NTNU
Det skapende universitet

Brukergrensesnitt for beregning av tverrsnittsdata for komplekse bjelketverrsnitt

Bjørnar Eggesbø Askevold

Master i ingeniørvitenskap og IKT
Innlevert: juni 2014
Hovedveileder: Bjørn Haugen, IPM

Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR PRODUKTUTVIKLING
OG MATERIALER

MASTEROPPGAVE VÅR 2014
FOR
STUD.TECHN. BJØRNAR EGGESBØ ASKEVOLD

BRUKERGRENSESNITT FOR BEREGNING AV TVERRSNITTSDATA FOR KOMPLEKSE BJELKETVERRSNITT
User interface for computation of parameters of complex beam cross sections

Tverrsnittsdata for komplekse bjelketverrsnitt som f.eks. et vindturbin-blad er vanskelig å beregne analytisk. Dynamiske beregninger av vindturbiner benytter ofte bjelkemodeller for turbin-bladene, og resultatene er avhengig av korrekte tverrsnittsdata.

Oppgaven tar sikte på å utvikle brukergrensesnitt delen av en applikasjon som skal beregne tverrsnittsdata for komplekse tverrsnitt ved hjelp av elementmetoden. Det er ønskelig at applikasjonen også kan håndtere kompositttmaterialer.

Brukergrensesnittet bør kunne brukes til å modellere tverrsnittet og også til å visualisere spenningstilstanden til tverrsnittet basert på spenningsresultanter som moment, skjærkraft og aksialkraft. Applikasjonen bør være enkel å bruke da en ønsker å benytte den i undervisningssammenheng. Brukergrensesnittet bør bygges rundt en numerisk elementmetode modul som kjøres som en separat prosess.

Oppgaven har fokus på modellering og resultatpresentasjon for en komplett applikasjon for beregning av tverrsnittsparametre og spenningsvariasjon over komplekse bjelketverrsnitt. Grensesnittet mellom det grafiske brukergrensesnittet og den numeriske koden må defineres i samarbeid med annen masteroppgave som har dette som hovedfokus.

Senest 3 uker etter oppgavestart skal et A3 ark som illustrerer arbeidet leveres inn. En mal for dette arket finnes på instituttets hjemmeside under menyen masteroppgave (http://www.ntnu.no/ipm/masteroppgave). Arket skal også oppdateres en uke før innlevering av masteroppgaven.

Arbeidet i masteroppgaven skal risikovurderes. Hovedaktiviteter som er kjent/planlagt skal risikovurderes ved oppstart og skjema skal leveres innen 3 uker etter utlevering av oppgavetekst. Alle prosjekt skal vurderes, også de som kun er teoretiske og virtuelle. Skjemaet må signeres av veileder. Risikovurdering er en løpende dokumentasjon og skal gjøres før oppstart av enhver aktivitet som KAN være forbundet med risiko. Kopi av signert risikovurdering skal være inkludert i vedlegg ved levering av rapport

Besvarelsen skal ha med signert oppgavetekst, og redigeres mest mulig som en forskningsrapport med et sammendrag på norsk og engelsk, konklusjon, litteraturliste, innholdsfortegnelse, etc. Ved utarbeidelse av teksten skal kandidaten legge vekt på å gjøre teksten oversiktlig og velskrevet. Med henblikk på lesning av besvarelsen er det viktig at de

nødvendige henvisninger for korresponderende steder i tekst, tabeller og figurer anføres på begge steder. Ved bedømmelse legges det stor vekt på at resultater er grundig bearbeidet, at de oppstilles tabellarisk og/eller grafisk på en oversiktlig måte og diskuteres utførlig.

Besvarelsen skal leveres i elektronisk format via DAIM, NTNUs system for Digital arkivering og innlevering av masteroppgaver.

Torgeir Welo
Instituttleder

Bjørn Haugen
Faglærer

NTNU
Norges teknisk-
naturvitenskapelige universitet
Institutt for produktutvikling
og materialer

Summary

Dynamic calculations often depend on detailed cross sections data, creating a demand for an application that enables easy modeling of complex cross sections. This master thesis describes the process in which a user interface for finite element analysis on complex cross sections were developed. The application developed is bundled as part of a larger software package, and the application enables the user to model cross sections as well as visualize results after analysis.

Modeling of cross sections is performed using advanced operations, and usability has been stressed as a key feature throughout the process. The application relies on Qt to deliver native cross platform support, and follows the pattern Model-View-Controller (MVC), a common architecture used in interaction-oriented software. It utilizes the library OpenCASCADE Technology for several advanced tools used in geometric operations, and the application make extensive use of The Visualization Toolkit (VTK) to deliver graphics.

The report describes the structure of modules in the software package, as well as the application's role in the workflow. The software libraries exploited for functionality is reviewed and their implementation is carefully described. The report also describes choices that were faced and decisions taken to design and in the overall development of the application.

Sammendrag

Dynamiske beregninger benytter seg ofte av bjelkemodeller, og er avhengig av korrekte tverrsnittsdata. Det er derfor et behov for et program som kan brukes til modellering av komplekse tverrsnitt. Rapporten omhandler utviklingen av et brukergrensesnitt til en programpakke for beregning av komplekse tverrsnitt. Den ferdige applikasjonen er en modul i en programpakke, og gir brukeren mulighet for modellering av tverrsnitt samt visualisering av resultatdata etter en analyse. Modelleringen kan utføres med forholdsvis avanserte operasjoner, og det har vært fokus på å lage en applikasjon som er brukervennlig og effektiv i bruk for flere brukergrupper.

Brukergrensesnittet er laget med kryssplattformstøtte ved hjelp av Qt og følger arkitekturmønsteret *Model-View-Controller (MVC)* for god lesbarhet. En del av modelleringsrammeverket *OpenCASCADE Technology (OCCT)* blir brukt for levere brukeren enkelte verktøy for modelleringen, og grafikkvinduet er basert på *The Visualization Toolkit (VTK)*, som er et omfattende rammeverk for visualisering.

Rapporten redegjør for programpakkens oppbygning og utvikling, og beskriver applikasjonens rolle i pakken. De til dels store rammeverk som blir brukt for å levere funksjonalitet blir gjennomgått og vurdert opp mot funksjonalitet. Til slutt følger det en detaljert beskrivelse av implementasjonen av både systemarkitektur og brukerinteraksjon. Rapporten inneholder også dokumentasjon bak designvalg og beslutninger tatt i utviklingsprosessen.

Forord

Denne masteroppgaven er et resultat av at Institutt for produktutvikling og materialer (IPM) ved Norges teknisk-naturvitenskapelige universitet (NTNU) høsten 2013 hadde et ønske om å lage en modularisert programpakke med mulighet for beregning av komplekse tverrsnitt. Programpakken skulle ta sikte på å utføre hurtige analyser på ulike tverrsnitt i forskjellige materialer, på lik linje med programmet «CrossX». Det var på sikt også ønsket at den nye programpakken skulle ha mulighet for å utføre analyser på tverrsnitt bestående av kompositttmaterialer. Utviklingen av programpakken er planlagt å gå over flere prosjekt- og masteroppgaver, og funksjonaliteten skal utvides gradvis over tid.

