er av en slik karakter at XP er en fordelaktig systemutviklingsmodell.

- XP muliggjør dynamisk endring av kravspesifikasjon, eventuelt en situasjon der programmets funksjonalitet ikke er fullstendig definert ved prosjektstart. Bruk av iterasjoner der kunden bestemmer ny funksjonalitet fortløpende i prosjektet gjør utviklingsmodellen godt egnet til dette formålet.
- XP minimaliserer prosjektrisiko, noe som er spesielt nyttig dersom prosjektet er av nyskapende karakter eller mål og rammer er uklare.
- Økt produktivitet fordi utviklerne kontinuerlig får tilbakemelding fra kunden, og forstår systemets bruksområder og brukernes krav bedre.
- Legger vekt på å tilfredsstille kundens ønsker. Funksjonstester sørger for at applikasjonen oppfyller kravene fra kunden, slik at man unngår uenighet vedrørende forskjeller mellom kravspesifikasjonen og resultatet.

XP baserer seg på 12 hovedregler for hvordan man utvikler et prosjekt[1] :

1. Planleggingsprosessen: Kunden og utviklerne samarbeider for å oppnå resultater raskest mulig. Kunden bestemmer hva som skal bli gjort, mens utviklerne bestemmer hvordan det skal gjøres (implementasjonen). Kravene fra kunden spesifiseres i såkalte bruker-historier, dvs. krav til ønsket funksjonalitet i programmet. I tillegg skal utviklerne prøve å estimere tidsforbruket som går med til de ulike bruker-historiene, slik at kunden kan foreta en prioritering av aktivitetene.

2. Små utgivelser: Gi ut en kjørbar utgave av applikasjonen tidlig i utviklingsfasen, for deretter å bygge videre på denne. Etter dette gis det med små mellomrom ut nye versjoner, der hver utgivelse inneholder få nye funksjoner, som har blitt spesifisert av kunden.

3. System metafor: Beskrivelse av helheten i systemet, hvordan det fungerer.

4. Enkel design: Gjør ting så enkelt som mulig til en hver tid, uten å tenke på framtidige krav.

   Grunnet enkelte krav til applikasjonens oppbygning (f.eks modularitet), føler vi det er riktig å ta hensyn til de endringer som nødvendigvis må oppstå, selv om dette ikke alltid gir den enkleste løsningen i et kortsiktig perspektiv.

5. Kontinuerlig testing: Hver programdel skal testes grundig. Det lages små testprogrammer før hver ny implementasjon av kode. Disse skal gå feilfritt, ellers skal koden skrives om. Aksepteringstester lages for å teste hele programmet, eller større deler av det. Når alle aksepteringstestene godtas for en gitt brukerhistorie, defineres den som ferdig.

   Det er i vårt tilfelle hverken hensiktsmessig eller mulig å skrive automatiserte tester for alle nye brukerhistorier som blir implementert i prosjektet. Vi vil gjennomføre automatiserte tester der dette er hensiktsmessig, men ellers basere oss på veldefinerte aksepteringstester utarbeidet i samarbeid med kunden.

6. Restrukturering: Etter hvert som systemet vokser, vil det oppstå overflødig eller tungvinn kode. Denne må skrives om eller fjernes, samtidig som testene forsikrer deg om at det ikke vil oppstå feil.

[1] punkter som ikke er kommentert anser vi for å være gjennomførbare uten modifikasjoner

7. Parprogrammering: All programmering skal foregå i par, den ene skriver koden, mens den andre foretar strategiske vurderinger for neste del av kode som skal lages og verifiserer koden som blir skrevet.

8. Kollektivt eierskap av kode: Koden som blir skrevet skal kunne endres av samtlige personer i utiviklingsteamet.

   Endringer av kode hvis korrekthet ikke kan verifiseres ved hjelp av automatiserte tester vil kun forekomme etter konsultasjon med personen(e) som opprinnelig skrev den.

9. Kontinuerlig integrasjon: Alle forandringer blir integrert inn i kodebasen daglig, men ikke før alle testene fungerer (du skal ikke legge til noe som ødelegger for noe du allerede har gjort).

10. 40 timers uker: Ingen i utiviklingsteamet skal arbeide mer enn 40 timers uker. Det skal ikke forekomme at noen har to overtidsuker i rad. Skjer dette, tyder det på at noe er alvorlig galt med prosjektet.

11. Lett tilgjengelig kunde: Det skal være god kommunikasjon mellom utviklingsteamet og kunden gjennom hele prosjektets gang. Kunden må ha kunnskaper innen feltet, samt ha mulighet til å ta beslutninger vedrørende prosjektets framdrift. Han skal også hele tiden kunne svare på aktuelle spørsmål fra utviklerne.

12. Kodestandarder: All kode blir skrevet etter samme standard, man skal ikke være i stand til å se hvem som har skrevet hvilken kode.

Siden vi vil basere oss på denne modellen vil fremtiden ligge noe uviss i forhold til mange av de andre systemutviklingsprosjektene. Vi tenker da på oppgaver hvor for eksempel fossefallsmodellen kan tas i bruk, og det er mulig å fastsette de fleste datoer og tidsfrister tidlig i utviklingsfasen. I XP vil dette være vanskelig grunnet dens arbeidsform og systemutviklingsmetodikk. Vi vil i samarbeid med oppdragsgiveren og veilederen sette opp brukerhistorier, for deretter å utvikle tidsestimater som blir utgangspunktet for en releaseplan. Releaseplanen vil fordele brukerhistoriene utover iterasjonene etter oppdragsgiverens prioriteringer, med de mest sentrale delene i første iterasjon. Kunden kan deretter foreta omprioriteringer, eventuelt spesifisere nye brukerhistorier gjennom hele prosjektets levetid. Iterasjonene vil i dette tilfellet typisk være noe kortere enn vanlig, da det er snakk om en temmelig konsentrert utviklingsperiode med stort innslag av læring og utprøving.

1.7 Arbeidsformer

Det ble tidlig avtalt at Anders Rindal skulle fungere som gruppeleder, da de resterende gruppemedlemmene uttrykte respekt for hans evne til å beholde oversikten i liknende situasjoner og fordele arbeidsoppgaver på en effektiv og rettferdig måte.

Gruppemedlemmene ble enige om at mye av arbeidet skulle gjennomføres mens hele gruppen var samlet. Dette er spesielt viktig for prosjektet vårt, hvor det kan være svært vanskelig å finne den beste løsningsmetoden for hver oppgave før man har eksperimentert med ulike alternativer. Prosjektet bærer preg av at det er sterkt forskningsrelatert, noe som medfører at gruppen må benytte mye av tiden til å sette seg inn i (for

oss) nye emner, og det var dermed viktig for prosjektets fremdrift at alle de involverte tilegnet seg de nødvendige kunnskapene så fort som mulig.

For å forsikre oss om at alle beholder motivasjonen og blir holdt oppdatert om prosjektets fremdrift, forsøker vi etter beste evne å følge ekstremprogrammeringens krav om delt eierskap av kode. Dersom uforutsette hindringer i form av for eksempel sykdom skulle inntreffe, vil de resterende gruppemedlemmene kunne fortsette arbeidet uavbrutt fordi parprogrammeringsprinsippet sørger for at ingen kunnskap om prosjektets virkemåte blir utilgjengelig.

Kildebruk ville bli viktig i arbeidet med å sette seg inn i emneområdet, og selvstendig søk etter løsningsmetoder på Internet og i bøker/dokumenter ga resultater i form av andre innfallsvinkler på problemer. Dersom en person fant informasjon som ville være hele gruppen til gode, måtte denne personen forklare dette for de andre gruppemedlemmene, ofte ved bruk av en tavle. Det samme var tilfellet dersom man fant en ny fremgangsmåte som man ville dele med resten av gruppen.

XP gir mange forslag og krav til arbeidsmetoder under systemutviklingsfasen. Vi forsøkte å benytte oss av disse forslagene, siden de er resultatet av andres erfaring med hva som er lønnsomt i andre utviklingsprosjekter av liknende natur. Dette ble beskrevet i avsnittet om valg av systemutviklingsmodell.

Rapporten ble gjennomført ved at man forsøkte å arbeide kontinuerlig gjennom hele den tildelte prosjektperioden, i form av statusrapporter, møtereferater og dokumentering av arbeidet som til enhver tid foregikk. Etter sluttføring av en del av prosjektet, diskuterte gruppemedlemmene betydningen av dette og skrev om hvordan løsningen fungerer. Designmessige avgjørelser ble foretatt i hver iterasjon, og produktets oppbygning utviklet seg gradvis. Dette dannet grunnlaget for rapportens drøfting av design.

Oppdragsgiver og veileder var meget involvert i prosjektdiskusjoner under møter annenhver uke, hvor fremdriften ble satt i søkelys, samtidig som nye oppgaver ble fremsatt. Gruppen forsøkte etter beste evne å gi fornuftige tidsanslag på brukerhistoriene, slik at oppdragsgiver kunne prioritere oppgavene riktig sett fra hans synspunkt.

Statusmøter hver måned medførte detaljerte beskrivelser av hva som hadde blitt fullført, og hva som ville bli viktig i tiden fremover. Dette hjalp gruppen med å holde fokus på oppgaven og se den større sammenhengen i prosjektet.

1.8 Organisering av rapporten

Denne rapporten vil være organisert i forskjellige hovedtemaer. Hver del tar for seg aspekter som vil være relevante for gitt tema. Høgskolen i Gjøvik har satt opp en mal for hvordan en rapport skal/bør være, og det er den som har ligget til grunn for vår oppbygningen. Siden det finnes mange forskjellige typer prosjekter, vil det være umulig å finne en rapportmal som vil tilfredsstille alle disse prosjektene. Dette er en av grunnene til at vi har modifisert malen en tanke slik at den, etter vår mening, vil passe vårt prosjekt best mulig.

Etter konsultasjon med veileder, ble det bestemt at rapporten skulle utformes etter følgende inndeling:

1. **Innledning**

   Rapporten begynner med en introduksjon av arbeidsområdet, problemstillingen og våre forutsetninger før arbeidet på prosjektet begynte. Innledningen forutsetter ingen spesielle forkunnskaper, og forsøker å gi en kort, klar og forståelig presentasjon av prosjektet.

2. **Analyse / Kravspesifikasjon**

   Ethvert prosjekt gjennomgår en fase hvor analyse står i hovedsentrum. Dette kapitlet behandler de forutsetningene som ble gjort i perioden før design og implementasjon ble iverksatt, og begrunner de valgene som ble tatt.

3. **Design**

   Oppbygning av logiske strukturer og funksjonalitet for applikasjonen i sin helhet. Hvilke grensesnitt vi har benyttet oss av for å realisere prosjektet.

4. **Implementasjon**

   En mer detaljert beskrivelse av hvordan vi kodemessig har løst ulike problemstillinger.

5. **Testing / kvalitetssikring**

   Tiltak som ble gjort for å sikre at kvaliteten på applikasjonen ble ivaretatt.

6. **Sluttdiskusjoner**

   Drøftinger og refleksjoner tilknyttet oppgaven og gruppen som helhet.

7. **Konklusjon**

   En oppsummering av resultatet som er oppnådd gjennom dette prosjektet

8. **Appendiks / Vedlegg**

   Terminologiliste, rapporter, kildekode etc.

Vi har tatt utgangspunkt i HiGs standard for strukturering av hovedprosjektsrapporter, men på grunn av de stadig skiftende kravspesifikasjonene prosjektet vårt gjennomgår har vi gjort visse tilpasninger. Da ingen av gruppemedlemmene har skrevet en liknende rapport tidligere, må dette med tas med i betraktningen når man gjennomgår det endelige resultatet.

1.9 Rapportlayout

Til en rapport av denne typen stilles det en del krav. Den skal være oversiktlig, lett å lese / finne frem i og ikke minst strukturert. For å gjøre denne jobben har vi benyttet oss av LyX, en tekstbehandler for Linux som baserer seg på LaTeX koding og er bygget på WYSIWYG(“What You See Is What You Get”) prinsippet. Vi var alle enige å benytte denne tekstbehandleren da vi synes den lager pene og ryddige dokumenter i tillegg til at den er enkel å jobbe med.

LyX tilbyr et mangfoldig sett med valgmuligheter for å organisere en rapport av denne typen. Ikke minst mtp. opprettelse av innholdsfortegnelse og referanser i rapporten, men også hvordan ting vil være og se ut.

Inndeling

Hoveddelene i rapporten vil være organisert i kapitler, hvor hvert kapittel har sine delkapitler eller seksjoner hvor forskjellige emner er beskrevet. For hvert delkapittel vil det også finnes finere inndelinger.

Nummereringen av delkapitler og de finere inndelinger vil alltid relatere seg til hovedkapitlene de ligger under. De vil også være uthevet. Hvordan denne inndelingen vil være vises nedenfor:

Kapittel [x]

Delkapittel [x].1

Delkapittel [x].2

Videre inndeling [x].2.1

Videre inndeling [x].2.2

Siste inndeling [x].2.2.1

I tillegg til denne inndelingen vil det finnes overskrifter som ikke vil være nummerert. Disse vil også være uthevet og vil være med på å organisere den enkelte seksjon eller inndeling den ligger under.

Fonter

Fonten som benyttes vil alltid være den samme og er standard. Dette gjør at rapporten hele tiden vil følge den samme stilen, og den vil bli mer oversiktlig og ryddig.

Kapittel 2

Analyse / Kravspesifikasjon

2.1 Kravspesifikasjon og forberedelser

Hovedmål

Oppgaven består i å lage en applikasjon som skal kunne lese inn og vise et bilde (så godt som mulig iht. enhetens ICC-profil). Det skal også være mulig å visualisere bildets farger på forskjellige måter i det tredimensjonale fargerommet. I tillegg skal utstyrsgamut visualiseres. Det er dermed viktig å finne en enkel måte å håndtere bildefiler på, og finne frem til de metoder som behøves for å vise en 3D-struktur.

Programmet skal også inneholde funksjonalitet for å gjøre endringer i fargerommet og i bildeplanet, og deretter visualisere hvordan det endelige bildet vil bli seende ut.

Applikasjonen skal være lett å utvide grunnet dens modulære struktur. Dette innebærer at det er viktig å foreta de rette beslutningene vedrørende de grensesnitt man innfører i applikasjonen, siden man må sørge for at programdelene får klare og veldefinerte ansvarsområder.

Delmål

Vi anser det for å være viktigere å gjøre et godt og grundig arbeid på basisfunksjonaliteten enn å diversifisere innsatsen over et større område. Gjennom prosjektet skal utviklingsmodellen XP og dens grunnprinsipper følges etter beste evne.

Det vil bli lagt vekt på dokumentasjon underveis for å få et best mulig utgangspunkt for eventuelle utvidelser av funksjonaliteten. Dersom tiden står til rådighet, etter at grunnlaget er lagt, vil vi jobbe med å utvide systemet basert på evt. ønsker fra oppdragsgivers side.

Vi setter oss som foreløpig mål at applikasjonen skal inneholde følgende funksjonalitet: (Det må påpekes at dette kan forandres underveis, avhengig av nye prioriteringer)

Endringer i 3D:

- Forflyttning av enkeltpunkter i rommet. Brukeren tar tak i ett punkt, og kan bevege dette rundt i 3D. Pikslene i bildet som har en tilhørende farge, skal oppdateres for å reflektere forandringene.
- Fjæreffekt på nærliggende punkter. Flere punkter flyttes en varierende distanse, avhengig av nærhet til det objektet brukeren flytter.
- Forflyttning av flere punkter samtidig. Det er ikke avklart hvilke typer forflyttninger som kan være aktuelt.
- Globale transformasjoner, f.eks. justering av kontrast, metning, hvitpunkter etc. Avklares nærmere underveis i prosjektperioden.

Forskjellige visualiseringer:

- Punktsky med lik størrelse på punktene. Alle forskjellige pikselverdier i bildet representeres som ett punkt i 3D.
- Redusert punktsky med forskjellige størrelse på punktene. En forenklet struktur der ett punkt i 3D kan tilsvare mange nærliggende farger i bilder.
- Convex hull. Bildet vises som et konvekst objekt bestående av en samling trekanter.
- Segment maxima. Bildepikslene deles inn i segmenter rundt et sentrum, og ekstrempunktene benyttes for å konstruere en triangularisert overflate.

Ulike fargerom (se i terminologilisten for en nærmere forklaring):

- RGB
- Lab
- XYZ

Sammenheng mellom 2D og 3D:

- Valg av punkter i 3D synliggjøres i 2D og motsatt.
- Endringer utført av brukeren vises både i 2D og 3D.

Rammer

Ut i fra dette har vi i utgangspunktet valgt å basere oss på Java og Java3D som hovedutviklingsmiljø, selv om dette medfører noen svakheter. Java3D er fremdeles under utvikling, noe som gir seg utslag i at Linux versjonen er noe ustabil og at det i skrivende stund ikke finnes noen versjon for MAC. Vi har valgt å ikke legge for stor vekt på dette, da det er all grunn til å tro at dette vil forandre seg i nær fremtid. En av fordelene ved å benytte Java, er den innebygde støtten for bruk av ICC profiler, som trolig vil lette arbeidet betraktelig. En av ulempene er at det krever mye minne, og er til tider tregt. Dersom det viser seg at enkelte operasjoner blir for krevende, vurderer vi å benytte OpenGL for visualiseringen. På grunn av ønsket om portabilitet vil vi i så fall bruke et plattformuavhengig API mellom Java og OpenGL.

XP

Ekstremprogrammering innser at en kravspesifikasjon kan være vanskelig å sette opp på forhånd for visse prosjekttyper. I vårt tilfelle er det umulig å definere en mer eller mindre konkret liste med krav som skal stilles til produktets funksjonalitet, og det som tilsvarer en mer generell kravspesifikasjon med anslag over tidsforbruk og liknende vil være å finne i vedlegget som beskriver de brukerhistoriene som har blitt gitt og gjennomført.

Sammendrag av brukerhistorier

I ettertid kan vi legge frem følgende sammendrag av alle brukerhistoriene vi har kommet fram til i samarbeid med oppdragsgiver og veileder

- Bilder skal leses inn fra filer med forskjellige formater, deretter skal bildepunktene visualiseres som et sky-objekt i RGB fargerom.
- Fargerommet skal kunne kvantifiseres, slik at tilnærmet like farger representeres som ett punkt i 3D. Detaljnivået justeres i brukergrensesnittet, og kvantiseringen skal valgfritt skje i det opprinnelige eller gjeldende fargerom. I tillegg til å kunne vises som punkter skal det også eksisterer en histogramvisning som viser kuler med varierende størrelse.
- Punktene skal kunne flyttes i henhold til brukerens bevegelser med musa. Disse forflyttningene skal kunne begrenses automatisk til forhåndsdefinerte bevegelser langs linjer eller i plan. Det skal også være mulig å flytte flere punkter samtidig ved hjelp av globale transformasjoner.
- Bilder skal bildepunktene visualiseres som et sky-objekt i Lab fargerom.
- Omfanget av RGB fargerommet skal vises som en deformert kube i Lab.
- Segment maxima metoden skal benyttes for å representere bildet som en overflate i 3D. Sentrum skal kunne velges av brukeren eller beregnes av programmet på bakgrunn av punktenes posisjon. Resultatet vises som solide trekanter og/eller wireframe, med muligheter for transparens.

- Segment maxima punktene skal kunne flyttes og samtlige punkter i segmentet skal skaleres i forhold til radius.
- Det skal bygges en tetraeder struktur ut fra segment maxima visualiseringen, det skal også være mulig å interpolere forflyttninger ved hjelp av denne strukturen (tetraederinterpolasjon). Interpoleringen skal også kunne fungere på tilsvarende løsninger basert på andre visualiseringer.
- Programmet skal lese inn og parse filformatet IT8.7/2 og et regulært filformat.
- Utstyrsgamuten man får ut av filene skal kunne vises samtidig med bildetgamuten.
- Segment maxima punktene skal kunne eksporteres til et gitt filformat.
- Aksekors og andre grenser for fargerom skal kunne vises i 3D.
- Kameraet skal kunne posisjoneres på forhåndsdefinerte steder i 3D. Det skal beholde sin posisjon mellom oppdateringer av scene-struktur.
- Alle forskjellige typer visualiseringer skal ha sitt eget parameterpanel.
- Programmet skal ikke behøve å kalkulere verdier på nytt mellom hver oppdatering dersom det ikke har skjedd noen forandringer.
- Alpha Shapes metoden skal benyttes til å representere bildet som en overflate i 3D. Brukeren skal interaktivt kunne velge alpha verdier.

For nærmere detaljer angående estimert og virkelig tidsforbruk se vedlegg. Det må bemerkes at gruppen i tillegg har implementert andre elementer enn det som kommer fram av brukerhistoriene, fordi vi har hatt interesse og glød for prosjektets resultat.

2.2 Java3D vs OpenGL - Utdypning av analysefasen

I utgangspunktet var oppdragsgiver, veileder og gruppemedlemmene mest interessert i å finne ut hvordan Java3D ville egne seg til å vise og behandle de relativt sett store mengder data applikasjonen i praksis skal håndtere. Det ble derfor lagt vekt på at man raskest mulig skulle kode en eller annen type visualisering for å få avklart dette, slik at man kunne ta grunnleggende avgjørelser vedrørende prosjektets arkitektur. Det ble ansett som viktig å starte arbeidet med implementasjonen raskest mulig, fordi det ellers var begrensede muligheter for å forutsi de problemene man kunne støte på. Dermed fulgte en periode med uttesting og eksperimentell analyse av oppgaven .

Vår bekymring økte da det viste seg at vår første prototype ikke oppførte seg tilfredsstillende med tanke på flytting av punkter. Dette første forsøket benyttet seg av bokser for å representere en unik pikselverdi, og Java3Ds innebygde støtte for flytting av objekter sørget for at flyttingen fungerte. Problemet oppstod når man forsøkte å benytte et reelt bilde i stedet for testbilder med begrenset antall pikselverdier. Java3D forlangte at hvert synlige objekt skulle ha sin egen transformasjon som anga riktig posisjon til objektet, og når antallet objekter økte, kunne selv ikke en relativt rask datamaskin holde orden på scenegrafen som dette resulterte i.

Vi ga raskt tilbakemelding til oppdragsgiver og veileder slik at de ble klar over problemet, og det ble bestemt at andre alternativer skulle utprøves, samtidig som arbeidet med eventuelle snarveier for Java3D-versjonen skulle fortsette. Ett gruppemedlem foreslo at man skulle se nærmere på muligheten for å benytte OpenGL i tillegg til Java. Det finnes flere eksisterende løsninger for dette, men vi mente at de ikke var tilfredsstillende til vårt formål. Det finnes derimot en måte å koble Java opp mot andre programmeringsspråk ved å benytte et grensesnitt og dynamisk lastede biblioteker. Ingen av de interesserte parter hadde noen erfaring med dette, men det ble avgjort at man skulle teste hvordan en slik løsning ville bli.

Kravet om plattformuavhengighet gjorde at det var utelukket å benytte plattformavhengig kode for 3D-visningen. Det var derfor klart at man måtte benytte et plattformuavhengig vindusbibliotek. Vi testet SDL, en løsning som eksisterer på alle relevante operativsystemer og maskinarkitekturer, men fant raskt ut at begrensninger på åpning av nye vinduer gjorde dette alternativet uegnet. GLUT, OpenGL Utility Toolkit, var mer lovende, selv om også dette var problematisk å bruke i forbindelse med Java. Vi greide imidlertid å kode en testapplikasjon som benyttet JNI (Java Native Interface) for å koble GLUT/C++ til Java. JNI lar et Java program kalle opp funksjoner i dynamiske biblioteker som kan skrives i andre språk, og håndterer kommunikasjon mellom de ulike delene av programmet.

Samtidig med denne uttestingen, jobbet gruppen med en løsning på problemene med Java3D. Etter en del eksperimentering fant vi en måte å unngå den store scenegrafen. Vi benyttet ett objekt, en array av punkter, slik at man ikke lenger behøvde det store antallet transformasjoner, men kunne angi punktenes posisjon direkte. Dette ga en uforholdsmessig stor hastighetsforbedring, og tillot oss å kunne vise mange flere punkter enn før.

På dette tidspunktet hadde vi allerede en fungerende løsning som benyttet GLUT, men det var visse ulemper ved denne måten å gjøre det på. Deler av programmet ville måtte rekompileres for ulike plattformer, og det ville bli mer arbeid med den grafiske siden av prosjektet. JNI ville komplisere modulariseringen av produktet, og det var ingen ekstreme ytelsesforskjeller mellom de to løsningene. Fordelen med GLUT var at det tillot mer direkte tilgang til det som ble vist på skjermen, men på den andre siden ville det kreve en egen instruksjonstråd, og man måtte dermed ha foretatt en del synkronisering av kommunikasjon mellom Java og skjermvisningen.

Etter å ha veid de ulike argumentene for og imot de to metodene, fant vi at det ikke var noen tungtveiende årsaker til å velge den mer kompliserte løsningen med GLUT. I samråd med oppdragsgiver og veileder kom vi frem til at vi skulle satse på Java3D, og heller arbeide rundt de eventuelle begrensningene som dette ville medføre.

2.3 Kildekode / kommentering

Vi følger vanlig Javanavnsetting:

- Klasser navngis slik: MyClass
- Objekter, variable og metoder har liten forbokstav og følger ellers navnsettingen til klasser: myObject

- Konstanter skrives med store bokstaver: MY_CONSTANT
- JavaDoc brukes for å kommentere kode. (se: `http://java.sun.com/javadoc`)
- En versjonshistorie for hver enkelt fil legges i starten på filen. Eks.

```
/* ************************************************************\
* File: MyClass.Java                                           *
* Date: xx.xx.xxxx                                             *
* Version: x.y.z                                               *
* History:                                                     *
* - Major:                                                     *
*   - date, description                                        *
* - Minor:                                                     *
*   - date, description                                        *
* - Bugfix:                                                    *
*   - date, description                                        *
\************************************************************/
```

Vi baserer oss videre på å utvikle applikasjonen i Windows. Dette grunnet en noe ustabil versjon av Java3D for Linux, som ellers hadde vært å foretrekke. Alle vil også benytte seg av samme editor.

