HOVEDPROSJEKT:

TITTEL

CMTF-3D

Complex 3D Model, Tiny File

FORFATTER(E): Lars Anundskås
Jon Erik Bakken
Thomas Austad

Dato: Gjøvik, 12 Mai 2005

Sammendrag av hovedprosjekt

Tittel:	CMTF3D	Nr: 10
	Complex 3D Model, Tiny File	Dato: 19.05.05
Deltaker(e):	Lars Halvor Anundskaas	
	Jon Erik Bakken	
	Thomas Austad	
Veileder(e)	Øyvind Kolås	
Oppdragsgiver:	Sunsteam Search DA ved Sjur Mathisen	
	sjur@sunsteam.com	
Kontaktperson:	Thomas Austad	
Stikkord (4)	3D modelering, software bibliotek, C og C++	
Antall sider: 109	Antall bilag: 0	Tilgjengelighet: Åpen

Kort beskrivelse av hovedprosjektet:
Prosjektet består av et software bibliotek for generering av komplekse 3D modeller, samt et verktøy for modellering av disse.

Biblioteket er plattform- og 3D API uavhengig, og leverer fra seg 3D modeller
Disse modellene har en struktur som lett kan integreres i
spill og andre typer realtime visualiseringsapplikasjoner. Genereringen av 3D modellene skjer ved at biblioteket leser inn filer som inneholder beskrivelsen av de operasjoner som ble utført under modelleringsfasen i verktøyet.

Modelleringsverktøyet setter 3D artister i stand til å lage komplekse
3D modeller ved bruk av enkle 3D-objekter som kan forandres og settes sammen.

Forord

cmtf3d - er utviklet av tre studenter på datalinjen ved Høgskolen i Gjøvik våren 2005. Prosjektoppgaven ble utformet høsten 2004 som en innleveringsoppgave i OOSU faget. Den ble videreutviklet i samarbeid med Sunsteam Search DA representert ved Sjur Mathisen, og har bestått i å utvikle et plattformuavhengig bibliotek for generering av 3D modeller, og et modelleringsverktøy som viser at biblioteket fungerer.

Vi ønsker å takke følgende personer for deres bidrag i prosjektet:

- Øyvind Kolås, vår veileder.
- Kristian Kirkesæter for velvillig utlån av Apple iBook og bistand under uttesting av biblioteket på denne.
- Kostas Pataridis (aka Navis/ASD).
- COPWALL gruppen for en fin LaTeX mal for sammendraget.

Innhold

Figurer

Tabeller

1 Innledning

Det har vært en trend de siste årene at utviklingen av teknologi på hardwaresiden stadig har eskalert i tempo. Dette gjelder spesielt teknologi rundt grafikkort og utvikling av mindre enheter som mobiltelefoner, håndholdte datamaskiner og spillkonsoller. Denne typen teknologi har i dag fått en prosesseringskraft som man for få år tilbake bare kunne drømme om.

Spillindustrien, blant andre, har tatt til seg denne rivende utviklingen, og prøver hele tiden å utnytte den til enhver tid gjeldende hardwareteknologi maksimalt. Dette har medført at spillverdener blir mer komplekse/detaljerte, og spillene har blitt veldig ressurskrevende både til hardware og lagringskapasitet. Det ser imidlertidig ikke ut som om spillprodusentene har anstrengt seg for å begrense den fysiske datastørrelsen. Spesielt med tanke på 3D modeller og verdeners størrelse på disk.

Man gjør i skrivende stund store fremskritt for å stadig forbedre hardwareakselerert 3D til mobile enheter(f.eks. mobiltelefoner og PDAer). Blandt annet så har norske Falanx Microsystems(`http://www.falanx.no/`) utviklet en grafikk prosessor kalt *Mali* for slike enheter.

Per i dag har vi ikke sett applikasjoner eller biblioteker for generering av 3D-modeller som tar sikte på å begrense filstørrelsen. Dette selv om vi vet at innen demoscenen[1], lages det demoer av kompliserte 3D verdener med grafikk og musikk på 4 og 64 kilobyte. Vi antar dermed at slike verktøy finnes, men de er ikke distribuert.

En løsning som tar for seg både modellering og lagring til filer av begrenset størrelse anser vi som nyttig for både PCer og mobile enheter. På PCer kan modeller til spill overføres raskt via internett som små filer, og genereres mens man spiller, og mobile enheter er avhengige av at ting ikke tar for stor plass grunnet begrenset lasgringskapasitet(per i dag). Nytteverdien av et slikt verktøy bør derfor være stor.

1.1 Oppgavedefinisjon og avgrensninger

Prosjektet består av ett software bibliotek for generering av komplekse 3D modeller ut fra små filer(her anses filer på 10 kilobyte eller mindre som små), samt et verktøy for modellering av disse.

[1] En datarelatert subkultur hvor folk kappes om å lage de beste grafiske presentasjonene.

1.1.1 Biblioteket

Biblioteket skal være plattform- og 3D API uavhengig, og må kunne levere 3D modeller. Disse må ha en struktur som lett kan integreres i spill og andre typer visualiseringsapplikasjoner.

Denne strukturen kalles med faguttrykket *mesh* og i vårt prosjekt har vi definert *mesh* til å være er en samling 3D punkter og polygoner som knytter punktene sammen til flater. Tilsammen representerer dette hvordan en 3D modell skal tegnes opp. Der hvor ordet *meshing* er brukt henspiller dette på generering av en mesh.

Genereringen av mesh ut fra 3D modellene skal skje ved at biblioteket leser inn filer hvis innhold beskriver de steg som ble utført under modelleringsfasen. Genereringen av modellene bør ikke ta for lang tid, så valg av algoritmer og datastrukturer i biblioteket er kritisk.

1.1.2 Modelleringsverktøy

Verktøyet skal sette 3D artister i stand til å lage komplekse 3D modeller ved bruk av enkle 3D-objekter som kan forandres og settes sammen. Ved lagring av en 3D-modell skal kun stegene som ble utført for å lage modellen skrives til fil. Dette for å bruke minst mulig diskplass.

Siden de fleste 3D artister bruker Windows 2000/XP, må verktøyet kunne kjøres på denne plattformen, men det er også ønskelig at verktøyet kan brukes på Linux og/eller UNIX generelt.

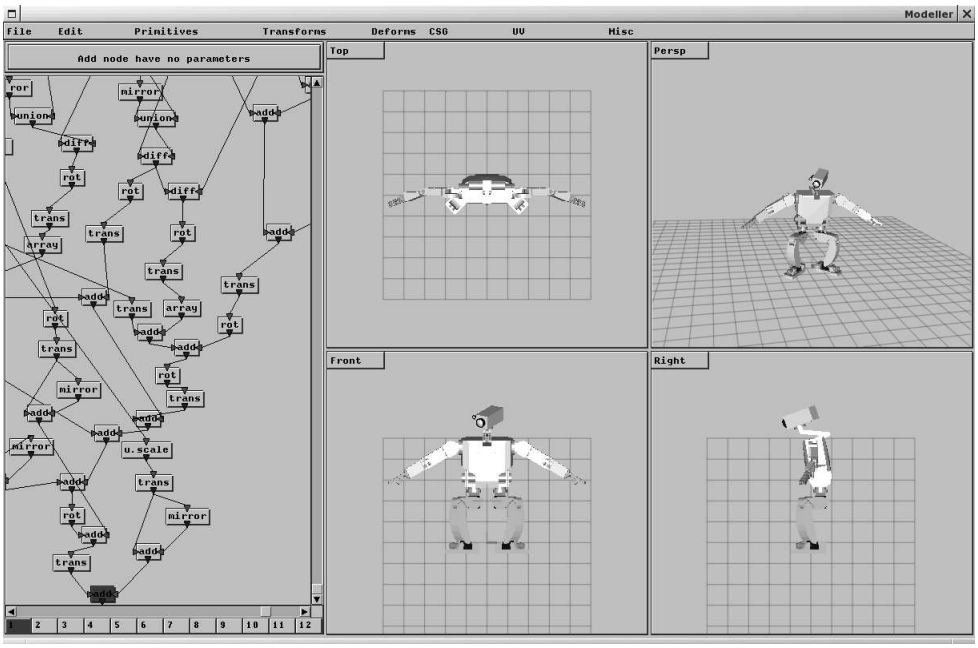
Modelleringsverkøyet skal gi brukeren mulighet til å kunne jobbe med modelrepresentasjonen på en best mulig måte. For å muliggjøre dette må brukeren kunne se modellen fra flere synsvinkler[2] samtidig. Det skal også være enkelt å redigere 3D-modellen, men vi vil understreke at verktøyet ikke skal være noen konkurrent til for eksempel 3D Studio Max.

1.2 Målgruppe

1.2.1 Rapporten

Målgruppen for rapporten er i hovedsak oppdragsgiver, veileder og sensor som skal vurdere prosjektet, men også andre interessenter. Språket i rapporten er av en teknisk art og gir et innblikk i hele prosjektprosessen. Noe forståelse for programmering og systemutvikling vil være en fordel, men er ingen nødvendighet for å lese den.

[2]med synsvinkler mener vi egentlig en samling orthografiske og perspektiviske views



Figur 1: Modellering av en robot.

1.2.2 Bibliotek

Biblioteket er rettet mot utviklere av 3D-programvare som for eksempel spillutviklere.

1.2.3 Applikasjonen

Applikasjonen er rettet mot 3D-modellerere og personer med erfaring fra tidligere modelleringsverktøy.

1.3 Formål med oppgaven

Bibliotekets hovedmål er å krympe 3D-modellbeskrivelser slik de er lagret på filnivå. Dette gir en bedre utnyttelse av kapasiteten som er tilgjengelig på lagringsmedier(harddisk, CD, DVD). En bieffekt av dette er at det gir f.eks. spillutviklere mulighet til å legge ved enda flere modeller, eller til å benytte denne plassen til andre ting som f.eks. grafikk.

Vi valgte denne oppgaven fordi vi fant temaet interresant og anså det som en god mulighet til å fordype oss innen flere fagområder(Meshing, objektorientert-systemutvikling og programmering samt strukturert programmering).

1.4 Egen kompetanse og bakgrunn

Studentene studerer bachelor i ingeniørfag, data ved Høyskolen i Gjøvik og har tatt fordypning innen programmering.

Som et ledd i studie har alle programmert en god del, og til sammen har det blitt utviklet bl.a. to IRC klienter i programmeringsspråket Java som prosjektoppgaver i Programvareutvikling, samt tre interpretere for språket MinimL i faget kompilator teknikk.

En av gruppemedlemmene har også kodet flere grafiske demoer, og verktøy for å sette sammen slike programmer.

Under gjennomføring av hovedprosjekt har vi dratt spesiell nytte av matematikk, objektorientert systemutvikling og algoritmiske metoder slik de har blitt undervist ved HiG.

I løpet av prosjektet har vi måtte lære oss Constructive Solid Geometry[3](CSG), platformuavhengighet samt bruk av utviklingsverktøy og feilrettingsmetoder.

1.5 Rammer

1.5.1 Arbeidsmetoder

Tidsmessig har vi hatt ca fire og en halv måned til rådighet for gjennomføring av hovedprosjekt; fra 5/1-2005 til 19/5-2005. I denne perioden har vi stort sett hatt kontortidfra 10:00 til 15:00 på hverdager. Det har også blitt jobbet utover dette etter behov. Vi har ført daglog med oversikt over arbeidstimer og utført arbeidsoppgave. Denne oversikten er vedlagt i appendix I .

Vi har jobbet tett sammen om de ulike arbeidsoppgavene, men fordelt hovedansvaret for forskjellige deler av prosjektet som vist på Ganttskjema (Appendix B)

For å strukturere arbeidsprosessen valgte vi å bruke en inkrementell utviklingsmodell, men supplementerte med teknikker fra RUP og UML i design fasen.

Når det gjelder valg av inkrementer så delte vi opp prosjektet i fire moduler, hvor to og to moduler ble utviklet parallelt.

[3]En teknikk for å modellere komplekse objekter.

1.5.2 Prosjektorganisering

Deltakere
Gruppen består av Thomas Austad(Gruppeleder), Jon Erik Bakken og Lars Anundskås. Hvis det skulle oppstå uoverenstemmelser innad i gruppen er det gruppeleder sitt ansvar å greie opp i disse.

Ansvarsforhold
Under utvikling av de forskjellige modulene i prosjektet har gruppemedlemmene jobbet sammen under utviklingsfasene. Når moduler så har gått over i en vedlikeholdsfase har hovedansvar for modulene blitt fordelt som vist under:

- GUI - Jon Erik Bakken
- CMTF3D - Thomas Austad
- MOD - Lars Anundskås

1.5.3 Fremdriftsplan

I fremdriftsplanen ble prosjektet planlagt så nøye og realistisk som mulig. Dette ble rammen for hva som skulle gjøres til en hver tid, og ble i sin helhet fulgt. Dette er vist i Ganttdiagram(Appendix B). Vi gjennomførte møter med prosessveileder etter behov. Ett av disse var statusmøte og statusrapporten er vedlagt i Appendix H. Det ble også gjennomført møter med arbeidsgiver, men det meste av kommunikasjonen foregikk ved hjelp av epost og instant messaging.