Programpakken ble påbegynt i en prosjektoppgave for Bjørnar Askevold, Fredrik Nordmoen og Kristian Strømstad under veiledning av Bjørn Haugen. Den ble delt opp i henholdsvis «solver» og «brukergrensesnitt» for å også gjøre dem mer anvendelige hver for seg.

Utviklingen av brukergrensesnittet til programpakken ble startet fra bunnen av i denne oppgaven, med en videreføring av tankegangen påbegynt i prosjektoppgaven. Oppgaven har vært koordinert med masteroppgaven til Kristian Strømstad, som bygget videre på vårt arbeid lagt i «solveren» høsten 2013. Arbeidet har vært utført med ukentlige møter med min veileder, professor Bjørn Haugen, som jeg vil takke for innspill underveis i prosessen.

Trondheim
Juni 2014

Bjørnar Eggesbø Askevold

Innhold

Tabeller

Figurer

Forkortelser

VTK The Visualization Toolkit. Et rammeverk for å visualisere alle former av forskningsdata

XML Extensible Markup Language

GUI Graphical User Interface

OCCT OpenCASCADE Technology. Et 3D-modelleringsrammeverk med biblioteker for blant annet avanserte geometriske operasjoner

GPL General Public Licence

LGPL Lesser General Public Licence

BSD Berkeley Software Distribution Licence

Definisjoner

Mesh En sammenhengende nett/graf bestående av kanter og noder

Preprosessor Program som lar brukeren tegne geometriske figurer og meshe dem

Solver Program som tar inn meshdatafil samt innspenninger og krefter, og regner ut meshets egenskaper

Postprosessor Viser resultatet fra solveren. Blant annet grafisk representasjon av krefter og forskyvninger i meshet

CrossX Program for beregning av tverrsnitt utviklet av Institutt for konstruksjonsteknikk (KT)

Brukergrensesnitt Flaten av programmet som brukeren ser og interagerer med

Plattformuavhengig Et program som kjører på flere operativsystemer(plattformer)

Qt Rammeverk for plattformuavhengige brukergrensesnitt og applikasjoner

Gmsh Program for generering av mesh

OpenCASCADE Bedrift som utvikler 3D-modelleringsrammeverket OpenCASCADE Technology

Widget Et sett av elementer som definerer en enestående del av brukergrensesnittet

C++ Programmeringspråk brukt i utviklingen av denne applikasjonen

Kapittel 1

Innledning

I oppgaveteksten er det beskrevet et brukergrensesnitt som vil være en del av en større programpakke laget for beregning av tverrsnittsdata for komplekse tverrsnitt. Rapporten tar for seg utviklingen og valgene tatt under utviklingen av brukergrensesnittet. Et brukergrensesnitt er «*kontaktflaten mellom brukeren og datamaskinens operativsystem og programmer, og avgjør hvordan brukeren styrer programmene*»[4]. Til tross for at et brukergrensesnitt er definert som en kontaktflate, blir applikasjonen i denne oppgaven omtalt som et brukergrensesnitt fordi det er det grafiske grensesnittet mot de andre modulene. I rapporten vil den utviklede modulen bli omtalt som «applikasjonen», og grensesnittet vil ha to hovedbruksområder. Det skal kunne virke som henholdsvis;

1. Preprosessor: Et grensesnitt for å modellere tverrsnittsdata og konfigurere materialparametre. Preprosessoren er også ansvarlig for å lage meshet.

2. Postprosessor: Leser resultatdatasettet og presenterer resultatet fra beregningene som en 3D-modell. Den viser blant annet forskyvninger og spenningsresultanter visuelt som et lag med farge på figuren, men den er ikke på noen måte begrenset til kun å vise disse resultatene.

Preprosessor og postprosessoren er implementert som ulike modus i applikasjonen. I tillegg til å gi et brukergrensesnitt med modelleringsfunksjonalitet, håndterer også applikasjonen de andre modulene og styrer kommunikasjonen mellom dem.

1.1 Bakgrunn

Det finnes mange andre programmer med mulighet for å gjøre analyser av tverrsnitt. Flere av programmene som finnes er store med og det kreves derfor omfattende arbeid for å få ut enkel data om et tverrsnitt. CrossX er utviklet ved Institutt for konstruksjonsteknikk(KT) med fokus på å utregne tverrsnittsparametre for bjelketverrsnitt, og dekker nettopp mye av dette behovet. Ulempen med CrossX er at det ikke er planlagt støtte for utregninger av tverrsnitt med kompositttmaterialer. Brukergrensesnittet fremstår også som noe utdatert, og litt tungvint i bruk. Det var derfor et ønske om å utvikle et program med samme begrensede kjernefunksjonalitet og med støtte for avanserte materialer, som kan brukes til undervisning. Programmene skal kunne kjøres i et skript for å løse større oppgaver, og det var derfor fordelaktig å bygge opp programmet bestående av flere moduler.

1.2 Programpakkens oppbygning

Programpakken består av modulene «Mesher», «Solver» og brukergrensesnitt, og representerer ulike steg i arbeidsprosessen ved bruk av elementmetoden.

1.2.1 Mesher

Enhver geometrisk figur som det utføres beregninger på, må først gjøres om til et «mesh» bestående av en mengde små elementer. Det er meshprogrammets oppgave å dele figuren inn i elementer. Elementene i de ulike dimensjonene forestilles som henholdsvis; en bjelke(1D), flate(2D) eller kube(3D). Alle elementene er bygget opp av noder og kanter. Kantene fungerer som koblinger mellom nodene og utgjør meshet. Koblingene i et element får tildelt styrke basert på området elementet dekker og det underliggende materialets egenskaper. I denne oppgaven blir det kun brukt mesh basert på triangelelementer med tre noder, men ett element kunne også ha bestått av flere noder.

1.2.2 Solver

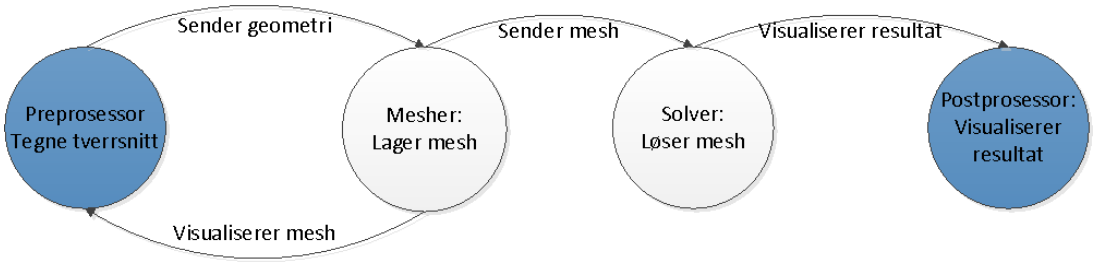
Solveren utfører beregningene på meshet av tverrsnittet. Den regner ut spenningsresultater gitt en kraft, vridning, eller et moment. Belastning av tverr-

snittet vil være beregnet med enhetsverdier (1N, 1M, 1rad). Bruken av enhetskrefter er en fordel for brukeren fordi det kreves færre steg fra modelleringen av et tverrsnitt til et ferdig resultat. Forskyvninger og stress i tverrsnitt kan enkelt regnes ut i etterkant ved å multiplisere enhetskraften og resultatet. Solveren er en modul i programpakken og kan kjøres som en selvstendig applikasjon.