2.3.1 Versjonshåndtering

Vi bruker major og minor versjonsnummerering pluss en "buggfiks". Eks. 1.3.7

Forandring av:

- "buggfiks" skjer når en feil er korrigert.
- minor skjer når en mindre vesentlig funksjon er lagt til
- major skjer når en betydelig forandring er gjort

Katalogstrukturen og backup filer navngis med dato for å unngå misforståelser vedrørende hva som er den siste versjonen.

2.3.2 Web

Under prosjektets gang har vi disponert en webside hvor vi har lagt ut løpende informasjon og opplysninger rundt prosjektet. Dette har vært opplysninger som:

- Oppgavebeskrivelse. Bakgrunn og hoved-, delmål.
- Kontaktinformasjon til oppdragsgiver og veileder.
- Kontaktinformasjon til gruppedeltakerene.

- Statusrapporter.
- Mulighet for å laste ned ulike versjoner av applikasjonen. Denne delen har fortløpende blitt oppdatert etterhvert som nye utgaver har vært klare. Vi har også valgt å fjerne eldre versjoner når nye har kommet.
- Testbilder til bruk i applikasjonen og screenshots for å vise hvordan den aktuelle versjonen ser ut.
- Linker til andre programpakker vi benytter oss av og som er nødvendig for å kjøre applikasjonen.

Vi har valgt å la en person på gruppen være ansvarlig for oppdatering og utforming av websiden for på denne måten å unngå konflikter. Dette har vist seg å fungere fint da vi ikke har opplevd noen problemer.

2.4 Plattformuavhengighet

Hovedgrunnen til at vi valgte å lage programmet i Java, var ønsket fra oppdragsgiver om at dette skulle være plattformuavhengig. Dette var også noe vi var overbevist om Java ville kunne tilby oss, mye grunnet dets oppbygging (interpretering og JavaVM i bunn). Allikevel kom vi over problemer som kan være avgjørende for sluttbrukeren. De største problemene hadde vi med Java3D som virker noe uferdig.

Kapittel 3

Design

3.1 Sessjonshåndtering

I et program der det kan arbeides i flere vinduer samtidig, og de kan inneholde helt forskjellige produkter, er det viktig å kunne håndtere de forskjellige instansene fra hverandre.

Dette har vi løst ved å innføre sesjoner. Sesjonene blir håndtert av en egen klasse, Session, som det opprettes et nytt objekt av for hver gang man åpner ett nytt bildet. Session klassen inneholder all informasjon image2D objektet, og tilbyr også GUI for denne. De forskjellige Session objektene blir lagret i en array, som vedlikeholdes av klassen ICC3D. Altså den som starter og avslutter hele programmet. Hvert canvas 3D objekt som opprettes (m.a.o når man åpner en ny 3D-view) får tilsendt sitt respektive session objekt.

På denne måten klarer vi å håndtere flere forskjellige bilder og 3D-views samtidig, og brukeren får større fleksibilitet i arbeidet.

3.2 Layout, style, fonter

3.2.1 Generelt

Under utviklingen av applikasjonen har vi hele tiden prøvd å legge vekt på å lage et grafisk brukergrensesnitt (GUI) som er mest mulig intuitivt og enkelt å bruke. Fonter, ikoner og layout generelt er valgt med bakgrunn i dette. Vi har også prøvd å holde oss til det som mange mener og føler er “standard utseende” på GUI. Dette er gjort for at flest mulig brukere skal kunne føle seg “hjemme” og i kjente omgivelse nå de bruker applikasjonen.

3.2.2 Vinduene

Vi har valgt å bruke et multi-vindusmiljø, der nye vinduer opprettes etterhvert som man ber om det eller applikasjonen krever det. Layouten på vinduene følger vanlig

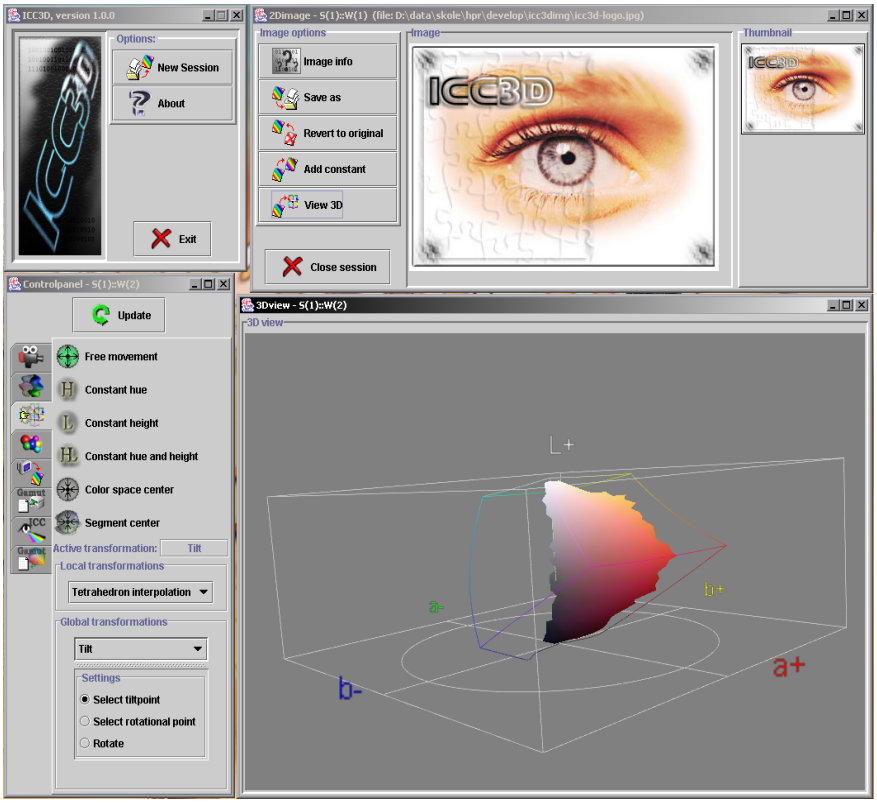
Java standard. Denne løsningen valgte vi mye på grunn av at Java3D ennå ikke er helt kompatibelt med resten av Java når det gjelder muligheten for å kunne legges inne i tabber eller MDI (Multi) vinduer. Derfor bør den ha et eget vindu å arbeide i.

Oppbygning, virkemåte og sammenheng mellom de enkelte vinduene ligner litt på stilen som brukes i Adobe PhotoShop og The Gimp (bildebehandlingsprogram for Linux). Når man åpner et nytt bilde, starter man automatisk en ny session. Ut fra tilgjengelige valg i denne session kan man få frem en 3D modell av det aktuelle bildet. Det er mulig å åpne flere 3D modeller av det samme bildet fra den samme session. Selv om disse 3D modellene stammer fra samme session og samme bilde, er de samtidig frittstående. Dvs. at man kan vise 3D modellene med forskjellige visualiseringer uten at de vil påvirker hverandre. Dette er hensiktsmessig dersom det er ønskelig å vise hvordan et bilde kan representeres med forskjellige visualiseringer samtidig.

Når en 3D modell åpnes for første gang, vil også et kontrollpanel automatisk åpnes. Dette vinduet inneholder alle mulige og tilgjengelige valg og innstillinger for den aktuelle 3D modellen. Det er via dette panelet man velger visualisering (og innstillinger for denne), fargerom osv.

Det vil til enhver tid bare være ett kontrollpanel uansett hvor mange 3D modeller og sessions man har oppe (forutsatt at man har minst en session og en 3D modell oppe). For å endre innstillinger på en 3D modell (n), må man først aktivere dette vinduet slik at kontrollpanelet vet hvilket vindu (n) den skal jobbe med. Vil man så endre på en annen 3D modell (n+1), aktiverer man så dette vinduet og kontrollpanelet bytter automatisk til det valgte vinduets (n+1) innstillinger. Dersom man lukker en 3D modell og det ikke finnes flere (verken fra samme eller andre sessions), lukkes også kontrollpanelet automatisk. Dette fordi det nå ikke finnes noen modell å gjøre endringer på og det er derfor ikke behov for å vise dette vinduet.

En generell oversikt over programmets brukergrensesnitt.



Figur 3.1: ICC3D og alle dets vinduer.

Navnsetting av vinduer

Tittelen på de enkelte vinduene følger en standard vi har valgt å bruke. Denne ser slik ut:

<type vindu> - S(n)::W(n) [file: absolutt path til fil]

hvor

- <type vindu> kan være “2D Image”, “3D view” eller “Controllpanel”
- S(n) er Session(session nr n).
- W(n) er Window(vindu nr n) for den aktuelle session S(n).

f.eks.: “2D Image - S(1)::W(1) (file: foo.tif)”.

Figur 3.2: Tittel i “2D Image” vinduet.

Hovedvinduet vil inneholde: <navn på applikasjon> <, versjonsnummer>,
f.eks.: “ICC3D, beta 0.5.0”.

Figur 3.3: Tittel i hovedvinduet til ICC3D.

3.2.3 Ikoner

Alle ikoner som vises i applikasjonen har vi laget selv. Dette for å sette vårt eget preg på applikasjonen og for å kunne visualisere en funksjon best mulig. Ikonene er lagret på PNG format og har størrelse 30x35 eller 50x55 (avhengig type ikon og hvor de blir brukt). Alle ikoner er også transparente slik at bakgrunnsfarge på applikasjonen i seg selv, som kan bestemmes av gjennom valg i operativsystemet, ikke vil interfere med hvordan ikonene vises.

3.2.4 Fonter

Vi har ikke brukt noen spesiell type font, men basert oss på den som er standard i Java. Dette gjør at vi unngår konflikter dersom applikasjonen kjøres på forskjellige plattformer, da disse i noen tilfeller vises litt forskjellige. Siden applikasjonen skal være mest mulig plattformuavhengig kom vi frem til at dette var den beste løsningen.

3.3 Fargerom

Et sentralt emne for dette prosjektet er fargerom og hvordan disse behandles. Et fargerom definerer en bestemt måte i angi forskjellige farger på. Skal det gjøres kalkulasjoner og beregninger på en farge spesifisert i et gitt fargerom er det helt avgjørende at formler tilknyttet det angitte fargerormmet benyttes. Dersom dette ikke gjøres vil man endre den aktuelle fargens verdi og få totalt feil resultat.

ICC3D er i skrivende stund i stand til å manipulere farger for tre forskjellige fargerom. Disse er:

- Lab
- sRGB
- XYZ

(se terminologiliste for ytterliger informasjon om de forskjellige fargerommene)

Bruken av de forskjellige fargerommene i ICC3D gjenspeiler seg i hvordan representasjonen av et gitt bilde skal vises og/eller manipuleres i 3D. Alle endringer som gjøres i 3D, utføres i et gitt fargerom bestemt av bruker. Vil man gjøre endringer i et annet fargerom, bytter man til ønsket fargerom og all nødvendig konvertering av fargeverdier ol. gjøres automatisk av programmet slik at brukeren bare behøver å forholde seg til den manipulerende delen av representasjonen. Denne måten å behandle forskjellige fargerom på gjør ICC3D fleksibel og enkel å jobbe med.

3.3.1 Struktur

I Java finnes det innebygd støtte for konvertering av forskjellige fargerom. Funksjonaliteten for dette finnes i klassen ICC_ColorSpace, og den gjør det mulig å konvertere til og fra sRGB og XYZ fargerom. I begynnelsen baserte vi oss på disse funksjonene for vår konvertering. Men etterhvert som kravet om hastighet og effektivitet meldte seg, fant vi ut at disse konverteringsrutinene var for tunge å arbeide med. En av grunnene til dette er at konverteringen baserer seg på store og kompliserte ICC profiler. For å komme rundt dette “problemet”, så vi oss nødt til å gjøre konverteringen på en annen måte. Resultatet av dette ble at vi lagde oss nødvendige klasser og skrev optimaliserte konverteringsrutiner for konverteringen selv. Dette medførte en drastisk endring i behandlingstid. På grunnlag av dette oppnådde vi en forbedring på opptil 75% i behandlingstid i visse tilfeller.

Måten vi gjorde dette på var å lage et lite hierarki for fargerommene. Den abstrakte baseklassen **ColorSpace3D** deklarerer funksjonalitet som aktuelle fargerom er nødt til å implementere. Dette er funksjoner for konvertering til og fra bestemte fargerom, oppbygning av aksesystem og hvordan det nye fargerommet blir visuelt fremstilt i applikasjonen. Tenker da på hvordan de enkelte opsjonene blir tilgjengelige for brukeren.

Tabellen viser hierarkiet:

Baseklasse	*Virksomhetsklasse*
ColorSpace3D	
	LabColorSpace3D
	SRGB_ColorSpace3D
	XYZ_ColorSpace3D
	...

Tabell 3.2: Arvehierarki for fargerom.

Ved å strukturere oppbygningen på denne måten vil det være enkelt å lage nye fargerom som utvidelse til ICC3D.

3.3.2 Aksesystem

I tillegg til lage funksjonene for konvertering til og fra andre fargerom, vil de også være nødt til å implementere en funksjon for å visualisere aksesystemet for det gitte fargerommet. Dette aksesystemet viser omfanget til fargerommet og vil være en ren visualisering som brukeren kan velge om skal vises eller ikke.

I de fleste tilfeller vil aksesystemet være veldig nyttig da det viser om farger i et bildet ligger innenfor eller utenfor fargerommet. Det vil også gi et bedre visuelt bilde av hvordan fargene er representert/plassert i det aktuelle fargerommet.

3.4 2D

Den todimensjonale delen av ICC3D er ikke den som er mest fremtredende. Hovedsakelig er denne delen for å illustrere resultat på bildet du gjør endringer på.

Når man åpner et bildet kommer det opp ett nytt vindu, dette er 2D vinduet, og inneholder noen funksjoner. Selve bildet kommer opp i en tumbnail view og et felt der du ser selve utsnittet av det du har valgt i tumbnailen. Det er her verdt å merke seg at når man gjør forandringer på bildet, vises det originale bildet i tumbnailen, mens utsnittet viser forandringene som blir gjort.

På venstre side har man en knapperad som gir grunnlaget for videre navigasjon innover i programmet. Knappene er

- Image info
- Save as
- Revert to original
- Add constant
- View 3D

Image info gir deg ren bilde informasjon. Det er her snakk om filnavn, filtype, størrelse på bildet, samt fargerom og fargedybde. Dette er mer eller mindre oppramsing av informasjon som er hentet inn når bildet lastes, og vil ikke forandre seg under kjøring av programmet.

Save as er en viktig funksjon. Det er her du lagrer bildet når du er ferdig med redigering eller vil fortsette senere. Viktig er dog at den kun lagrer bildet, og ingen informasjon om forflyttningene som er gjort i 3D (sluttresultatet av pixelverdiene, men ikke posisjonen som for øyeblikket eksistere i 3D canvaset). Du får opp en fildialog boks og valg mellom å kunne lagre i fire forskjellige filformater, henholdsvis bmp, jpg, png og tif.

Revert to original gjør endringer både i 3D og 2D. Funksjonen henter fram det originale bildet og fjerner all informasjon om det som er blitt gjort tidligere. 3D canvaset blir oppdatert og satt tilbake slik det var fra starten av.

Add constant tilbyr en funksjon der man kan legge til konstante rgb verdier til bildet. De nye verdiene som blir lagt inn blir lagt til for hver eneste pixel i bildet, og en bør derfor bruke denne funksjonen med varsomhet.

View 3D tilbyr selve portalen til resten av programmet, der hele funksjonaliteten i programmet egentlig ligger.

3.5 Dynamisk klasseloading og modularitet

3.5.1 Generelt

En av de store fordelene som ICC3D tilbyr er dynamisk klasseloading. Dette skjer under oppstart og kan sammenlignes med bruk av plugins. Grunnen til at vi bruker denne måten å laste visse typer klasser på er at vi oppnår en større grad av modularitet som medfører at det er lett å legge til ny funksjonalitet og utvidelser av programmet. Et av de sentrale punktene i oppgavedefinisjonen var nettopp at applikasjonen skulle være mest mulig modulær i sin oppbygging og virkemåte.

Nedenfor er en oversikt over hvilke type klasser som lastes dynamisk under oppstart:

- **Colorspaces / Fargerom** (Lab, XYZ, sRGB)
- **Visualizations / Visualiseringer** (AlphaShapes, SegmentMaxima, Quantized ...)
- **Interactions / Interaksjoner** (GlobalTransformScaleL, SegmentInteraction3D, BasicInteraction3D ...)
- **Constraints / Begrensninger** (ConstantAngleConstraint, ConstantHeightConstraint, SegmentCenterConstraint ...)

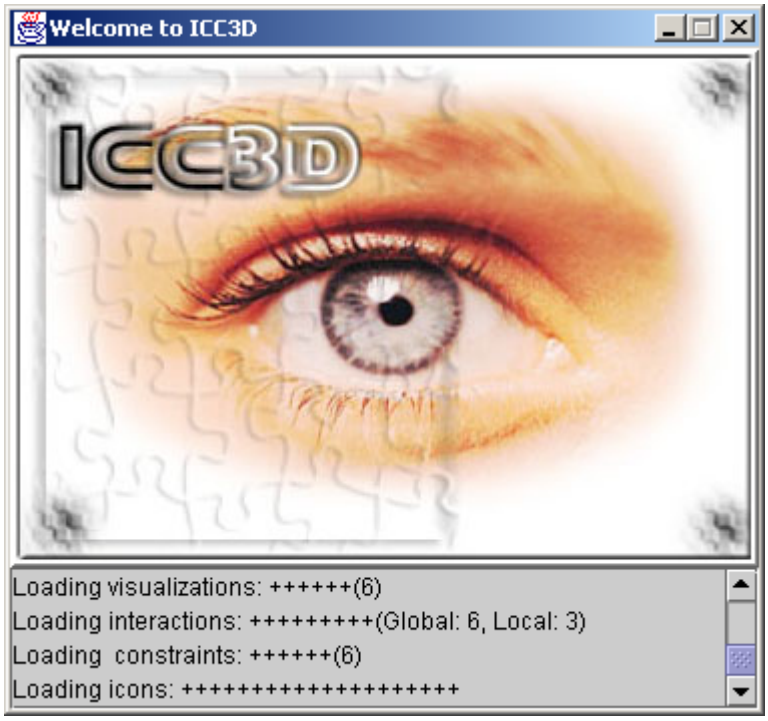
3.5.2 Virkemåte

For å kunne benytte oss av dynamisk klasseloading er vi nødt til å bruke visse funksjoner i Java.

Under oppstart av applikasjonen vil de aktuelle klassene hentes ut fra bestemte kataloger. Vi har laget en katalogstruktur for disse som er delt inn i type funksjonalitet. I de gitte katalogene vil alle klasser som tilhører en bestemte type funksjonalitet ligge. Programmet vil så gå gjennom disse katalogene, lese klassene og implementerer dem slik at de blir tilgjengelige etter oppstart.

Dersom man vil utvide applikasjonen med feks. en ny visualisering, plasserer man enkelt og greit den nye klassen på rett sted i katalogstrukturen og den vil lastes automatisk neste gang programmet startes (med forbehold at den er bygd opp rett og fungerer!). Innstillinger og valg for den nye visualiseringen (som defineres i de enkelte klassene) vil så være tilgjengelig fra samme sted som de andre eksisterende visualiseringene og den vil være klar til å brukes i applikasjonen. Det samme gjelder dersom man vil fjerne en funksjonalitet; fjern klassen fra rett katalog, og programmet vil ikke tilby den gitte funksjonaliteten neste gang den startes.

Det er viktig å legge merke til at nye klasser lastes under oppstart av programmet. Dersom nye klasser legges til under kjøring av programmet vil disse ikke være tilgjengelige før applikasjonen lukkes og startes på nytt.



Figur 3.4: Dynamisk lasting av plugins

Denne måten å behandle ny funksjonalitet på, både legge til og fjerne, styrker applikasjonens modularitet og videreutviklingspotensiale.

3.5.3 Struktur

Alle klasser som lastes dynamisk er nødt til å følge en gitt struktur for at de skal kunne brukes. Dette har vi gjort ved å lage abstrakte baseklasser som de aktuelle klassene må arve fra. Disse baseklassene spesifiserer et rammeverk med nødvendige funksjoner som må være tilstede i arvede klasser. Eksempler på slike funksjoner kan være hvordan innstillinger og valg av opsjoner vil være eller se ut, samt navn på den nye funksjonaliteten. Hvilke funksjoner som er definert som abstrakte, og dermed må implementeres i nye klasser, vil selvfølgelig variere fra type til type. Nedenfor er det gitt en oversikt over hvordan klassehierarkiet er bygd opp:

Baseklasse	Virksomhetsklasse	Virksomhetsklasse
ColorSpace3D		
	LabColorSpace3D	
	SRGB_ColorSpace3D	
	XYZ_ColorSpace3D	
Visualization3D		
	NoneVisualization3D	
	StandardVisualization3D	
	DetailedVisualization3D	
	SegmentMaximaVisualization3D	
	AlphaShapesVisualization3D	
	ConvexHullVisualization3D	
Interactions3D		
	GlobalTransformFancyRadius	
	GlobalTransformMoveAB	
	GlobalTransformMoveL	
	GlobalTransformRadius	
	GlobalTransformScaleL	
	GlobalTransformTilt	
	SimpleInteraction3D	
		BasicInteraction3D
		SegmentInteraction3D
		TetrahedronInterpolationInteraction3D
InteractionConstraint		
	NoConstraint	
	PlaneConstraint	
		ConstantAngleConstraint
		ConstantHeightConstraint
	LineConstraint	
		CenterConstraint
		ConstantAngleAndHeightConstraint
		SegmentCenterConstraint

Tabell 3.4: Arvehierarki for klasser som lastes dynamisk

3.6 Interaksjoner

Kanskje det viktigste målet med denne applikasjonen er å ha muligheten til å flytte på objektene som representerer bildet i 3D. Dette innebærer at designet bak forflyttninger av forskjellige typer er viktig for å ha størst mulig fleksibilitet. Det bør være mulig for brukeren å velge mellom de ulike operasjonene på en enkel måte, og brukerens aksjoner bør resultere i forventede og forståelige reaksjoner, både når det gjelder objektene i 3D og bildet som blir manipulert.

Vi fant det hensiktsmessig å benytte vårt dynamiske og modulære system for lasting av plugins, slik at også de interaksjonene som er tilgjengelig er helt separert fra resten av programmet. Dernest skilte vi mellom globale og lokale transformasjoner, der de globale opererer på alle pikslene i et bilde, mens de lokale kun forandrer på de pikslene som er assosiert med det spesifikke objektet brukeren forsøker å flytte på i 3D.

Alle manipulasjonsmetodene må ha de samme funksjonene for å flytte på objektene i 3D, samt oppdatere bildet når dette trengs. Det ble vurdert forskjellige valg for når bildet bør oppdateres, som om dette bør skje fortløpende mens brukeren drar på objekter i 3D, eller om det bør vente til etter at brukeren er ferdig med å flytte. Av ytelseshensyn er det ikke hensiktsmessig at oppdateringene skjer kontinuerlig, selv om dette nok ville vært bedre sett fra brukerens synspunkt. Det er imidlertid lagt inn støtte for at oppdateringene kan utføres etter hvert, selv om det ikke er et valg for dette i programmet.

Det antas at de lokale transformasjonene fungerer på samme måte når det gjelder oppdatering av objektet som flyttes i 3D, noe som gjenspeiles i klassehierarkiet som sørger for at denne koden skal være felles. Derimot er det opp til de globale transformasjonene å implementere de ønskede forflyttninger både i 3D og 2D, siden det er lite sannsynlig at disse vil ha noe felles med unntak av den generelle oppbygningen.

3.6.1 Globale transformasjoner

Vi har lagt inn følgende globale transformasjoner

- Move ab
- Move L
- Scale L
- Radius
- Fancy Radius
- Tilt

Hvis man ikke vil flytte på punktene/segmentene hver for seg og skal gjøre store endringer er disse godt egnet. De muliggjør at man kan gjøre relativt store endringer på bildet i løpet av kort tid, og kanskje slipper tidskrevende småjusteringer. Det er viktig å tenke på å ikke ha aktivert noen av begrensningene når man bruker noen av disse transformasjonene. Dette har vi løst ved at det alle begrensninger automatisk skrus av når en velger en av de globale transformasjonene.

Move ab gjør at alle punktene beveger seg i AB-planet. De oppfører seg likt som om et punkt skulle bli flyttet med constant height begrensningen slått på, altså i med konstant L. Transformasjonen gjøre at man kan forandre samtlige farge verdier på en enkel måte.

Move L transformasjonen flytter alle punktene med konstant A og B, men med variabel L verdi. Du kan m.a.o. bestemme luminansen til alle punktene samtidig.

Scale L skalerer punktene i forhold til L-aksen. En kan med denne altså justerer kontrasten i bildet. Den splitter opp punktene midt på L aksen (L=50), slik at de over midten går mot maks L, og de under mot min L.

Radius er også en skalering. Den skalerer punktene i forhold til radius, like mye i både a og b retning. De flyttes altså med en felles vektor, som regnes ut av forflyttningen i forhold til L-aksen. L verdien til punktene er her altså konstant.

Fancy Radius er en avart av radius transformasjonen. Forskjellen er i det store å hele at du her ikke finne en felles vektor, men holder A og B vektorene separat. Resultatet blir at dersom brukeren flytter det valgt objektet bort fra L-aksen i retning +A, vil alle objektene strekkes bort fra L-aksen i retning +A eller -A avhengig av hvilken side av LB-planet de er på. Dette gir en litt annen form for forflyttning og kan være nyttig ved visse justeringer samt at den gir deg ganske stor frihet med modellen.