1.6 Øvrige roller

Prosjektets oppdragsgiver er firmaet Sunsteam Search DA representert ved Sjur Mathisen.

Prosjektets prosessveileder har vært Øyvind Kolås som har bidratt med mange gode innspill.

1.7 Organisering av rapporten

1.7.1 Kapitteloppsummering

Kapittel 1 - Innledning
Inneholder en overordnet beskrivelse av prosjektet, samt en beskrivelse av annsvarsforhold og innvolverte.

Kapittel 2 - Kravspesifikasjon

Inneholder alle krav til biblioteket og applikasjonen som forelå før utviklingen startet.

Kapittel 3 - Design
Inneholder designet av biblioteket og applikasjonen.

Kapittel 4 - Implementasjon
Inneholder en beskrivelse av hvordan vi har implementert løsningen i forhold til kravspesifikasjonen og designet, samt en beskrivelse av utviklingsmiljø, verktøy og valg av disse.

Kapittel 5 - Kvalitetssikring og testing
Inneholder en beskrivelse av de rutiner og aktiviteter som er fulgt og utført i forbindelse med testing og kvalitetssikring av løsningen.

Kapittel 6 - Drøfting / diskusjoner
Inneholder kritikk og evaluering av det utførte arbeidet, og en beskrivelse av mulig videreutvikling.

Kapittel 7 - Konklusjon
Inneholder prosjektets konklusjon og oppsummering.

Appendix er gitt etter kapittel 7, og viser til literatur og nettsteder som er benyttet til fordypning, og ressurser til arbeid med prosjektet.

1.7.2 Terminologi

Oppgaven er skrevet på bokmål og inneholder faguttrykk som er oppført i terminologilisten. Denne er vedlagt i appendix 7. Vi vil utdype noen sentrale begreper i dette kapitlet.

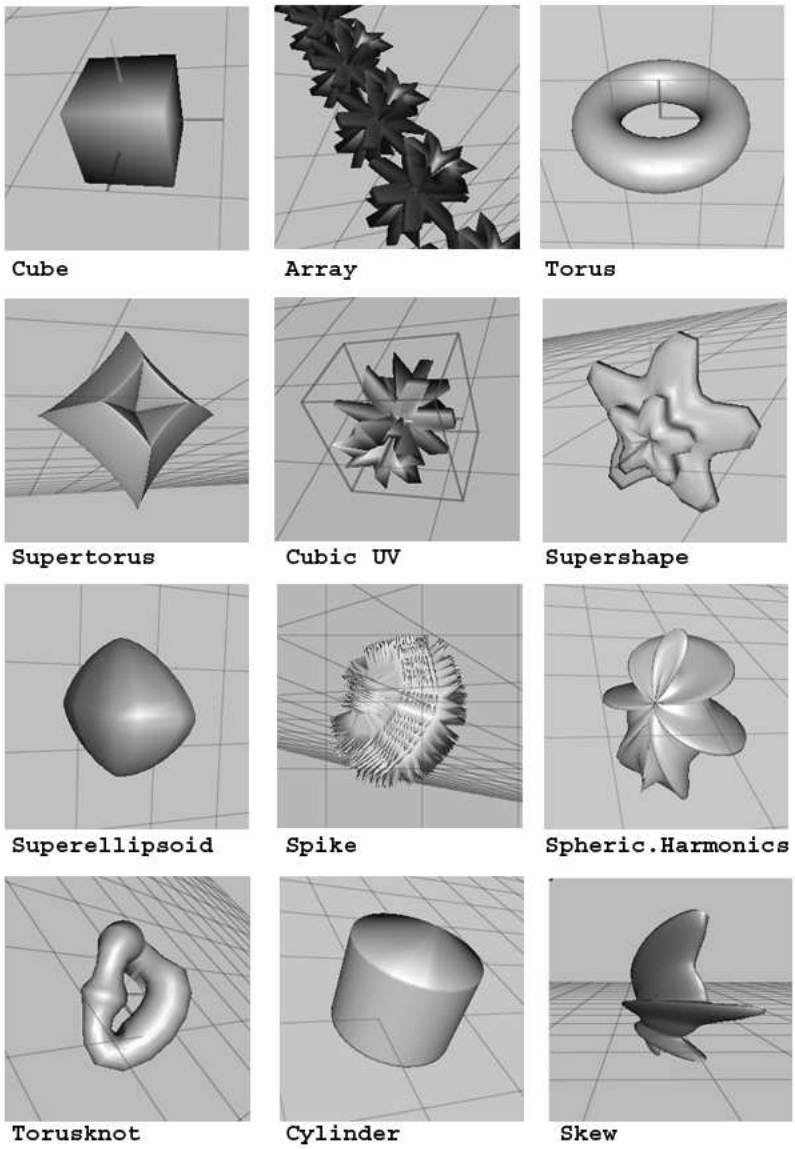
1.7.3 Layout

Rapportens hoveddeler vil være organisert i kapitler, der hvert kapittel vil ha underkapitler for forskjellige emner innen hver hoveddel. Disse underkapitlene vil også inneholde videre inndeling. Det første er nummerert og finnes også i innholdsfortegnelsen.

Ord som finnes i terminologilisten er markert med 'referanser' til terminologlisten. Utover dette vil den samme standardfont bli benyttet gjennom hele rapporten for å gi ett helhetlig inntrykk.

1.7.4 Presentasjon av modelleringsverktøy

Modelleringsverktøyet, som er vårt sluttprodukt, ligger på vedlagte CD-rom. Under har vi lagt ved noen bilder som viser noen enkle modeller og operasjoner utført i verktøyet.



2 Kravspesifikasjon

2.1 Introduksjon

Løsning består av ett software bibliotek for generering av komplekse 3D modeller, samt et verktøy for modellering av disse.

2.1.1 Krav til biblioteket

Biblioteket skal kunne generere 3D-modeller ut fra en beskrivende datastruktur. Denne strukturen skal innholde primitiver og operasjoner på disse.

Biblioteket skal også ta seg av lagring og innlesing av 3D-modellbeskrivelser på en platformuavhengig måte. Det er også et krav at filen skal ta minst mulig diskplass.

Biblioteket skal ha et API som kan brukes på to nivåer, hvor det ene består av enkel innhenting og generering av 3D-modeller fra fil/minneområde, mens det andre nivået gir full tilgang til selve 3D-modell generatoren.

2.1.2 Krav til modelleringsapplikasjonen

Med applikasjonen skal man kunne modellere 3D-modeller ved å bruke bibliotekets funksjonalitet.

Primitiver/operasjoner skal vises som en graf, og brukeren skal selv kunne lenke sammen primitiver/operasjoner ved å dra linjer mellom disse.

Brukeren skal kunne se forandringer på 3D-modellen etterhvert som han/hun redigerer.

Det skal være mulig å importere eksisterende modeller som er laget i modeleringsverktøyet fra fil, og sette disse inn i den modellen som redigeres.

Det skal være mulig å ha flere modeller åpne for å kunne jobbe på disse samtidig.

Verktøyet skal kunne kjøres på Linux og Windows XP. Det skal også være mulig å porte verktøyet til andre operativsystemer på en enkel måte.

2.2 Bruker beskrivelse

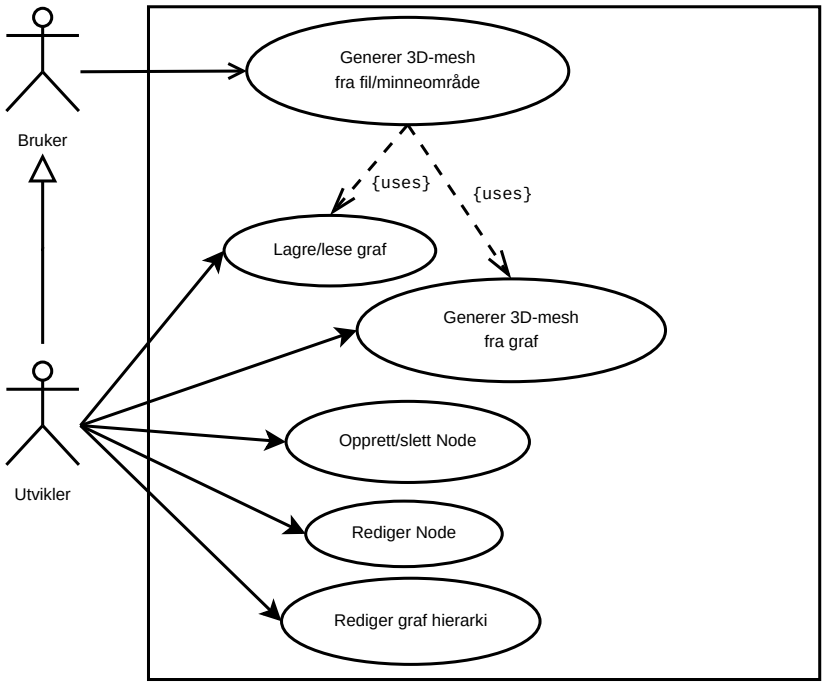
2.2.1 Systemets brukere

Modellerere
Modellerere har ofte bakgrunn fra verktøy som Lightwave, 3D Studio eller Maya.

Utviklere
Utviklere er vant til å lese API dokumentasjon for å kunne bruke biblioteker. Derfor er de vant til teknisk jargong som vil forkomme i våre APIer.

2.2.2 Funksjon

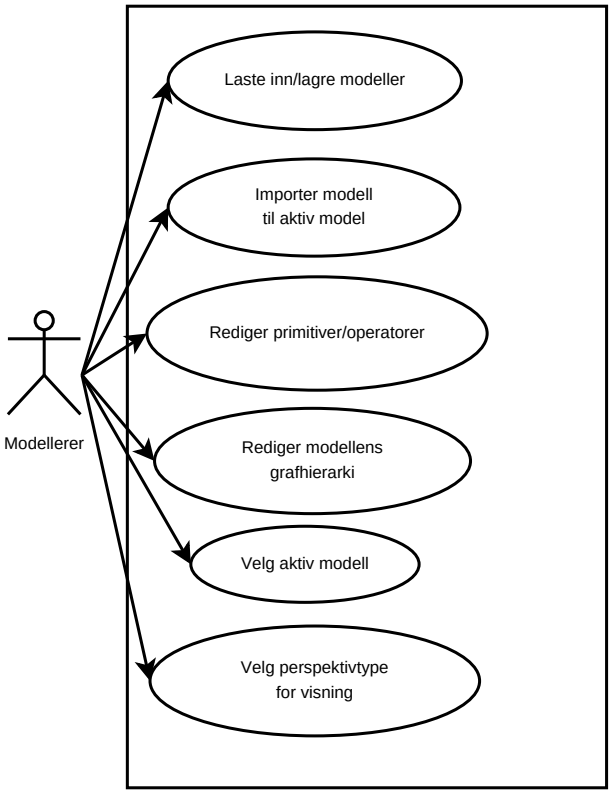
CMTF3D biblioteket



Figur 2: Use Case for bibliotek

Figur 2 illustrerer de to forskjellige måtene biblioteket kan brukes på. En *Bruker* kan be biblioteket om å laste inn og generere en 3D-mesh, mens *Utvikler* har full tilgang til hele bibliotekets funksjonalitet(API). Dette for å kunne bruke selve APIen til å sette sammen og generere modeller(3D-mesh).

Modelleringsverktøy



Figur 3: Use Case for modelleringsverktøyets funksjonalitet

Figur 3 viser i hovedtrekk de operasjoner brukeren av modelleringsverktøyet kan utføre. I korte trekk kan brukeren opprette modeller, noder og redigere disse, samt lagre og laste inn fra fil.

2.2.3 Ytelse

Biblioteket skal kunne generere 3D-mesh på optimal tid. Det må også ta minst mulig plass for å kunne bli brukt på systemer/hardware med lite resurser.

Modelleringsverktøyet trenger en relativt rask CPU(1.6 Ghz eller raskere) for å kunne gi en optimal brukeropplevelse, samt et 3D-kort med støtte for OpenGL.

2.2.4 Begrensninger

CMTF3D biblioteket
Biblioteket skal i prinsipp kunne kompileres og brukes på alle platformer, men vi vil begrense oss til kun å teste biblioteket på Linux, Windows XP og Mac OS X.

Modelleringsverktøy
Operativsystemer som verktøyet skal fungere på er Linux og Windows XP.