1.2.3 Modulenes sammenheng

De selvstendige modulene «mesher», «solver» og brukergrensesnittet kan ses på som en rekke steg utført fra modellering av problemet til resultatet, som vist i Figur 1.1. Brukeren vil starte med å lage et tverrsnitt i preprosessoren. Tverrsnittet blir deretter sendt til mesher som lager et mesh. Hvis brukeren er fornøyd med meshet sendes det videre til solver for analyse. Når solveren er ferdig lagres resultatet i en fil og lastes inn av postprosessoren som lar brukeren visualisere de ulike resultatene. Bruken av Qt gjør at også filhåndtering og prosesstyring skal virke på alle operativsystemer støttet av Qt.

Programpakken er delt opp i moduler for at brukeren skal kunne starte på en vilkårlig plass i denne prosessen og for å ha mulighet til å bruke en modul alene. Dette tillater for eksempel at brukeren kan løse en mengde ferdiggenererte tverrsnitt fortløpende uten å kreve at brukeren nødvendigvis går via et brukergrensesnitt hver gang.



Figur 1.1: Arbeidsflyten i programpakken starter med å modellere problemet i tverrsnittet og lage et mesh. Når meshet ser bra ut løses det i solveren og vises i postprosessor. Sirklene uthevet i blå, er modulene omfattet av denne oppgaven

1.3 Avgrensninger

Det ble satt noen begrenseninger i oppaven for at den ikke skulle bli for stor. Applikasjonen starter Gmsh som en ekstern prosess og utveksler data med denne ved lagring til fil. Det er flere fordeler forbundet med å integrere meshprosessen i applikasjonen, men det kreves et meshprogram som er åpent og med en lisens som tillater slik bruk. Gmsh har en begrenset lisens og koden til Gmsh viste seg å være krevende å kompilere. Løsningen valgt i prosjektet fungerer godt, og integrasjon av mesher ble derfor ikke ansett som nødvendig før en mer egnet mesher er tilgjengelig.

Det var ønsket at programpakken skal støtte tverrsnitt bestående av blant annet kompositttmaterialer. På dette tidspunktet støtter ikke solveren kompositttmaterialer, og det ble bestemt å utelate slik funksjonalitet for å i stedet fokusere på helheten av brukergrensesnittet.

Det var også nødvendig å begrense antall verktøy for bruk i modellering av tverrsnitt. Det tar lang tid å implementere verktøy, og arbeidet i oppgaven har derfor blitt begrenset til å tegne figurer og å utføre «Cut» og «Fuse» operasjoner.

Kapittel 2

Arbeidsprosess

2.1 Prosjektoppgave

Prosjektoppgaven ble utført høsten 2013, i samarbeid med Fredrik Nordmoen og Kristian Strømstad. Oppgaven omhandlet implementasjon av en solver skrevet i C++, av Kristian Strømstad og undertegnede. Den jobbet med mesh bestående av triangelelementer, og regnet ut tyngesenter, skjærsenter samt prinsipialakser. Resultatet ble skrevet ut i et eget format, som ble lest av et enkelt brukergrensesnitt. Brukergrensesnittet ble utviklet av Fredrik Nordmoen ved bruk av Java på grunn av problemer med å kompilere VTK med Qt. Det ble konkludert med at det ville være en stor fordel å bytte til C++ så fort VTK fikk støtte for Qt.

2.2 Masteroppgave

Rapporten er en oversikt over utviklingsprosessen med brukergrensesnittet våren 2014. Masteroppgaven startet med en verifisering av at VTK6.1 og Qt5.2 kunne kompileres og kjøres sømløst, for så å kartlegge hvilke funksjoner i VTK som kunne utnyttes i applikasjonen. Støtten for VTK kombinert med Qt er nødvendig for å kunne opprettholde ønsket om at applikasjonen skal ha kryssplattformstøtte. Under testingen av VTK var det spesielt interessant å se hvordan VTK kunne benyttes til modellering av geometriske figurer.

Grunnet byttet av programmeringsspråk fra Java til C++ og Qt, var det

lite av Fredrik's tidligere arbeidet som kunne overføres fra prosjektoppgaven. Utviklingen av brukergrensesnittet må derfor anses for å ha startet fra grunnen av våren 2014. Det har ført til at det underveis har vært lagt ned arbeid med å lage en arkitektur som det kan bygges videre på, og som er ryddig og lett å vedlikeholde. Det har også vært fokus på å gjøre koden tydelig med god dokumentasjon. På grunn av brukergrensesnittets omfattende funksjonalitet ble det underveis i utviklingen diskutert hva som er kjernefunksjonalitet og hvilken funksjonalitet som hadde høyest prioritet. Arbeidet ble til slutt evaluert med en brukertest.

2.3 Utviklingen

2.3.1 Verktøy

Visual Studio

Visual Studio er utviklet av Microsoft, som en utviklingsplattform for Windows. Den har god støtte for C++, og er derfor et naturlig valg. Visual Studio støtter ikke Qt som standard, men har mulighet for dette gjennom tillegg. Visual Studio 2012 med tillegg som gir støtte for utvikling med bruk av Qt5.2 var installert. Plugin for Qt støtter i skrivende stund ikke en høyere versjon enn Visual Studio 2012.

Qt-Designer

Skallet til brukergrensesnittet er utformet i Qt-Designer, som er et enkeltstående program i Qt-rammeverket. Qt-Designer bygger opp brukergrensesnittet visuelt, av elementer som flyttes og konfigureres. Skallet lagres i en XML-fil, og gir derfor en ønsket frikobling mellom utseende av programmet og interaksjonskoden. Dette gjør det mye enklere å utføre endringer i utseende uten å måtte gå gjennom koden. Bruken av Qt-Designer krever derfor ingen forståelse av programmering, og personen som jobber med dette kan fokusere på design og brukervennlighet. Se seksjon 4.2.1 for en mer detaljert gjennomgang av fordeler ved en slik frikobling.

2.3.2 Dokumentasjon

For å gjøre overgangen til en ny utvikler lettere og på grunn av prosjektets størrelse var det nødvendig med en gjennomgående dokumentasjon. I tillegg til at denne rapporten skulle kunne fungere som dokumentasjon for applikasjonen, er det utarbeidet en en mer detaljert dokumentasjon som er skrevet direkte inn i definisjonsfilene i koden med egne «tags». Disse blir lest av et program kalt Doxygen. Doxygen henter ut kommentarene samt funksjons- og variabeldefinisjonene, og lager en nettside og en pdf-fil med den ekstraherte strukturen samt kommenteringen. Fordelen med kommentering i definisjonsfilen er at det er lettere å holde dokumentasjonen oppdatert fordi kommentarene kan leses og endres sammen med koden. Dokumentasjonen fra Doxygen er ment som en mer detaljert gjennomgang av koden og har vært utarbeidet parallelt med utviklingen av applikasjonen. På grunn av kildekodens størrelse er kun dokumentasjonen tatt ut og vedlagt i eget vedlegg (Appendiks F).