Tilt er nyttig dersom hvit-punktet i bildet ikke befinner seg på riktig plass i forhold til svart, vil det være aktuelt å tilte alle objektene slik at man får forandret på dette. Dette gjøres ved at man først velger et tiltpunkt, dvs. et punkt som skal stå i ro og være sentrum for rotasjonen, for deretter å velge et rotasjonspunkt som skal dreies rundt tiltpunktet.

3.7 Visualiseringer

Det er viktig at både bildet og de forskjellige typene enhetsgamuter blir vist på best mulig måte i 3D. Dette er utgangspunktet for brukerens evne til å ta beslutninger vedrørende de endringer han eller hun vil utføre, og den effekten disse forandringene har på den grafiske representasjonen av bildet.

Vi anser det som et fortrinn at både bilder og enhetsgamuter kan benytte de samme visuelle fremvisningsmodusene, siden dette gir et helhetlig inntrykk, og gjør at vi kun trenger å implementere en versjon av hver visualisering i stedet for to veldig like utgaver. Dette er tidsbesparende, og det motsatte ville bryte med applikasjonens krav om modularitet. Hver visualisering må dermed ta hensyn til dette, siden representasjonen av bildet i motsetning til enhetsgamuten kan forandres av brukeren, og det derfor må bygges opp en mer avansert datastruktur i dette tilfellet.

3.8 Visualiseringer og interaksjoner - en studie i samhandling

Kanskje den viktigste problemstillingen vedrørende programmets oppbygning, er hvordan visualiseringene og de forskjellige forflyttningsoperasjonene kobles sammen. En dårlig løsning her kunne forårsaket store problemer ved utvidelser, og hindret at konstruksjonen av nye moduler foregikk uavhengig av andre deler av systemet.

Målet er at interaksjonene ikke behøver å ta hensyn til hvilken visualisering som er i bruk, så lenge scenestrukturen støtter de operasjoner som er nødvendig for å flytte objektene på den ønskede måten. Dette medfører at en interaksjon som er basert på for eksempel interpolasjon av tetraederoppdelingen ikke er avhengig av en spesiell visualisering, men kun krever at den visualiseringen som er i bruk har lagd objekter med den ønskede tetraederoppdelingen.

Vi har benyttet grensesnitt (i dette tilfellet Java interfaces) for å løse denne problemstillingen. Visualiseringene definerer et antall punkter som kan flyttes uavhengig av hverandre, og lager deretter ett objekt per punkt. Objektet implementerer ett eller flere grensesnitt, hvorav det mest grunnleggende gir støtte for å forandre posisjonen og fargen til objektet. Visualiseringen som bygger strukturen bestemmer hvordan operasjonen skal utføres, for eksempel om en forandring av posisjonen til et objekt skal medføre at koordinatene til flere trekanter skal beveges, eller om det er sentrum av en kule som skal forflyttes.

Interfacet som gir klassene forflyttningsegenskapene, ser slik ut:

```
public interface PickableObject {
  public void setPosition(float pos[]);
  public float[] getPosition();
  public int[] getPixels();
  public int getLength();
  public byte[] getColor();
  public void setColor(byte col[]);
}
```

Det interne handlingsmønsteret til objektene er irrelevant for interaksjonen, som kun trenger å undersøke om objektet støtter det grensesnittet den behøver, for deretter å kalle de respektive funksjoner når brukeren velger og flytter på punktet som representerer objektet i 3D. Det er opp til visualiseringene å lage objekter som som implementerer grensesnittene.

```
public class BasicObject3D implements PickableObject {
  private int pixelIndex[];
  ...
  public BasicObject3D(int addCount, ...) {
    ...
  }
  public float[] getPosition() {
    return new float[]{position[0], position[1], position[2]};
  }
  public byte[] getColor() {
    return new byte[]{color[0], color[1], color[2]};
  }
  public void setColor(byte col[]) {
    color=new byte[]{col[0], col[1], col[2]};
    pointArray.setColor(index, color);
  }
  public void setPosition(float pos[]) {
```

```
        position=new float[]{pos[0], pos[1], pos[2]};
        pointArray.setCoordinate(index, position);
      }
    }
```

3.9 Begrensninger

Dataskjermer er todimensjonale, i ICC3D arbeider man i tre dimensjoner og dermed oppstår det noen problemer. Det er ikke alltid like lett å kunne forflytte et punkt på skjermen som egentlig foregår i et 3D dimensjonalt rom. Derfor har vi lagt inn visse begrensninger og globale transformasjoner som skal gjøre arbeidet med å navigere punktene lettere.

Vi har lagt inn følgende begrensninger/constraints

- Free movement
- Constant hue
- Constant height
- Constant hue and height
- Color space center alignment
- Segement maxima center alignment

Free movement er akkurat det det står. Man kan fritt flytte punktet rundt, og det er ingen begrensning på forflyttningen. Denne bør kun brukes i sammenheng med de globale transformasjonene, eller hvis en har meget god forståelse av hvordan punktet beveger seg.

Constant hue begrenser punktets bevegelse slik at det kun beveger seg i det planet som gir konstant hue. Dette gjør at punktet beveger seg i det planet som går parallelt med L-aksen i lab fargerommet, men som forandre a og b verdi. Planet som går parallelt med L-aksen står vinkelrett på normalvektoren fra punktet mot L-aksen. Dette utgjør da selvfølgelig et vertikalt plan.

Constant height gjør at punktet kun kan bevege seg i et plan der høyden er konstant. Altså slik at L-verdien i lab fargerommet er konstant og punktet beveger seg fritt rundt i dette horisontale planet.

Constant hue and height begrenser punktets bevegelse ganske betraktelig, og gjør at punktet kun kan bevege seg i en linje. Denne linjen går med konstant L samtidig som den går parallelt med L-aksen. Retningen på denne linjen er da den samme som når man bruker constant hue.

Color space center alignment og Segment maxima center alignment er nyttig når man skal forandre farge-metningen (saturation, se **??** på side ??) og kontrasten på punktene/segmentene samtidig. Dette gjør at punktene kun kan forflytte seg mot sentrum av enten fargerommet eller segment maxima modellen. Den siste av dem krever selvfølgelig at du viser modellen i segment maxima. Denne linjen som du kan forflytte punktet i er normalvektoren mot det gitte sentrum.

3.10 Utstyrsgamut

Muligheter for å vise forskjellige typer gamut tilhørende enheter blir en sentral del av applikasjonen, siden dette danner grunnlaget for brukerens avgjørelser vedrørende flytting av bildepunktene. Det er viktig at brukeren kan se hvilke områder i bildet som det vil bli problematisk å vise på en enhet, slik at han eller hun kan gjøre de endringene som vil forhindre at dette skjer.

Det er også naturlig at brukeren kan sammenligne to forskjellige enhetsgamuter, slik at det er mulig å oppdage de forskjellene som finnes mellom de to. For at dette skal kunne gjennomføres på en fornuftig måte, må man kunne benytte forskjellige visualiseringer på de elementene som er med i 3D-scenen for å skille disse fra hverandre. Vi har lagt inn valg for å gjøre objekter delvis gjennomsiktige, samt funksjoner som gir objektene farger som identifiserer ett objekt fra et annet.

I tillegg har vi lagt inn støtte for flere metoder for innhenting av de data som danner grunnlaget for konstrueringen av enhetens gamut. Dette dreier seg om et filformat som gir overflaten som en regulær struktur, men også et som kun gir en del fargeverdier i Lab som enheten kan representere. Disse formatene støttes av en del apparatur som måler enheters evne til å gjengi farger, og input til programmet er derfor tilgjengelig for flere typer utstyr.

I dagens databaserte samfunn har man blitt mer oppmerksom på problemene vedrørende fargereproduksjon på for eksempel printere, og det har dermed blitt vanlig at slike enheter kommer med sin egen ICC-profil i form av en fil som gir forskjellige konverteringsmuligheter mellom et utstyrsuavhengig fargerom, og det som blir benyttet av enheten. ICC-profiler kan også inneholde informasjon om enhetens fargegamut, og hvis dette er tilfelle er applikasjonen vår i stand til å hente ut data om dette.

Ved å la programmet ha muligheten til å inkludere et ubegrenset antall forskjellige enhetsgamuter med ulike filformat og visualiseringer i samme 3D-visning, bør vi ha dekket de krav brukerne stiller til denne funksjonen.

Kapittel 4

Implementasjon / koding / produksjon

4.1 Grensesnitt / API

4.1.1 Java

Java ble utviklet på midten av nittitallet av Sun, og er i dag ett av de store programmeringsspråkene. Det benyttes i mange prosjekter i hele verden, og har rukket å få en stor mengde tilhengere.

Det er et objektorientert språk, og syntaktisk nokså lik C++. Det har utviklet seg masse i løpet av årenes løp, og innbefatter nå et solid API med et hav av funksjoner. I de senere utgavene, bl.a. swing pakken, fikk GUI'et seg en sold oppussing. Samtidig som overgangen til Java2 har rensket opp i ukonsekvente funksjonsnavn. Dermed har språket blitt mer oversiktlig.

Java er plattformuavhengig. Dette er løst ved å gi ut en JavaVM (Java Virtual Machine), eller runtime, for nesten samtlige plattformer tilgjengelig i dag. Alle Java programmer kjører på denne VM'en og trenger derfor ikke å rekompileres fra plattform til plattform. Grunnet denne interpreteringen som skjer må man også måtte regne med en hvis hastighetsreduksjon i forhold til språk som blir kompilert helt ned til maskinkode (c, c++, pascal etc.), men Java klarer allikevel å holde et forbausende høyt ytelsesnivå i forhold til disse.

Språket er sterkt knyttet opp i mot Internett. Ved å lage Java applets (som kjøres i webserveren) eller servlets (som ligner mer på cgi) kan man få dynamiske websider. Mellom applets og vanlige programmer skiller det veldig lite, og samme kode kan kjøres både på web og på maskinen som et vanlig program. Dette gir programmererne store muligheter.

I de senere årene har det kommet alternativer til Java. Microsoft promoterer sin .net teknologi med C# (uttales C-sharp) og også andre firmaer som f.eks. norske Trolltech leverer utviklingsmiljøer som kan kjøres på flere plattformer (c++ widget settet Qt). Om Java klarer konkurransen får tiden viser, men det ser ut til at de har fått godt fundament. Særlig på serversiden har Java blitt meget populært.

4.1.2 Java3D

Java3D er et grensesnitt (API) for å visualisere og skape et sofistikert og interaktivt tredimensjonalt grafikk og lydsystem. Det er en standard utvidelse til Java 2 JDK (Java Developement Kit). Grensesnittet tilbyr et sett med høynivå byggestener for å lage og å manipulere 3D geometri og strukturer for å rendre dette. Java3D kan brukes i både applikasjoner og applets.

En Java3D applikasjon (eller applet) lager og manipulerer 3D objekter og plasserer disse i en scenegraf struktur. Denne scenegrafen er bygget som et tre (DAG) og den beskriver fullstendig innholdet av et virtuelt univers og hvordan dette skal rendres.[1]

Siden Java3D er i stand til å håndtere komplekse og prosessorkrevende scener skulle man tro at hastighet ville være et problem fordi Java blir interpretert via JVM. Dette er (til en viss grad) ikke tilfellet da Java3D igjen bruker eksisterende hardware akselerasjon gjennom andre lavnivå API kall til feks. OpenGL eller Direct3D. I tillegg finnes det funksjoner for optimalisering av scenegrafer dersom dette skulle være nødvendig eller situasjonen krever det. Dersom verken OpenGL eller Direct3D er støttet av hardware, vil den nødvendige beregningen foregå via software, noe som vil sette en betydelig lavere grense for kompleksiteten til en scene.

Java3D er i skrivende stund bare tilgjengelig for Windows i distribusjon fra Sun. Det finnes derimot en distribusjon for Linux laget/portet av Blackdown. Denne distribusjonen virker noe ustabil da den uten forvarsel avslutter en brukers session i X.[2]

Oppbygning av en scenegraf i Java3D

For å bygge en komplett scenegraf i Java3D er man nødt til å ha visse basisnoder tilstede i scenegrafen. Dette gjelder “view branch” delen. Den delen som beskriver hvordan og hvorfra scenen skal rendres i det virtuelle universet. Den andre delen består av noder som beskriver hvordan scenen skal se ut; hvilke objekter som henger sammen, hvordan de henger sammen og hvordan de påvirkes av hverandre, rotasjoner ol. Det finnes mange forskjellige typer noder. Noen er synlige og andre er usynlige. De usynlige nodene er der for å styre og kontrollere oppførselen til de synlige.

[1] Scenegrafen vil mange ganger se ut som en DAG (Directe Acyclic Graph, se terminologiliste). Dette er fordi noder ofte vil ha referanser til andre noder, men det vil fortsatt være et tre siden det ikke er snakk noen mor-barn releasjon mellom nodenen.

[2] NB: Dette er ut fra vår egen erfaring og prøving, og vi har ikke funnet noe dokumentasjon fra verken BlackDown eller andre om at dette er et generelt problem. Versjonen vi henviser til av Java3D fra BlackDown (1.3 beta1) er tross alt en beta, så det er mulig dette bare er et midlertidlig problem.

Ved å sette sammen og kombinere forskjellige noder vil man til slutt danne et virtuelt univers. I grafen er det bare bladnodene som vil være synlige. Noen av noden er:

- **VirtualUniverse**

  En VirtualUniverse node er en top-nivå container for alle scene grafer. Den består av et sett med Locale noder. En applikasjon eller applet kan ha flere VirtualUniverse noder, men de fleste trenger bare en.

- **Locale**

  En Locale node definerer en posisjon i en VirtualUniverse node, og fremstår som en container for en samling av BranchGroup subgrafer (branch graphs), ved denne posisjonen.

- **BranchGroup**

  Denne noden fremstår som en peker til roten av en scene graph branch; BranchGroup noder er de eneste nodene som kan legges inn under en Locale nodes sett med noder.

- **ViewPlatform**

  En ViewPlatform node kontrollerer posisjonen, orientasjonen og størrelsen til en Viewer node. Det er til denne noden (ViewPlatform) i scene grafen en View node refererer.

- **Viewer**

  Denne type node inneholder all informasjon som beskriver den fysiske og og virtuelle "tilværelsen" i et Java 3D univers.

- **View**

  En View inneholder alle parametre som trengs for å rendre en 3 dimensjonal scene fra en bestemt posisjon.

- **TransformGroup**

  Spesifiserer en enkeltstående spatial transformasjon, via en Transform3D node, som kan posisjonere, orientere og skalere alle dens barn.

- **Transform3D**

  Brukes til å utføre translasjoner, rotasjoner, skaleringer og klippe effekter.

- **Geometry**

  Denne noden er abstrakt og spesifiserer geometriinformasjonen som en Shape3D node behøver.

- **Shape3D**

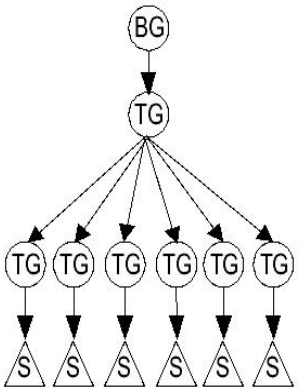
  Shape3D (blad)noden spesifiserer alle geometriske objekter. Den inneholder en liste av en eller flere Geometry komponent noder og en Appearance node. Geometry nodene definerer Shape3D nodens geometriske data.

- **Appearance**

  En node som spesifiserer informasjon som farge, materiale, tekstur osv.

Noder som arver fra Shape3D og Geometry er bla. de som danner de visuelle objektene, feks. en kube eller en kule eller andre mer komplekse objekter. Disse kan så settes sammen vha. en BranchGroup node slik at kuben og kulen logisk sett vil bli ett objekt (selv om de i virkeligheten er to adskilte objekter). På denne måten kan man styre begge objektene som ett. For at man skal kunne gjøre transformasjoner (skalering, rotasjon, forflytting osv), er man også nødt til å lage en TransformGroup node som kobles til (i dette tilfellet) den aktuelle BranchGroupen. Både BranchGroup-, og TransformGroup nodene er eksempler på usynlige noder. Videre kan man fra denne TransformGroupen lage en referanse til en Transform3D node som beskriver hvordan en evt. rotasjon vil bli utført på det aktuelle subtreet.

Figuren nedenfor viser hvordan flere enkle Shape3D(S) noder kan kobles sammen via TransformGroup(TG) noder og en BranchGroup(BG) node. Resultatet av dette treet vil kunne være en kube bygget opp av 6 vegger.



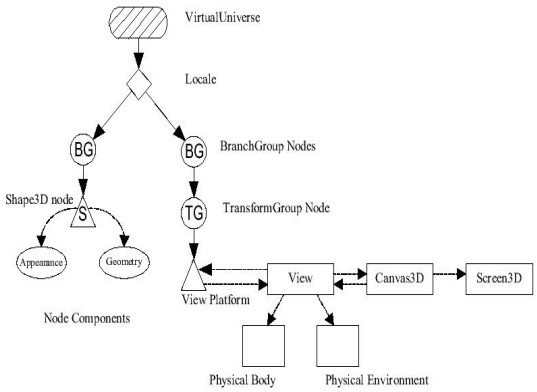
Figur 4.1: Eksempel på et BranchGroup subtre.

Dersom man vil forandre på utseende til et visuelt objekt, kan man gjøre det ved å legge inn en referanse fra den aktuelle noden til en Appearance node. Denne inneholder informasjon om hvordan objektet skal rendres.

Siden scenegrafen er bygget med en trestruktur, vil det til enhver tid bare finnes en vei (scenegraf path) fra rot og til en bladnode. Denne veien beskriver i helhet hvordan bladnoden vil se ut, oppføre seg osv. Under rendring traverseres treet fra rot til bladnodene og på denne veien vil all nødvendig informasjon om enkeltnoder eller subtrær være beskrevet. Hvordan treet traverseres bestemmes av Java3D, bortsett fra om man bruker visse typer noder som legger begrensninger på rekkefølgen.

I tillegg til å bygge den delen av grafen som beskriver hvordan objektene henger sammen og hvordan de oppfører seg(“content branch”), er man også nødt til å lage en “view branch”. Disse to subtrærne kobles så sammen via en Locale node. Et knutepunkt eller holdepunkt i den virtuelle verden, som siden kobles til en “Virtual Universe” node.

Figuren nedenfor viser hvordan et “Virtual Universe” i Java3D kan se ut. Her er både “content branch” og “view branch” tatt med:



Figur 4.2: Eksempel på et “virtuelt univers”.

4.1.3 JAI - Java Advanced Imaging

Java Advanced Imaging er et grensesnitt (API) fra Sun (med fler) for bruk i grafikk og bildehehandlingssystemer. Grensesnittet gjør det lett å utføre forskjellige og komplekse operasjoner for å manipulere bilder. Det er et kraftig verktøy som gir utviklere en god mulighet til å lage skalerbare og plattformuavhengige bildehandlingsverktøyer (både applikasjoner og applets). I skrivende stund finnes det versjoner for Windows, Linux og Solaris.

JAI fungerer ved å tilby et sett med objektorienterte grensesnitt som støtter høynivå programmeringsmodeller. Dette gjør det lett å implementere og utnytte JAI både i ny og i allerede eksisterende kode. Den er skrevet fullstendig i Java, men mange av operasjonene er optimalisert for raskest mulig prosessering (dette gjøres ved å benytte native code skrevet i C). Noen av funksjonene er også flertrådede for å utnytte kraften i et flerprosessorsystem. Hvilke operasjoner og hvor mye de er optimalisert kommer an på hvilken plattform JAI kjøres under.

JAI har støtte for en rekke kjente filformater som brukes innenfor bildebehandling. Versjon 1.1.1 (i skrivende stund den nyeste versjon og den vi har basert oss på gjennom prosjektet) har støtte for BMP, GIF, FPX, JPEG, PNG, PNM og TIFF. Det er ennå noen begrensninger innenfor enkelte av formatene mtp. om det bare er mulighet for lesing eller skriving (encoding eller coding respektive), men dette er noe vi regner med vil forsvinne i fremtidige utgivelser av JAI.

Grunnen til at vi har valgt å bruke JAI er fordi den gjør mye av jobben vi ellers ville vært nødt til å lage selv. Dette ville vært arbeidskrevende da vi ville vært nødt til å sette oss inn i de forskjellige filformatene og hvordan disse behandles. Det ville dermed pga. dette blitt mindre tid til selve oppgaven i prosjektet.

I prosjektet brukes JAI i hovedsak til å lese inn, lagre og vise bilder. Ut fra strukturen som bygges når et bilde leses inn via JAI, kan vi så hente ut relevant data og bygge vår

egen struktur for å representere disse dataene for videre behandling i ICC3D. Når så all manipulering (via ICC3D) er ferdig, snur vi prosessen ved å bygge opp strukturen for JAI igjen, og bruker så JAI til å lagre de aktuelle dataene. Dette gjør at vi slipper å tenkte på evt. komprimering, hvordan bildet skal representeres iht. det aktuelle filformatet osv. Dette er noe JAI vil ta seg av.

Vår oppfatning av JAI i denne sammenhengen har vært positiv, nettopp fordi den forenkler en del av rutinene vi er avhengig av for å få utført vårt arbeid definert i oppgaven.

4.1.4 JMF - Java Media Framework

Å slippe å skrive kode er en fin ting. Og i så henseende er det fint å kunne hente ned tilleggspakker som inneholder mye av den funksjonaliteten vi behøver. Vi har tatt med Java Media Framework for å kunne lagre 3D-modellen som film.

JMF tilbyr muligheten for lyd og video opptak, avspilling og streaming. Den utvider multimedia aspektet til Java ganske kraftig, og gir utviklere et solid fundament man kan arbeide videre på. JMF API'et er utviklet av Sun, IBM, Silicon Graphics og Intel Corp.

Med JMF slipper vi å sette oss inn avanserte videokomprimerings teknikker, og vi kan tilby et mye større spekter. Isteden for kun å basere seg på en komprimeringsmetode, kan brukeren velge mellom alle de "codecene" som ligger på tilgjengelig på den respektive maskinen. Dette gjelder også DivX og MPEG4.

I vår Implementasjon har vi ikke med lyd eller streaming, da dette ikke har noen hensikt.

4.2 Plattformer

Grunnet forskjellige plattformenes oppbygging er det også andre ting som må ta hensyn til, dette er nærmer beskrevet under.

4.2.1 Microsoft Windows

Windows versjonen av Java3D kommer i to forskjellige versjoner, en for OpenGL og en for DirectX. Både vi og Sun anbefaler at windows brukere installere OpenGL versjonen. Dette med hensyn på at opptegningen av selve 3D canvaset ikke er helt likt for begge to. Siden vi under utviklingen av programmet baserte oss på å bruke OpenGL versjonen, er ICC3D knyttet såpass tett opptil denne at bruk av DirectX versjonen vil gi uheldige resultater. En vil kunne oppleve at punkter som skulle være sirkler vil blir opptegnet som prikker o.l.

4.2.2 Linux

Som det meste i Linuxverdenen er ting litt uferdig, og lite støttet av de store firmaene. I våre tilfeller gjelder dette Java3D og Java Media Framework, der begge disse ikke lages

av Sun selv, men firmaet Blackdown. Disse er kun laget for maskiner med Intel kompatible prosessorer. Dette resulterer i noe ustabile versjoner av disse to bibliotekene og kan gi brukeren lite hyggelige opplevelser. Vi har ved en god del tilfeller opplevd at ved lukking av 3D canvaset har den tatt med seg hele X-Windows systemet, og i enda mer kritiske tilfeller faktisk logget ut brukeren. Dette er selvsagt noe ingen ønsker å oppleve, og det kan se ut at problemet har blitt mindre ved Java2 1.4.0, XFree 4.2.0 og bruk av direct rendering i X. Vi kan allikevel ikke garanterer at problemet har forsvunnet. JMF'en ser heller ikke ut til å fungere helt tilfredsstillende, uten at vi umiddelbart har noen løsning på hvordan man skal ordne dette problemet.

Andre ting som kan virke forstyrrende ved bruk av Linux er at kontroll vinduet kan ende opp med feil størrelse i forhold til windows versjonen, og kan være vanskelig å få i riktig størrelse. Andre problemer er at noen av de såkalte Windows Manager'ene i Linux (dette er da snakk om WindowMaker, KDE, Sawfish, Enlightenment, Afterstep osv.) har en tendens til å ta i mot vindus informasjon på forskjellige måte. De kan til tider finne ut å minimere hele kontroll panelet. Dette har oppstått når noen lukker 3D canvaset, for å så åpne ett nytt (frustrasjonen blir her stor når man ikke får tak i kontrollpanelet). ICC3D gjemmer automatisk kontroll panelet når det ikke gjenstår noen 3D canvaser på skjermen, og skal i teorien ta det fram igjen når det åpnes et nytt canvas 3D objekt. Her kan det som sagt at panelet ikke dukker opp igjen, men derimot ligger minimert. Hvis man er klar over dette går det bra, det er da bare å maksimerer dette vinduet. Dette er noe vi ikke kan gjøre noe med.

4.2.3 Apple Macintosh

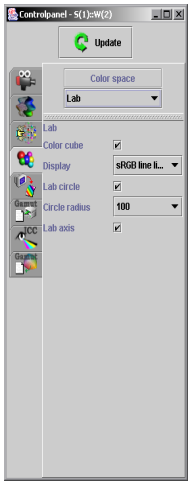
I skrivende stund finnes det ingen versjon av Java3D for Mac. Sun har lovt at det snart skal komme en versjon for OS X (det siste operativsystemet for Mac, ikke bakoverkompatibelt med tidligere utgaver) en gang om ikke alt for lenge. Vi kan derfor ikke si hvordan denne versjonen vil fungere.