2.3 Detaljert kravspesifikasjon

2.3.1 Funksjonell spesifikasjon, struktur og tverr-relasjoner

Biblioteket
En bruker skal kunne:

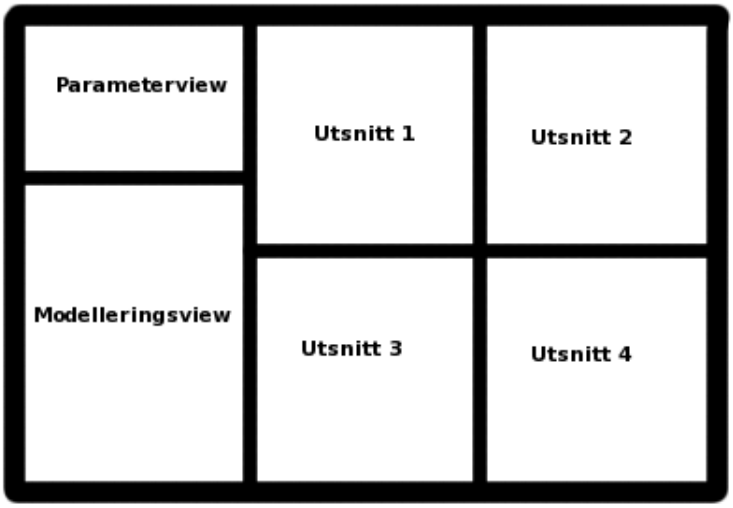
- Lese en graf fra en fil/minneområde og få opprettet datastrukturen for denne.
- Genere en 3D-mesh ut ifra en opprettet graf.
- Forandre grafhierarkiet ved å opprette/slette noder, redigere disse eller forandre relasjonene mellom dem.

Modelleringsverktøyet
En bruker skal kunne:

- Opprette en ny modell.
- Laste inn/lagre modeller.
- Importere modeller inn til aktiv modell.
- Se 3D-modellen fra forskjellige synsvinkler.
- Redigere 3D-modellen ved å endre graf representasjonen eller ved redigering av node paramtre.
- Velge aktiv modell, blant flere åpne modeller.

Brukergrensesnitt
Brukergrensesnittet skal ha følgende hovedkompontener (se Figur 4):

- 4 utsnitt av 3D-modellen.
- Grafrepresentasjonen av 3D-modellen.
- Parameterrepresentasjonen av aktiv node i grafrepresentasjonen.



Figur 4: Mockup for modelleringsverktøy

2.3.2 Funksjonell spesifikasjon

Biblioteket
Biblioteket skal være plattform og 3D API uavhengig, og må kunne generere 3D modeller som mesher. Disse meshene må ha en struktur som lett kan integreres i spill og andre typer visualiseringsapplikasjoner.

Genereringen skal skje ved at biblioteket leser inn filer som inneholder beskrivelsen av de operasjoner som ble utført under modelleringsfasen i verktøyet. Selve genereringen av modellene bør ikke ta for lang tid, så valg av algoritmer og datastrukturer i biblioteket er kritisk.

Biblioteket må støtte følgende:
Primitiver:

- Cube.
- Cylinder.
- Super Ellipsoid.
- Torus.
- Super Torus.

Operasjoner:

- Translation.
- Rotation.
- Scale.
- Skew.
- Bend.
- Twist.
- Taper.
- CSG.
- Array.
- Subdivide.
- Surface.

Modelleringsverktøyet
Modelleringsverktøyet(proof-of-concept) skal sette 3D artister i stand til å lage komplekse 3D modeller ved bruk av enkle 3D-primitiver og operasjoner på disse. For å bruke minst mulig diskplass skal kun de operasjonene som ble utført for å lage modellen skrives til fil.

Verktøyet må kunne kjøres på Windows XP og Unix plattformer generellt.

Modelleringsverktøyet skal ha et så enkelt og konsist grafisk brukergrensesnitt som mulig (se Figur 4). Det er ønskelig å innenfor en gitt tidsramme å utvikle et eget, evt bruke et kryssplatform GUI bibliotek.

Det skal ikke brukes ikoner i GUI, det skal istedet brukes knapper med forklarende tekst.

GUI skal gi brukeren mulighet til å kunne jobbe med modellrepresentasjonen på en best mulig måte. Det skal også være enkelt å redigere modelhierarkiet og opsjoner for valgte elementer.

Modelleringsviewet er et panel hvor alle byggestener er visualisert(som knapper med tekst), og det er her brukeren lenker sammen byggestener etc. Når brukeren velger en byggesten, skal modelleringsview kun tegne opp modelleringsprosessen frem til og med den valgte byggestenen. Når den er valgt vil alle opsjoner vises i elementopsjoner panelet, og disse kan så redigeres enten i modelleringsviewene, eller direkte i dette panelet.

UI
Tanken bak verktøyets UI er at brukeren ikke skal heftes unødvendig med inntasting av tastekombinasjoner, men istedet kunne bruke høyre hånd til modellering med mus, og venstre hånd til hotkeys på tastatur. Dette skal gå igjen i alle deler av verktøyet.

2.3.3 Data spesifikasjon og dataordliste

Data input/output
Biblioteket skal benytte sitt eget filformat for lagring og lesing av data. Dette filformatet er egendefinert (se kapittel 4.6).

2.3.4 Overordnede operasjonelle systemkrav

Modelleringsvertøy
Modelleringsverktøyet har kun ett *modus operandi* for redigering av modeller. Brukeren skal i dette modi kunne opprette nye modeller, redigere allerede eksisterende modeller, samt sette sammen en model ved å importere eksisterende modeller for så å lenke disse sammen.

2.4 Begrensninger

2.4.1 Software design begrensninger

Generelt:

- Animasjon skal ikke støttes.
- Det skal ikke være mulig å legge teksturer på 3D-modeller.
- OpenGL: for platformuavhengig visualisering i modelleringsverktøyet.

Operativsystemer:
Modelleringsapplikasjonen skal utvikles for UNIX og Windows. Den plattform spesifikke koden skal være skilt ut slik at det er enkelt å utvikle for nye operativsystemer.

2.5 Aspekter omkring livssyklus

2.5.1 Dokumentasjon

Både CMTF3D bibliotekets API og modelleringsverktøyets dokumentasjon skal leveres i HTML. Dette for å kunne være tilgjengelig på alle platformene på en enklest mulig måte.

2.5.2 Modul- og integrasjonstesting

Etter at hvert inkrement er ferdig skal den nye modulen testes for å sikre at den oppfyller ønskede krav og i tillegg utfører ønskede oppgaver på en tilfredstillende måte. I tillegg skal modulen testes i sammenheng med den totale løsningen, for å sikre at modulen ikke gir uønskede bieffekter i dette samspillet.

2.5.3 Krav til utvidelser

Biblioteket skal kunne utvides med nye primitiver og lignende.
Modelleringsverktøyet skal kunne utvides med GUI for disse nye primitiver.

Modelleringsverktøyet må kunne portes til andre operativsystemer på en enkel måte, og all funksjonalitet i programmet skal være godt dokumentert.

2.6 Utgivelser underveis

Det skal utgis en ny versjon etter hver milepæl. Versjonene skal bli gjort tilgjengelige for oppdragsgiver og veileder, slik at disse kan teste applikasjonen og biblioteket.

3 Design

3.1 Innledende Design

Løsningen vil bestå av ett bibliotek og en applikasjon. Selve biblioteket anses som en modul og vil deles inn i to undermoduler (CMTF3D og CSG) hvor CMTF3D tar seg av mesh generering og I/O, mens CSG skal tar for seg boolske operasjoner på mesher. Når det gjelder applikasjonen vil den deles opp i tre moduler(GLF, GUI og MOD). Hvor GLF modulen skal ta seg av grunnleggende funksjonalitet(oppretting av vindu m.m.) på en operativsystemuavhengig måte. GUI modulen skal ta seg av det grafiske brukergrensesnittet, og MOD modulen har som oppgave å ta seg av modellering på en grafisk måte.

3.2 Bibliotek

3.2.1 CMTF3D

Biblioteket er en triangel mesh generator med to grensesnitt. Ett grensesnitt for enkel innlasting av mesh fra fil og ett grensesnitt for avansert bruk. Det avanserte grensesnittet vil bli brukt av modelleringsverktøyet, og man vil ha tilgang til all funksjonalitet ved hjelp av dette grensesnittet.

3D modeller vil bli representert som grafer(DAG), og ved å prosessere hver node i grafen vil en triangel mesh gradvis genereres.

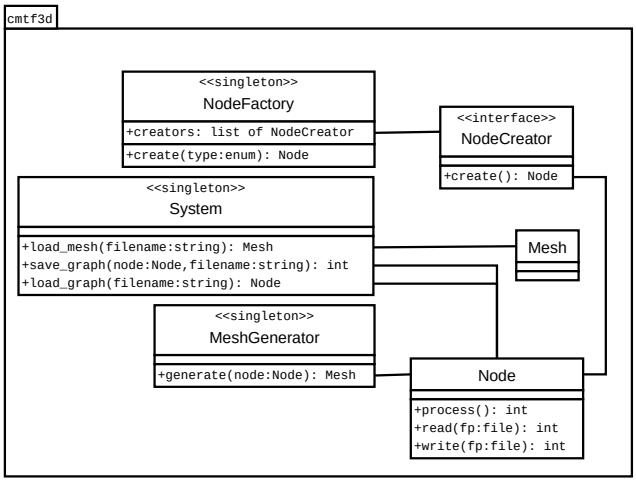
Noder i grafen kan ha null, en eller to input noder, og det er noen enkle regler for hvordan de forskjellige typene noder opererer.

Bladnoder(noder uten input) må generere en mesh slik at foreldrene til noden har noe å jobbe med. Vi kan si at bladnoder lager mesher som de interne nodene opererer på.

Noder med en input kan i hovedsak ta mesh som barnet genererte og gjøre operasjoner på denne.

Noder med to input gjør i hovedsak operasjoner hvor input fra barn nr 2 brukes til å gjøre operasjoner på input fra barn 1. Resultatet fra dette er således nodens output.

Nodetyper vil bli registrert i en database slik at vi kan bruke en creator/-factory pattern som beskrevet i 'Applying UML and Pattern´ [1] av C. Larman. Dette for å ungå at kjernen i biblioteket skal trenge å vite noe om nodetyper og hva de skal utføre.



Figur 5: Overordnet klassediagram for bibliotek

3.2.2 CSG

CSG modulen tar seg av boolske operasjoner på mesher og skal kun ha en kallbar funksjon som tar seg av oppsett og utføring av selve CSG operasjonen. Denne funksjonen kan være på formen:

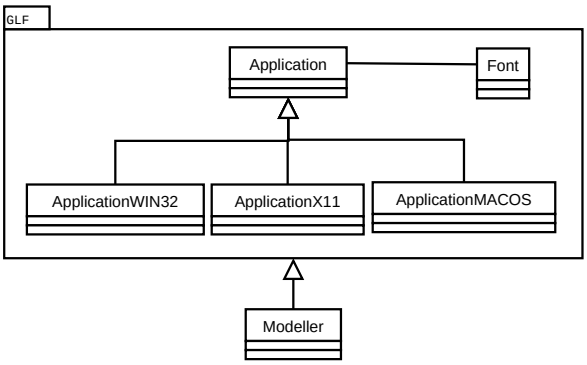
```
enum Operation {
        UNION,
        DIFFERENCE,
        INTERSECTION
};
int perform_csg_operation(
                Operation op,
                Mesh A,
                Mesh B,
                Mesh output);
```

Dette for å skille selve implementasjonen fra API designet slik at modulen kan omskrives ved endrede behov. F.eks. hvis en implementasjon ikke skalerer så bra man hadde håpet.

Modulen skal bruke datastrukturer og funksjonaltet i CMTF3D så langt det er mulig.

3.3 GLF

GLF er et rammeverk for applikasjoner som benytter seg av OpenGL. Det tar seg av den plattformspesifike koden, som å få satt opp et vindu med OpenGL kontekst, innhenting av mus og tastetrykk samt behandle hendelser fra operativsystemet.



Figur 6: Overordnet klassediagram for GLF.

Rammeverket skal være plattformuavhengig. Vi har derfor valgt å skille ut dette i en egen modul. Denne bør være så gjenbrukbar som mulig slik at det kan brukes til andre typer prosjekter.

Dette blir på følgende måte:

- Vi definerer en baseklasse for applikasjoner. Her finnes funksjoner for å ta seg av de forskjellige hendelser som sendes fra Operativsystemet. Disse må gjøres om til et fornuftig format, og videresendes til applikasjonen.
- Baseklassen har også virtuelle funksjoner som må implementeres av operativsystemspesifike applikasjonsklasser. Det er disse som tar seg av oppsett av vindu, innhenting av hendelser og behandling av disse.
- Oppretting av bitmap font.