2.3.3 Revisjonskontroll

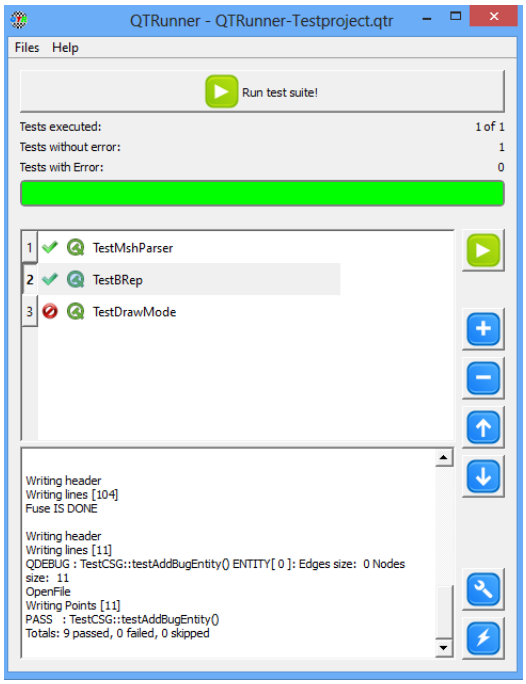
Revisjonskontroll er helt nødvendig ved systemutvikling, og det har vært utbredt bruk av revisjonskontroll gjennom utviklingen av både prosjektoppgaven og masteroppgaven. I prosjektoppgaven ble Microsofts Team Foundation Service brukt til revisjonskontroll og deling, men under utviklingen av denne applikasjonen var SourceTree med Git brukt. Både Team Foundation Service og Git har omtrent den samme funksjonaliteten og er regnet som avtakere etter SVN. Fordelen er størst ved samarbeid i store grupper, men det skal ikke legges skjul på at det kan være veldig fordelaktig også ved kun èn bruker. Det har vært til stor nytte i dette prosjektet fordi man lett kan gå tilbake til en tidligere revisjon og kjøre programmet i et annet stadium for å lokalisere opprinnelsen til eventuelle feil.

2.3.4 Testing

Testdrevet utvikling er en vanlig praksis for systemutvikling i dag. I sin reneste form betyr det at en gruppe utviklere først blir enige om hvilke egenskaper et program skal ha. Deretter blir det skrevet tester for programmets egenskaper. Når egenskapene er implementert kan de kjapt verifisieres ved hjelp av testene. Noen foretrekker å skrive testene etter at de har begynt å skrive koden. Uavhengig av om testene skrives før eller etter at egenskapene er

implementert, gir testdrevet utvikling store fordeler. Risikoen for å ødelegge kode en annen person har skrevet reduseres ved at testen i så fall vil avdekke feilen umiddelbart. Refaktorering av kodebasen øker ryddigheten, som igjen fører til økt produktivitet og kvalitet på programvaren. I et større miljø kan også en testdrevet utvikling gi fordelen av automatisk testing og kompilering av programmet.

2.3.4.1 Testdrevet utvikling i prosjektet



Figur 2.1: Skjermbilde av testprosjekt brukt til automatisk testing av enkelte moduler i applikasjonen for bruk til feilsøking og for å gi kjapp tilbakemelding etter endringer.

Bruk av testdrevet utvikling i dette prosjektet var for å gjøre overgangen til andre utviklere lettere. Testingen har hovedsaklig foregått på klasser der det lett kan oppstå feil, og som kan bli testet uten noen form for brukerinput. Modulene som ble utviklet på denne måten var operasjoner for lesing og skriving samt boolske operasjoner som klipping og liming av geometriske figurer utført gjennom biblioteket fra OCCT.

På grunn av Qt sin oppbygging er det nødvendig å bruke Qmake for å gjøre om Qt-kodeord til C++ før kompilering. Det var derfor ikke mulig å benytte Visual Studio sin godt integrerte enhetstesting. I stedet har Qt et eget system, men som dessverre viste seg å være mer tilrettelagt for bruk i Qt sitt eget utviklingsverktøy. Det kan vise seg å være komplisert å videreføre, men det er likevel laget noen tester som kan benyttes. Videre bruk av testdrevet utvikling anbefales hvis testverktøyene i Visual Studio får bedre støtte for Qt.

Kapittel 3

Avhengigheter

Applikasjonen har brukt en rekke rammeverk og APIer. Qt blir blant annet brukt for å gi grensesnittet kryssplattformstøtte, VTK blir brukt for grafikkvisning og OCCT for modelleringsverktøy. De nevnte rammeverkene er alle store, og en gjennomgang av funksjonalitet samt oppbygning av disse rammeverkene anses derfor som nødvendig.

3.1 Qt

For å oppfylle kravet om kryssplattformstøtte og bruk av programmeringsspråket C++ var det nødvendig å benytte et rammeverk som bygger og tilpasser brukergrensesnittet til de ulike plattformene. Det er flere rammeverk som gjør denne oppgaven, men Qt skiller seg ut fordi den gjennom tidene har hatt god støtte for VTK. Rammeverket gir universal tilgang til en rekke plattformspesifikke funksjonaliteter, som filsystem, nettverk og prosesstyring. Det hjelper også at Qt er under utvikling og opplever en økende popularitet. For utvikleren gir Qt støtte for blant annet bruk av «Object Ownership» og «Signal & Slots» gjennom egen syntaks, som ved hjelp av en kodegenerator blir konvertert til ren C++ kode før kompilering på de ulike plattformene. «Signals & Slots» gjør at man kan utforme klasser med friere koblinger, og «Object Ownership» virker som en enkel form for «Garbage Collection» ved at Qt automatisk sletter et objekt når foreldreobjektet blir slettet. Qt gjør det også mulig å skille design av brukergrensesnitt ut i en egen definisjonsfil, noe som har vært mye benyttet i prosjektet. Ulempen er at programmene utviklet med Qt får mer «overhead» i forhold til om alt var skrevet i C++, men

likevel ikke nok til at det er av negativ betydning for denne applikasjonen.

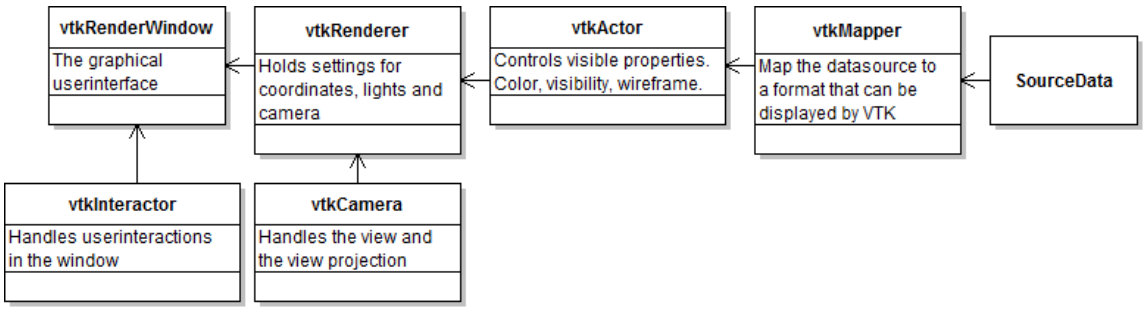
3.2 The Visualization Toolkit

The Visualization Toolkit er et open source rammeverk for visuell presentasjon av data som blir brukt i alt fra prosessering av medisinske bildedata til informatikk og geometriske problemer. Utviklingen av VTK startet på grunnlag av et ønske om å lage et system for 3D-visualisering av forskningsdata som ikke skulle være begrenset av lisenser og patenter. Resultatet er et portabelt rammeverk med støtte for flere programmeringsspråk og som i stor grad er utviklet av rammeverkets brukere. Fordelen med sistnevnte er at den har innebygd støtte for mange dataformater og gjør presentasjon av avansert data enkelt. Dessverre er denne allsidigheten også et stort problem for VTK. Rammeverket har flere lag med abstraksjoner, noe som krever at man har god kjennskap til arkitekturen før man kan tilpasse funksjonalitet. I tillegg varierer kvaliteten på dokumentasjonen. Den er veldig bra på de vanlige presentasjonene som i tillegg har mange eksempler, men kvaliteten blir fort kraftig redusert på for eksempel egendefinert brukerinteraksjon.