Et annet problem med Mac er at de kun har en museknapp, dette gjøre at navigeringen i 3D canvaset vil bli vanskelig (zoom, forflyttning etc). Vi baserer oss på at brukeren har tre knapper, evt to der den tredje blir emulert (ved å trykke begge samtidig), men ved en knapp må programmet skrives om for å få det til å funke tilfredsstillende.

4.2.4 Foretrukket system

Alt i alt vil Microsoft Windows være den plattformen vi vil anbefale å bruke. Vi har riktignok ikke prøvd ICC3D på andre Unix varianter (vi har ikke hatt tilgang til slike maskiner), og det er grunn til å tro programmet vil fungere tilfredsstillende på SUNs eget Solaris system. Programmet vil fungere på Linux, men man må være obs på et par småting og ha riktig prosessortype, vi håper og tror at programmet snart er kjørbart på Macintosh systemer, men for øyeblikket er det altså Windows plattformen som gjør den beste jobben. Er Java plattformuavhengig ? tja, si det...

4.3 Fargerom



Figur 4.3: Kontrollpanel og valg tilknyttet fargerom.

4.3.1 Generelt

Ut fra den gitte strukturen beskrevet i design av fargerom på side 22, vil alle nye fargerom være nødt til å arve fra baseklassen **ColorSpace3D**.

Nedenfor vises et utsnitt av denne baseklassen for å illustrere oppbygningen.

```
public abstract class ColorSpace3D {
   //...
   public ColorSpace3D() { }
   public abstract float[] toCIEXYZ(float color[]);
   public abstract float[] fromCIEXYZ(float color[]);
   public abstract byte[] toSRGB(float color[]);
   public abstract float[] getPosition(float color[]);
   public abstract BranchGroup getAxis();
   public abstract String getName();
   public abstract Component getParameterPanel();
   public abstract Transform3D getTransform();
   //...
}
```

4.3.2 Utvidelse

Vil man utvide programmet med et nytt fargerom, feks. Luv, vil dette kunne gjøres ved å lage en ny klasse som arver fra **ColorSpace3D** slik:

```
public class LuvColorSpace3D extends ColorSpace3D {
```

```
   // lokale variable
   public LuvColorSpace3D() { /* ... */ }
   // implementer abstrakte funksjoner fra ColorSpace3D
   // samt evt. andre lokale funksjoner
}
```

Legg så **LuvColorSpace3D** klassen på rett sted i katalogstrukturen (icc3Dpackage/colorspaces) og start programmet på nytt. Programmet vil nå være i stand til å manipulere og visualisere bilder i Luv fargerom.

4.3.3 Konvertering av fargeverdier

Konverteringen til og fra de forskjellige fargerommene implementeres gjennom funksjonene

```
public abstract float[] toCIEXYZ(float color[]);
public abstract float[] fromCIEXYZ(float color[]);
public abstract byte[] toSRGB(float color[]);
```

Som er deklarert i **ColorSpace3D**.
Det er dermed opp til den enkelte klasse å skrive den aktuelle koden som sørger for denne konverteringen som i seg selv er spesifikk for hvert fargerom.

Hvordan foregår konverteringen

Et vesentlig punkt er at vi alltid går veien om CIEXYZ for å konvertere fargeverdier. Dette gjøres ved at vi benytter oss av klassen **ColorSpaceConvertor** vist nedenfor:

```
public abstract class ColorSpaceConvertor {
   public static float[] convert( float color[],
                                  ColorSpace3D from,
                                  ColorSpace3D to) {
      if (from.getName().equals(to.getName())) {
         float retColor[]=new float[color.length];
         System.arraycopy(color, 0, retColor, 0, color.length);
         return retColor;
      }
      return to.fromCIEXYZ(from.toCIEXYZ(color));
   }
}
```

Denne klassen inneholder bare en statisk funksjon **convert** som tar imot:
1. fargen som skal konverteres
2. hvilket fargerom den skal konvertere fra
3. hvilket fargerom den skal konvertere til

Dersom man vil konvertere en farge spesifisert i sRGB til Lab vil følgende kode utføre dette:

```
SRGBColorSpace srgb = ColorSpaceManager.getColorSpace("sRGB");
LabColorSpace lab = ColorSpaceManager.getColorSpace("Lab");
float sRGBColor[] = new float[]{0.5f, 0.5f, 0.5f};
float labColor[] = ColorSpaceConverto.convert(sRGBColor, srgb, lab);
```

Det som vil skje når convert kalles, er at denne vil igjen kalle de aktuelle funksjonene i srgb og lab for konvertering. Disse funksjonene vil så konvertere til og fra CIEXYZ med funksjonene toCIEXYZ(...) og fromCIEXYZ(...) for srgb og lab respektive. Mao. vil srgb konverterer fargen til CIEXYZ som vil bli parameter til lab sin fromCIEXYZ(...) som til slutt vil returnere angitt farge i Lab fargerom.

4.3.4 Formler / kode for konvertering

De følgende avsnittene vil gi en detaljert beskrivelse av formlene som benyttes for konverteringen i de enkelte fargerommene.

SRGB_ColoSpace3D

Alle formler gjelder for hvitpunkt = D50.
RGB (RGB') henviser til de enkelte R, G og B (R', G' og B') verdiene hver for seg.

4.3.4.1 sRGB → sRGB

Benyttes av SRGB_ColorSpace.toSRGB(...).

Konverteringen av sRGB til sRGB vil være å se til at *RGB* verdiene ligger i rett intervall: $0.0 <= RGB <= 1.0$. Dersom dette ikke er tilfellet vil de bli klippet til enten 0.0 eller 1.0 respektive. Deretter vil de bli skalert for å få en verdi i intervallet: $0 <= RGB <= 255$.

if $(RGB < 0.0)$

$RGB' = 0.0$

if $(RGB > 1.0)$

$RGB' = 1.0$

$RGB' = RGB * 255.99$

(Grunnen til at en verdi som 255.99 benyttes har å gjøre med fordelingen av verdier mellom 0-255)

4.3.4.2 sRGB → CIEXYZ

Benyttes av SRGB_ColorSpace.toCIEXYZ(...).

Konverteringen kan gjøres i 2 steg:

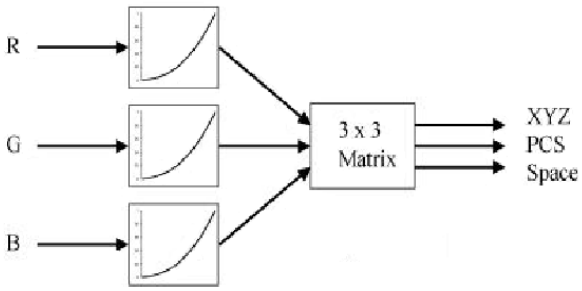
1. Forsikre seg om at verdiene til *RGB* er korrekte. De enkelte R, G og B verdiene går derfor gjennom en sjekk som sørger for dette.

   if $(RGB <= 0.03928)$
   $RGB' = \frac{RGB}{12.92}$
   else
   $RGB' = (\frac{(0.055+RGB)}{1.055})^{2.4}$

2. Deretter utføres følgende matrisemultiplikasjon mellom 3x3 matrisen for sRGB profilen og *RGB* verdiene.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4361 & 0.3851 & 0.1431 \\ 0.2225 & 0.7169 & 0.0606 \\ 0.0139 & 0.0971 & 0.7141 \end{bmatrix} * \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Resultatet av følgende operasjoner vil være fargeverdier spesifisert i CIEXYZ fargrom. Vises vha. figuren nedenfor:



Figur 4.4: Konvertering av fra sRGB til CIEXYZ fargerom.

4.3.4.3 CIEXYZ → sRGB

Benyttes av SRGB_ColorSpace.fromCIEXYZ(...).

Konverteringen utføres på samme måte som for toCIEXYZ(...) men i motsatt rekkefølge og med formlene invers:

1. Første må man finne inversmatrisen til

$$\begin{bmatrix} 0.4361 & 0.3851 & 0.1431 \\ 0.2225 & 0.7169 & 0.0606 \\ 0.0139 & 0.0971 & 0.7141 \end{bmatrix}$$

   som gir følgende matrise:

$$\begin{bmatrix} 0.4361 & 0.3851 & 0.1431 \\ 0.2225 & 0.7169 & 0.0606 \\ 0.0139 & 0.0971 & 0.7141 \end{bmatrix}^{-1} = \begin{bmatrix} 3.1336 & -1.6168 & -0.4907 \\ -0.9786 & 1.9160 & 0.0335 \\ 0.0720 & -0.2290 & 1.4053 \end{bmatrix}$$

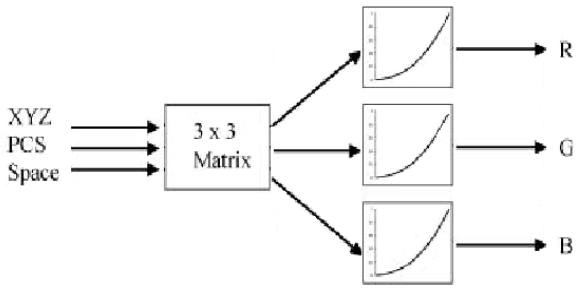
Videre må man utføre samme operasjon som for sRGB→CIEXYZ i pkt. 2, men nå med *RGB* verdier som resultat:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 3.1336 & -1.6168 & -0.4907 \\ -0.9786 & 1.9160 & 0.0335 \\ 0.0720 & -0.2290 & 1.4053 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

2. Neste steg er å gjøre en tilsvarende sjekk som for sRGB→CIEXYZ i pkt. 1:

   if ($RGB <= 0.0030402$)
   $RGB' = RGB * 12.92$
   else
   $RGB' = 1.055 * RGB^{\frac{1}{2.4}} - 0.055$

Resultatet av følgende operasjoner vil være fargeverdier spesifisert i sRGB fargrom. Vises vha. figuren nedenfor:



Figur 4.5: Konvertering fra CIEXYZ til sRGB fargerom.

LabColorspace3D

4.3.4.4 Lab→sRGB

Benyttes av LabColorSpace.toSRGB(...).

Bruker **ColorSpaceConvertor** for å utføre konverteringen. Denne vil så bruke SRGB-ColorSpace.toSRGB(...) for videre konvertering. (Se SRGB_ColorSpace.toSRGB(...) på side 41)

```
public byte[] toSRGB(float color[]) {
   ColorSpace3D rgbSpace=ColorSpaceManager.getColorSpace("sRGB");
   return rgbSpace.toSRGB(ColorSpaceConvertor.convert(color, this, rgbSpace));
}
```

4.3.4.5 Lab→CIEXYZ

Benyttes av LabColorSpace.toCIEXYZ(...).

Konvertering av Lab verdier til CIEXYZ verdier kan utføres i 3 steg:
Definerer L, a, b som inputvariable og X, Y, Z som outputvariable. Xx, Yy og Zz er midlertidige variable.

1. $\begin{array}{ll} Yy = & \frac{(a*100)+16}{116} \\ Xx = & (\frac{(b-0.5)*256}{500}) + Yy \\ Zz = & Yy - (\frac{(L-0.5)*256}{200}) \end{array}$

2. Følgende utføres for både Xx, Yy og Zz hvor respektive er substituert med f .
   if $(f^3 > 0.008856)$
   $f = f^3$
   else
   $f = \frac{f-(\frac{16}{116})}{7.787}$

3. $\begin{array}{ll} X = & Xx*0.9642 \\ Y = & Yy*1.0 \\ Z = & Zz*0.8249 \end{array}$

4.3.4.6 CIEXYZ→Lab

Benyttes av LabColorSpace.fromCIEXYZ(...).

Konvertering av CIEXYZ verdier til Lab verdier utføres på samme måte som for to-CIEXYZ(...) men i motsatt rekkefølge og med formlene invers:
Definerer X, Y, Z som inputvariable og L, a, b som outputvariable. Xx, Yy og Zz er midlertidige variable.

1. $\begin{array}{ll} Xx = & \frac{X}{0.9642} \\ Yy = & \frac{Y}{1.0} \\ Zz = & \frac{Z}{0.8249} \end{array}$

2. Følgende utføres for både Xx, Yy og Zz hvor respektive er substituert med f .

   if $(f > 0.008856)$
   $f = f^{\frac{1}{3}}$
   else
   $f = (7.787 * f) + \frac{16}{116}$

3. $L = \frac{200*(Yy-Zz)}{256} + 0.5$
   $a = \frac{(116-Yy)-16}{100}$
   $b = \frac{500*(Xx-Yy)}{256} + 0.5$

XYZ_ColorSpace3D

Konverteringen i denne klassen vil rett og slett være å returnere den samme fargeverdien som skal konverteres siden begge er i XYZ fargerom, bortsett fra CIEXYZ→sRGB. Det som gjøres er å kopiere fargeverdiene i inputarrayen over i en ny variabel, og så returnere denne. Bruker optimalisert funksjon for raskest mulig kopiering av array.

4.3.4.7 CIEXYZ→sRGB

Benyttes av XYZ_ColorSpace.toSRGB(...).

Bruker **ColorSpaceConvertor** for å utføre konverteringen. Denne vil så bruke SRGB-ColorSpace.toSRGB(...) for videre konvertering. (Se SRGB_ColorSpace.toSRGB(...) på side 41)

```
public byte[] toSRGB(float color[]) {
   ColorSpace3D rgbSpace=ColorSpaceManager.getColorSpace("sRGB");
   return rgbSpace.toSRGB(ColorSpaceConvertor.convert(color, this, rgbSpace));
}
```

4.3.4.8 XYZ→CIEXYZ

Benyttes av XYZ_ColorSpace3D.toCIEXYZ(...).

```
public float[] toCIEXYZ(float color[]) {
   float retColor[]=new float[color.length];
   System.arraycopy(color, 0, retColor, 0, color.length);
   return retColor;
}
```

4.3.4.9 CIEXYZ→XYZ

Benyttes av XYZ_ColorSpace3D.fromCIEXYZ(...).

```
public float[] fromCIEXYZ(float color[]) {
   float retColor[]=new float[color.length];
   System.arraycopy(color, 0, retColor, 0, color.length);
   return retColor;
}
```

4.4 2D

Implementasjon

I 2D delen har vi benyttet oss av kode laget av Sun. Dette gjelder selve thumbnail viewen som ligger fritt for nedlasting på Suns hjemmeside. Vi følte at denne klassen gav oss det resultatet vi ønsket oss, og så ingen grunn til å måtte lage dette selv (hvorfor finne opp hjulet på nytt ?).

Når det gjelder selve oppdateringen av bilde har vi forsøkt å få dette til å gå så kjapt som mulig. Vi har tatt i bruk et såkalt Memory Image Source som gir oss direkte tilgang til pixlene som en integer array, og vi kan på en enkel måte oppdatere de ønskede pixlene (f.eks. pixel ved 200x300 = array[300*width+200]). Vi har allikevel ikke klart å levere såpass at bilde blir oppdatert mens man flytter musa, men her er det hastigheten til Java som er begrensningen. Vi ser oss derfor fornøyd med at oppdateringen er rimelig kjapp etter at museknappen slippes. Det er verdt å merke seg at kun de pikslene som har forandret seg blir oppdatert i bildet.

Canvas 3D objektet har fått direkte tilgang til informasjonen i hvert sitt respektive image2D objekt, og å oppdatere image2D objektet blir da en rimelig affære. Punktposisjonene i 3D view'en blir konvertert til farge verdier, v.h.a. en invers Lab konverteringsalgoritme.

4.5 Oppbygning av 3D struktur

4.5.1 Generelt

En av de sentrale delene av ICC3D er visualisering av data i 3 dimensjoner. Det er i denne delen hvor all manipulering av et data blir utført. Det har derfor vært en veldig viktig oppgave å strukturere/organisere den slik at det vil være lettest mulig å arbeide med en 3-dimensjonal representasjon.

I prosjektet har vi valgt å benytte oss av Java3D. Mye av jobben med å visualisere data vil derfor behandles av nettopp Java3D. Men selv om det er denne delen som hele tiden vil ligge til grunn for visualiseringen i 3D, vil det være vår oppgave å bygge den nødvendige strukturen som Java3D benytter seg av for å rendre det ferdige resultatet.

Java3D benytter seg av et mangfoldig sett med objekter/noder for å bygge en korrekt struktur. Det har derfor vært en nødvendig del av prosjektet å sette seg inn i hvordan denne strukturen bygges.

4.5.2 Struktur

Alle visualiseringene er bygget opp rundt et hierarki, med **Visualization3D** som abstrakt baseklasse. Hierarkiet er vist nedenfor:

Baseklasse	*Virksomhetsklasse*
Visualization3D	
	NoneVisualization3D
	StandardVisualization3D
	DetailedVisualization3D
	SegmentMaximaVisualization3D
	AlphaShapesVisualization3D
	ConvexHullVisualization3D

Figur 4.6: Arvehierarki for visualiseringer.

For å bygge den nødvendige strukturen benytter vi oss av forskjellige type objekter/noder alt ettersom hvilken representasjon som er ønskelig. Feks. om det skal vises mange små objekter eller et solid objekt. Det vil derfor være den enkelte visualisering sin oppgave å bygge den nødvendige strukturen som representerer de aktuelle data.

4.5.3 Implementasjon

Den abstrakte baseklassen **Visualization3D** deklarerer (blant annet) en funksjon

```
public abstract BranchGroup[] createSceneObject(...);
```

som må defineres og implementeres i alle klasser som arver fra denne. Det er denne funksjonen som vil være ansvarlig for å bygge opp strukturen for den aktuelle visualisering.

Denne funksjonen vil returnere en eller flere BranchGroup noder i en array alt ettersom hvordan representasjonen bygges opp og hva som skal være med. Dette bestemmes også i den aktuelle visualisering og kan være parametre som beskriver oppbygningen.

Resultatet av dette vil bli en BranchGroup node som hektes inn på rett sted i Java3D sin struktur slik at den kan rendres.

Under denne oppbygningen vil også andre type noder legges inn i aktuelle subtrær. Dette er noder som beskriver hvordan objektet skal se ut og hvordan det skal oppføre seg ved en eventuell rotasjon ol.

I figuren nedenfor er det vist hvordan hierarkiet (fra hovedvindu til 3D modell) er organisert for å bygge en 3 dimensjonal modell av et 2D bilde:

Klasse	Funksjon	Beskrivelse
ICC3D		
|		
MainGui	createNewSession(...)	Starter en ny session fra et 2D bilde
|		
Session	show3D(...)	Initialiserer opprettelse av 3Dmodell fra 2D bildet
|		
Image3D	createBG(...)	Image3D vil representere 3D modellen av bildet og kaller nødvendig funksjon i
|		
ImageManager	getBranch(...)	ImageManager som Image3D har referanse til. ImageManager vil til enhver tid vite hvilken
|		
[aktuell visualisering]	createSceneObject(...)	visualisering som er aktiv og kaller så aktuell funksjon i denne for å fullføre opprettelsen av 3D modellen.

Tabell 4.3: Funksjonsoppbygning av 3d struktur.

4.6 Parsere

4.6.1 Generelt

En parser er et program eller en del av et annet program som har til hensikt å lese og tolke en eller flere filformater.

I dette prosjektet finnes det i hovedsak tre parsere. Dette er:

- Parser.Java
- Parser2.Java
- IccReader.Java
  - IccHeader.Java
  - LutReader.Java

De to første er frittstående, mens sistnevnte bruker også andre filer for å parse en gitt fil. Perserne blir benyttet til å hente ut relevante opplysninger og bygge en veldefinerte strukturer av innlest data slik at andre deler av koden vil kunne bruke disse i sin behandling.

Informasjonen fra disse filene er av meget stor interesse, da en stor del av oppgaven til ICC3D baserer seg nettopp på gamutdata og ICC profiler fra diverse enheter for å utføre nødvendig manipulering av bilder.

4.6.2 Filformater

4.6.2.1 IT8.7/2 - Parser.Java

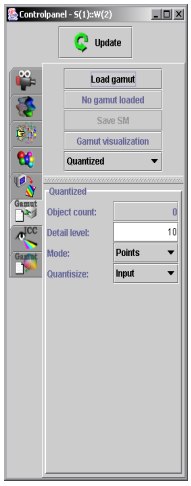
Denne blir brukt til å lese filer på et format beskrevet som **IT8.7/2**. Disse filene inneholder fargedata fra tester utført på forskjellige enheter. Dette kan være enheter som feks. printere, scannere, monitorer osv. Data fra disse testene brukes for å bestemme gamut området til den enkelte enhet, det vil si hvilke fargeverdier den er i stand til å gjengi.

Testene utføres ved at en rekke fargeverdier tilføres den enkelte enhet, verdiene som enheten reproduserer måles og sammenliknes med de tilførte. Ut fra disse resultatene vil det være mulig å bestemme om enheten har gjengitt de aktuelle verdiene korrekt eller ikke. De verdiene som gjengis korrekt vil være med på å bestemme enhetens gamut.

Filformatet spesifiserer data vha. en bestemt syntaks. Som regel er både XYZ og Lab fargeverdier tilgjengelige fra disse filene. Nedenfor er et eksempel på en slik fil og dens struktur/syntaks:

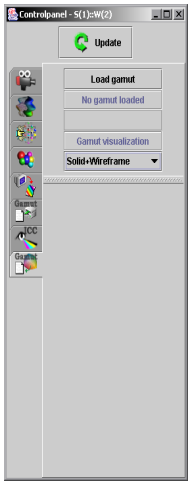
```
IT8.7/2
ORIGINATOR "FotoTune"
DESCRIPTOR "Agfa IT8"
MATERIAL "Agfacolor"
MANUFACTURER "Agfa"
CREATED "Wed Oct  7 07:43:00 1998"
PROD_DATE "1998:010"
SERIAL "5x7 c80731xx"
NUMBER_OF_FIELDS 9
BEGIN_DATA_FORMAT
SAMPLE_ID  XYZ_X  XYZ_Y  XYZ_Z  LAB_L  LAB_A  LAB_B   LAB_C  LAB_H
END_DATA_FORMAT
NUMBER_OF_SETS 288
BEGIN_DATA
A1     2.93    2.51    1.90   17.93    9.69    1.58    9.82    9.23
A2     3.46    2.43    1.55   17.64   19.94    4.86   20.53   13.70
A3     4.77    2.97    1.57   19.95   28.67    8.62   29.94   16.74
A4     5.35    3.28    1.66   21.12   30.75    9.59   32.22   17.31
.
.
.
GS20    1.13    1.16    0.97   10.26    0.58   -0.13    0.60  347.67
GS21    0.94    0.96    0.76    8.66    0.44    0.49    0.67   47.83
GS22    0.78    0.80    0.72    7.18    0.76   -1.21    1.43  302.47
GS23    0.47    0.48    0.66    4.36    0.25   -5.06    5.07  272.82
END_DATA
```

Parseren henter i skrivende stund bare ut Lab verdiene fra disse filene for videre prosessering.



Figur 4.7: Kontrollpanel og valg tilknyttet gamut for IT7.8/2 formatet.

4.6.2.2 Regulært - Parser2.Java



Figur 4.8: Kontrollpanel og valg tilknyttet regulært filformat.

Denne parseren brukes for å lese overflatedata av en gitt struktur i 3D. Dvs. ytterpunktene av det visuelle objektet.

De aktuelle dataene som representerer overflaten er gitt i et filformat som er organisert etter strukturen vist nedenfor:

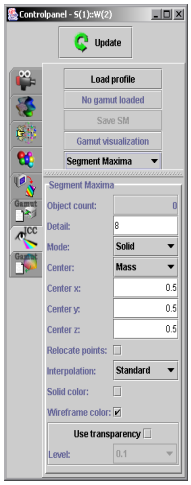
```
[INDEX NR] [R] [G] [B] [LAB_L] [LAB_A] [LAB_B]
```

Dette formatet blir også benyttet av andre programmer noe som gjør ICC3D i stand til å operere sammen med disse for å visualisere data.
Eksempel på hvordan en fil av denne typen er bygd opp vises nedenfor:

```
0    255    255    255    93.9300    0.1500     -1.9100
1    255    255    223    93.1562    -1.5662    10.4350
2    255    255    191    92.5500    -2.9775    22.5925
3    255    255    159    91.9825    -3.9563    34.3625
.
.
.
725    0    0    95     21.4800    2.2362    -5.9725
726    0    0    64     21.7000    1.1975    -1.2000
727    0    0    31     21.2200    0.7887    0.4325
728    0    0    0      20.5500    0.5000    1.3300
```

Strukturen som denne parseren bygger brukes for å visualisere hvordan objektet som dannes ut fra gitte data vil se ut i 3D.

4.6.2.3 ICC profiler - IccReader.Java



Figur 4.9: Kontrollpanel og valg tilknyttet ICC profiler.

Oppgaven til denne og de tilhørende filene er å hente ut gamut data fra ICC profiler dersom dette er tilgjengelig i den aktuelle ICC filen. Alle ICC profiler vil nemlig ikke inneholde gamut data, og det vil dermed ikke være mulig å hente ut relevante data.

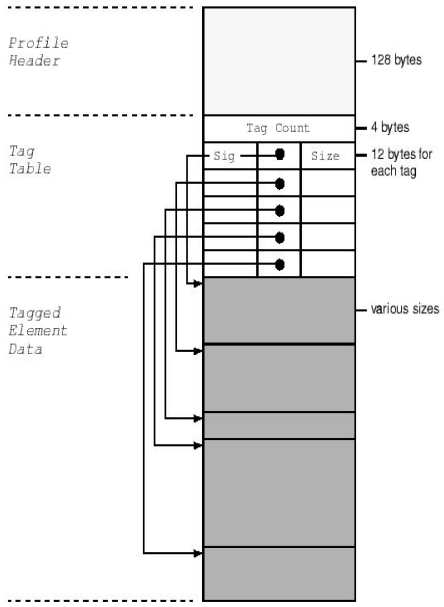
Selve gamut informasjonen ligger ikke lagret i profilene, men det vil være mulig å bestemme om en fargeverdi ligger innenfor eller utenfor gamutområdet til den aktuelle profilen. Dette gjøres ved at man “spør” profilen om en verdi ligger innenfor eller ikke.