3.4 GUI

Siden modelleringsverktøyet skal være plattformuavhengig vil det ikke være fornuftig å benytte seg av OS spesifikke bibliotek for GUI. Siden dette kompliserte oppgaven vår skisserte vi grensesnitt for modelleringsverktøyet for å finne ut hvilke GUI elementer vi virkelig hadde behov for. Deretter kartla vi hva som fantes av plattformuavhengige GUI bibliotek. Det ble evaluert flere GUI biblioteker med OpenGL støtte(FLTK, wxWidgets, libUFO), men fant at disse bibliotekene var for omfattende i forhold til de GUI elementer modelleringsverktøyet trengte. Valget falt derfor på å implementere et eget GUI Toolkit.

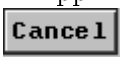
I første omgang fant vi at vi trengte: knapper, tekstfelt, menyer, paneler og scrollbar.

Designet til GUI Toolkit vil være tuftet på Java SWING(som igjen har aner tilbake til Smalltalk), men vårt Toolkit er ikke like omfattende og funksjonsrikt. Det er skreddersydd for å lage vår applikasjon.

3.4.1 Grafiske elementer

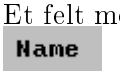
Biblioteket støtter følgende grafiske elementer.

Knapper
Knapper med tekst som man kan trykke på for at noe skal skje.

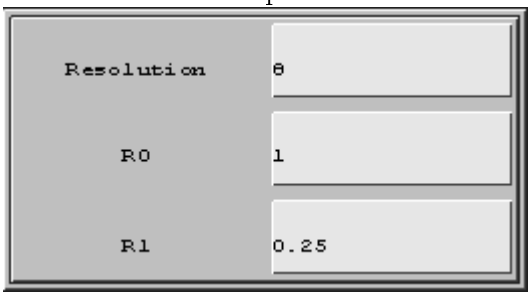


Tekstfelt
Et felt der man kan skrive inn tekst.

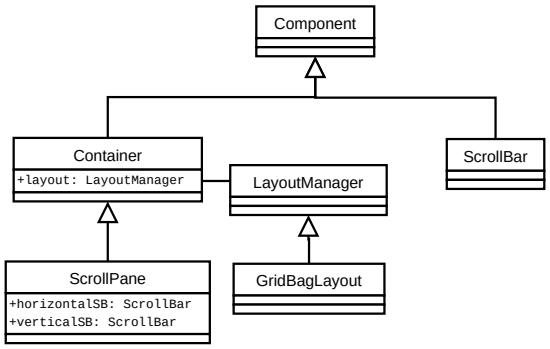
Label
Et felt med en beskrivende tekst.



Panel
Et element der man plasserer andre GUI elementer.



Meny
En menylinje øverst på skjermen.



Figur 7: Utdrag fra overordnet klassediagram for GUI.

ScrollPane
Et element der man plasserer andre gui elementer, som panel, men dette inneholder også muligheter for å scrolle vertikalt og horisontalt.

3.4.2 Oppbygging av klassehierarki

Baseklassen i hierarkiet er `Component`(se figur 7). I denne klassen spesifiseres alt om komponenters plassering, størrelse og grunnleggende funksjoner. For å lage et element trenger man bare å arve fra `Component` og legge til den ekstra funksjonaliteten man trenger.

En mer avansert type komponent er `Container`, som kan inneholde andre komponenter. `Container` kan også inneholde en `LayoutManager`.

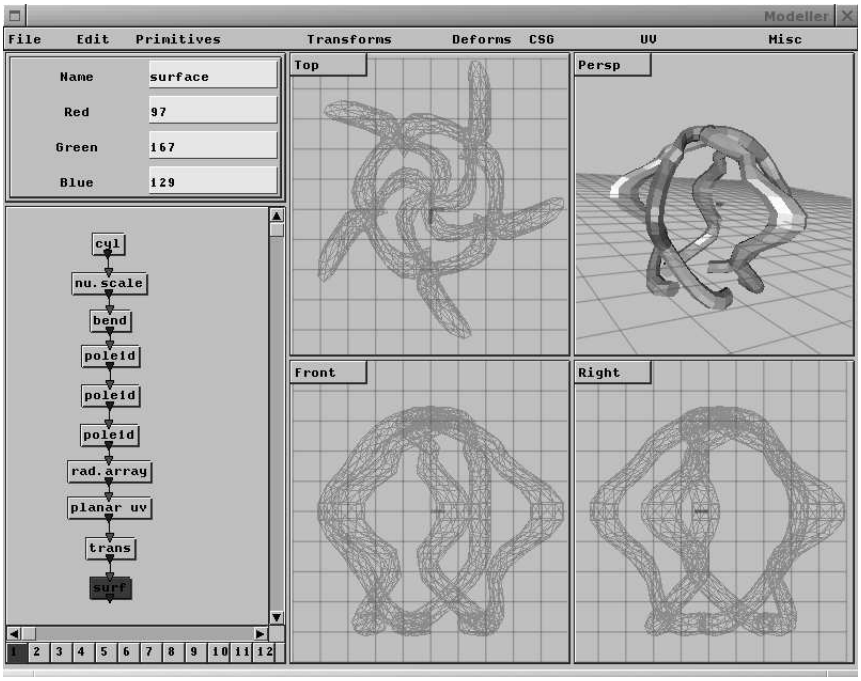
`LayoutManager` er baseklasse for posisjonering og endring av størrelse på komponenter innenfor en containers område.

Vi har valgt å plassere alle tegnefunksjoner i singleton-klassen `Painter` siden mange komponenttyper benytter seg av de samme tegneoperasjonene.

3.5 Modelleringsverktøy

Dette er visualiseringsverktøyet hvor brukeren drar nytte av CMTF3D-biblioteket for å generere, redigere og se på 3D-modeller. Brukere med erfaring fra andre

modelleringsapplikasjoner skal raskt kunne sette seg inn i verktøyet.



Figur 8: De forskjellige views i modelleringsapplikasjonen.

3.5.1 Meny

Menylinjen skal inneholde hovedmenyene File, Edit, Primitives, Transforms, Deforms, CSG , UV og Misc. Disse menyene brukes for å last inn, lagre, opprette og slette noder, samt sentrering av valgt node i `DAGView`.

3.5.2 DAGView

`DAGView` skal vise 3D-modellen representert som en graf, og man skal kunne koble sammen noder ved å lenke dem sammen. Det skal også være mulig å flytte på noder/subtrær og velge node slik at modellen blir vist i `OrthoView` og `PerspView` samt at nodens parametre blir vist i `ParamView`. Det skal være 12 `DAGView` å velge mellom slik at brukeren kan jobbe på flere 3D-modeller samtidig.

3.5.3 ParamView

Viser parametrene til noden som er valgt i `DAGView`, og man kan redigere disse.

3.5.4 OrthoViews

Viser 3D-modellen i ett ortografisk perspektiv(foran, bak, over, under, høyre og venstre). Det kan velges forskjellige opptegningsmodus for 3D-modellen(solid, linjer og punkter) som kan kombineres med teksturvisning.

For noden valgt i `DAGView` kan man i `OrthoView` bruke mus for å justere parametrene til denne.

3.5.5 PerspView

Her viser objektet perspektivisk, og modellen kan roteres og zoomes. Det er også mulig å velge mellom de samme opptegningsmodus som i `OrthoView`.

3.6 Filstruktur

Modelleringsapplikasjonen og biblioteket skal benytte samme filformat for lagring av 3D-modeller. Dette filformatet skal være leselig på flere hardware-platformer og det skal være mulig å gjøre enkel feilsjekking under innlesing av data.

Selve filformatet skal være binært og det skal ha header-, footer- og innholdsblokker.

Headerblokken skal inneholde informasjon on filformatets versjon, footer-blokken brukes til å signalisere slutten på filen, og innholdsblokkene består av informasjon om noder og hvordan disse er lenket sammen til en graf.

4 Implementering

4.1 Verktøy

4.1.1 Valg av programmeringspråk

Kodingsomfanget i prosjektet ble vurdert til en slik størrelse at valg av programmeringsspråk måtte falle på et språk som alle gruppemedlemmene hadde erfaring med fra før(C, C++ og Java).

Siden løsningen vil gå ut på å behandle store mengder data matematisk anså vi hastighet som en kritisk faktor ved valg av språk. Etter å ha sett på en sammenligning mellom språkene utført av Ulrich Stern [2], ble Java forkastet.

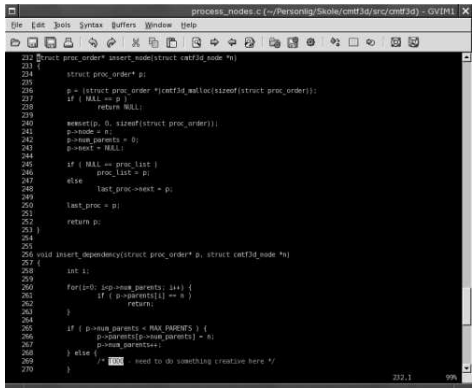
I biblioteket har vi lagt vekt på hastigheten og lenking opp imot andre språk, derfor ble C valgt fremfor C++. Applikasjonen derimot skal kodes i C++ slik at vi kan dra nytte av OO teknikker for å strukturere og innkapsle koden best mulig.

4.1.2 Valg av Kompilatorer

MS Visual Studio 2003 ble valgt for å kompilere under Windows, og GCC 3.3.x under Linux. For å effektivisere build prosessen under *NIX har det blitt brukt et makefile system.

4.1.3 Valg av editor

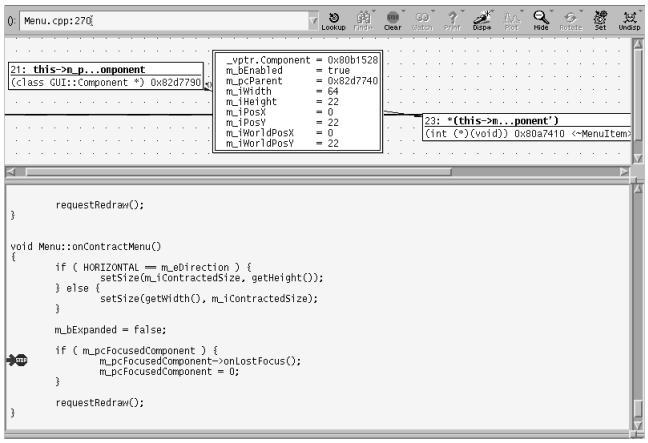
Vi standardiserte på VIM som editor for kildekode under *NIX. Denne har syntax highlighting, automatisk indentering av blokker og folding av funksjoner. Under Windows valgte vi MS Visual Studio siden dette er integrert sammen med kompilatoren.



Figur 9: Tekst editoren VIM

4.1.4 Debuggere og minnehåndteringsverktøy

Debugging i Windows ble gjort i MS Visual Studio. I Linux har GNU DDD blitt brukt, denne er en grafisk front-end for kommandolinje debuggeren GDB. For debugging av minnehåndteringsfeil under Linux har Valgrind blitt brukt.



Figur 10: Debuggeren DDD

4.1.5 Verktøy for dokumentasjonsgenerering

For automatisk generering av dokumentasjon ut fra kode valgte vi Doxygen. Dette er et dokumentasjonssystem for C++, C, Java og andre språk.

Doxygen ble valgt fordi:

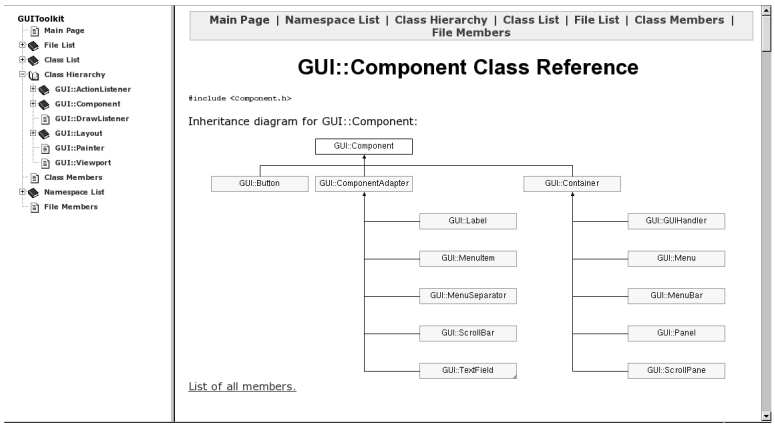
- Den genererer dokumentasjon i mange formater, blant annet HTML.
- Hendig oversikt over klasser og deres medlemmer.

4.1.6 Systemutvikling og planlegging

Dia ble benyttet til design av klassediagrammer for de forskjellige delene av løsningen, mens Gantt-Project ble brukt til prosjektstyring.

4.2 Inndeling i inkrementer

Arbeidsmengden ble delt inn i 4 inkrementer som er beskrevet nedenfor.