3.2.1 Rammeverkets oppbygning

VTK er bygget opp av flere kjerneklasser som tar for seg håndteringen av data fra en kilde og frem til den tegnes på skjermen. Dette gjøres ved at dataene først «mappes» som en geometri til et vtkobjekt som kan tegnes. Deretter settes dette vtkobjektet til en «vtkActor». Dette er den grafiske representeringen av geometrien. I «vtkActor» konfigureres blant annet farge, posisjon og synlighet. En eller flere «vtkActor» kan legges til i en renderer, som kontrollerer koordinatsystemet, lys og kamera/perspektiv. Til slutt vises det i renderen på skjermen gjennom «vtkRenderWindow». I likhet med de fleste klassene har «vtkRenderWindow» ytterligere klasser for å konfigurere blant annet hvordan interaksjonen skal behandles.

Dette representerer «pipelinen» til VTK - altså hvilke steg dataene går gjennom før de blir vist på skjermen, og er sentralt for å forstå hvordan deler av denne applikasjonen er bygd opp.



Figur 3.1: Oversikt over «pipeline» i The Visualization Toolkit. Kildedata mappes som en geometri og får gitt visuelle egenskaper gjennom en «vtkActor». Flere «vtkActor» kan legges til i en «vtkRenderer», som konfigurerer verdenen og hvordan gjenstandene blir tegnet. Til slutt legges renderen til et «vtkRenderWindow», som viser det på skjermen og håndterer interaksjon.

3.3 OpenCASCADE Technology

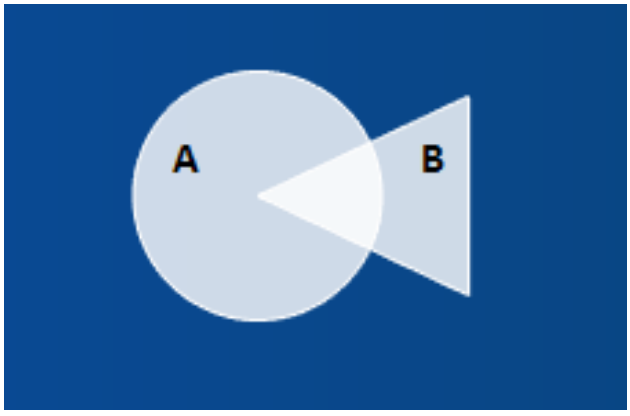
OpenCASCADE Technology (OCCT) er en open source 3D-modelleringsplattform utviklet av firmaet OpenCASCADE. Det er bygget opp av flere selvstendige kodebiblioteker, og det er kun et fåtall av bibliotekene fra OCCT som er brukt i denne applikasjonen; Det er hovedsakelig modelleringsalgoritmer og de nødvendige datastrukturene knyttet til disse. I applikasjonen brukes bibliotekene for å utføre boolske operasjoner, som å klippe eller lime overflater i entiteter. Det finnes flere programmer som også kan utføre boolske operasjoner, som er mindre komplekse enn OCCT (Se bruk av OCCT i seksjon 4.4.1). Det som likevel gjør at OCCT ble valgt, er at det er mye brukt og godt testet, og har ganske mye dokumentasjon. Utviklernes valg om å bygge opp programmet i mange relativt selvstendige biblioteker gjør det lett å plukke enkelte funksjoner. Samtidig gjør strukturen at det noen steder er redundans i koden, i tillegg til at enkelte datastrukturer ikke er kompatible. I applikasjonen er det veldig nyttig at rammeverket er delt opp på denne måten fordi man da enklere kan utvide bruken av OCCT, og implementere flere biblioteker hvis det skulle være ønsket. Det skal nevnes at det finnes ytterligere verktøy til modellering som kan integreres fra de allerede importerte bibliotekene fra OCCT.

3.3.1 Boolske operasjoner

Som nevnt brukes OCCT til å utføre boolske operasjoner. Med boolske operasjoner menes samme som algebra for mengder kjent fra matematikken. Formene kan beskrives som venndiagram, og operasjonene kan beskrives med matematiske operatorer. Operatorene for boolske operasjoner er «NOT», «UNION», «AND», «INTERSECTION» og «XOR».

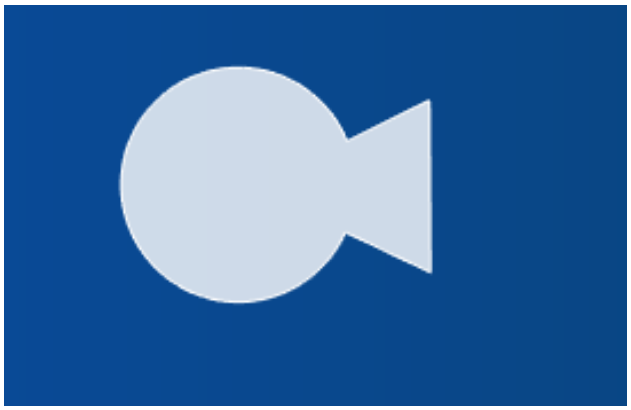
«NOT» og «UNION» er mest vanlige å bruke i modellering, og tilsvarer henholdsvis operasjonene «Cut» og «Fuse». Disse var derfor ansett som viktigst og ble implementert først. Figur 3.2, 3.3 og 3.4 illustrerer resultatet av operasjonene.

Før enhver operasjon starter brukeren med to former. Deretter velges operasjonen «A NOT B», vist i figur 3.3, der A og B er formene. Resultatet viser formen til A der formen til B er kuttet ut. Tilsvarende i figur 3.4, der operasjonen A UNION B er utført. Her er formene A og B limt sammen uten overlapp.



Figur 3.2: To overlappende former før utført boolsk operasjon.

Figur 3.3: En form etter utført boolsk operasjon «NOT». Her er overflaten fra den andre formen subtrahert fra den første.



Figur 3.4: En form satt sammen av to former etter utført boolsk operasjon «UNION». Her er overflaten fra begge formene addert sammen til èn overflate uten overlapp.

3.4 Gmsh

Applikasjonen benytter programmet Gmsh for å generere mesh av tverrsnittet. Gmsh har stor mulighet for konfigurasjon i tillegg til en bred støtte for filformater. Modelleringsfunksjonaliteten er begrenset til enkle geometriske figurer, og utvikleren av Gmsh anbefaler bruk av et CAD-program for å modellere avanserte former. Modelleringen i Gmsh har likvel vært nyttig, og har gjentatte ganger vært brukt for å lage enkle filer brukt til testing. I tillegg til å være et selvstendig program med et eget brukergrensesnitt kan Gmsh startes gjennom en konsoll. Den godtar flere kommandoer for å spesifisere filer og kontrollere generering av meshet. Dette er en veldig stor fordel og gjør at Gmsh kan opereres som en ekstern prosess, og styres fra den utviklede applikasjonen.