For å finne ut dette, sender vi n antall forskjellige verdier inn til profilen (en oppgave som gjøres av LutReader), og får et svar tilbake som sier om angitt verdi ligger innenfor eller utenfor.

Basert på dette vil vi være i stand til å bestemme gamut området for den enkelte ICC profil ut fra den mengden med verdiene vi sender inn.

Oppbygning av ICC profil

ICC profiler er tag basert. Dvs. at informasjonen er lagret elementvis. Strukturen i en ICC profil er organisert slik:



Figur 4.10: Organiseringen i en ICC profil.

Tag tabellen vil virke som en innholdsfortegnelse for den gitte profil, og det er her man finner lokasjonen til de aktuelle data i profilen. Tabellen inneholder en tag signatur, offset adressen for begynnelsen og størrelsen til hvert enkelt taged element.

Elementene kan akseseres i tilfeldig rekkefølge gjennom hele profilen. Dette gjør at man vil spare mye prosesseringstid ved å først slå opp i tag tabellen før man leser data da mange ICC profiler vil være veldig store og kompliserte.

4.6.3 Eksportering av data

ICC3D har mulighet for å eksportere overflatedata fra en bestemt profil (ICC eller IT8.7/2) til et gitt filformat. Disse dataene beskriver ytterpunktene til objektet som dannes når angitte profiler er lest og visualisert med vha. Segment maxima.

Dette er en funksjon som grafikere ved HiG har benyttet seg av i sine beregninger/målinger. Bruken av denne funksjonen lettet deres arbeid, da de fikk en lettfattelig oversikt over aktuelle data som de kunne forholde seg til.

Filformatet er organisert etter følgende struktur:

```
Segment Maxima Information
Detail
[DETAIL]
Center L;Center a;Center b
[CENTER L];[CENTER A];[CENTER B]
L;a;b
[L][A][B]
[L][A][B]
...
[L][A][B]
[L][A][B]
```

4.7 Forandring av posisjon og zoom i 3D

For at brukeren skal kunne flytte på og rotere objektene som vises, lytter programmet etter musehendelser. Dersom man holder nede venstre museknapp og drar, vil man kunne rotere scenen. Høyreklikk velger objektene som kan flyttes i scenen, mens midtre museknapp zoomer inn og ut. I tillegg til de manuelle endringene, har vi realisert automatiske overganger til predefinerte synsfelt som befinner seg på forskjellige steder i forhold til scenens sentrum.

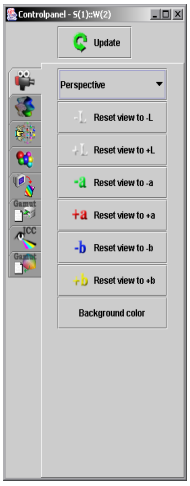
Alle translasjoner og liknende transformasjoner utføres av såkalte transform groups, og de overgangene vi har lagt inn fungerer ved at man interpolerer lineært mellom den matrisen som tilsvarer nåværende transformasjon, og den som gir den ønskede posisjonering. Vi henter ut transformasjonsmatrisen fra Java3D, lager den ønskede destinasjonsmatrisen og kaller en funskjon som foretar selve interpolasjonen. Pseudokode for denne operasjonen vil se omtrent slik ut:

```
void displayChange(Matrix source, Matrix destination) {
  Matrix change = destination - source;
  change/=numberOfFrames;
  for (numberOfFrames) {
    Java3Dscene.setTransform(source += change);
    sleep(numberOfMilliseconds);
  }
}
```

Mens applikasjonens hovedtråd sover, vil Java3D oppdatere 3D-visningen i en annen tråd.

Vi har også lagt inn mulighet for å skifte mellom bruk av perspektiv, det vil si en visning hvor objekter vil bli mindre desto lengre fra synspunktet de er, og ortografisk fremvisning som lar et objekt være samme størrelse på skjermen uansett hvor langt vekk fra kameraets posisjon det befinner seg.

Bakgrunnsfargen kan endres ved at brukeren velger fra en fargepalett, slik at man unngår eventuelle problemer med at deler av objektene kan forsvinne inn i bakgrunnen. Ved å sette de nødvendige tillatelser på Java3Ds scenestruktur, kan bakgrunnen oppdateres uten at hele scenegrafen må bygges på nytt.



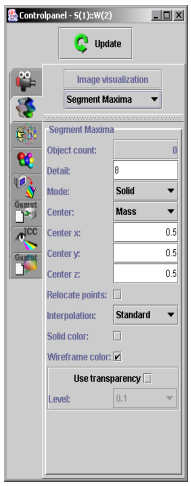
Figur 4.11: Kontrollpanel og valg tilknyttet synsvinkel.

4.8 Visning av aksesystemer i 3D

Vi har lagt inn muligheten for at man kan vise tekst som definerer aksene i Lab-fargerommet. Denne teksten settes inn på de relevante stedene i 3D, men må deretter roteres slik at den alltid sees rett forfra. For å sørge for dette benytter vi en metode kalt billboarding, som blir benyttet i ulike sammenhenger der 2D-objekter uten volum skal vises i 3D. Vi finner summen av alle transformasjonene som påvirker teksten, for så å invertere denne matrisen. Så fjerner vi alle translasjoner og skaleringer som ligger implisitt i matrisen, slik at det eneste som er igjen er rotasjonskomponenten. Denne matrisen definerer en ny transformasjon som legges over alle tekstkomponentene som skal vises i scenen. Det må bemerkes at selv om Java3D har en innebygd funksjon for denne typen oppgaver, var det ikke mulig for oss å benytte denne da den førte til en kontinuerlig oppdatering av 3D selv når det ikke skjedde endringer. Det er ikke nødvendig at en tråd skal benytte all tilgjengelig prosessorkraft til alle tider, og med vår metode blir transformasjonen kun beregnet på nytt etter at brukeren eller programmet har endret på synsvinkelen som er i bruk.

For å vise omfanget til fargerommet som bildet benytter, har vi lagd noen funksjoner som sørger for å vise den kuben som gir omrisset av rommet konvertert til fargerommet som benyttes i 3D. Vi har lagd en funksjon som konstruerer den nødvendige Java3D-strukturen gitt et start- og sluttpunkt for en linje med et visst antall inndelinger. Denne funksjonen blir kalt opp for hver av kantene i kuben, og lager objekter bestående av linjestykkene man får når man beveger seg langs linjen og konverterer posisjonene mellom fargerommene. På grunn av den generelle implementasjonen kan man også bruke funksjonen for å vise en mer detaljert oversikt over hele kubens interne oppbygning og overflate.

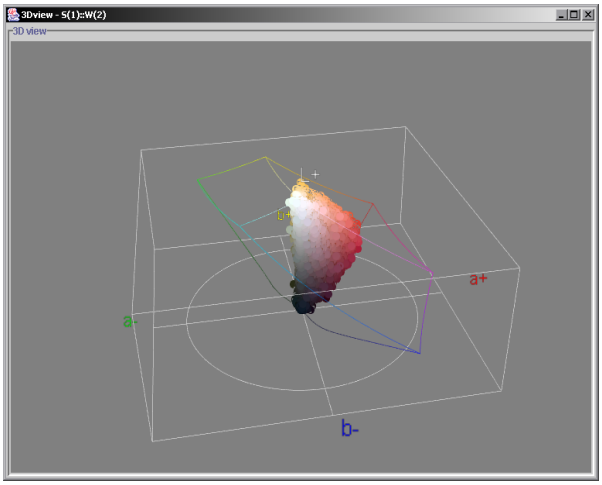
4.9 Visualiseringer



Figur 4.12: Kontrollpanel og valg tilknyttet visualiseringer.

4.9.1 Standard

Den første typen visualisering som vi implementerte i applikasjonen var en enkel sky av punkter. Selv om det allerede fra begynnelsen var åpenbart at dette ikke ville være veldig praktisk når man skal gjøre endringer på bildet, var det et naturlig startpunkt av den grunn at selve implementasjonen ville være enkel å gjennomføre. I tillegg ville denne visningsmodusen kunne brukes som et utgangspunkt for de andre modulene, slik at man for eksempel kunne begynne arbeidet på funksjonalitet for konvertering av farger.



Figur 4.13: Standard visualisering.

Ved å finne de unike pikselverdiene som er i bruk i bildet, og deretter å konvertere disse til det fargerom som skal danne basis for visningen, har man allerede beregnet de nødvendige koordinater for å vise pikselverdiene som punkter i 3D. Det var aktuelt å vise punktene som kuler, bokser eller punkter uten dybde. Etter en del eksperimentering viste det seg at det var totalt uaktuelt å benytte bokser eller kuler, da det i praksis var uholdbart å vedlikeholde den scene grafen som ville vært nødvendig. Derfor ble det til at vi bruker en enkel punktrepresentasjon der alle punkter har samme størrelse.

Brukeren har dermed muligheten til å se den virkelige spredningen av pikslene, noe som med fordel kan tas i bruk i tillegg til mer sofistikerte visualiseringer som forenkler måleverdiene eller på andre måter kan vise en form som ikke er representativ for bildet.

4.9.2 Kvantisert

Selv om punktskyen gir en meget god oversikt over bildets utbredning i 3D, vil det være naivt å tro at brukeren vil være i stand til å manipulere denne fremstillingen slik at det gir den ønskede virkningen på originalen. Et middels stort bilde med jevne fargeoverganger vil gi så mange punkter at brukeren blir overveldet, og det er ikke hensiktsmessig å flytte flere tusen punkter enkeltvis.

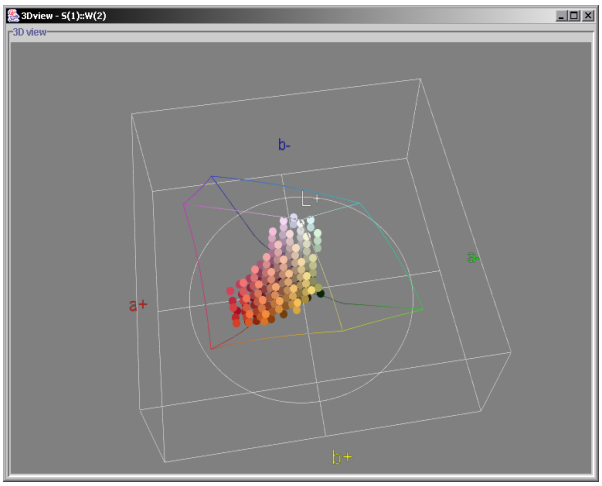
Dette førte til at vi kodet en kvantisert visualisering, som forenkler representasjonen slik at færre punkter vises i 3D. Kvantiseringen foregår ved at man deler fargerommet opp i kuber, der antallet kuber angis av brukeren ved å spesifisere antall oppdelinger langs hver av de tre aksene. Deretter går man igjennom alle punktene som opprinnelig representerer en eller flere piksler i bildet. For hvert punkt finner man hvilken kube som inneholder denne posisjonen, og markerer at det finnes piksler i den.

Etterpå lager man en ny 3D struktur ved å lage et nytt objekt for hver kube som er markert, slik at det nye punktet befinner seg midt i kuben. Dette medfører unøyaktigheter i form av at de nye objektene ikke nødvendigvis gir brukeren inntrykk av at spredningen av punktene er den samme, men det er nødvendig å foreta en slik operasjon for å unngå de store datamengdene som ellers oppstår. De objektene man sitter igjen med etter denne forenklingen kan så benyttes for å bygge opp en Java3D struktur. Vi har gitt brukeren valget mellom å vise punktene på samme måte som ved standardvisualiseringen, dvs. at hvert objekt vises som et punkt, og å representere hvert objekt som en kule med volum. Kulens volum vil være lineært avhengig av antallet piksler den er assosiert med i bildet, slik at kulens radius r vil være:

$$r = numberOfPixelsInCube^{\frac{1}{3}}$$

Dette gir brukeren et godt inntrykk av bildets fargedekning, da brukeren kun behøver å se på de ulike kulene for å se hvor utbredt fargen er i bildet. Det er imidlertid enkelte tilfeller hvor store kuler dekker mindre kuler som befinner seg i nærheten, noe som ikke er heldig. Dette gjelder spesielt når de mindre kulene befinner seg utenfor gamuten til en printer som man sammeligner med, da man ikke oppdager at pikslene som befinner seg ved den mindre kulen i realiteten er utenfor det fargespektrum som enheten kan vise. Det er nødvendig å begrense kulenes volum, slik at for eksempel kulen som representerer hvitt ikke blir altfor stor når man har et lyst bilde, mens en kule som har få piksler blir så liten at den nesten forsvinner.

Et annet problem som oppstår når man foretar denne oppdelingen i det fargerommet den vises i, er at enkelte av objektene kan bli utenfor utstrekningen til det fargerommet som bildet opprinnelig befinner seg i. Dette kan være misledende for brukeren, og det er ikke en heldig situasjon. Dette forbedrer man ved å la inndelingen foregå i det fargerom som bildet benytter, og deretter bringe de punktene man får ut av denne oppdelingen over i fargerommet som benyttes i 3D. Dette er en enkel modifikasjon, og denne valgmuligheten ble raskt implementert.



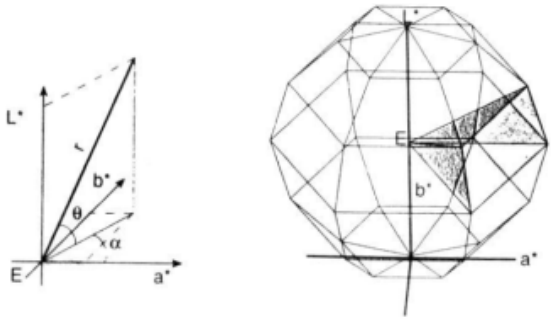
Figur 4.14: Kvantisert visualisering.

4.9.3 Segment maxima

Som en av våre visualiseringer har vi med segment maxima. Segment maxima går ut på å kunne gjengi omhyllingslegemet basert på segmenter, der det ytterste punktet i hvert segment representerer det punktet som det respektive segmentet skal ha på overflaten. Det er i denne modusen brukeren har de fleste iterasjonene og mulighetene for å endre på bildet han har åpnet. Ved enkle håndgrep kan man i segment maxima modus flytte på segmentene/punktene, velge interpolasjons metode og oppnå de resultatene man ønsker. En vil oppnå et høyre ønsket detaljnivå enn for eksempel ved å flytte på punkter i quantized modus, samt at man har muligheten for tetraeder interpolasjon. Segment maxima kan vises som kun solid, solid+wireframe eller kun wireframe. En kan også sette på gjennomsiktighet, noe som i det fulle og hele gir stor valgfrihet og mulighet for detalj justeringer av bildet. Det kan være greit å variere mellom disse mulighetene når en skal sammenligne fargerom.

4.9.3.1 Algoritmen

Det første som gjøres er å dele opp fargerommet i n*n segmenter, man finner så ut hvor disse segmentene befinner seg ved hjelp av alfa og theta vinklene ut i fra det punktet som er valgt som sentrum.



Figur 4.15: Segment maxima oppdeling

Deretter går man igjennom punktene og regner ut deres posisjon ved å bruke følgende formler (kulekoordinater) :

$$r = [(L - L_E)^2 + (a - a_E)^2 + (b - b_E)^2]^{1/2}$$

$$\alpha = \arctan\left(\frac{b - b_E}{a - a_E}\right)$$

$$\Theta = \arctan\left[\frac{L - L_E}{[(a - a_E)^2 + (b - b_E)^2]^{1/2}}\right]$$

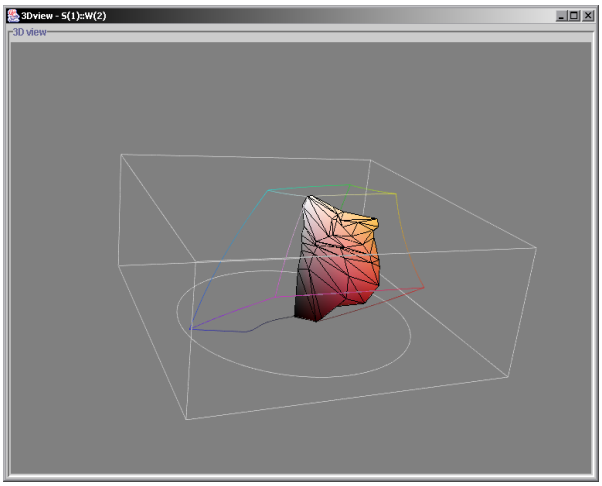
Her er r distansen punktet (fargen) har fra det gitte sentrum, α er hue vinkelen som strekker seg 360 grader rundt og Θ er vinkelen som går i et plan med konstant alpha (180 grader). E er definert for å være sentrum. Ut i fra dette er det enkelt å finne ut hvilket segment punktet befinner seg i, og lengden fram til det (som til slutt er det avgjørende).

Man tar så og lager trekanter mellom de punktene som har størst r i hvert segment (altså, lengst fra sentrum) og bygger således opp overflaten for legemet. Dette gjøres på følgende måte :

- Velg et ekstremal punkt (punkt 1)
- Dra en linje til ekstremalpunktet i segment ved siden av (punkt 2)
- Velg så ekstremalpunktet i segmentet under punkt 1, og dra en linje til punkt 2

- Forsett å gjør det samme fra punkt nummer 2, og når man har kommet rundt, går man ned et segment

Det oppnås særtilfeller ved topp og bunn punktene, der dras linjer fra samtlige punkter i den første segment rekken under toppunktet (eller over bunnpunktet) til topp eller bunn punktet. Man oppnår en vifte form som er nødvendig for å lukke legemet.



Figur 4.16: Segment maxima visualisering.

4.9.3.2 Unøyaktighet

Segment maxima metoden er ikke feilfri, og man vil komme opp med punkter som ligger lenger ute enn det legemet viser. Dette grunnet at man kun velger ett punkt i hvert segment. Ved flere punkter som ligger langt ute i et segment, og noe lenger inn i segment ved siden av vil de punktene som ligger mellom de to ekstremverdiene være på utsiden av det endelige legemet. Når alt kommer til alt vil nøyaktigheten til segment maxima ligge i hvor mange segmenter man deler legemet opp i (n - verdien), jo flere segmenter, jo mer nøyaktig, men sansynligheten for tomme segmenter øker.

4.9.3.3 Problemer

Etter hvert som arbeidet steg fram kom vi over en del spesielle situasjoner som det måtte gjøres noe med (det ble bl.a. et problem for tetraeder oppbyggingen). Nesten ved samtlige bilder fikk vi problemer med at trekanter fra forskjellige segmenter overlappet hverandre. Dette ble et særlig stort problem i topp og bunn, da dette punktet ikke nødvendigvis ligger rett overfor eller nedenfor det gitte sentrum. Dette er ikke definert i segment maxima metoden, og vi måtte derfor finne fram til en metode som ville fungere tilfredsstillende på egenhånd. Det ble eksperimentert med en del metoder, bl.a. å øke antall segmenter i topp og bunn, eller å slå sammen en del av segmentene så det ble færre punkter. Den endelige løsningen ble å dra topp/bunn punktet inn mot sentrum. Dette blir ikke optimalt med hensyn til at legemet skal være så korrekt som mulig, men

gir en tilfredsstillende løsning da denne denne forflyttningen ikke gjør noen store endringer på strukturen. Brukeren har dog muligheten til å velge om han vil gjøre det eller ikke.

4.9.3.4 Interpolering

Et annet problem er i de segmentene der det ikke eksisterer punkterer. Strukturen får da hull som er lite heldige. Vi gir her brukeren mulighet til å interpolere punkter som ikke eksisterer. Altså ut i fra områdene rundt, regne ut nye punkter som passer inn i strukturen, og gjøre overflaten hel. Det eksisterer for øyeblikket fire forskjellige interpoleringsmetoder i ICC3D. Disse valgene er standard, alternativ, sm fix radius og sm fix chroma. De to siste var opprinnelig et forsøk på å løse problemet skrevet om i forrige punkt, men viste seg å kun fungere delvis. Det de gjør er å interpolere de ikke eksisterende punktene med hensyn til chroma og radius

De to andre kanskje mer vanlige valgene er standard og alternativ interpolering. Det er ikke så mye som skiller disse to fra hverandre, da de har nokså lik oppbygging. Forskjellen er at alternativ interpolerer med alle punktene rundt seg, og hvis det ikke skulle finnes et punkt i et av segmentene, setter den dette punktet til å være i sentrum. Standard interpolering har en litt annen innfallsvinkel og velger å se bort i fra de punktene som ikke eksisterer. Den interpolerer m.a.o. kun med de punktene rundt som faktisk eksisterer.

4.9.3.5 Implementasjon

Segment maxima klassen er stor og tung, og er en av de største klassene vi har laget. Den har støtte for forskjellige antall theta og alpha segmenter, selv om dette ikke blir brukt. Vi har implementert segment maxima etter å nesten slavisk følge algoritmen beskrevet over, men en lettere modifisering av kalkuleringene måtte til for at vi skulle få ting til å passe.

Siden den arver fra Visualization3D bruker den samme oppbygging som de andre visualiseringene som også arver fra denne klassen. Hoveddelen av koden ligger i funksjonen

```
public BranchGroup[] createSceneObject(int numPoints,
                                       float pointCoord[][],
                                       int numEqual[],
                                       byte pointColor[],
                                       int pixelIndex[][],
                                       ColorSpace3D colorSpace3D[]) {....}
```

Her skjer selve oppbyggingen av 3D strukturen, den går igjennom samtlige punkter og finner radius til punktene og hvilket segment de ligger i. Alt dette er, som sagt tidligere, ut i fra det gitte sentrum. Hvis ikke midtpunktet blir gitt av bruker, blir dette kalkulert ut ved å bare regne middelverdien for alle punktene. Den tar i bruk en indeksert TriangleStripArray, som er en primitiv i Java3D, og tilbyr en sammenhengende triangel-struktur. Denne triangel-strukturen er lik den vi har behov for i oppgaven, og passet oss derfor meget bra.

Etter at alle punkter er kalkulert og gitt posisjoner settes alle attributter (farger, gjennomsiktighet etc). Hvis det er valgt noen annet enn kun solid, altså enten om det skal tas med en wireframe eller ikke, bygges dette også opp. Hvis en har valgt wireframe + solid, legges linjene utenpå selve solid modellen. Den legges såpass tett opptil resten av modellen, at det ikke skal være noe synlig mellomrom ved vanlig bruk. Grunnen til at vi ikke kunne legge den på samme plan som den andre modellen, er at den da ikke ville være synlig. Vi anser denne forskjellen for å være så lite at det ikke går utover troverdigheten til modellen.

Til slutt bygger den opp tetraeder strukturen som brukes ved tetraederinterpolasjonen. Denne oppbyggingen ser bl.a. slik ut:

```
// second top row
for (d=thetaDetail-3; d<thetaDetail-2; d++) {
   for (c=0; c<alphaDetail; c++) {
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2].setNeighbor(
         0,tetra[alphaDetail+(d-1)*alphaDetail*2+((alphaDetail+c-1)%alphaDetail)*2+1]);
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2].setNeighbor(
         1,tetra[alphaDetail+2*alphaDetail*(thetaDetail-3)+c]);
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2].setNeighbor(
         2,tetra[alphaDetail+(d-1)*alphaDetail*2+c*2+1]);
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2+1].setNeighbor(
         0,tetra[alphaDetail+(d-1)*alphaDetail*2+((c+1)%alphaDetail)*2]);
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2+1].setNeighbor(
         1,tetra[alphaDetail+(d-2)*alphaDetail*2+c*2]);
      tetra[alphaDetail+(d-1)*alphaDetail*2+c*2+1].setNeighbor(
         2,tetra[alphaDetail+(d-1)*alphaDetail*2+c*2]);
   }
}
```

Dette viser at nabolisten til tetraederne oppdateres ved å gå igjennom alle segmentene.

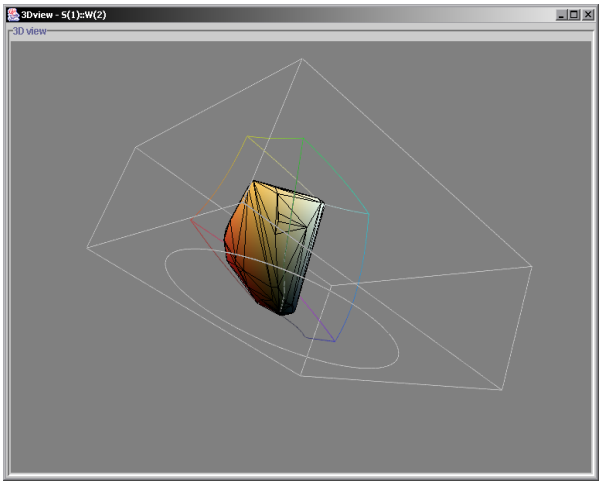
4.9.4 Convex Hull (3D)

Det vil alltid være interessant å finne en best mulig måte å representere en samling punkter som ett objekt lagd av trekanter i 3D. En måte å gjøre dette på, er å finne de trekantene som er slik at det ikke eksisterer punkter som befinner seg utenfor de planene trekantene danner. Disse trekantene danner det man kaller et konvekst legeme, altså et objekt som ikke har noen konkave områder der overflaten ser ut som en grop. Det finnes mange velbrukte algoritmer for å konstruere slike objekter, hvorav en av de mest brukte kalles Quick Hull. Som navnet indikerer regnes den for å være rask.