Figur 11: Dokumentasjon generert med Doxygen

4.2.1 GLF inkrementet

I GLF så ble det arbeidet med å abstrahere bort den plattformavhengige delen av applikasjonen.

- **UNIX** - Utvikle plattformspesifikt rammeverket for UNIX.
- **WIN32** - Utvikle plattformspesifikt rammeverket for WIN32.

4.2.2 GUI inkrementet

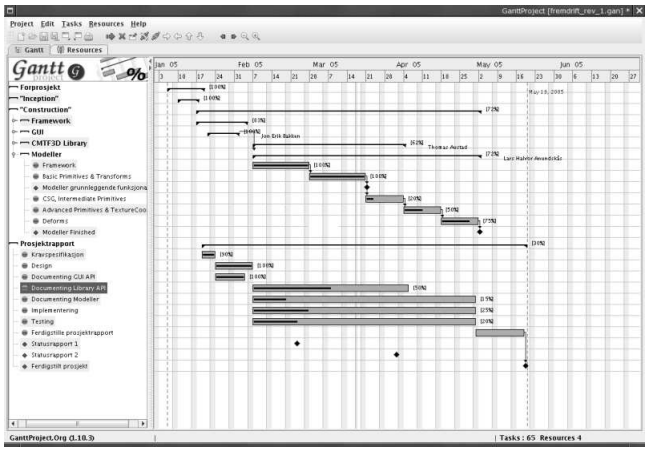
Her ble rammeverket for applikasjonens grafiske brukergrensesnitt utviklet.

- **Design** - Her fastlegges software arkitektur.
- **Implementasjon** - Koding av modulen og dokumentasjon av APIet.
- **Testing** - Uttesting av funksjonaliteten.

4.2.3 Bibliotek inkrementet

Utvikle mesh generatoren som er selve kjernen i prosjektet.

- **Basis** - Design og implementering av kjernen.
- **Basic primtitives** - Implementere grunnleggende 3D-primitiver.
- **Transformasjoner** - Implementere transformasjoner.
- **Tekstur koordinater** - Implementere teksturkoordinatgenerering.
- **Deformasjoner** - Implementere deformasjoner.



Figur 12: Gantt-Project

- **CSG** - Implementere CSG operasjoner.
- **Intermediate primitives** - Implementere intermediate 3D-primitiver.
- **Advanced primitives** - Implementere avanserte 3D-primitiver.

4.2.4 Modelleringsverktøy inkrementet

Her ble modelleringsverktøyet implementert.

- **Rammeverk** - Design og implementering av verktøyets rammeverk.
- **Grunnleggende primitiver og transformasjoner** - Implementere støtte for bibliotekets primitiver og transformasjoner
- **CSG og intermediate primitiver** - Implementere støtte for bibliotekets CSG operasjoner og intermediate primitiver
- **Avanserte primitiver og tekstur koordinat generering** - Implementere støtte for bibliotekets avanserte primitiver og tekstur koordinat generering
- **Deformasjoner** - Implementere støtte for bibliotekets deformasjoner

```
#if defined(win32)
#include "ApplicationWIN32.h"
#define DECLARE_APPLICATION public GLF::ApplicationWIN32
#define APPLICATION_CTOR GLF::ApplicationWIN32
#define INSTANTIATE_APPLICATION(AppCls)                    \
int WINAPI WinMain(HINSTANCE hInstance,                    \
                   HINSTANCE hPreInstance,                 \
                   LPSTR lpCmdLine,                        \
                   int nCmdShow)                           \
{                                                          \
        AppCls* pcApp = new AppCls;                        \
        pcApp->run();                                      \
        delete pcApp;                                      \
        return 0;                                          \
}
#elif defined(linux)
...
#else
#error Unknown platform
#endif
```

Figur 13: Bruk av makro for å gjemme bort platformspesifikk baseklasse.

4.3 Applikasjonsrammeverk

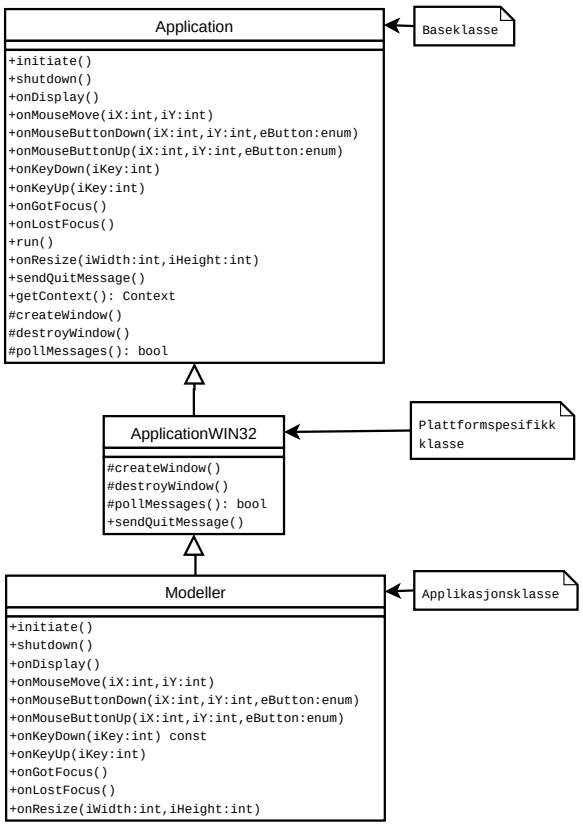
Vi har laget en baseklasse `Application` som inneholder all plattformuavhengig kode. Denne klassen har definert virtuelle funksjoner for oppretting av vinduer, håndtering av input og hendelser. Disse implementeres av de plattformspesifikke applikasjonsklassene.

Baseklassen inneholder virtuelle callback-funksjoner for forskjellige typer hendelser. Disse blir kalt hver gang operativsystemet sender en beskjed til applikasjonen.

For at applikasjonsklassen skal avledes fra riktig plattformspesifikk baseklasse så brukes det litt makromagi(se figur 13) for å gjemme bort valget mellom de OS-spesifikke klassene. Når man så koder en applikasjon med rammeverket brukes dette på følgende måte:

```
#include "Framework.h"
class MyApplication : DECLARE_APPLICATION {
public:
        MyApplication() :
                APPLICATION_CTOR("applikasjon", 800, 600, false)
        {
                ...
        }
};
INSTANTIATE_APPLICATION(MyApplication);
```

Ved oppstart av en applikasjon vil `Application` sin `run` funksjon bli kjørt. Denne oppretter et vindu, sender beskjed om at applikasjonen skal initialisere seg selv. Så kjører den en beskjedsløkke der det tas imot beskjeder fra operativsystemet. Disse beskjedene konverteres og videresendes til callback funksjonene som applikasjonen har implementert. Når beskjedsløkken termineres vil applikasjonen få beskjed om å avslutte.. 4



Figur 14: Overordnet klassediagram for applikasjonsrammeverket

4.4 3D Bibliotek

Siden biblioteket skal ta seg av alt som har med oppretting av mesher, behanding av disse samt IO til fil, ble dette delt inn i to grensesnitt, ett for enkel innlastning av mesh og ett for avansert bruk. Vi fant det hensiktsmessig å implementere det enkle grensesnittet som en enkeltstående funksjon(se figur 15). Denne tar seg av innlasting av en mesh-beskrivelse enten fra fil eller et minneområde. Det avanserte grensesnittet derimot gir tilgang til alle bibliotekets funksjoner og datastrukturer. Dette omfatter minnehåndtering, matematikk, mesh-behandling, input/output og grafbehandling.

I kapitlene under blir hovedelementene til biblioteket gjennomgått.

4.4.1 Oppstart og avslutning

Vi valgte å ha funksjoner for oppstart og avsluttning av biblioteket. Dette ble gjort for å forenkle registrering av nodetyper i nodefabrikken(beskrevet i underkapittel 4.4.6).

```
struct cmtf3d_mesh* cmtf3d_load_mesh(
        uint8   mem,
        void    *file,
        uint32  offset,
        uint32  length,
        uint8   flags);
```

Figur 15: Grensensitt for innlasting av mesh.

Før biblioteket kan brukes må `cmtf3d_initiate()` funksjonen kalles. Denne tar seg av initialisering av nodefabrikken.

Etter at man er ferdig med å bruke biblioteket må `cmtf3d_shutdown()` funksjonen kalles. Dette for å deallokere minne som bibliotekets nodefabrikk og andre subsystemer har brukt. For å forsikre seg om at denne funksjonen blir kjørt kan man f.eks. registrere den med `atexit()`[4] slik at den blir kjørt ved programterminering.

4.4.2 Minnehåndtering

For å holde ett godt øye med minne håndtering(allokering og deallokering) har vi valgt å lage våre egne funksjoner for dette. `cmtf3d_malloc()` tar seg av allokering og `cmtf3d_free()` frigjøring av minne. Foreløpig er disse wrappere rundt ANSI-C funksjonene `malloc()` og `free()`, og holder rede på hvor mye minne som til en hver tid er allokert, hvilke fil og funksjon som har allokert minne, samt nummer på kildekodelinjen som allokerte minne. Det er også implementert en funksjon som går gjennom minneallokeringslisten når `cmtf3d_shutdown()` kalles. Denne funksjonen frigir minne som enda ikke er deallokert, og skriver minnehåndteringsfeilbeskjeder til `stderr`[5] for hver minneblokk som ikke er frigitt.

Forbedring av minnehåndteringen kan implementeres f.eks. ved pooling av minneområder. Dette er noe vi ikke fikk tid til å implementere, men siden hele biblioteket bruker våre minnehåndteringsfunksjoner skulle det ikke by på store problemer å implementere en pooling algoritme.

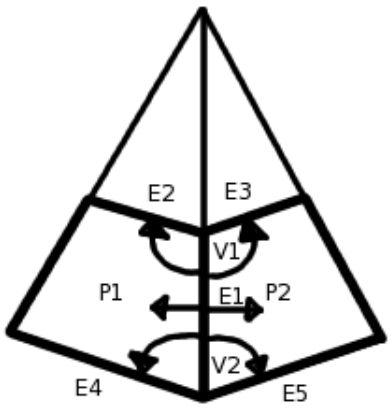
4.4.3 Mesher

Mesher er representert ved winged-edge mesh representasjonen, og implementetasjonen av denne er basert på beskrivelsen i kapittel 12.5.2 'Compu-

[4] atexit() er en POSIX C funksjon som brukes til å registrere en funksjon som kalles ved programterminering.

[5] stderr - brukes til å skrive feilbeskjeder til terminal/konsoll.

ter Graphics 2nd ed´ [3] med støtte i orginal artikkelen til Bruce G. Baumgart [4]. Dette er en data struktur som egner seg godt til behandling av mesher fordi den eksplisit innholder informasjon om alle tilknyttninger mellom punkter, kanter og polygoner(Se figur 16). Dette gjør det for eksempel enkelt å finne hvilke polygoner som er tilknyttet en kant, noe som eksempelvis kan brukes for å sjekke om en mesh er solid(d.v.s. alle kanter må ha tilknyttning til to polygoner).



Figur 16: Winged-edge data struktur for kanten `E1`. Punktenen `V1`, `V2` og polygonene `P1`, `P2` har også pekere til tilbake til kanten(disse er ikke vist).