Det at Gmsh kjøres som en ekstern prosess fremfor som en integrert del av applikasjonen kan være en fordel ved at applikasjonen i nåværende tilstand er friere knyttet til valg av mesher. Det vil nemlig være mye lettere å bytte den ut ved en senere anledning.

3.5 Lisenser

Ved bruk av rammeverk er det nødvendig å ta hensyn til kopibeskyttelse og rett bruk. De nevnte rammeverkene er lisensiert under enten LGPL eller BSD - med unntak av Gmsh som er lisensiert under GPL. Lisensiering kombinert med hvordan rammeverkene er brukt medfører at applikasjonen ikke tvinges til åpen kildekode. Den kan dermed distribueres lukket, noe det har vært uttrykt et ønske om. Se appendiks A for utdypende informasjon om lisensene som er benyttet.

Kapittel 4

Implementasjon av programmet

4.1 Valg av programmeringsspråk

Programmeringsspråket C++ var et naturlig valg for solveren, til dels på grunn av dets hastighet og støtte for biblioteker for å utføre matematiske operasjoner. Det var ønsket at programmene i pakken skal bruke det samme språket for å unngå at brukeren er nødt til å installere støtte for java. Bruken av C++ i begge modulene gjør at prosjektet fremstår som mer oversiktlig for fremtidige utviklere, og det åpner for å kunne dele kode mellom prosjektene. Forfatteren har tidligere kun hatt fag om objektorientert Java, ikke C++, og så på det som en mulighet for å utvide sine programmeringskunnskaper.

4.2 Systemarkitektur

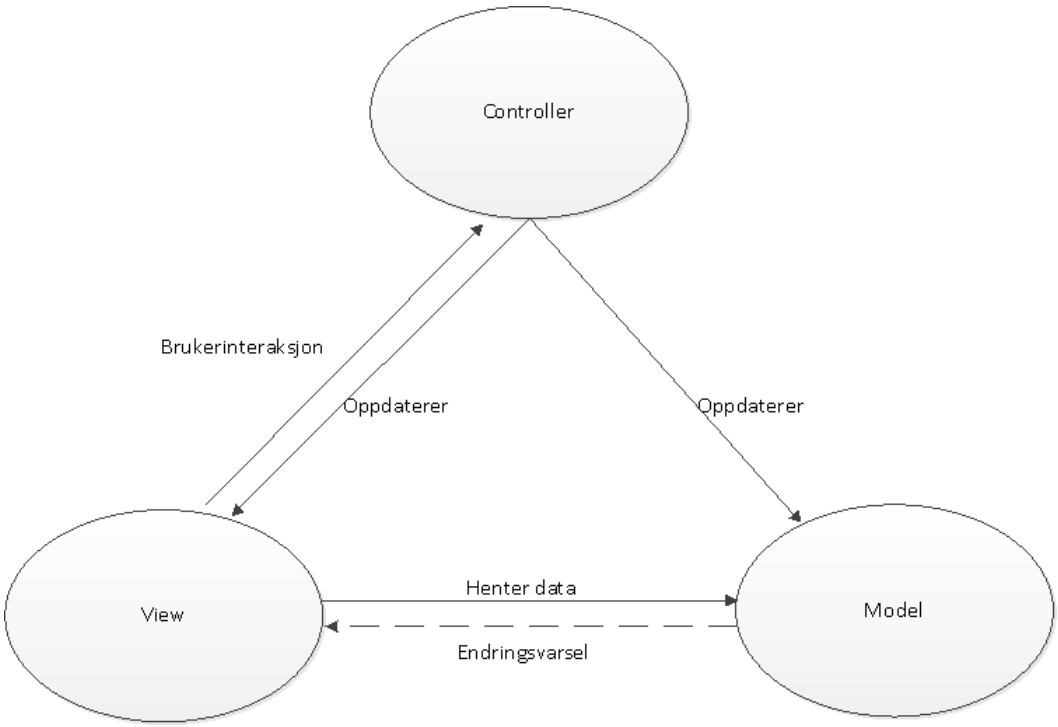
Det ble lagt ned mye arbeid i å lage en fleksibel systemarkitektur som støtter utvidelser, og som utnytter egenskapene i valgte rammeverk. Grunnet programmets interaksjonsorienterte natur var det å foretrekke å velge en arkitektur som klart skiller design fra logikk.

4.2.1 MVC - Model-View-Controller

Det mest utbredte mønsteret for interaksjonsorienterte programmer er «Model-View-Controller(MVC)» vist i Figur 4.1, og varianter som bygger på det, som Microsofts «Model-View-ViewModel(MVVM)». Felles for dem er at de

strukturerer programmet i tre deler. Design, programmets overflate, skal ligge lagret i et «View» og skal være separert fra koden i så stor grad som mulig. Det andre kravet er at data som vises til brukeren skal ligge lagret i et eget objekt kalt en «Model». Tilslutt er det en «Controller» som kobler sammen «View» og «Model». Figur 4.2 gir en oversikt over hvordan MVC er implementert i applikasjonen på et overordnet nivå.

Kontrolleren inneholder logikken som driver applikasjonen. Den reagerer på hendelser og oppdaterer «Model» og «View» når en handling utføres. Fordelen med MVC er blant annet ryddighet. Det tilrettelegger også for at de ulike delene kan utvikles av personer med spesialisert kompetanse. Ved bruk av MVC-arkitektur i en applikasjon er det mulig å ha en designer som utelukkende spesialiserer seg på design og brukervennlighet. På denne måten er man ikke avhengig av at utviklerne har interesse og kompetanse både innen programmering og design. Qt oppfordrer til MVC arkitektur ved å definere i brukergrensesnittet i ui-filer. Disse er basert på XML, og fører til at de fleste views kan defineres helt uten kjennskap til C++.



Figur 4.1: Komponenter i MVC er Model, View og Controller. Modellen inneholder data i applikasjonen, mens Viewet definerer utseende og inneholder funksjoner for hvordan dataene blir presentert. Kontrolleren koordinerer de andre komponentene og oppdaterer view og model etter hendelser