Quick Hull fungerer ved at man først konstruerer en simpleks, som i n dimensjoner er et objekt som består av n+1 punkter og n+1 overflatedeler. Flatene vil bestå av n punkter, som til sammen definerer et hyperplan med en gitt orientering. Deretter assosierer man hvert enkelt av de resterende punktene med den første av flatene hvis hyperplan dette punktet befinner seg over. Man kan her definere dette som synlighet, dvs. at et punkt kun kan se de flatene som det er på utsiden av. Alle andre flater vil ikke være synlig fra dette punktet, noe som kommer av at det opprinnelige legemet også nødvendigvis

vil være konvekst. Dersom punktene befinner seg på innsiden av alle hyperplan, kan man forkaste punktet da det ikke er en del av overflaten til det konvekse legemet. (Det befinner seg inne i objektet, og ingen flatedeler er synlig fra denne posisjonen.)

Etter at man har konstruert en simpleks av n+1 punkter og assosiert punktene med de initielle flatene, vil man gå gjennom alle de konstruerte flatene i den rekkefølgen de ble lagd. Dersom den gjeldende flaten er merket som forkastet, dvs. at den ikke lenger er en del av det konvekse legemet, vil det ikke være nødvendig å gjøre noe, ellers vil man se på flatens liste med assosierte punkter. Hvis denne listen er tom vil flaten være en del av det resulterende legemet, og man trenger heller ikke gjøre noe. En liste med punkter indikerer at disse punktene befinner seg på utsiden av flaten, og flaten vil dermed ikke være en del av det konvekse legemet. Man vil dermed markere flaten som forkastet, og finne det punktet i listen som befinner seg lengst vekk fra planet som flatepunktene danner. Det er deretter nødvendig å lage en ny liste med punkter, hvor man legger inn de andre punktene.



Figur 4.17: Convex hull visualisering.

Det ekstrempunktet man finner vil deretter være utgangspunktet for en gjennomgang av det nåværende objektets overflate. Man tar utgangspunkt i flatedelen som punktet var assosiert med, og undersøker deretter rekursivt naboene til denne flaten. Det vil derfor være hensiktsmessig å vedlikeholde en nabostruktur for legemets overflate. Dette rekursive søket finner deretter alle flater som kan sees fra ekstrempunktet, og merker disse som forkastet. Eventuelle assosierte punkter vil bli lagt til i listen over punkter som ikke lenger tilhører en flate. Når man finner en flatedel som ikke er synlig, tar man vare på den kanten av n-1 punkter som definerer overgangen fra den synlige flaten til den ikke synlige delen av objektet, samt hvilken flatedel som vil være nabo på den usynlige siden av kanten. Disse kantene vil utgjøre horisonten av det synlige området, og kalles derfor gjerne horisontkanter. Dersom man traverserer nabostrukturen på en angitt måte, vil man kunne finne horisonten i en veldefinert rekkefølge dersom dimensjonen er 3. Den anbefalte måten å gjøre dette på er å utføre et bredde først søk, der man hele tiden holder rede på fra hvilken flatedel man kom til den gjeldene flaten. Deretter undersøker man alltid de nærmeste flatenaboene først, slik at kantene som definerer horisonten finnes i etterfølgende orden. Dette kan være tidsbesparende ved at man slipper unødvendig arbeide for å konstruere den nye nabostrukturen.

Etter at man på denne måten har prosessert ekstrempunktet, vil man danne en ny overflatestruktur bestående av de delene av det opprinnelige objektet som ikke har blitt forkastet, samt nye flater som blir dannet av ekstrempunktet og hver av horisontkantene. Deretter finner man naboene til alle nye flater, samt oppdaterer nabopekerne til de flatene som ikke har blitt forkastet, men er en del av den synlige horisonten.

Man vil så gå gjennom listen over punkter som ikke er assosiert med en flate, og finner man en flate som er synlig fra posisjonen vil man assosiere punktet med denne flaten. Dersom man har undersøkt listen over flater fra gjeldende indeks til slutten uten å finne en synlig flate, vil man kunne forkaste punktet, siden det vil befinne seg inne i det konvekse legemet.

Vår implementasjon

Vi lagde de datastrukturer som vi anså som nødvendige for å gjennomføre disse operasjonene, og forsøkte samtidig å disse så generelle som mulig slik at de kunne brukes i lignende algoritmer som vi så for oss kunne bli aktuelle. Selve algoritmen ble testet grundig med forskjellige typer input, før den ble innlemmet i selve hovedprogrammet. Vi valgte å lage en utgave av algoritmen spesifikk for 3D, siden det er snakk om mange meget komplekse og regnekrevende operasjoner som i utgangspunktet ikke passer veldig bra for Java. Ved å spesialisere algoritmen noe, var det mulig å forbedre ytelsen betraktelig. Våre tester har vist at vår versjon er fullt ut kapabel til å operere på store bilder (200000+ unike pikselverdier) uten å bli oppfattet som uforholdsmessig treg for brukeren, selv på datamaskiner som ikke er raske.

Den rekursive funksjonen som traverserer overflaten har vi konstruert som den etterfølgende koden viser:

```
/**
* Recursively visit all visible facets. Discard vertices as
* necessary, and build a horizon consisting of all
* surrounding ridges.
* @param face The facet that should be visited.
* @param discardedVertices The vertices that are discarded.
* @param horizon The horizon ridges.
* @param vertex The vertex being processed.
* @param sequence The sequence number.
*/
private void visit(Facet face,
                   OptimizedVector discardedVertices,
                   OptimizedVector horizon,
                   Vertex vertex,
                   int sequence) {
  discardedVertices.append(face.associatedVertices);
  Facet neighbor;
  for (int i=0; i<3; i++) {
    neighbor=face.neighbor[i];
    if (!neighbor.discarded) {
      if (neighbor.lastVisit<sequence) {
        neighbor.lastVisit=sequence;
        if (neighbor.outside(vertex)) {
          neighbor.discarded=true;
```

```
            visit(neighbor,
                  discardedVertices,
                  horizon,
                  vertex,
                  sequence);
          }
          else {
            Edge edge=new Edge(face.corner[i],
                               face.corner[(i+1)%3],
                               neighbor);
          horizon.addElement(edge);
          }
        } else {
          Edge edge=new Edge(face.corner[i],
                             face.corner[(i+1)%3],
                             neighbor);
          horizon.addElement(edge);
        }
      }
    }
  }
```

Vårt første utkast til en løsning fulgte algoritmen som beskrevet over, med det ene unntak at vi ikke benytter bredde først søk for å finne horisonten. Vi valgte også å begynne med to flater (i realiteten samme flate med motsatt orientering av punktene), ikke en tetraedersimpleks (fire flater) som angitt over, men dette gir ingen variasjon fra metoden som benyttes for å konstruere det konvekse legemet. På grunn av at enkelte funksjoner er noe langsommere enn andre i Java, lagde vi også en modifisert utgave som ikke baserer seg på nabostrukturen for å finne horisonten.

Den nye versjonen benytter en stack for å finne horisontkantene, og er noe raskere enn den første utgaven i mange tilfeller, avhengig av inputdata. I stedet for å traversere overflaten, sjekker man bare om hver flate er synlig eller ikke. Dersom den er synlig, forsøker man å legge de tre kantene denne flaten består av på stacken. Stacken fungerer på den måten at man benytter seg av kanter bestående av 2 punkter som dyttes på stacken dersom en tilsvarende kant ikke eksisterer på stacken fra før. Gitt kanten AB vil den nøytralisere kanten BA dersom den finnes fra før. Dette følger av det faktum at to flater deler en kant, og dersom en kant legges på stacken to ganger er den ikke med i horisonten.

Selv om konvekse legemer er en velkjent måte representere en bilde- eller enhetsgamut på, har den sine svakheter. Dette gjelder spesielt for bildegamuter, der man mister en god del informasjon i områder som egentlig ikke inneholder punkter, men som grunnet mangelen på konkave områder ser ut til å tilhøre objektet. Enhetsgamuter vil typisk forårsake færre problemer, fordi de generelt sett har en form uten store innhogg i objektet, men det er likevel på langt nær en perfekt metode for disse punktsamlingene. Det har blitt forsket på metoder for å unngå disse situasjonene, og et av resultatene er det man kaller Alpha Shapes.

4.9.5 Alpha Shapes

Alpha Shapes er en benevnelse som benyttes om en veldefinert type objekter i 3D som består av punkter, kanter, trekanter og tetraedere. Disse objektene benyttes i mange sammenhenger for å vise en gitt samling punkter som en mer konkret overflate, der overflaten baserer seg på en Delaunaytriangulering av punktene, samt et alpha-parameter som angir det man kan kalle detaljnivået til gjenstanden. En algoritme som konstruerer alle forskjellige overflater man kan finne ut i fra et sett med punkter ble publisert av Herbert Edelsbrunner og Ernst P. Mücke i 1994, og vår implementasjon tar utgangspunkt i deres forslag. Alpha shapes er en videreutvikling av konvekse legemer, og overkommer problemet med at disse objektene ikke blir nok detaljert i områder hvor punktsamlingen har en grop innover mot sentrum.

Konseptet baserer seg på at man først finner Delaunaytrianguleringen til punktene, for deretter å finne de intervallene av alphaverdier som medfører at hver geometriske bestanddel er en del av objektet. Edelsbrunner og Mücke har benyttet en flipmetode for å finne Delaunaytrianguleringen, og vi begynte også med å implementere denne løsningen.

4.9.5.1 Delaunaytriangulering i 3D (ved flipmetoden)

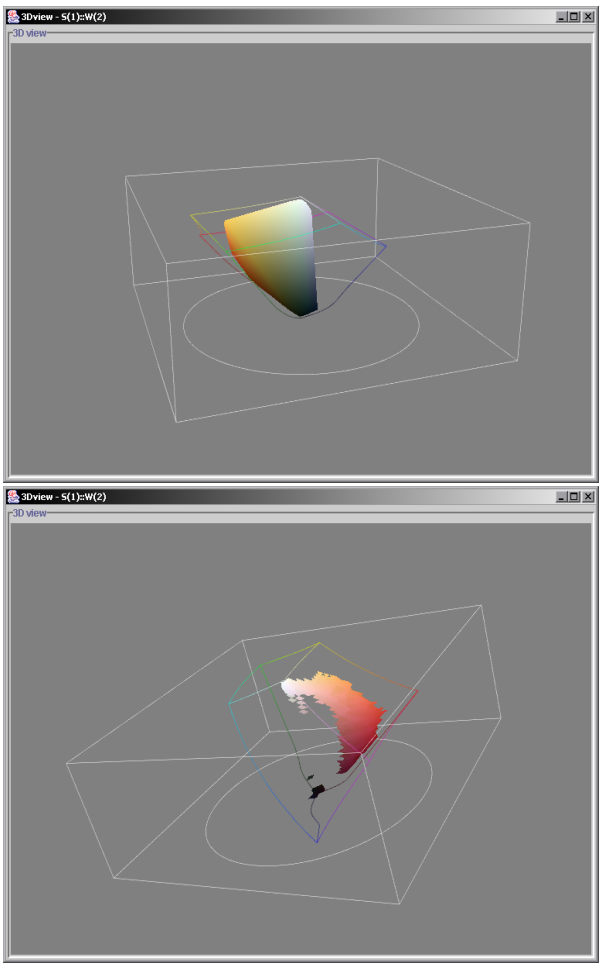
Delaunaytrianguleringen av et gitt antall punkter med generell posisjon vil være unik, og består av de tetraederne og tilhørende trekanter som gir den minste samlede kantlengde totalt sett.

Edelsbrunner og Mücke benytter en algoritme publisert av Joe for å finne denne triangulerингen i 3D. Først sorterer man punktene langs en akse, slik at de nye punktene hele tiden vil ligge på utsiden av det oppbygde objektet. I vårt tilfelle bør man i tillegg ha et sekundært sorteringskriterium som medfører at man tar punktene nærmest aksen først, da de kvantiserte verdiene ellers kan skape flere problemer ved at mange punkter ligger på samme plan.

Deretter tar man de første fire punktene og lager et tetraeder bestående av fire trekanter. Man merker disse trekantene som synlig, og fortsetter deretter å prosessere punktene ett om gangen. Man finner de overflatetrekantene som er synlig fra det nye punktet, og lager deretter nye tetraedere av det nye punktet og disse trekantene. De trekantene som behandles på denne måten vil heretter ikke være en del av objektets overflate, og behøver ikke lenger å prosesseres i søket etter synlige trekanter. De nye tetraederne kobles mot sine naboer, som man finner ved å se på de trekantene som de ble laget av, samt de andre tetraederne som ble lagd når man behandlet det nye punktet.

Selv om dette gir en triangularisering, er det ikke nødvendigvis en Delaunaytriangulering. Joe fremsetter en påstand om at hvis man prosesserer ett punkt om gangen, og deretter foretar en lokal test for å forsikre seg om at de nye tetraederne er lokalt Delaunay, vil den resulterende strukturen også være en Delaunaytriangulering. Testen som gjøres, er å se om de nye tetraederne og deres naboer tilfredsstiller et krav om at punktet som er en del av tetraeder A men ikke tetraeder B som er As nabo, ikke skal være innenfor den kulen som dannes av de fire punktene i B.

Dersom man definerer M til å være determinanten



Figur 4.18: Alpha shapes visualisering. Øverst: stor alphaverdi. Nederst: liten alphaverdi.

$$M \begin{matrix} i1, i2 \ldots ik \\ j1, j2 \ldots jk \end{matrix} = \det \begin{pmatrix} \pi_{i1,j1} & \pi_{i1,j2} & \cdots & \pi_{i1,jk} \\ \pi_{i2,j1} & \pi_{i2,j2} & \cdots & \pi_{i2,jk} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{ik,j1} & \pi_{ik,j2} & \cdots & \pi_{ik,jk} \end{pmatrix}$$

hvor $\pi_{k,0} = 1$ for alle k, kan man uttrykke denne testen gitt tetraederet T bestående av punktene i,j,k,u og nabopunktet v som

$$M \begin{matrix} i,j,k,u \\ 1,2,3,0 \end{matrix} \cdot M \begin{matrix} i,j,k,u,v \\ 1,2,3,4,0 \end{matrix} > 0 \Leftrightarrow \textit{Punktet v ligger inne i kulen rundt T}$$

Dersom testen viser at tetraederet ikke er lokalt Delaunay, må man forsøke å forandre på dette ved å foreta en av to transformasjoner kalt flips. Dersom de to tetraederne til sammen danner et konvekst objekt, kan man løse situasjonen ved å fjerne trekanten som deler A og B, og heller erstatte den med kanten/linjen som går mellom de to punktene i A og B som ikke er en del av den felles trekanten. Denne kanten deler det konvekse legemet i tre nye tetraedere, og man erstatter den gamle strukturen med disse tre og deres tilhørende trekanter. Dette er mulig fordi det kun er den interne strukturen lokalt som forandrer seg, de tre tetraederne vil, sett utenfra, ha samme overflate som de to tidligere hadde.

Den andre transformasjonen foretas hvis objektet ikke ble konvekst, og det i tillegg finnes et tetraeder som er nabo med begge de to tetraederne. Dersom man finner et slikt tredje tetraeder, kan man erstatte disse tre med to nye tetraedere. Dette vil heller ikke være et problem for den omliggende tetraederstrukturen, siden det også her er kun den interne oppdelingen som forandrer seg.

Selv om vi kodet en relativt effektiv utgave av denne algoritmen, viste det seg at det ikke ga en hastighet som var tilfredsstillende. Algoritmen har i verste fall en kompleksitet som er kvadratisk med antall punkter den behandler, og selv om det ikke er riktig så ille i praksis, tok det for lang tid å benytte flipalgoritmen på et middels antall punkter. Etter å ha undersøkt en del om dette emnet, bestemte vi oss for å forsøke en annen fremgangsmåte.

4.9.5.2 Delaunaytriangulering i 3D (via convex hull i 4D)

Delaunaytriangulering og det å finne et konvekst omhyllingslegeme er i realiteten to sider av samme sak. Dette manifesterer seg i at man kan finne en Delaunaytriangulering for punkter i dimensjon n ved å beregne det konvekse legemet til punktene i dimensjon n+1. Dette gjøres ved at man legger til en koordinat til alle punkter som beregnes til å være summen av kvadratene til de andre koordinatene. Dette kan man tenke seg ser ut som en paraboloide dreiet rundt den nye aksen. Deretter finner man det konvekse legemet i dimensjon n+1, og de overflatedelene som danner plan med negativ sistekoordinat (peker nedover, tenkt i 3D) vil være med i Delaunaytrianguleringen. For å utføre Delaunay i 3D vil man derfor beregne en fjerde koordinat, utføre for eksempel Quick Hull i 4D, og dermed få ut de tetraederne som tilhører trianguleringen. Som en bonus vil man få nabostrukturen til tetraederne, da tetraederne vil ha forkastede tetraedere

eller tetraedere med positiv sistekoordinat som nabo på sider som danner overflaten i 3D, mens de andre naboene på overflaten i 4D vil være virkelige tetraedernaboer i 3D.

Quick Hull i 4D følger akkurat samme algoritme som i 3 dimensjoner, den eneste forskjellen ligger i måten vi har implementert de på ved at datastrukturene blir mer avanserte. Det er noe mer komplisert å forestille seg denne prosessen i 4D, men prinsippene er de samme. Kantene i 4D vil bestå av 3 punkter, mens 4 punkter vil utgjøre en flate. Hyperplanet til en flate i 4D finnes ved å løse planlikningen for fire punkter, og deretter finne distansen fra origo langs plannormalen ved å benytte at prikkproduktet mellom normalen og ett av punktene gir distansen. Horisonten finnes på tilsvarende måte som i 3D, men man vil ha 4 naboer per flate i stedet for 3, og konstrueringen av den nye nabostrukturen vil bli noe tyngre regnemessig.

Hyperplan utregnes ved å benytte likningen

$$\begin{array}{rcrcrcrcl} n_1*x_1 & + & n_2*y_1 & + & n_3*z_1 & + & n_4*w_1 & = & k \\ n_1*x_2 & + & n_2*y_2 & + & n_3*z_2 & + & n_4*w_2 & = & k \\ n_1*x_3 & + & n_2*y_3 & + & n_3*z_3 & + & n_4*w_3 & = & k \\ n_1*x_4 & + & n_2*y_4 & + & n_3*z_4 & + & n_4*w_4 & = & k \end{array}$$

der planet består av normalvektoren n_1, n_2, n_3, n_4 og distansen k. Man benytter forutsetningen om at lengden av normalen er 1 for å finne k.

4.9.5.3 Alpha Shapes fortsetter

Koden for å kjøre en beregning av det konvekse legemet gitt punktdata, benytter man denne koden:

```
for (int i=0; i<numPoints; i++) {
  vertices[i] = new Vertex(
    pointCoord[1][i*3] + Math.random()/generalPositionConst,
    pointCoord[1][i*3+1] + Math.random()/generalPositionConst,
    pointCoord[1][i*3+2] + Math.random()/generalPositionConst);
  vertices[i].addDimension();
}
OptimizedVector list = quickHull.build4D(vertices);
int tetraCount = 0;
for (int i = 0; i < list.elementCount; i++) {
  Facet tetra = (Facet)list.elementAt(i);
  if (!tetra.isDiscarded() && tetra.normalW <= 0.0) {
    tetraCount++;
    tetra.calculateRadiusOfSmallestCircumsphere();
  }
}
```

Vi kan se at man først forskyver alle koordinater med en liten verdi, for deretter å kjøre funksjonen addDimension() som kalkulerer og legger til den fjerde koordinaten. quickHull.build4D(...) konstruerer en liste med alle **Facets** (flatedeler) som er blitt konstruert i prosessen med å finne det konvekse legemet. Alle delene som ikke har blitt forkastet,

og har en positiv fjerdekoordinat, vil tilsvare gyldige tetraedere i Delaunaytriangulering i 3D. I samsvar med Edelsbrunner og Mückes forslag, beregnes deretter radiusen til kulen som omgir tetraederet. (Mer om dette senere)

Etter å ha testet koden for Quick Hull i 4D, kunne vi konstatere at den var en god del raskere enn flipalgoritmen. Likevel er den en kompleks algoritme som man ikke bør bruke på et stort antall punkter. Derfor implementerte vi en kvantifisering av fargerommet, som fungerer på samme måte som i den kvantiserte visualiseringen. Dette er også et viktig poeng for eventuelt fremtidig arbeid på prosjektet, da man uansett ikke kan arbeide på den ubearbeidede versjonen som gir altfor mange små trekanter dersom man skal inkludere støtte for interaksjon med strukturen.

Når man har den triangulariserte Delaunaystrukturen, kan man beregne de alphaverdiene som definerer grensene for om trekantene skal være med i 3D-visningen eller ikke. Dette gjøres ved først å beregne radiusen til tenkte kuler som omgir tetraederne. Deretter avgjør man om den tilsvarende kulen som omgir hver trekant inneholder andre punkter.

Den største alphaverdien til trekanter som befinner seg på overflaten til objektet, vil være maksimalverdien til et flyttall. Dette betyr at trekanten vil være synlig for alle alphaverdier større enn minimumsverdien, som vil være radiusen til kulen rundt de tetraederet som den tilhører hvis det finnes punkter i trekantens tenkte kulevolum, eller radiusen til kulen som omfatter trekanten i andre tilfeller.

Alle andre trekanter vil ha radius til den største kulen rundt nabotetraederne som maksimumsverdi, mens minimumsverdien enten er den minste av radiene til de to kulene, eller radius til kulen rundt trekanten. Dette er også avhengig av om kulen som omfatter trekanten inneholder andre punkter.

I tillegg til de ovenfor nevnte trekantene, vil en normal alpha shape bestå av kanter og punkter, avhengig av alphaverdien. Det er i vårt tilfelle lite hensiktsmessig å vise punkter eller kanter, men det vil ikke kreve noen større modifikasjoner dersom dette skulle være ønskelig i fremtiden, for eksempel i forbindelse med flytting av punktene i 3D.

Alpha shapes er mye benyttet innen mange bruksområder som viser et gitt antall punkter som en gjenstand med form i 3D. Utregningen av alphaverdiene kan sammenlignes med at man ved gradvis mindre verdier skjærer bort stykker fra et objekt som til å begynne med er konvekst. Etter hvert som volumet synker, vil man se stadig mindre flater, og det vil bli hull i objektet. Objektet kan bli splittet opp i flere deler, og til slutt vil man sitte igjen med kun de punktene man startet med, uten trekanter eller kanter.

Dette egner seg for en interaktiv presentasjon, noe som vi har implementert i form av en slider som bestemmer alpha-verdien. Når brukeren flytter posisjonen til slideren, vil objektet i 3D oppdateres. Det er ikke nødvendig å foreta nye beregninger under denne fasen, man trenger bare å undersøke maksimums- og minimumsverdiene til hver enkelt byggedel for å finne ut om den skal tegnes opp eller ikke.

Dersom man vil forbedre ytelsen til alpha shapes, kan man bruke et plattformuavhengig tilleggsbibliotek skrevet i C eller C++ som støtter Delaunaytrianguleringer og koble dette til Java ved hjelp av JNI.

4.10 Generell posisjon

Alle algoritmer som finner Delaunaytrianguleringen (eller konvekse legemer) til et sett punkter, er avhengig av at punktene befinner seg i det man kaller generell posisjon, dvs. at dersom man benytter tilstrekkelig nøyaktighet vil ingen punkter gi motstridende resultater ved de geometriske testene som benyttes. Dette medfører at ingen 4 punkter kan ligge i samme plan, og ingen 5 punkter kan ligge på overflaten til en kule. Dette vil selvfølgelig ikke medføre riktighet i vårt tilfelle, da inndata er gitt i kvantiserte utgaver, og dette vil kunne skape problemer for beregningene. En annen betingelse for å kunne benytte disse punktene, er selvfølgelig at man ikke har flere punkter på samme posisjon. Dette kravet sikres ved at piksler med samme fargeverdi blir behandlet, og at man kun genererer ett punkt av alle disse pikslene. Vi har for enkelhets skyld ikke benyttet noen av de mulige teknikkene som er oppfunnet for å forhindre slike situasjoner. Disse blir beskrevet i blant annet Edelsbrunner og Mückes dokumenter, men oppfattes som unødvendig nøyaktige og feilsikre for vårt prosjekt. Vi har i stedet valgt å forskyve koordinatene til alle punkter med et tilfeldig (men lite) tall, noe som unngår at uheldige tilfeller oppstår. Dette dreier seg først og fremst om Quick Hull i 4D, som uten dette tiltaket i enkelte tilfeller ville forårsaket at strukturen man får ut av algoritmen ikke ville vært en Delaunaytriangulering, eller at algoritmen ville feilet på grunn av ugyldige tetraedere.

Ulempen med denne løsningen er at det teoretisk sett fremdeles finnes en (veldig) liten risko for at det oppstår feil. Vi betrakter denne sannsynligheten som svært liten. Forskyvningen vi foretar er også så liten at den ikke har en synlig eller praktisk effekt på punktene som man får ut av algoritmen. Hvis dette hadde vært et problem, ville man kunnet erstatte de nye koordinatene med de opprinnelige dersom dette hadde vært ønskelig.

4.11 Interaksjoner

4.11.1 Interaksjoner generelt.

For at det skal være aktuelt å flytte på objekter i 3D, må det være mulig for brukeren å velge hvilket objekt som skal flyttes, samt indikere hvordan det skal flyttes. Java3D har en rekke muligheter for flytting av objekter, men da disse ikke er raske nok ble vi nødt til å finne vår egen løsning. Når brukeren høyreklikker på skjermen, foretar vi først en test på om det befinner seg noe under, eller i nærheten av, musepekeren. Dette dreier seg i første omgang om en test mot et objekt av typen bounding box, dvs. ikke en test mot virkelig geometri, men mot en tenkt gjenstand som inneholder alle valgbare objekter i scenen.