Algoritme for winged-edge mesh

Hvis winged-edge strukturen skal kunne brukes må informasjon om kantene i en mesh være tilgjengelig. Denne informasjonen kan enten opprettes sammen med en mesh eller den kan legges til ved behov. Vi fant det hensiktsmessig å kunne legge til dette ved behov for å spare minne. Dette gjorde så vi trengte en algoritme som fant alle kanter og la til relevante data i mesh strukturen. Algoritmen vi kom opp med bygger rundt ideen om å gå gjennom polygonene en etter en for å opprette kanter. Kildekoden for algoritmen er presentert på neste side.

```
void cmtf3d_build_edges_mesh(
                struct cmtf3d_mesh *m)
{
        struct cmtf3d_face   *f;
        struct cmtf3d_vertex *v;
        struct cmtf3d_edge    e;
        int i, j, k, eid;

        /* bailout conditions */
        if ( NULL == m || NULL == m->faces || NULL == m->vertices )
                return;

        if ( NULL == m->edges )
                m->edges = cmtf3d_create_edge_list(2);
        else
                m->edges->last_edge = 0;

        /* set vertex edge counters to zero */
        for(i=0; i<m->vertices->last_vertex; i++) {
                m->vertices->vertices[i].num_edges = 0;
        }

        /* add edges for all faces */
        f = m->faces->faces;
        for(i=0; i<m->faces->last_face; i++, f++) {
                /* loop around the edges of the face */
                for(j=0; j<3; j++) {
                        /* check if edge have been added previously */
                        eid = cmtf3d_find_edge_in_list(m->edges,
                                                       f->vertices[j],
                                                       f->vertices[(j+1)%3]);
                        if ( (uint32)-1 == eid ) {
                                /* create new edge */
                                e.vertices[0] = f->vertices[j];
                                e.vertices[1] = f->vertices[(j+1)%3];
                                e.faces[0] = i;
                                e.faces[1] = (uint32)-1;
                                cmtf3d_add_edge_to_list(m->edges, &e);
                                /* set face edge-index */
                                f->edges[j] = m->edges->last_edge - 1;
                        } else {
                                if ( i == m->edges->edges[eid].faces[0] ||
                                     i == m->edges->edges[eid].faces[1] )
                                {
                                        /* this branch should never execute */
#ifdef _DEBUG
                                        printf("EDGE_HAS_ALREADY_BEEN_ADDED\n");
#endif
                                } else {
                                        /* update face info */
                                        m->edges->edges[eid].faces[1] = i;
                                        f->edges[j] = eid;
                                }
                        }
                }
        }

        /* update vertex/edge sharing info in vertex struct */
        for(i=0; i<m->edges->last_edge; i++) {
                for(j=0; j<2; j++) {
                        if ( (uint32)-1 == m->edges->edges[i].vertices[j] ) {
                                /* this is not a true edge i.e. it's only linked
                                 * to one vertex and hopefully not linked to any
                                 * faces :) */
                                continue;
                        }
                        v = &m->vertices->vertices[m->edges->edges[i].vertices[j]];
                        /* check if edge 'i' is already in the vertex edge list */
                        for(k=0; k<v->num_edges && v->edges[k] != i; k++) ;
                        /* add edge to vertex if it wasn't found */
                        if ( k >= v->num_edges ) {
                                assert(v->num_edges < CMTF3D_MAX_VERTEX_EDGES);
                                v->edges[v->num_edges++] = i;
                        }
                }
        }
}
```

Polygoner
Vi valgte å bruke triangler og ikke andre typer polygoner i mesh strukturen. Dette fordi polygoner generelt ikke behøver å være planare, noe som kan gjøre en del mesh operasjoner noe vanskeligere å implementere siden man hele tiden må sjekke om polygoner er planare.

Det er selvsagt mulig å bruke en generell polygonrepresentasjon hvor man internt bruker triangler(med planepsilon) for å representere polygonet, men vi fant at dette bare førte til flere kodelinjer, og ikke til en enklere polygonbehandling. Det må også sier at trianglet er det eneste polygonet som garantert er planart.

4.4.4 Modellgraf

Grafen som beskriver 3D-modeller er en DAG og består av noder(se figur 17) som utfører oppgaver på input mesher og genererer output i form av nye mesher. Se figur 18 for en oversikt over hvordan en graf for en enkel 3D-modell kan se ut.

Noder har kun kunnskap om sin relasjon til sine barn, og om formatet på mesher. M.a.o. så vet ikke noder noe om hvordan mesh-genereringen foregår. Dette for å forenkle koding av node typer, og for å forenkle mesh-genereringen som sådan.

Noder har tre funksjoner `process()`, `read()` og `write()`. Hvor `process` tar seg av prosessering av noden basert på dens input, mens `read` og `write` tar seg av fil I/O.

4.4.5 Mesh-generator

Mesh-generatoren er selve kjernen i biblioteket, og har som hovedoppgave å traversere en DAG for å generere en triangel-mesh representasjon av 3D-modellen DAGen representerer. Dette gjøres ved følgende algoritme:

Først utføres en post order traversering av DAGen for å lagre alle noder i en lenket liste kalt prosesseringslisten(se figur 19). Før en node traverseres sjekkes det om noden allerede er oppført i prosesseringslisten, hvis den ikke er det så traverseres barnene til noden. Etter at barnene til noden er traversert blir pekeren til noden lagt til i barnenes oppføring(i en liste over foreldre). Vi itererer så gjennom prosesseringslisten (head to tail) og for hver oppføring kjøres oppføringens node sin `process()` funksjon. Deretter dupliseres nodens output-mesh slik at første node i foreldrelisten får orginal meshen, og de etterfølgende foreldre får kopier av denne. Når alle oppføringer i prosesseringslisten er prosessert vil output-mesh til den siste

```
/** Maximum number of input nodes */
#define CMTF3D_MAX_INPUTS    2

/**
 * This structure represents a node in a graph(DAG).
 *
 * When coding a node type there are a few important things:
 * - Primitive nodes i.e. leaf nodes must create meshes.
 * - Nodes with only one input that only modifies existing mesh data can safely
 *   pass a reference counted mesh as it's output. To do this use
 *   cmtf3d_clone_mesh() with the CMTF3D_CLONE_NONE option.
 * - Nodes with more than one input that only modifies existing mesh data or add
 *   to it can pass a reference counted mesh as it's output. Which input mesh it
 *   passes is left to the node developer to judge.
 **/
struct cmtf3d_node {
    /** node types are defined in node_type.h */
    uint8                type;
    /** used internally by the library */
    uint8                flags;
    /** number of input nodes */
    uint8                num_inputs;
    /** the children of this node */
    struct cmtf3d_node *input[CMTF3D_MAX_INPUTS];
    /** these are set by the library and points to the
     * meshes created by the input nodes */
    struct cmtf3d_mesh *input_mesh[CMTF3D_MAX_INPUTS];
    /** store new mesh here or a ref.counted pointer to input_mesh[?] */
    struct cmtf3d_mesh *output;
    /** pointer to node specific data */
    void               *data;
    /**
     * Process node
     * @param n pointer to node
     * @return zero on success
     */
    int (*process)(struct cmtf3d_node *n);
    /**
     * Read node
     * @param fp pointer to a cmtf3d_file struct
     * @param n pointer to node
     * @return zero on success.
     */
    uint8 (*read)(void *fp, struct cmtf3d_node *n);
    /**
     * Wwrite node
     * @param fp pointer to a cmtf3d_file struct
     * @param n pointer to node
     * @return zero on success.
    */
    uint8 (*write)(void *fp, struct cmtf3d_node *n);
};
```

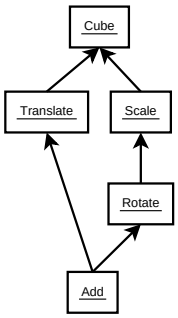
Figur 17: Bibliotekets nodestruktur.

oppføringen være meshen som representerer hele DAGen.

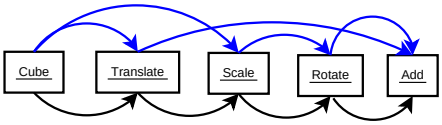
4.4.6 Nodefabrikken

Nodefabrikken benyttes til oppretting av noder basert på enumererte node typer, og all oppretting og frigjøring av noder gjøres via denne fabrikken. Oppretting av en node gjøres med `cmtf3d_create_node()` og frigjøres med `cmtf3d_destroy_node()`.

For at biblioteket skal kunne opprette noder av en spesiell type må det registreres en `creator` funksjon. Dette har som formål å fylle inn en node struktur med de data/funksjonspekere som noder av den spesielle typen trenger. Selve registreringen gjøres vha `cmtf3d_register_creator()`.



Figur 18: Graf som beskriver en 3D-modell



Figur 19: Prosesseringsliste for mesh generator (videreføring av figur 18). Blå linjer angir lenker til foreldre-noder, og svarte linjer angir lenking i prosesseringslisten.

4.4.7 I/O

På bakgrunn av at I/O må fungere på little- og big-endian hardware, samt at det skal være mulig å lese/skrive til minneområder valgte vi å implementere en uniform API for dette. Denne APIen lagrer alle data på network-byte-order og konvertering til/fra hardwarens representasjon gjøres transparent slik at man slipper å tenke på endianness ved skriving av innlesings- og utskrivingsfunksjoner for f.eks. nodene. For en gjennomgang av filformatet som biblioteket bruker se kapittel 4.6.

4.4.8 Nodetype implementasjon

Siden biblioteket bruker en nodefabrikk for oppretting av noder, så valgte vi å legge all funksjonalitet som trengs for å implementere en nodetype i en egen headerfil(`node_toolkit.h`). Dette for å gjøre det enklere å legge til nye nodetyper.

Nedenfor beskrives noen av hovedtypene av noder som er implementert.

4.4.9 Primitiver

Primitiver er bladnoder i grafen og generer mesher. Disse er uttrykt ved matematiske formler og under implementasjon har vi benyttet oss av Paul Bourke sine web-sider [5] om former og flater.

4.4.10 Transformasjoner

Dette er interne noder i grafen og utfører transformasjoner på sin input node. Under implementasjon av disse har vi benyttet oss av matematiske formler på matriseform slik de er definert i boken 'Computer Graphics 2nd ed´ [3].

4.4.11 Deformasjon

Deformasjoner er interne noder i grafen og utfører deformasjoner på sin input node. Basis deformasjonene ble implementert etter beskrivelsene i kapitlet '*Deformation independent of representation´* fra boken 'Advanced Animation and Rendering Techniques - *Theory and Practice´* [6] og ble supplementert med Alan H. Barrs '*Global and Local Deformations of Solid Primitives´* [7] som denne bygger på.

Vi så også på muligheten for å implementere FFD[6] slik dette er beskrevet i SEDE86 [8] og en mindre restriktiv variant kalt Extended FFD beskrevet i COQU90 [9], men fant at selv om disse deformeringsmetodene ga større modelleringsfrihet, så var de avhengig av mye mer minne en Alan H. Barrs metode.

4.4.12 CSG

Dette er interne noder i grafen som utfører CSG operasjoner på to input noder. En nærmere beskrivelse av CSG er å finne i kapittel 4.5.

4.5 Constructive Solid Geometry

CSG er implementert som ett eget bibliotek som benytter seg av datastrukturene i 3D biblioteket. Dette gjør så implementasjon av CSG nodetypene er triviell siden det bare er å bruke CSG bibliotekets API. Denne APIen er laget for å være enkel og består av en funksjon. Denne tar seg av oppsett og bruk av CSG bibliotekets interne datastrukturer, og er definert som følger:

```
enum csg_boolean_op {
        CSG_UNION,
        CSG_INTERSECTION,
        CSG_DIFFERENCE,
        CSG_CLASSIFY
```

[6] Free Form Deformation

```
};

int csg_perform_boolean_operation(
                enum csg_boolean_op  operation,
                struct cmtf3d_mesh  *first_mesh,
                struct cmtf3d_mesh  *second_mesh,
                struct cmtf3d_mesh **final_mesh);
```

4.5.1 BSP basert CSG implementasjon

Første forsøk på en implementasjon var å bruke BSP-trær[7] for å gjøre CSG operasjonene raskest mulig. Vi implementerte en algoritme som først ble beskrevet av Thibault & Naylor [10]. Algoritmen er også omtalt i 'Computer Graphics 2nd ed´ [3] kapittel 12.5.3.

Vi fant det det ønskelig at T-junctions(se kapittel 4.5.3) skulle motvirkes slik at polygoner ikke skulle få artifakter under lyssetting. Dette gjorde koden for splitting av polygoner en del vanskeligere å implementere, og medførte at minneforbruket eskalerte drastisk i en del tilfeller der BSP-trærne ikke var balanserte.

For å motvirke det økte minneforbruket grunnet motvirkning av T-junctions fant vi det hensiktsmessig å balansere BSP-trærne bedre. Dette viste seg å være CPU intensivt uten å forbedre minneforbruket alt for mye.

I tillegg til de ovenfornevnte problemene må det legges til at den BSP baserte algoritmen trengte tre kopier av hver input mesh. Noe vi fant kostbart rent minneforbruksmessig, samt at vi også problemer med numeriskstabilitet under polygonklassifiseringsfasen. Sistnevnte fant vi ingen løsning på, og det ble derfor nødvendig å forkaste implementasjonen for heller å satse på en enklere algoritme.

4.5.2 BBOX basert CSG implementasjon

Andre forsøk på CSG implementasjon var å bruke binære AABB-trær[8] med 'merdian fit´. Hoveddelen av denne algoritmen er beskrevet i Laildaw, Trumbore & Hughes [11]. Algoritmen er også omtalt i 'Computer Graphics 2nd ed´ [3] kapittel 12.6.4.

Under implementasjon av algoritmen valgte vi å ikke ta hensyn til T-junctions siden dette enkelt kan legges til ved en senere anledning. Vi konsentrere oss istedet om å få algoritmeimplementasjonen så rask og stabil som mulig.