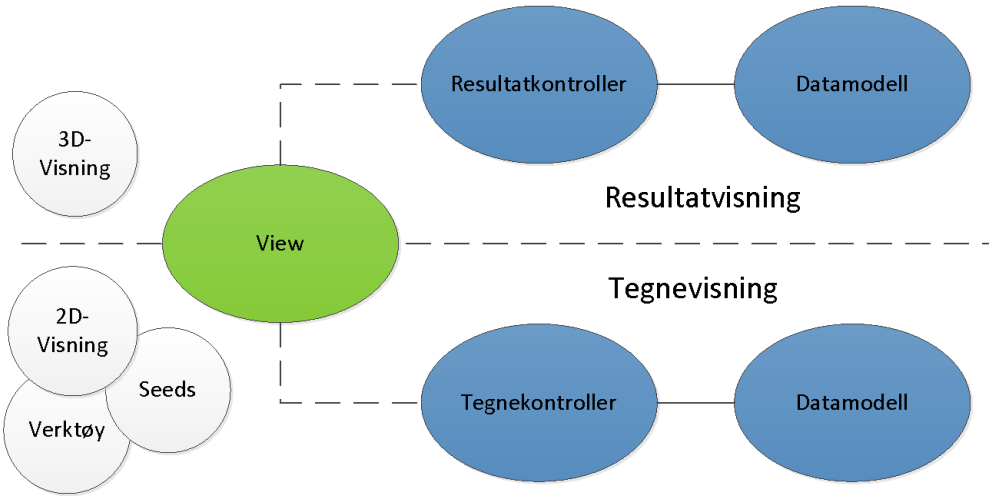
4.2.1.1 Observatørmønster

En sentral del av MVC er observermønsteret. Mønsteret går ut på at et objekt holder på en liste av observatører. Observatørene får beskjed hver gang det skjer en endring i objektet. Dette er implementert i MVC ved at «View» legger seg til som en observatør i «Model», og oppdateres når data endres. Dette tillater at programmet kan ha flere views som oppdateres automatisk ved en endring i modellen. Qt gjør det enkelt å implementere dette, men gjør det i en litt annen form. Qt bruker «Signals» og «Slots» som gjør at enhver klasse kan bli observert eller virke som en observatør. Enhver klasse kan dermed sende ut signaler og et signal kan kobles til flere slots.

4.3 Implementasjon av arkitektur

Som vist i Figur 1.1 inneholder applikasjonen to tilstander. Den ene har funksjonen til en preprosessor og den andre til en postprosessor. Byttet mellom tilstandene er implementert slik at ikke tunge GUI-komponenter som VTK blir byttet ut. VTK er en fast «widget» som viser grafikk, og det er i stedet valgt å kun bytte ut kontrolleren som aktiverer og deaktiverer innhold og funksjonalitet. Det er en litt utradisjonell implementering av MVC, men reduserer tiden betraktelig for å bytte mellom visningene.

På grunn av stor forskjell på data i tegnevisning og resultatvisning er det valgt å bruke en datamodell for hver tilstand. Tilstandene og kontrollene er veldig forskjellige, og blir derfor gjennomgått hver for seg.



Figur 4.2: Oversikt over MVC implementert i applikasjonen. Blå indikerer modeller og kontrollere som blir erstattet ved visningsbytte. «GraphicsView» i grønn, blir stående fast. Helt til venstre i grå, er egenskaper som aktiveres eller deaktiveres i «GraphicsView».

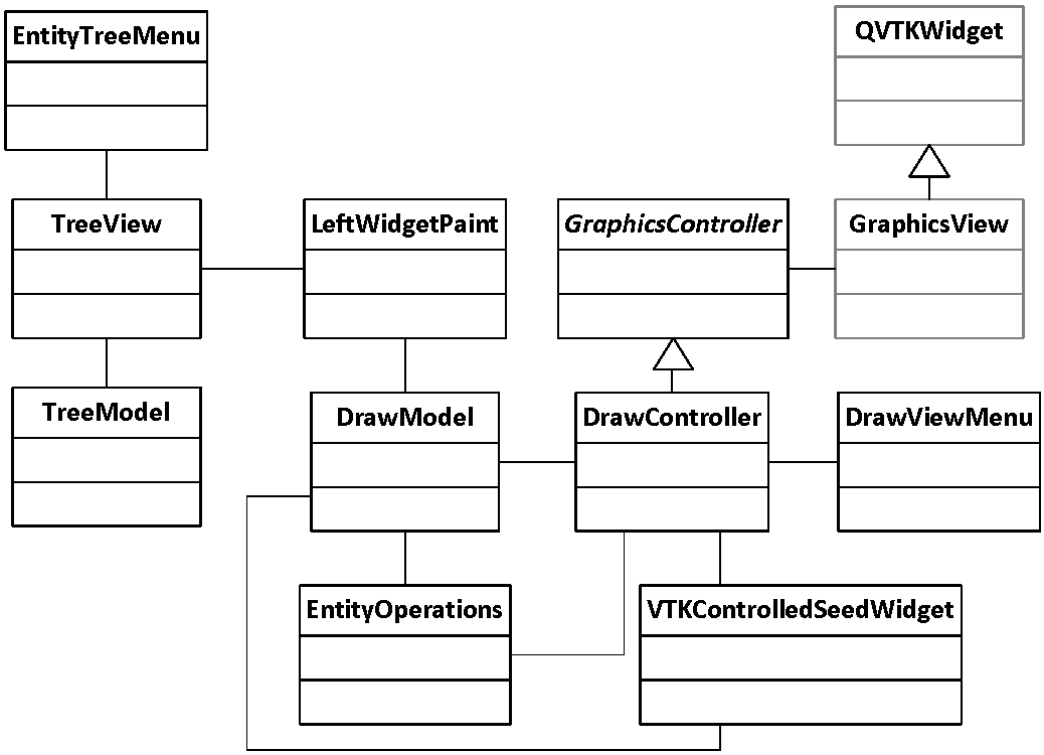
4.3.1 Tegnevisning

Tegnevisning er programpakkens preprosessor. Visningen har som oppgave å la brukeren modellere tverrsnitt ved blant annet å tegne, endre og slette entiteter. Antallet mulige handlinger i denne tegnevisningen gjør den omfat-

tende, og visningen er derfor delt opp i flere klasser. Klassene er delt opp etter funksjonalitet og oppgave, og skal i så stor grad som mulig være begrenset til den. Dette er for å gjøre «kontrolleren» mer oversiktlig, ved at en utvikler slipper å måtte lete gjennom store klasser. Grunnet antallet klasser i tegnevisningen vil den bli illustrert gjennom de to klassediagrammer, i henholdsvis Figur 4.3 og Figur 4.4.

«DrawController» setter visningsmodus for grafikk i «GraphicsView» og lager en «VTKControlledSeedWidget». «VTKControlledSeedWidget» utvider VTK sin «vtkSeedWidget» og håndterer brukerinteraksjon med noder. «vtkSeedWidget» manglet kontroll til å kunne styre om det kun skulle være mulig å flytte på en eksisterende node, eller også legge til nye. Det var derfor nødvendig å utvide klassen med slik funksjonalitet.

«DrawController», «VTKControlledSeedWidget» og «LeftWidgetPaint» er koblet til «DrawModel» og henter eller endrer data direkte i datamodellen. «GraphicsView» utvider «QVTKWidget» med forhåndsdefinerte egenskaper som kan velges av kontrollerne for å unngå redundant kode i kontrollerne.