Dersom man ved å benytte denne testen finner at det er mulig at brukeren har valgt noe, foretar vi en ytterligere test hvor man representerer hvert valgbare objekt som en kule med en gitt radius. Deretter er det opp til Java3D å finne ut hvilket av disse objektene som brukeren har klikket på, eller om det likevel ikke finnes objekter som har denne posisjonen på skjermen.

Når man først finner ut at brukeren har valgt et objekt, må man finne ut hvordan brukeren forsøker å flytte det. Vi finner først den 4x4 matrisen som beskriver transformasjonen av objektenes posisjon fra det lokale koordinatsystemet for hvert objekt til det

virtuelle Java3D universet. Deretter inverterer vi denne, slik at vi finner den motsatte transformasjonen, for deretter å benytte denne matrisen til å regne ut hvor 4 tenkte objekter befinner seg. Dette er objektene som befinner seg i (0, 0, 0), (1, 0, 0), (0, 1, 0) og (0, 0, 1). Ved hjelp av disse 4 posisjonene kan man finne 3 vektorer som til sammen gir alle mulige kombinasjoner av mulige bevegelser: Til høyre/venstre, opp/ned og fremover/tilbake.

Så definerer man bevegelse med musepekeren i y-retning som en tenkt bevegelse opp/ned, og bevegelse i x-retning som bevegelse til høyre eller venstre. Dersom brukeren holder nede shift, foretar man forflytning fremover/tilbake i stedet for opp/ned. Hvor stor forflytning man foretar, er avhengig av hvor mye brukeren flytter musepekeren.

4.11.2 Standard interaksjon

Den mest grunnleggende måten å flytte objekter på i 3D, er rett og slett å flytte ett objekt (det som brukeren har valgt) i den retning som man kommer frem til gitt den ovenfor nevnte metoden. Dette er en typisk lokal transformasjon.

Situasjonen er ikke like enkel når man ser på oppdateringen av pikslene i 2D. Ett objekt i 3D kan representere mange forskjellige fargeverdier, og det er et poeng å beholde den innbyrdes forskjellen mellom disse verdiene. Hvis man satte fargen til alle assosierte piksler til fargen man får når man konverterer objektets nye posisjon tilbake til bildets fargerom, ville man kunne miste mye detaljer dersom man benytter for eksempel en kvantisert visualisering med få inndelinger.

Av denne grunn konverterer man alle pikslene til fargerommet benyttet i 3D, foretar en forflytning, og konverterer tilbake for å få den nye fargen. Man behøver ikke konvertere mer enn en gang for hver unike farge som er blant de valgte pikslene, fordi vi lagrer indeksene i rekkefølge sortert etter fargeverdi.

En forflytning av denne typen gir ikke et bra resultat i praksis. Pikslene som blir forandret, vil forandre seg for mye i forhold til de pikslene som befinner seg i nærheten. Derfor oppstår det unaturlig sterke fargeoverganger i bildet, dersom brukeren ikke er meget nøyaktig og flytter alle objekter som er i nærheten av dette objektet.

4.11.3 Segmentinteraksjon

Dersom man har en oppbygd struktur i 3D som inkluderer segmenter, for eksempel segment maxima, men også andre visualiseringer der man har ett sentrumspunkt med en overflatestruktur rundt, kan man benytte denne forflytningsmodusen til å flytte alle pikslene som tilhører et segment skalert med radius til punktet.

$$newPosition = oldPosition + (pointRadius/movedPointRadius) * \overrightarrow{movementVector}$$

Dette gir som resultat en mer jevn overgang mellom pikslene som befinner seg i ett segment, men gir en skarp overgang til områder i bildet hvor pikslene befinner seg i et annet segment.

4.11.4 Tetraederinterpolasjon

Det finnes en god og veldokumentert løsning for problemet med at det oppstår skjæringslinjer i bildet mellom de pikslene som ble flyttet og de som står i ro. Dette er å dele fargerommet inn i tetraedere, og å beregne punktenes posisjon ved å interpolere forflyttningen man gjør av hjørnepunktene over alle tetraedere som punktet er med i.

Denne interpolasjonen krever at man foretar inndelingen av rommet i tetraedere, samt beregner vektene som benyttes for å finne ut hvor mye en forandring av hjørnepunktet endrer de punktene som befinner seg inne i tetraederet. Implementasjonen av denne interaksjonen er avhengig av at visualiseringen har bygd opp de nødvendige strukturene. Oppbyggingen er helt separat fra selve forflyttningen, slik at det ikke spiller noen rolle hvilken visualisering som benyttes, så lenge objektene som kan velges implementerer grensesnittet som muliggjør tetraederinterpolasjonen.

I vårt tilfelle vil for eksempel segment maximavisualiseringen gi tetraedere som består av en trekant fra overflaten, samt midtpunktet av objektet. Først finner man volumet til de opprinnelige tetraederne før man flytter på punktene, deretter går man igjennom listen over unike punkter og traverserer tetraedernes nabostruktur for å finne et tetraeder som inneholder punktet. Siden segment maxima algoritmen tillater at punkter kan befinne seg utenfor overflaten, må man godta at enkelte punkter er utenfor alle tetraedere. Man kan da assosiere punktet med det nærmeste tetraederet, og ekstrapolere med de samme formlene som man ellers ville benyttet.

Når brukeren forsøker å flytte et objekt, finner man alle de tetraedere som har objektet som hjørnepunkt. Deretter beregner man nye posisjoner og farger for alle piksler som befinner seg i tetraederet som følger:

```
newPosition = oldPosition+weight1*cornerMovement1+
              weight2*cornerMovement2+...
```

Vekten til et hjørnepunkt beregnes som volumet av det tetraederet som dannes av de 3 andre hjørnepunktene og det punktet som skal interpoleres, delt på volumet til tetraederet. Slik får man jevne overganger som virker mer naturlig i bildet, i alle fall så lenge man unngår å bevege et hjørnepunkt over i tetraedere som det ikke tilhører.

4.12 Globale transformasjoner

De globale transformasjonene har ikke så ulike implementasjon som begrensninger, men noen fundamentale forskjeller er det. Base klassene har ikke samme oppbygging og virkemåte som i begrensninger, de tilbyr ingen funksjon utenom å bygge opp et rammeverket som er felles for underklasssene. Mye av funksjonaliteten i de forskjellige klassene er såpass ulik de andre, at dette ble det mest hensiktsmessige.

En typisk implementasjon for forflyttningstransformasjoner er :

```
public void update3D(int index, PickableObject objects[],
                     float start[], float end[],
```

```
                    ColorSpace3D colorSpace3D) {
...
float xres = end[0] - start[0];
for (int i = 0; i < objects.length; i++) {
     ...
     pos = objects[i].getPosition();
     npos[0] = pos[0] + xres;
     ...
     objects[i].setPosition(npos);
     objects[i].setColor(colorSpace3D.toSRGB(npos));
   }
}
```

Mens skalering foregår noe i likhet av :

```
newPosition=oldPosition+lMovement*(distanceToPlane/MovedObjectDistanceToPlane)
```

Ved tilting finner normalen til det planet som dannes mellom tiltpunktet og drapunktet. Dermed har man den aksen man ønsker å tilte/rotere rundt.

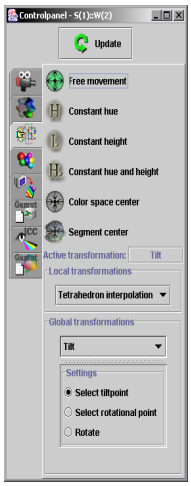
For å rotere rundt en vilkårlig akse, foretar man først en translasjon slik at tiltpunktet blir i origo. Deretter roterer man slik at rotasjonspunktet kommer inn på en av standardaksene, for deretter å utføre rotasjonen som brukeren ønsker å utføre rundt denne aksen. Så gjør man det hele i motsatt rekkefølge, slik at man gjør om alle operasjonene med unntak av den ene rotasjonen.

Arvehierarki for transformene ser slik ut:

Baseklasse	***Virksomhetsklasse***
Interaction3D	
	GlobalTransformMoveAB
	GlobalTransformMoveL
	GlobalTransformScaleL
	GlobalTransformRadius
	GlobalTransformFancyRadius
	GlobalTransformTilt

Tabell 4.5: Arvehierarki for transformasjoner

4.13 Begrensninger



Figur 4.19: Kontrollpanel og valg tilknyttet constraints/begrensninger.

For å implementere begrensninger på interaksjonene, har vi måttet lage en del nye klasser. Det er hovedtyper med begrensninger, de som låser punktet til å gå i en linje, og de som tvinger det til å kun bevege seg i et plan. Med dette i bakhode har vi laget to base klasser som hver av begrensnings klassene arver fra. Dermed er det bare for underklassene å sette forutsetningene som base klassen skal arbeide under, og returnere svaret den gir.

Typisk implementasjon av en funksjon i disse klassene er:

```
public float[] getRealEndPoint(float start[], float stop[], PickableObject objects[]) {
  linePoint=new Vector3f(0.5f, start[1], 0.5f);
  return super.getRealEndPoint(start, stop, objects);
}
```

Siden alle disse klassene arver fra enten line eller plane constraint, sørger mor-klassen for selve utregningen, mens de avgjørende variablene (protectede) settes av de respektive klassene.

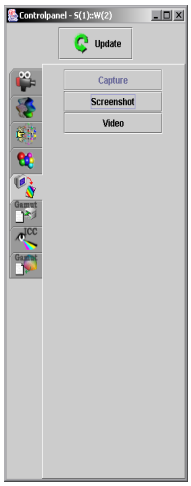
Tabellen viser arvehierarkiet for begrensningene

Baseklasse	*Virksomhetsklasse*
LineContstraint	
	ConstantHueAndHeightConstraint
	CenterConstraint
	SegmentCenterConstraint
PlaneConstraint	
	ConstantHeightConstraint
	ConstantAngleConstraint

Tabell 4.7: Arvehierarki for begrensninger.

4.14 Capture

Brukere som benytter programmet i presentasjonssammenheng vil finne det nyttig at man har muligheten til å lagre stillbilder av 3D scenen. Sammen med muligheten til å forandre bakgrunnsfargen, gjør dette at brukeren kan lage bilder som egner seg for å bli skrevet ut på papir. Bildene som blir tatt, kan lagres i ethvert format som støttes av den versjonen av JAI som benyttes på ulike plattformer.



Figur 4.20: Kontrollpanel og valg tilknyttet capture. Stillbilde og animasjon.

Det er ikke helt uproblematisk å kode rutinen som lager disse bildene, grunnet Java3D som i utgangspunktet ikke gir tilgang til de data som man behøver. Det finnes ingen måte å aksessere det canvaset som viser 3D på skjermen slik at man får tak i de pikslene som blir vist. Vi klarte å unngå dette problemet ved å lage ett nytt canvas som lagrer sin output til en bildebuffer hver gang brukeren ber om et stillbilde. For at dette skal gi samme resultat som det som blir vist på skjermen, setter vi opp alle parametre riktig ut fra skjerminnstillingene. I ettertid nevnte oppdragsgiver at det muligens kunne være interessant å lagre en animasjon av 3D-bildet til en videosekvens. Selv om dette ikke

ble oppgitt som en prioritert oppgave, fant vi tid til å implementere en løsning som bør være tilfredsstillende.

Vi valgte å benytte JMF, Suns anbefalte bibliotek for Javaprogrammer som trenger støtte til opptak eller avspilling av video. Dette frigjør oss fra å måtte gjøre enkodingen selv, noe som kunne vært en mindre relevant oppgave sett i forhold til programmets hovedmål og ekstraarbeidet dette ville medføre. Først måtte vi bestemme oss for en standard metode for å spesifisere hva slags bevegelse animasjonen skulle utføre.

Grunnet tidspress og det faktum at oppgaven er perifer i forhold til prosjektets hovedmål, valgte vi å unngå å lage et komplisert brukergrensesnitt. Vi implementerte i stedet et scriptspråk som muliggjør rotasjoner og translasjoner av objektene i 3D. Brukeren kan dermed definere animasjonen ved en sekvens av kommandoer. Kommandoene det dreier seg om er disse:

- **frames X**

  Denne kommandoen bestemmer antall bilder som skal lages i animasjonen for den nåværende oppgaven. **X** vil da være et heltall > 0 som angir antall bilder.

- **rotate X Y Z**

  Denne kommandoen angir at nåværende oppgave skal inkludere en rotasjon. **X Y Z** angir antall grader (ikke radianer) som det skal roteres rundt de respektive akser. **X Y Z** er alle av typen flyttall.

- **move X Y Z**

  Denne kommandoen angir en translasjon av objektene. **X Y Z** vil indikere hvor langt du ønsker å flytte objektene langs de respektive aksene. Typiske verdier for **X Y Z** vil ligge mellom -1.0 og 1.0.

- **next**

  Denne kommandoen indikerer at en oppgave er komplett. De rotasjoner og translasjoner som er lagt inn i den nåværende oppgaven, vil da bli utført som en operasjon over det angitte antall bilder.

- **go**

  Denne kommandoen angir slutten på scriptet.

Typisk vil scriptet se ut som dette:

```
rotate 0 360 0
frames 100
next
rotate 11.1 0 0
frames 10
next
move 1 0 0
frames 10
next
rotate 0 360 0
frames 100
go
```

Etter at programmet har parset scriptet, vil hver operasjon representeres som et objekt i en liste. Deretter kan vi bare be om at hvert objekt skal kjøres på den eksisterende scene grafen, og benytte en egenlaget klasse for å lage animasjonen.

Vi benytter samme teknikk som ved stillbildene for å få tilgang til 3D-visningen. Selve videoenkodingen overlates til en klasse som er i stand til å lage filmer av alle typer bildedata. Dette gjør at man kun behøver å spesifisere et filformat og diverse parametre, for så å starte enkodingen. Deretter kan man sende ett og ett bilde til enkoderen, som utfører enkodingen og skriver resultatet til fil. Når man ikke har flere bilder som skal med i animasjonen, kjører man en funksjon som avslutter enkodingsprosessen. Pseudokode for enkodingsprosessen sett utenfra blir da:

```
encoder.initialize(parameters);
encoder.start();
while (moreImages)
    encoder.encode(image);
encoder.end();
```

Dette gjør enkoderen til en meget generell modul som egner seg til mange situasjoner.

Selve implementasjonen av videoenkoderen er bygd på JMF rammeverket, men har blitt forbedret en god del. Vi har gjort prosessen multithreadet, dvs. at enkodingsprosessen startes i sin egen tråd. Den benytter en FIFO (First In, First Out) kø for bildene, slik at programmet som starter enkoderen kan legge bildene inn i køen, mens enkoderen tar de ut i den rekkefølgen de kommer inn. Selve køen er av begrenset størrelse for å unngå store minnekrav, og håndteringen er synkronisert slik at kun en tråd har tilgang til køen om gangen. Dersom køen er full, vil hovedtråden vente på at enkoderen skal hente et bilde. Hvis enkoderen er ferdig med de data den har fått utlevert, og køen er tom, vil den vente på flere bilder eller evt. en melding om at enkodingen er ferdig. Bildene må gjøres om til rå data i form av en RGB databuffer før JMF kan håndtere de, og i tillegg er enkelte andre aspekter ved både enkoding og rendering av 3D tidkrevende. Den flertrådede løsningen med køsystem sørger for minimalt med venting, da det er noen faser som ikke utnytter CPUen maksimalt og slik kan utføres samtidig.

Enkoderen er i stand til å støtte de komprimeringsalgoritmer som er installert på datamaskinen den kjører på, slik at den for eksempel kan lage video i DivX-format. Beklageligvis er ikke JMF fullstendig stabilt på alle plattformer, og det er også enkelte problemer ved konvertering mellom forskjellige formater. Det kan derfor være nødvendig for brukeren å teste seg frem til han/hun finner et egnet format.

4.15 Generell optimalisering av kode

Å optimalisere koden har i seg selv vært en utfordring. Java er et interpretert språk og er derfor ikke av det kjappeste vi kunne valgt. Java er treg til å tegne seg selv opp, og gevinsten av å benytte seg av assembler “trick” som shift reduksjoner isteden for multiplikasjon etc. er ikke like stor som i ikke-interpreterte språk. Vi har allikevel funnet punkter der vi har klart å få ting kjappere.

Mye av å få ting kjappere har gått ut på å lage ting selv, ikke bruke andre moduler som kan implementeres. Dette viste seg spesielt gunstig i konverteringen av fargerom.

Først brukte vi ICC profilens konverteringsmodul, noe som viste seg å bruke opptil 10 sekunder på vår utviklingsmaskin. Ved å skrive denne kode-snutten selv oppnådde vi å komme ned i ca. 0.6 sekunder på samme operasjon.

Ved flere tilfeller av tunge regne operasjoner prekalkulerer vi visse verdier for at maskinene skal slippe å regne ut de samme verdiene flere ganger. Vi har også tildels bruk av såkalte look-up-tables.

Andre ting vi har valgt, selv om gevinsten som sagt ikke er enorm, er å bruke logiske operasjoner der det har vist seg hensiktsmessig. Dette gjelder særlig i bilde oppdatering og forandring. Det har blitt laget en optimalisert vektor klasse for å lagrer en type objekter, brukt i bl.a. Delaunaytrianguleringene.

Alt i alt har vi klart få programmet til å gå i en brukbar hastighet, og føler at dette ikke er et problem med tanke på brukervennligheten. Der vi føler det er hensiktsmessig, har vi benyttet en egenutviklet progress bar som viser kalkulasjonenes fremgang til brukeren. For å illustrere konseptene vi har benyttet, kan man se på koden som indekserer bildet og finner de unike pikselverdiene:

```
// int imagePixel[] is the image data. Each pixel is represented as
// an int consisting of four bytes (r, g, b, a respectively)
int uniquePixel[] = new int[nPixels];
  // The unique pixelvalues (colors)
byte pixelCount[];
  // pixelCount[i] = count of pixels whose color = uniquePixel[i]
BitSet bitSet = new BitSet(16777216);
  // one bit per color (24 bit)
int numUnique = 0;
  // number of unique colors
int index;
int q;
ProgressBar bar = new ProgressBar(0, 7, "Processing image");
bar.setDescription("Counting number of unique colors.");
for (i=0; i < nPixels; i++) {
  index = imagePixel[i]&0x00ffffff;
  if (!bitSet.get(index)) {
      // new color
    uniquePixel[numUnique] = imagePixel[i];
    numUnique++;
    bitSet.set(index);
  }
}
bar.inc();
bar.setDescription("Sorting colors.");
Arrays.sort(uniquePixel);
bar.inc();
bar.setDescription("Building vertex and color information.");
byte color[] = new byte[numUnique*3];
float vertex[][] = new float[2][];
vertex[0] = new float[numUnique*3];
```

```
for (i=0; i < numUnique; i++) {
  q = i*3;
  vertex[0][q+2]=(float)((uniquePixel[i]&0xff)/255.0f);
  vertex[0][q+1]=(float)(((uniquePixel[i]&0xff00)>>8)/255.0f);
  vertex[0][q]=(float)(((uniquePixel[i]&0xff0000)>>16)/255.0f);
  color[q+2] = (byte)(uniquePixel[i]&0xff);
  color[q+1] = (byte)((uniquePixel[i]&0xff00)>>8);
  color[q]   = (byte)((uniquePixel[i]&0xff0000)>>16);
}
bar.inc();
bar.setDescription("Converting pixels "
  + "from input to output color space");
vertex[1] = ColorSpaceConvertor.convert(
  vertex[0], image2D.getColorSpace3D(),
  image3D.getColorSpaceManager().getColorSpace3D());
bar.inc();
int numEqual[] = new int[numUnique];
bar.setDescription("Counting number of equal colors.");
for (i=0; i < nPixels; i++) {
  numEqual[Arrays.binarySearch(uniquePixel, imagePixel[i])]++;
}
bar.inc();
int imageIndex[][] = new int[numUnique][];
int lastImageIndex[] = new int[numUnique];
bar.setDescription("Building necessary structure.");
for (i=0; i < numUnique; i++) {
  imageIndex[i] = new int[numEqual[i]];
}
bar.inc();
bar.setDescription("Indexing image pixels.");
for(i=0; i < nPixels; i++) {
  index = Arrays.binarySearch(uniquePixel, imagePixel[i]);
  imageIndex[index][lastImageIndex[index]++] = i;
}
bar.inc();
bar.setVisible(false);
bar.dispose();
```

4.16 Installasjon

Figur 4.21: Utseende på installasjonsprogrammet

De siste fasene i et produkts utvikling blir ofte undervurdert, noe som kan resultere i manglende informasjon om bruken av produktet og kompliserte eller ikke-eksisterende rutiner for installasjon av et program. Dette gjelder spesielt applikasjoner utviklet i Java, da de aller fleste småprogrammer som er fritt tilgjengelig fra Internett forutsetter at brukeren selv skal laste ned og installere de nødvendige tilleggs-biblioteker. Deretter må brukeren benytte et shell eller liknende og skrive de nødvendige kommandoer for å starte programmet, eller eventuelt starte et shell script eller en batch fil.

Dette er ikke en tilfredsstillende løsning i vårt tilfelle. For det første er programmet avhengig av flere tilleggspakker, og det er ikke åpenbart hvilke versjoner man bør benytte av disse. For det andre er programmets målgruppe ikke nødvendigvis kjent med teknologien som ligger bak Java, og det vil dermed stilles krav til brukerstøtte for å sørge for at alle får installert de filene riktig. Det er også rimelig å forvente at et slikt program skal legges inn i operativsystemets menyer automatisk, slik at brukeren kun behøver å velge en link for å starte det. Dette ble bekreftet av en gruppe grafikerstudenter på HiG, som har brukt tidlige betaversjoner av programmet for å finne de ekstreme fargeverdiene i gamuten til en skriver. Da også oppdragsgiveren ytret et ønske om slik funksjonalitet ved prosjektets slutt, ble det klart at vi burde se nærmere på dette problemet.

Vi implementerte menyen som tillater brukeren å velge mellom de forskjellige delene som skal installeres i Borland Delphi 6. Delphi er et Pascal-basert utviklingsverktøy som kun kan benyttes under Windows, noe som begrenser nytteverdien for et plattfor-

muavhengig system. Borland har imidlertid lansert Kylix, en tilsvarende versjon for Linux. Dersom man benytter seg av plattformuavhengige funksjoner og unngår å ta i bruk Win32 API, vil man dermed kunne rekompilere et Windows program utviklet i Delphi med Kylix og få en Linux versjon. Vi har dessverre ikke tilgang til Kylix på det nåværende tidspunkt, så installasjonsprogrammet for Linux måtte lages på annet vis. Samtidig med dette er strukturen i Linux og Windows så stor at mye av programkoden måtte skrives om. Vi har ikke sett på mulighetene for å lage noe tilsvarende under Mac OS X, men det ville ikke under noen omstendigheter være et aktuelt spørsmål grunnet manglende støtte for Java3D osv. slik som situasjonen er i dag.

4.16.1 Windows

Vi har utviklet et lite tilleggsprogram hvor brukeren kan velge de produktene han eller hun ønsker å installere. På denne måten kan man levere en cd som automatisk åpner en meny når man setter den inn i CD-ROMen. Denne effekten oppnås ved at man lager en tekstfil kalt *autorun.inf* og plasserer den i rotkatalogen på cden. Når Windows oppdager at en ny cd er plassert i en CD-ROM eller tilsvarende diskstasjoner, vil den lete etter denne filen og kjøre eventuelle programmer som er spesifisert der. I vårt tilfelle vil filen inneholde

```
[autorun]
open=setup.exe
```

noe som medfører at installasjonsprogrammet startes. Fra den menyen som da dukker opp, kan brukeren velge å installere Java, Java3D, JMF, JAI og hovedprogrammet. Hovedprogrammets installasjons modul har vi lagd ved å benytte InstallAnywhere, et standardprogram som lager brukervennlige installasjonsveivisere for flere operativsystemer. Siden dette er spesialtilpasset Java, betrakter vi det som et godt valg for vår applikasjon. Det vil da være relativt enkelt for brukeren å velge en installasjonskatalog, og bestemme hvor man skal legge eventuelle ikoner.

Internet har blitt en nyttig kilde til informasjon og programvare. Det var derfor naturlig å tilrettelegge produktet for en web-basert installasjon, slik at dette ble automatisert. Dette ble gjennomført ved å benytte PackageForTheWeb, et program som lar brukeren pakke ned filer til en eksekverbar fil. Når sluttbrukeren laster ned den genererte filen, vil filene pakkes opp i en midlertidig katalog. Så starter programmet et annet installasjonsprogram, som deretter fullfører installasjonen.

4.16.2 Linux (x86)

Å installere programmer under linux kan til tider være en litt problematisk oppgave. Det er få programmer som tilbyr brukere installasjonsprogrammer for linux, og brukerne er vandt til å ofte måtte kompilere programmene selv. Nå er ikke ICC3D et open source produkt, ei heller skrevet i et språk som må rekompileres så vi så det som en mulighet å kunne tilby Linux brukere nesten de samme forutsetningene som Windows brukerne. Siden Sun ikke har laget noen fullverdig installasjonsprogrammer for Linux versjonen, så måtte vi fravike litt fra Windows versjonen. Likevel føler vi at brukerne her har et lettfattet alternativ til å måtte installere hver pakke selv.