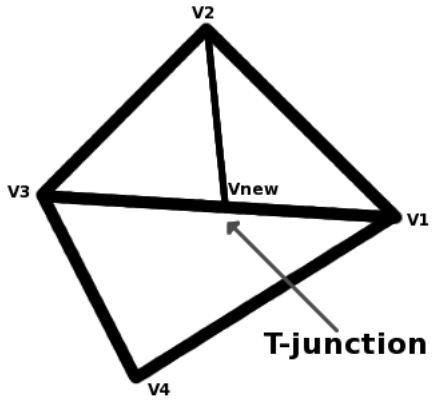
Etter å ha implementert basisalgoritmen fant vi at denne ikke var spesielt

[7] Binary Space Partitioning Trees
[8] Axis-Aligned Bounding Box trestruktur

rask. Algoritmen ble derfor utvidet med overlaptabeller[9] slik at det kunne gjøres oppslag på hvilke polygoner som skjærte gjennom ett spesifikt polygon. Dette gjorde klassifisering og splitting av polygoner betraktelig raskere, men det var fortsatt noen små problemer med numeriskstabilitet. Løsningen på dette var å innføre ϵ-verdier når det ble sjekket om punkters avstand fra et plan var null.

4.5.3 T-Junction problemet

Dette problemet oppstår når ett polygon skal deles i to eller flere polygoner (f.eks. ved å splitte polygon med et plan). Under selve splittingen blir det innsatt nye punkter langs det opprinnelige polygonets kanter. I tilfeller hvor det opprinnelige polygonet er tilknyttet andre polygoner vil ikke disse deles opp(se figur 20). Det vil da oppstå artifakter når ett polygon(som deler kanten som ble splittet) skal tegnes på skjerm med lyssetting.



Figur 20: Eksempel på en T-junction

4.6 .cmtf3d filformat

Dette filformatet lagrer beskrivelsen til 3D-modeller. Nærmere bestemt så lagres grafen til modellene. Rekkefølgen nodene til grafen lagres i har ikke noe å si, men implementasjonen går gjennom og lagrer nodene post-order.

Siden pekere ikke egner seg særskilt godt til å lagre informasjon om hvilke noder som er barn til hvilke noder, så brukes i stedet en 32 bits ID. Denne kalkuleres ved lagring av grafen, og brukes ved innlesing av grafen for å lenke sammen nodene. Hvordan IDene genereres er beskrevet i kapittel 4.6.3.

[9] Ideen om overlaptabeller fikk Thomas presentert av Kostas Pataridis(Navis/ASD) på TG05.

4.6.1 Header

Filformatet har en header på to bytes. Den første brukes til å identifisere filformatet. Den andre byten er delt i to nibbles(4 bits grupper) hvor den øverste er versjonsnummer, og den nederste er revisjonsnummer. Se tabell 1 for verdier disse skal inneholde.

Felt	Datatype	Størrelse	Verdi
Identifikator	uint8	1 byte	`0xED`
Version	nibble	4 bits	`0x1`
Revision	nibble	4 bits	`0x0`

Tabell 1: Headerfelt i filformatet.

4.6.2 Footer

Etter siste node i filen er det en *EOF*[10]-markør på en byte. Denne brukes til å signalisere at siste node er lest, og grunnen til at vi bruker en slik magisk-verdi er at innlesing av noder termineres når en nodetype med denne verdien lese inn.

Felt	Datatype	Størrelse	Verdi
EOF-markør	uint8	1 byte	`0xFF`

Tabell 2: Footer i filformatet.

4.6.3 Node

Grafens noder ligger sekvensielt etter hverandre i filformatet, og den noden som ikke er barn til noen av de andre nodene anses for å være grafens rotnode.

ID til input noder er gjort slik at nodene får stigende verdier fra 1 og

Felt	Datatype	Størrelse
Node type	uint8	1
ID til X input noder	uin32	4 * X
Node data	ukjent	ukjent

Tabell 3: Hvordan noder lagres i filformatet.

[10]EOF er en forkortelse for 'end-of-file'

oppover til $2^{32} - 1$. ID verdiene tilordnes nodene etter som de blir lest fra fil. Slik at første node som lese får ID 1, neste node får ID 2 o.s.v.

På overordnet nivå så vil nodene lagres som i tabell 3, men for å få ett litt bedre innblikk så kan vi se på lagring av en .cmtf3d fil som inneholder en kube og en bend. Først så har vi filen vist som en hex-dump:

```
ED 10 01 18
00 00 00 00
01 00 00 00
00 00 00 00
00 00 00 00
00 00 00 80
BF 00 00 80
3F 00 00 00
00 00 00 00
00 FF
```

Vi kjenner igjen de to første bytes som header blokken, og den siste som footer-blokken. I mellom disse ligger kube og bend nodenes data, og formaterer vi hex-dump så kommer disse klarere frem:

```
ED 10       Header
01          Node-type for kuben (denne har ingen parametre)
18          Node-type for bend
00 00 00 00 ID til input-node (32 bit unsigned integer)
01          Bøyningsakse (byte)
00 00 00 00 Posisjon (3 flyttall)
00 00 00 00
00 00 00 00
00 00 80 BF Til (flyttall)
00 00 80 3F Fra (flyttall)
00 00 00 00 Bøyningsgrad (flyttall)
00 00 00 00 Vridningsgrad (flyttall)
FF          Footer
```

4.7 GUI Toolkit

GUI er implementert som en SceneGraph(variant av en DAG). Dette fordi vi anså denne trestrukturen som enkel å implementere. I tillegg er det enkelt og intuitivt å posisjonere komponenter, siden hver enkelt komponent blir posisjonert i forhold til sin forelders koordinatsystem.

4.7.1 Component

Baseklassen `Component` inneholder all grunnleggende informasjon om en komponents koordinater relativt til forelder, dens størrelse og koordinater relativ til applikasjonsvinduet. Den inneholder også callback-funksjoner som blir kjørt dersom det registreres musehendelser, tastaturhendelser,

fokusskifte, og ved gjenopptegning.

For å slippe å implementere alle funksjoner i alle klasser som arver fra `Component` klassen har vi valgt å bruke et adapter som implementerer alle virtuelle funksjoner.

4.7.2 ActionListener

`ActionListener` er et interface til å motta meldinger. En klasse som skal behandle en melding implementerer dette interfacet og registrerer seg hos ett objekt av en klasse den vil motta meldinger fra. Når en melding blir sendt vil mottakerens `onActionEvent` funksjon bli kalt og meldingen kan behandles.

En klasse som vil sende beskjeder må implementere funksjonen `attachActionListener` for å registrere meldingsmottakere.

4.7.3 Button

Knapper benytter seg av `ActionListener` interfacet for å sende meldinger når de blir trykket på.

4.7.4 TextField

Bortsett fra å skrive inn tekst vil `TextField` sende en melding når `ENTER` blir trykket ned. For å gjøre dette brukes `ActionListener` interfacet for å sende teksten som ble skrevet inn.

4.7.5 Container

En `Container` er en utvidelse av `Component` som gjør det mulig at den kan inneholde andre komponenter. For å holde rede på hvilke komponenter en `Container` inneholder benyttes en `std::list`. Når en `Container` mottar hendelser (som f.eks. musebevegelse) sendes disse videre til den fokuserte komponenten.

4.7.6 ScrollBar

En `ScrollBar` er et scrollfelt, horisontal, eller vertikalt. Man kan scrolle ved å benytte musen eller piltastene. Når det scrolles vil `ScrollBar` benytte seg av `ActionListener` interfacet for å sende beskjed om dette.

4.7.7 MenuBar, Menu, MenuItem, MenuSeparator

En `MenuBar` er en menylinje. Når man beveger musen over elementene i menylinjen vil det dukke opp en `Menu` som består av ett eller flere `MenuItem`

adskilt med `MenuSeparator` objekter. Om man klikker på felter i menyen vil dette benytte seg av `ActionListner` interfacet. En `MenuSeparator` er en ren grafisk komponent som brukes til å skille menyelementer.

4.7.8 GUIHandler

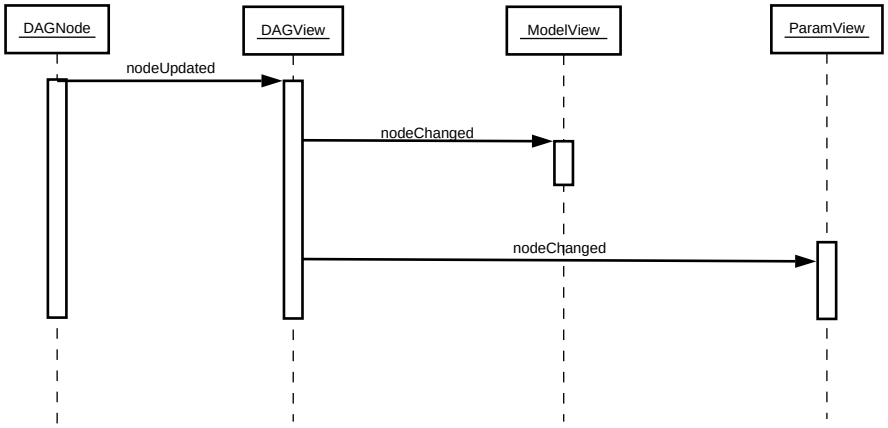
For å forenkle integrasjon med applikasjon med tanke på input har vi laget en klasse som tar seg av videresending av hendelser til de komponenter som skal få dem. M.a.o. sørger `GUIHandler` klassen for at komponentene på skjermen får de beskjeder de skal.

4.8 Modelleringsverktøy

Modelleringsverktøyet er bygget opp ved hjelp av GLF og GUI modulene, samt CMTF3D biblioteket. Mye av implementasjonen bestod i å utvide komponentene i GUI modulen, samt å lage kommunikasjonsårene mellom de forskjellige delene i verktøyet.

4.8.1 Meldingssending

For å hele tiden vise en oppdatert 3D-modell benytter vi oss av Model View Controller pattern. Dette gir en lav kobling mellom datamodellen og viewene i applikasjonen, og er illustrert på figur 21. For å få til dette har vi implementert følgende to interface.



Figur 21: Meldingssending

DAGNodeListener:
`DAGNodeListener` er et interface for å kunne motta meldinger om at en node sine parametre har blitt forandret. DAGView implementerer dette interfacet og registrerer seg hos enhver node som blir opprettet.

DAGViewListener:
`DAGViewListener` er et interface for å motta meldinger om at aktiv node har blitt forandret. Mottaker får i tillegg melding når en node som skal vises i bakgrunnen har blitt valgt.

4.8.2 DAGController

En `DAGController` er et interface som brukes til å modifisere parametrene til aktiv node i de ortografiske viewene med mus og tastatur interaksjon.

4.8.3 DAGView

`DAGView` er en utvidelse av `ScrollPane` der man kan plassere `DAGNode` objekter og knytte dem sammen til en graf. En `DAGNode` er en wrapper rundt en node i 3D-biblioteket.

4.8.4 ParamView

`ParamView` er en utvidelse av `Panel` som viser parametrene til valgt node. Den implementerer DAGViewListener interfacet for å kunne bytte parametervisning etter hvilken nodetype som er valgt. Parametervisningen for hver nodetype er implementert som klasser avledet fra `NodeParam`.

4.8.5 OrthoView

`OrthoView` er en utvidelse av `ModellView` og tar seg av opptegning av modellen i ortografisk perspektiv. Dersom valgt node har en `DAGController` vil alle musebevegelser og tastetrykk sendes til denne kontrolleren. `OrthoView` implementerer i tillegg DAGViewListener interfacet for å motta beskjed om hvilken node som skal vises.

4.8.6 PerspView

`PerspView` er en utvidelse av `ModellView` og tar seg av opptegning av modellen i perspektiv. `PerspView` implementerer også DAGViewListener interfacet for å motta beskjed om hvilken node som skal vises.

5 Kvalitetssikring og testing

Dette kapitlet beskriver de metoder og verktøy vi har benyttet for å sikre kvaliteten til vårt sluttprodukt.

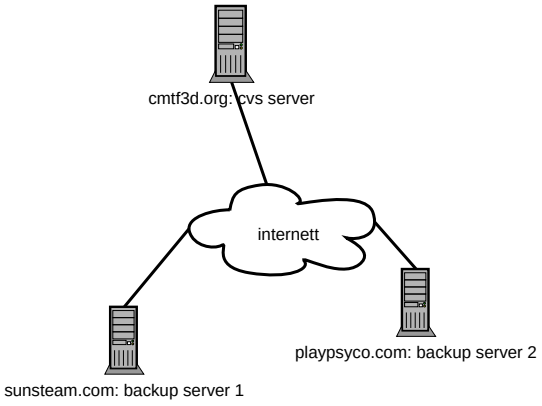
5.1 Versionsstyring

For å samkjøre kildekode har vi valgt å benytte oss av versjonstyring. Dette hjelper oss med å holde rede på historien til alle filer som ligger inni systemet, samt at det egner seg til kollaborasjon.

Fra tidligere prosjekter hadde gruppens medlemmer svært god erfaring med bruk av versjonstyringsprogrammet CVS. Siden vi alle kjente dette programmet godt, og var klar over de restriksjoner man må leve med. Undersøkte vi også alternative programmer som for eksempel ARCH(`http://www.gnu.org/software/gnu-arch/`) , men det viste seg at de alternative programmene ikke var like velutviklede som CVS.

5.2 Backup

Siden tap av data ville ført til mange bortkastede timer og i tillegg gitt konsekvenser for innleveringen av prosjektet, anså vi det som svært nødvendig å innføre gode rutiner for backup. Dette har blitt gjort ved at et backup script ble skrevet for så å bli satt igang på to fysisk adskilte servere. Scriptet tar backup av alt arbeidet til fastsatte tider. Den første serveren tar backup hver onsdag og lørdag, og den andre hver torsdag og søndag.



Figur 22: Server oppsett for backup

5.3 Testing

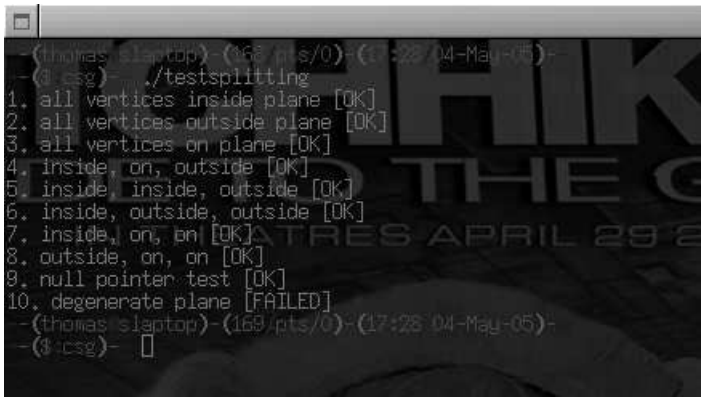
Som et ledd i testingen har både whitebox- og blackbox-testing blitt brukt. Hovedforskjellen på disse testmetodene ligger i om testeren har innsikt i løsningens oppbygging eller ikke.

Blackbox-testing vil si at man tester med hovedvekt på input og output. Testeren har ikke kjennskap til selve implementasjonen av funksjonalitet som skal testes, men kan teste hva resultatet av å eksekvere funksjonaliteten blir.

Whitebox-testing drar nytte av at testeren har god kunnskap(ved innsyn i kildekode) om hvordan funksjonalitet er implementert. Dette gjør så testeren kan utføre tester som maksimerer antall codepaths som blir kjørt/testet. Hvis feil oppstår under en slik test kan testeren lett lokalisere hvor feilen oppsto.

5.3.1 Whitebox

Whitebox-testing ble utført av utviklerne d.v.s. vi som har skrevet rapporten. Testingen har blitt utført etter hvert som funksjonalitet har blitt ferdige. Det har også blitt gjennomført testing av samspill mellom forskjellige deler av løsningen der dette har vært hensiktsmessig. Under disse testene har det blitt logført hvilke deler som har ikke har fungert som forventet slik at dette kunne utbedres senere.



Figur 23: Kjøring av test for splitting av polygoner.

Eksempelvis så ble det skrevet ett enkelt testprogram for testing av polygonsplittekoden som brukes av CSG. Dette programmet inneholdt en samling polygoner og plan disse skulle splittes av. Denne polygonsamlingen

var satt sammen slik at alle *branches* i oppsplittingskoden ble kjørt. En database med forventet resultat var også inkludert slik at kodens riktighet kunne verifiseres. Hvis ett polygon ikke ble riktig splittet, så skrev testprogrammet ut hvilken codepath som polygon/plan testen var designet for(se figur 23).

5.3.2 Blackbox

Blackbox-testing har hovedsaklig blitt gjort av arbeidsgiver. Han har testet applikasjonen uten innblanding fra prosjektgruppen, og har fylt ut ett testskjema(se appendix F) som var vedlagt modelleringsverktøyet. På dette skjema var all funksjonalitet som skulle testes ført opp, samt felter for svaring på opp funksjonalitet fungerte som forventet.

6 Drøfting av resultat

I dette kapittelet diskuterer vi resultatet satt opp mot kravspesifikasjonen og drøfter de valg vi har tatt underveis.

6.1 Kritikk av oppgaveløsningen

Prosjektgruppen er relativt fornøyd med utfallet av prosjektarbeidet, og har ikke angret på oppgavevalget. Prosjektarbeidet har blitt gjennomført i tråd med fremdriftsplanen uten store avvik, og løsningen innfrir de krav som ble satt i kravspesifikasjonen.

Underveis i prosjektet har arbeidsgiver kommet med ønsker om ny funksjonalitet. Disse ønskene har vi tatt tak i, og en oversikt over ekstrafunskjonalitet er gjennomgått i kapitlet om avvik(se kap. 6.2). Vi ser at dette kunne vært unngått hadde vi jobbet bedre med kravspesifiseringsfasen, og vært mer pågående i forhold til å innhente tilbakemelding fra arbeidsgiver.

Kommunikasjon mellom prosjektgruppen og arbeidsgiver foregikk hovedsaklig ved hjelp av epost og instant messaging. Dette fungerte i hovedsak bra, men det fordret at beskjeder ble konsist skrevet, slik at misforståelser ikke oppstod. I ettertid ser vi at vår utstrakte bruk av disse kommunikasjonsårene kunne vært supplementert med flere møter.

Når vi ser tilbake på valget av inkrementellutviklingsmodell kan vi konkludere med at dette stod godt til vårt prosjekt. Modellen hjalp oss til å ta opp ny funksjonalitet underveis. Dette gjorde så arbeidsgiver hele tiden var oppdatert på fremgangen ved at nye utgivelser ble utgitt etter hvert som inkrementene ble fullført. I designfasen valgte vi å dra inn modelleringsteknikker fra RUP[11] siden vi er godt kjente med disse, og mente at de ville forenkle design av løsningen. Bruken av RUP-teknikkene fungerte bra i samkvem med den inkrementelle modellen.

6.2 Avvik fra kravspesifikasjon

Vi har en fungerende løsning som støtter alle primitiver og operasjoner som fremgår av kravspesifikasjonen. I tillegg dukket det opp nye krav underveis. Disse ble implementert og her følger en oversikt:

Primitiver

- Knot

[11] Rational Unified Process

- Supershape
- Spherical Harmonics

Operasjoner

- Normal scale
- Pole
- Vortex
- Sin wave
- Wave
- UV mapping
- Radial array
- Smooth
- Mirror
- Spike
- Explode

Ønske om den nye funksjonaliteten kom frem under testing gjort av arbeidsgiver, og siden implementasjon av endel funksjonalitet tok noe mindre tid enn forventet ble det besluttet å implementere disse. Dette ga utvidede muligheter for modellering av komplekse modeller.

6.3 Videre arbeid

Siden vi har prioritert å få funksjonalitet på plass er det endel områder innen brukergrensesnittet som har blitt nedprioritert. Som f.eks korrespondanse mellom musebevegelser og den visuelle feedback brukeren forventer.

Angre funksjonalitet vil være svært nyttig å legge til. Ut fra feedback fra arbeidsgiver ville dette forenkle redigering av både grafer og nodeparametre.

Valg av flere noder i DAGView for flytting av disse. Dette ville forenkle opprydding i grafrepresentasjonen av 3D-modeller, og i følge arbeidsgiver er dette ønskelig å legge til i en evt. videreutvikling av løsningen.

Forbedre hastigheten til koden som tegner opp 3D-modeller på skjermen.

Slik koden fungerer i dag så vil 3D-modeller med mer en 40000 polygoner tegnes opp så langsomt at modelleringsverktøyet virker som det henger.

Visuelt sett er GUI mindre attraktivt, da det ikke er lagt vekt på design av dette. Det er derfor rom for store forbedringer, og det er lagt til rette for at koden for opptegning av GUI elementer kan omskrives.

Biblioteket bør videreutvikles slik at det støtter flere plattformer, og det bør satses på å få ned minneforbruket ytterligere under generering av mesher. Dette med tanke på bruk i mobile enheter og i spillkonsoller.

6.4 Evaluering av verktøyvalg

Siden vi allerede var vant til å jobbe med endel av verktøyene fra tidligere, vil vi her kommentere hvordan det var å bruke de øvrige verktøyene.

Valgrind
Denne ble brukt til debugging av minnehåndteringsfeil under Linux og når man blir vant til hvordan den presenterer sine funn på skjerm, er dette et effektivt verktøy. Et problem med Valgrind er at man må sørge for at den ikke følger funksjonskall til grafikk-kort moduler i kjernen, da dette gjorde at systemet hang seg.

Doxygen
Doxygen fungerer svært likt Javadoc, noe vi er kjent med fra tidligere prosjekter. Vi syntes derimot at Doxygen er bedre på mange måter ved at den har større muligheter for konfigurering, støtter flere programmeringsspråk, dokumentasjon på flere formater og automagisk generering av klassediagrammer.

Under utvikling av løsningen ble den genererte dokumentasjonen brukt flittig som oppslagsverk.

Gantt-Project
Til prosjektstyring på et prosjekt av vår størrelse fungerte Gantt-Project helt greit, men det hadde en feil. I visse tilfeller måtte skalere applikasjonsvinduet for å få den til å tegne gantt skjemaet korrekt.

6.5 Evaluering av arbeid

Gjennom prosjektperioden har gruppen jobbet strukturert og samarbeidet innad har vært godt. Alle har klart å holdt motivasjonen oppe selv når vi har hatt store utfordringer underveis. Dette har gjenspeilet seg i fremdriften til prosjektet ved at vi hele tiden har ligget godt ann i forhold

til fremdriftsplanen.

6.6 Fordeling av arbeid

Fordeling av arbeid på dagsbasis har blitt gjort hver morgen gjennom hele prosjektperioden. Dette har fungert fint og alle har hatt nok å gjøre. En gang i blandt har det blitt litt skjevfordeling av arbeidsmengde, men alt i alt har dette rettet seg opp i løpet av prosjektets gang. Når det gjelder fordeling av hovedansvar for de forskjellige moduler(se kap. 1.5.2) har dette fungert godt, og det har bidratt til å holde oversikten i de tilfeller hvor vedlikehold har vært aktuelt.

7 Konklusjon

Gjennom et halvt års prosjektarbeid har vi hatt muligheten til å følge applikasjonen fra å være en visjon, til å bli en ferdig fungerende, dokumentert og testet applikasjon. Dette har stilt store krav til planlegging og evne til å foreta veloverveide beslutninger.

Vi har lært mye av dette prosjektet, først og fremst programmeringsmessig, men også en del om systemutvikling og hvordan det er å jobbe på, og ha ansvaret for et større prosjekt. Vi har blant annet blitt bedre kjent med C, C++ og OpenGL som vi hadde erfaringer med fra før.

Vi er selv godt fornøyd med resultatet vi endte opp med, hvis vi ser bort i fra problemene rundt OS uavhengighet. Vi håper resultatet av prosjektet vil være til nytte for arbeidsgiver.

Nomenclature

*NIX Fellesnevner for alle UNIX varianter, page 29

API Application Programming Interface, page 8

Baseklasse En baseklasse er en klasse som andre klasser arver fra, page 46

Bladnode Nodene i endene av en DAG, page 22

Callback-funksjon Funksjoner som blir kalt når en hendelse inntreffer, page 33

DAG Directed Acyclic Graph, page 22

DDD Data Display Debugger, page 30

GCC GNU Compiler Collection, page 29

GDB GNU Project Debugger, page 30

Graf En samling noder og stier mellom dem., page 22

GUI Graphical User Interface, page 19

HTML Hyper Text Markup language, page 20

MS Microsoft, page 29

Node En datastruktur som kan inneholde forskjellig informasjon, page 22

OO Objekt orientering, page 29

OpenGL Et grafikk bibliotek, page 16

Referanser

[1] Craig Larman. *Applying UML and Patterns - an introduction to Object-Oriented Analysis and Design, second edition.* 2002.

[2] Ulrich Stern. Java vs. c++. 2001. `http://verify.stanford.edu/uli/java_cpp.html`.

[3] Foley van Dam et al. *Computer Graphics 2nd ed.* WHO KNOWS, 1995.

[4] Bruce G. Baumgart. Winged edge polyhedron representation. 1972.

[5] Paul Bourke. Surfaces. `http://astronomy.swin.edu.au/~pbourke/surfaces/`.

[6] Alan & Mark Watt. *Advanced Animation and Rendering Techniques - Theory and Practice*, chapter 17. ACM Press, 1992.

[7] A. H. Barr. Global and local deformations of solid primitives. *Proceedings of SIGGRAPH*, Computer Graphics 18(3):21–30, July 1984.

[8] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *Proceedings of SIGGRAPH*, Computer Graphics 20(4):151–159, August 1986.

[9] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. *Proceedings of SIGGRAPH*, Computer Graphics 24(4):187–196, August 1990.

[10] Thibault & Naylor. Set operations on polyhedra using binary space partitioning trees. July 1987.

[11] Trumbore & Hughes Laidlaw. Constructive solid geometry for polyhedral objects. 1986.