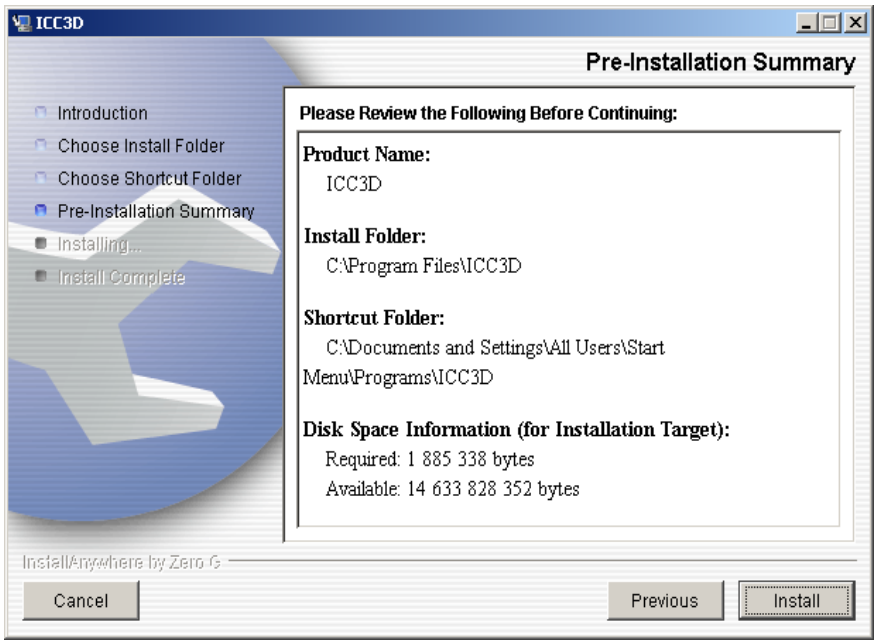
Selve programmet er skrevet i Qt (widget toolkit for c++) og er statisk linket slik at det ikke må rekompileres for å kunne kjøres. Selve GUI'et er så og si likt det som er i windows versjonen, men her ender likheten. Java pakkene for Linux har som sagt ikke noe som helst installasjons program av særlig brukervennlighet, så her måtte vi gjøre litt om. Ved å omdirigere outputen som egentlig går til terminalvinduet til en tekst viser, og tilby to knapper med valg om brukeren godtar lisensavtalen til Sun eller ei, har vi oppnådd å nesten lage et grafisk installasjonsprogram for Java pakkene. Her er det allikevel et unntak for Java3D som selv bringer opp ett terminal vindu som du må svare 'yes' eller 'no' i, noe vi ikke ser det store problemer med.Katalogen der hvor de respektive bibliotekene skal installeres velges før hvert bibliotek , men hvis man velger å installere Java2 1.4.0 fra programmet, slipper brukeren og velge riktig katalog for underpakkene (Java3D, JAI og JMF) selv.

Ett typisk funksjonskall i installeren for å få instalert en av Java pakkene (skrevet i C++, vha Qt):

```
void Installer::installJava() {
  openDialog("Pleace select where you want
              Java2 1.4.0 to be installed,
              and do make sure you got the
              correct file permissions !");
  if (dialog->result() == QDialog::Rejected)
  /* Checks if user wants to cancel the operation */
    return;

  loc = location->text();

  process = new QProcess();
  /* Starts a new process, for copying */
  process->addArgument("cp");
  process->addArgument("j2sdk-1_4_0-linux-i386.bin");
  process->addArgument(loc);

  openActionDialog();
  /* Opens up the install input dialog */

  edit->append("Copying the binary to your destination...\n");
  if (!process->start()) {
  /* Copies the file to the chosen destination */
    edit->append("Could not start copy process\n");
    return;
  }
  connect(process, SIGNAL(processExited()),
          this, SLOT(runJavaInstaller()));
}
void Installer::runJavaInstaller() {
  if (!process->normalExit()) {
    edit->append("Something went wrong, giving up...");
    delete process;
    return;
```

```
    }
    secondProcess = new QProcess(this);
    /* If all went well, it opens up another process */
    secondProcess->setWorkingDirectory(QDir(loc));
    secondProcess->addArgument("./j2sdk-1_4_0-linux-i386.bin");
    edit->append("Running the Java installer...");
    /* For redirection of output */
    connect(secondProcess, SIGNAL(readyReadStdout()),
            this, SLOT(slotDataOnStdout()));
    connect(secondProcess, SIGNAL(wroteToStdin()),
            this, SLOT(wrote()));
    connect(secondProcess, SIGNAL(processExited()),
            this, SLOT(finished()));
    if (!secondProcess->start()) {
    /* Installes the package */
      edit->append("Could not start the Java
                    install process\n");
      return;
    }
  }
```

4.16.3 InstallAnywhere



Figur 4.22: Installasjonsprogram laget av InstallAnywhere

For å installere ICC3D har vi valgt å bruke installAnywhere. Dette er et program som ligger fritt til nedlasting på nettet, og gir oss muligheten til enkelt å lage et installasjonsprogram for programmet. Dette er Java basert, og derfor er den lik for både Linux og

Windows versjonen. Dette krever selvfølgelig at du har Java installert på forhånd, så det er viktig at du gjør ting i riktig rekkefølge under installasjonen. InstallAnywhere pakken kjøres fra vårt eget install program.

Kapittel 5

Testing / kvalitetssikring

5.1 XP testing

For at man skal ende opp med et stabilt produkt som oppfører seg som forventet, er det viktig at man foretar testing og kvalitetssikring gjennom hele utviklingsprosessen. Dette er spesielt nødvendig i et komplekst systemutviklingsprosjekt som baserer seg på vanskelige algoritmer og inneholder mye funksjonalitet, slik som i vårt tilfelle. Dersom man gjennomfører testing kontinuerlig fra man begynner implementeringen til man er ferdig, reduserer man risikoen for å få et resultat som er preget av bugs og kode som ikke håndterer ulike unntakstilfeller som kan dukke opp en gang iblant, samtidig som det tillater optimalisering av kode i alle faser av prosjektet, ikke kun rett før prosjektets slutt.

Ekstremprogrammering erkjenner denne problemstillingen, og gir en helhetlig løsning som dekker de fleste aspekter ved systemutviklingen.

- Fortløpende realisering av nye versjoner med små mellomrom sørger for at problem blir oppdaget så raskt som mulig, og lar utviklerne ta tak i problemet før det påvirker andre elementer i applikasjonen.
- Kravet om at man hele tiden skal satse på et enkelt design såfremt dette er mulig, minsker muligheten for at deler av programmets kode kan bli tungrodd etter hvert som applikasjonen utvides. Dersom man når man skal legge til funksjonalitet ser at man kunne forenklet måten noe gjennomføres på, kreves det at man rydder opp i koden. Ved å benytte en viss andel av tiden til å foreta slike reorganiseringer, ender man opp med et bedre resultat som er mindre utsatt for bugs som oppstår grunnet misforståelser.
- Alle nye deler av programmet skal implementeres og testes utenfor selve applikasjonen før de inkluderes i resten av koden. Vi fant at dette ga mange fordeler med tanke på oppdaging av småfeil eller potensielle problemer ved de forskjellige algoritmene vi implementerte. For grundig uttesting av koden for Quick Hull, alpha shapes og Delaunaytrianguleringer lagde vi en klasse som kunne konstruere punkter med forskjellige tilfeldige fordelinger som følger et visst mønster, for eksempel punkter på overflaten til en kule. Dermed var det enkelt å gi

tilfeldig inputdata til algoritmene, samtidig som initialisering av generatoren som lager tilfeldige tall sørget for at vi kunne gjenta det samme forsøket flere ganger. Dermed var det enklere å optimalisere koden ved eksperimentering, fordi tidtakning av operasjoner var direkte sammenlignbare, noe det ikke ville vært med forskjellige tallverdier hver gang man kjørte testprogrammet.

- Vår erfaring var at parprogrammering medførte at sannsynligheten for bugs minket temmelig drastisk, da det ofte hendte at den personen som ikke benyttet tastaturet oppdaget feil mens de ble skrevet, i motsetning til å måtte foreta en kompiler-test-fiks syklus som er vanlig ved andre typer programmering. Det at to personer samarbeidet om å skrive en funksjon, førte også i flere tilfeller til en bedre implementasjon enn hva man ville fått dersom de arbeidet hver for seg, siden vi kunne foreslå forskjellige innfallsvinkler på problemene og diskutere oss frem til en løsning.

- Kunden, i vårt tilfelle oppdragsgiver og veileder, fungerte som betatestere for vår applikasjon. De var meget aktive og hjelpsomme, og kom stadig med innspill om feil og forbedringer som vi deretter forsøkte å ta hensyn til i den videre utviklingen av produktet.

- Automatiserte tester ble gjennomført for flere deler av applikasjonen. Vi kodet en test for ulike fargerom, som fungerer ved at farger spredt utover hele fargerommet ble konvertert til og fra XYZ. Dersom verdiene man får etter å ha utført disse to konverteringene er forskjellige fra de opprinnelige verdiene (distansen i 3D er større enn en valgt lengde), betyr dette at fargerommets konverteringsrutiner ikke fungerer som de skal. Tetraederstrukturen som bygges av flipalgoritmen og segment maxima visualiseringen blir testet for å se om alle tetraedernene er konvekse, samt at nabolinkene er korrekt plassert. Denne testen avslørte flere problemer ved segment maxima, som verken vi eller oppdragsgiver/veileder var klar over, men som vi etterpå kunne løse. I løpet av utviklingsfasen benyttet vi mange andre tester lagt inn i koden, som sørget for at vi fikk meldinger dersom det oppstod uventede situasjoner.

- Manuell testing i form av aksepteringstester ble foretatt for å forsikre at våre løsninger på brukerhistoriene ellers var i samsvar med spesifikasjonene.

- Vi konsentrerte oss om å optimalisere de funksjoner som var mest tidskritiske, og forsøkte å benytte de muligheter som eksisterer i Java på best mulig måte. Dette ga utslag i blant annet koden som prosesserer bildet, som benytter et BitSet (en lang rekke bits som kan settes til 1/0) og innebygde funksjoner for for rask sortering og binærsøk for å indeksere pikslene. Java er i seg selv ikke beregnet for programmer som krever høy ytelse, men etter optimaliseringen bør stort sett alle operasjoner utføres uten venting som oppfattes som slitsomt for brukeren.

5.2 Backup

Å miste hele prosjektet er for alle et mareritt, og vi har prøvd å beskytte oss på alle kanter når det gjelder dette. Å bruke en CVS (concurrency version control) server ble lenge vurdert helt til at vi fant ut at det var lite hendig i henhold til at vi kun var tre på gruppen, samtidig som XP krever at man skal programmere i par. Derfor valgte vi

en noe mer manuell backup modell, der koden med jevne mellomrom ble kopiert fra utviklingsmaskinen til flere andre forskjellige maskiner vi har kontroll over. Område som skolen har gitt oss på en server på skolen, ble ikke så mye brukt da vi har dårlig erfaring med hva It-tjenesten har å tilby av stabilitet.

De maskinene som ble brukt som backup maskiner, var en bærbar maskin hvis plassering selvfølgelig var variabel. Koden ble mailet til denne maskinen. Maskin nummer to befant seg på Sørbyen Studenthjem, dit koden blir kopiert gjennom skolens eget nettverk (selv om It-tjenesten nesten klarte å hindre oss i å gjøre dette også). Den siste maskinen som inneholdt koden var utvikler maskinen, som hele tiden befant seg på skolen under arbeidet.

Når koden ble kopiert til de andre maskinene varierte med hvor mye som var gjort, men mellom utvikler maskinen og den bærbare ble koden kopiert flere ganger i løpet av dagen. Den siste stasjonære maskinen på Sørbyen mottok kode ved dagens slutt.

Vi hadde ikke ett tilfelle av å miste kode, det nærmeste var da en av oss nesten klarte å kopiere over en del nyutviklet kode på utviklermaskin ved en feiltagelse. Heldigvis viste det seg at så ikke var tilfelle.

Kapittel 6

Sluttdiskusjoner

6.1 Drøftinger og diskusjoner underveis

Fordi prosjektets delmål og deres prioriteringsrekkefølge forandret seg etter hvert som nye momenter ble brakt på bane av oppdragsgiver, veileder og gruppens medlemmer, var det påkrevd med daglige diskusjoner om forskjellige valg som måtte gjøres. Det var imidlertid to avgjørelser som stod sentralt i gjennomføringen av prosjektet, og la retningslinjer for den arbeidsmetodikk som gruppen benyttet seg av, og den implementasjonen vi endte opp med.

Vi var fra starten av i tvil om hvorvidt Java og Java3D egnet seg som utviklingsverktøy for en applikasjon med de egenskapene som vi forutså ville være nødvendige. Likevel så vi klare fordeler ved en slik løsning, spesielt grunnet oppgavens krav om en utvidbar løsning som ikke baserte seg på en enkelt plattform. Som tidligere beskrevet undersøkte vi andre muligheter, men kom frem til at disse alternativene ikke tilfredsstilte de krav vi stilte, og at det trolig var overkommelig å benytte Java3D dersom man på beste måte unngikk de svakhetene dette medførte.

I etterkant må vi si oss godt fornøyd med dette valget. Vi har vært i stand til å implementere den funksjonalitet som var ønsket raskere enn hva vi opprinnelig hadde trodd grunnet språkets innebygde muligheter, fordi vi ikke behøvde å kode mange mindre detaljer men kunne konsentrere oss om andre sider ved implementasjonen. Det må bemerkes at andre APIer med en mer direkte kontakt med hardware kunne gitt flere muligheter for hastighetsforbedring, men vi anser det som svært lite sannsynlig at vi da hadde vært i stand til å holde et like raskt utviklingstempo.

Systemutviklingsmodellen man velger å følge i et prosjekt legger føringer på arbeidet som gjøres, samtidig som den bør gi gruppemedlemmene de styringsmuligheter som behøves i et større prosjekt. Tradisjonelle utviklingsmodeller passer ikke for systemutviklingsprosjekter hvor kravspesifikasjonen fastsettes underveis, og mangel på informasjon forhindrer en grundig analyse før man iverksetter implementeringen av applikasjonen. Det var derfor et relativt åpent spørsmål hvilken modell vi skulle benytte. Etter en periode med analyse av de tilgjengelige alternativene (for eksempel RUP, Rational Unified Process), konkluderte vi med at ekstremprogrammering var spesialtilpasset for vårt bruk.

Dette tillot oss å følge en kontinuerlig prosess, der iterasjonene førte til en stadig bedre forståelse av applikasjonens arbeidsfelt og de krav dette medfører. Både oppdragsgiveren og veilederen var villig til å delta under disse premissene, og deres aktive rolle var en forutsetning for prosjektets suksess.

6.2 Kritikk av oppgaven

Under utdelingen av oppgavene var det lite som virket som det kunne fenge oss. Vi hadde bestem oss på forhånd at vi ikke ville gjøre nok en "få en bedrift ut på nett" oppgave, men gjøre noe som kunne være litt utfordrende. Dette har absolutt denne oppgaven gitt oss, den har bydd på utfordringer som har gjort arbeidet både inspirerende og morsomt. Vi angrer ikke på valget.

Oppdragsgiver og veileder har gitt et godt inntrykk av å være oppriktig interessert i hva vi har gjort, og vist et stort engasjement i oppgaven. Det at oppgaven ikke hadde et fastspikret rammeverk på forhånd, gjorde at vi valgte å bruke XP som utviklingsmodell. Dette gjør at vi ikke kan sette fingeren på at noen har avviket fra det som var opprinnelig planlagt. Oppdragsgiver og veileder viste tidlig at de hadde et mål med oppgaven, og ikke ønsket at den skulle bli en av mange oppgaver som blir glemt i ettertid.

Oppgaven var bra, veileder og arbeidsgiver gjorde en bra jobb, og vi håper vi har levert et bra produkt.

6.3 Videre arbeid, nye (hoved)prosjekter

Vi ble tidlig klar over at dette ikke var ett program som vi i løpet av den perioden vi hadde til rådighet kunne bli ferdig med, men noe som kan bygges videre på i senere prosjekter. Derfor er det lagt stor vekt på at programmet er modulært og det skal ikke være en alt for stor oppgave å utvide det videre med nye funksjoner og muligheter. Oppdragsgiver har store forhåpninger med prosjektet, det skorter ikke på ideer om videre funksjonalitet, så det ligger absolutt et grunnlag for videre arbeid med programmet.

Oppdragsgiver har forespeilet muligheten for kommersialisering av produktet, noe som vil kreve videre arbeid med det som nå er gjort. Dette gjelder ikke bare på brukervennligheten, men også funksjonaliteten.

Det er også snakk om at en av oss skal ta over prosjektet på egen basis siden han skal studere videre på skolen. Oppgaven har mulighet til å kunne utvikle seg til å bli en diplomoppgave, så det gjenstår å se på viljen om han vil gjøre noe med det.

6.4 Evaluering av gruppens arbeid

6.4.1 Innledning

Det var med delte følelser og meninger vi gikk igang med dette prosjektet. Siden det er snakk om et hovedprosjekt, stilles det relativt store krav til resultat. Både av produkt og rapport. Dette medfører at det vil kreve mye innsats av den enkelte.

Valg av oppgave var ikke veldig vanskelig. Vi følte alle en viss tilknyttnig til oppgaven. Ikke minst fordi vi er interesserte i emnet, men også fordi den hørtes spennende ut og fordi det ville bli en utfordring.

6.4.2 Organisering

Gruppen består av 3 deltakere. Vi følte derfor at det passet seg å utnevne en prosjektleder som ville ha siste ord i eventuelle problemer eller uoverensstemmelser mellom deltakerene. Men denne problemstilling har vi ennå ikke opplevd under prosjektets gang. Noen vil kanskje si det er synd, da det også er en erfaring å ta med seg at ikke alt går på skinner.

Selv om vi valgte å utnevne en prosjektleder, har ikke denne personen hatt noe mer overordnet ansvar enn de andre på gruppen bortsett fra å skrive statusrapporter og å være kontaktperson mot oppdragsgiver og veileder. Dette er noe vi ble enige om for å ikke lage en "sjef" i gruppen. Det har hele tiden vært alles ansvar å gjøre den jobben den enkelte er blitt tildelt og å gjøre det innen angitte tidsfrister. Når det gjelder statusrapportene, har resten av gruppen vært med på å godkjenne eller kommet med forslag til endringer dersom de mener det har vært nødvendig.

6.4.3 Fordeling av arbeidet

Siden vi har valg å arbeide etter utviklingsmodellen XP, har vi ikke kunnet fordelt arbeidet mellom oss i like stor grad som andre utviklingsmodeller tillater. Dette pga. XP's arbeidsform som sier at det alltid skal arbeides parvis. Dette har også vært en fordel for oss da det ville medført problemer dersom vi skulle sittet alene og kodet hver for oss. Endringer ett sted medfører som regel endringer andre steder også, og dersom flere koder samtidig vil det bli mye unødvendig rot med å "sy sammen" den resulterende koden. Dette skjedde noen ganger, men vi satte raskt opp rutiner for å unngå det.

Men selv om parvis arbeid er en av grunnstenene i XP, har vi likevel kunnet fordele noen oppgaver oss imellom. Dette har vært typiske oppgaver som ikke har hatt noen relevans i forhold til annen kode. Mao. kode som er frittstående og som fungerer slik den står.

Utover dette har vi prøvd å fordele arbeidet innenfor områder vi føler den enkelte vil kunne bidra mest. Selv om vi har gjort dette, vil det være feil å si at en person har gjort det og en annen har gjort det, da det ikke er slik vi har jobbet. Vi har alle vært med på å utvikle samme produkt.

Likevel må det kunne sies at noen har jobbet mer på visse områder enn andre. Men dette gjelder igjen for alle. Dette er områder som feks. implementasjon av avanserte algoritmer, utforming av ikoner, GUI og generell layout og organisering / strukturering av kode.

6.4.4 Prosjekt som arbeidsform

Prosjektet som arbeidsform har stor relevans med hvordan arbeid blir utført i arbeidslivet. Det er derfor en veldig viktig og ikke minst nyttig erfaring å ta med seg videre.

Ikke bare mtp. at man får en oppgave/problemstilling som skal løses, men også at man jobber i en gruppe for å løse et problem sammen. Dette stiller større krav til evnen å sammarbeide og kommunisere og hvordan man forholder seg til andre mennesker. Både i med- og motgangstider. Et prosjekt vil som regel alltid oppleve problemer, enten det er relatert til problemstillingen eller det er konflikter innad i gruppen. Totalresultatet av prosjektet vil derfor i høy grad være avhengig av at disse to punktene fungerer.

Dersom problemer skulle oppstå mellom gruppemdlemmene, vil nok det beste være å løse denne konflikten fortest mulig da dette vil legge en demper for videre fremgang og utvikling av prosjektet.

Det er også en vesentlig fordel dersom gruppen består av mennesker med kompetanse innenfor forskjellige områder. Man vil dermed kunne komme lenger og vise til bedre resultater enn om alle hadde kunnet akkurat det samme. En annen fordel er at man kjenner de man jobber sammen med og vet hvor den enkelte står og hva han/hun er i stand til å gjøre. Det er også viktig at tidsfrister overholdes av den enkelte slik at videre arbeid kan påbegynnes/avsluttes som planlagt. Sistnevnte punkt vil være spesielt viktig i arbeidslivet da andre instanser ofte vil være avhengig av å kunne se (del)resultater for eventuelle avgjørelser videre.

Når det gjelder gruppen vår, må det sies at vi har fungert veldig bra sammen. Vi har hatt våre problemer, men dette har bare vært i tilknyttning til selve oppgaven og hvordan en problemstilling skal/bør løses. Vi har også raskt kunne fordele emner oss imellom mtp. kompetanse innenfor forskjellige fagområder.

6.4.5 Subjektiv opplevelse av hovedprosjektet

Gjennom 3 år på høgskolen har vi nesten hele tiden vært vant til å gå på forelesninger og evt. øvingstimer for å praktisere gjennomgått stoff. Vi har hatt noen mindre prosjekter, men omfanget av disse har ikke vært så stort og krevende som et hovedprosjekt.

Når vi nå skulle gå i gang med hovedprosjektet var det av alles oppfating at det ville bli lærerikt og spennende. Omfanget av hovedprosjektet er også økt med 2 vt. fra 4 til 6, noe som vil stille større krav til både produkt og rapport. Men dette har egentlig ikke vært noe vi har tenkt så mye over. Vi har hele tiden hatt i bakhodet at vi skal gjøre det beste ut av det og nå de målene vi har satt oss.

Kapittel 7

Konklusjon

Etter å ha gjennomført hovedprosjektet ved HiG sitter prosjektgruppen igjen med mange erfaringer. Vi har satt oss inn i nye arbeidsmetoder, samt et fagfelt som for oss var helt ukjent. Det var med en viss spenning vi påtok oss prosjektet, siden det var klart at det ville bli mange utfordringer underveis med en oppgave som skiller seg fra andre typiske prosjekter.

Vi mente at vi hadde satt oss nokså høye mål etter den foreløpige analysen i forprosjektet som ble gjennomført, men ved projektets slutt kan man se at vi har nådd alle hovedmål, samt de aller fleste av delmålene, med unntak av litt funksjonalitet som ble nedprioritert i løpet av prosjektets levetid til fordel for andre muligheter. I tillegg til de eksplisitte brukerhistoriene som ble gitt av oppdragsgiver, har vi også funnet tid til å legge inn støtte for mange andre operasjoner.

Gruppemedlemmene klarte å beholde motivasjonen på et høyt nivå gjennom hele prosjektfasen, noe som ga utslag i økt arbeidsinnsats og effektivitet. Samarbeidet innad i gruppen opplevdes som en positiv opplevelse, og kommunikasjonen med veileder og oppdragsgiver var bra. Møtene var med på å avklare problemer, og alle interessentene kunne komme med innspill til prosjektets fremdrift. Det ville ikke vært mulig å gjennomføre prosjektet uten de råd og ideer som veileder og oppdragsgiver kom med på disse møtene.

Applikasjonen som vi har utviklet gir brukeren mange valgmuligheter, og den egner seg godt for videreutvikling grunnet den interne oppbygningen vi har tatt i bruk. Det er ikke nødvendig med mye kunnskap om implementasjonen for å kunne legge til nye algoritmer og datastrukturer, det eneste som trengs er en forståelse for hvordan den dynamiske lastingen fungerer og grensesnittet mellom modulene. Programmet fungerer på en tilfredsstillende måte, og dekker de krav som ble fremsatt som en del av den fortløpende diskusjonen med oppdragsgiver. Det er likevel store muligheter for videre arbeid, fordi målgruppen har behov for mye funksjonalitet som det ikke er mulig å innføre i løpet av dette prosjektets levetid.

B. Joe construction of three-dimensional Delaunay triangulations using local transformations, Comput. Aided Geom. Design 8(1991), 123-142.

Bibliografi

[1] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. ACM Trans. Graphics 13(1994), 43-72.

[2] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graphics 9 (1990), 66-104.

[3] C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa, The Quick Hull algorithm for Convex Hulls. ACM Trans. on mathematical software (1996).

[4] Sun Microsystems, hjemmeside `java.sun.com`.

[5] Easy RGB, hjemmeside `easyrgb.com`.

[6] sRGB standard, hjemmeside `srgb.com`.

[7] ICC (International Color Consortium) spesifikasjonen, hjemmeside `www.color.org`.

Register

2D, 23, 46
3D struktur, 46

Ansvarsforhold, 3
Arbeidsformer, 7

Backup, 86
Begrensninger, 74

Capture, 75

Dynamisk klasseloading og modularitet, 24

Fargerom, 22, 39

ICC profil, 52
InstallAnywhere, 83
Installasjon, 80
Interaksjon, Segment, 71
Interaksjon, Standard, 71
Interaksjoner, 70
IT8.7/2, 49

JAI, 36
Java, 32
Java3D, 33
JMF, 37

Konvertering av fargeverdier, 40

Layout, style, fonter, 18

Oppdragsgiver, 2
Optimalisering, 77

Parsere, 48
Platformer, 37
Platformuavhengighet, 17
Prosjektorganisering, 2

Testing, 85
Tetrahederinterpolasjon, 72
Transformasjoner, globale, 72

Utviklingsmodell, 5

Versjonshåndtering, 16
Visualisering, Alpha shapes, 65
Visualisering, Convex Hull, 61
Visualisering, Kvantisert, 56
Visualisering, Segment maxima, 57
Visualisering, Standard, 55
Visualiseringer, 55

Web, 16