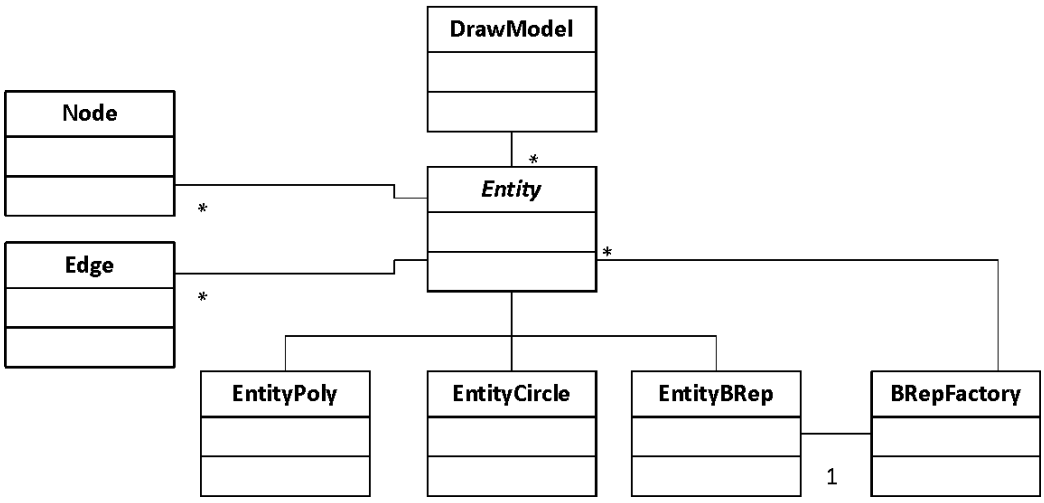


Figur 4.3: Forenklet klassediagram for klasser benyttet i tegnevisning. «DrawController» oppdaterer «GraphicsView» og datamodellen. Modellen er koblet til «TreeWiew» med entiteter og oppdateres når data endres i modellen. Dataobjektene som holder entitetsdata er ekskludert fra dette diagrammet for ryddighet.

Tegnemodellen inneholder data om flere entiteter. Entitetene skal kunne representere alt fra enkle figurer til avanserte, sammensatte former. De er implementert slik at flere figurerer kan forhåndsdefineres uten å måtte gjøre store forandringer på eksisterende kode. Data om formens noder og kanter lagres i en abstrakt klasse kalt «Entity». Alle entiteter som skal kunne manipuleres i brukergrensesnittet arver egenskaper fra denne, og utvider basisfunksjonaliteten.

En type entitet skiller seg litt ut. Den er lagt til for å kunne utføre en mer avansert form for modellering. Denne entiteten blir kalt «BRepEntity», der BRep står for «Boundary Representation». Navnet har den fått fordi den er

sammensatt av flere underliggende entiteter og benytter formene samt boolske operasjoner for å definere en overordnet form. Entiteten holder en referanse til «BRepFactory» for å kunne behandle formene og lage nye entiteter.



Figur 4.4: Klassediagram for klasser tilknyttet entiteter i tegnetilstand. Entiteter ligger lagret i «DrawModel» og inneholder data om noder og kanter til formen den representerer. Videre må en entitet representere et polygon, en sirkel eller en «Boundary Representation» bestående av flere former. Sistnevnte bruker BRepFactory til å utføre operasjoner på entitetene.

4.3.1.1 Sekvenser i tegnevisning

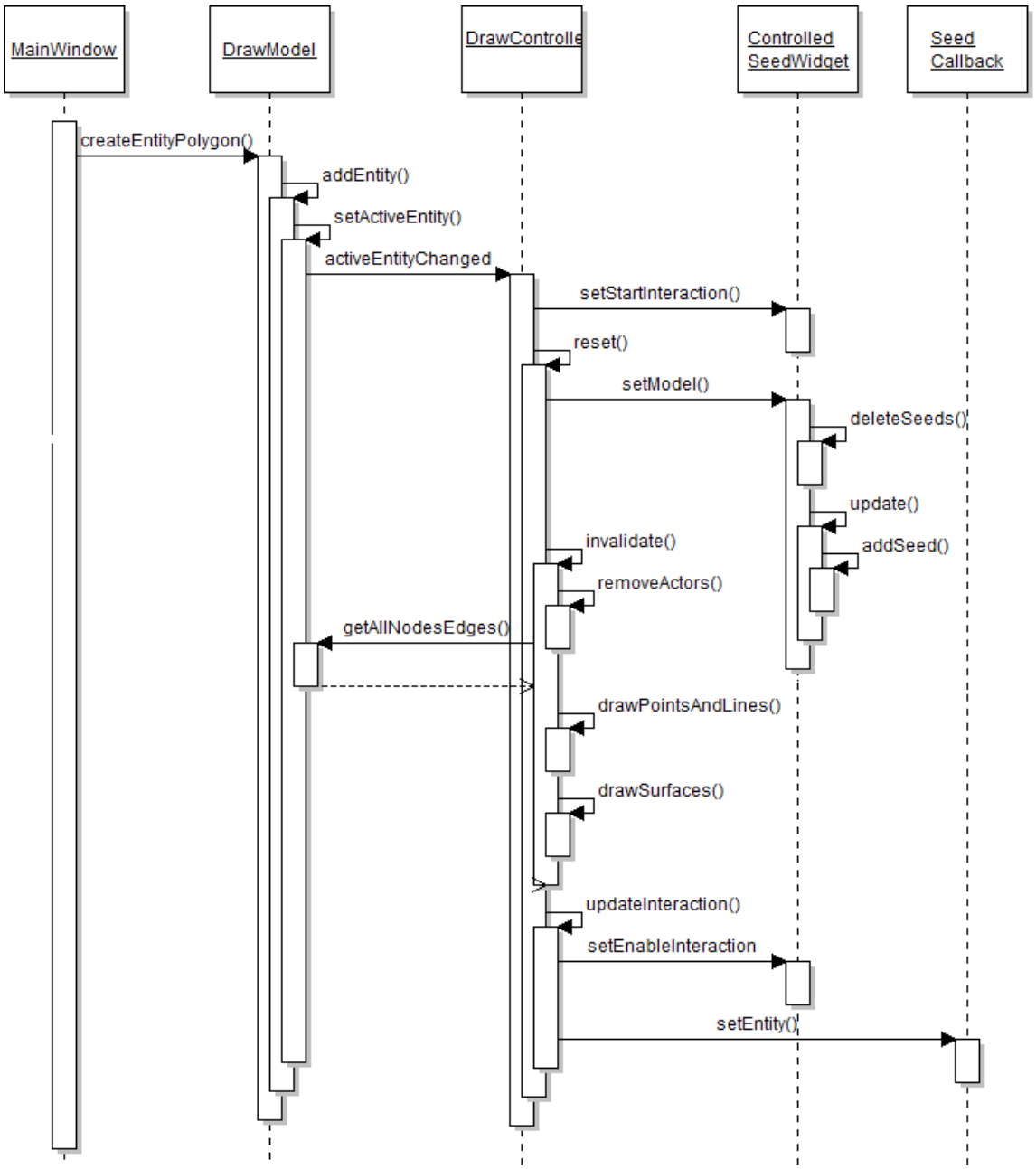
Hendelsesforløpet gitt tre ulike brukeroperasjoner blir her gjennomgått for å vise sammenhengen mellom klassene i tegnemodus. De er illustrert ved bruk av sekvensdiagrammer, som viser kall mellom klassene. På grunn av størrelsen og mengden av detaljer ble noen klasser utelatt fra sekvensdiagrammet. De skal likevel gi en god oversikt over hendelsesforløpet i programmet.

Når brukeren trykker på knappen for en ny entitet, håndteres det av «MainWindow» som videre kaller «DrawModel». (Se figur 4.5). «DrawModel» lager et nytt entitetsobjekt og sender ut signal. Signalet fanges opp av «TreeModel» og «DrawController», som oppdaterer visningene. «DrawController» setter «VTKControlledSeedWidget» til å håndtere interaksjon og oppdaterer den med eventuelle noder som skulle være entiteten. Deretter invalideres

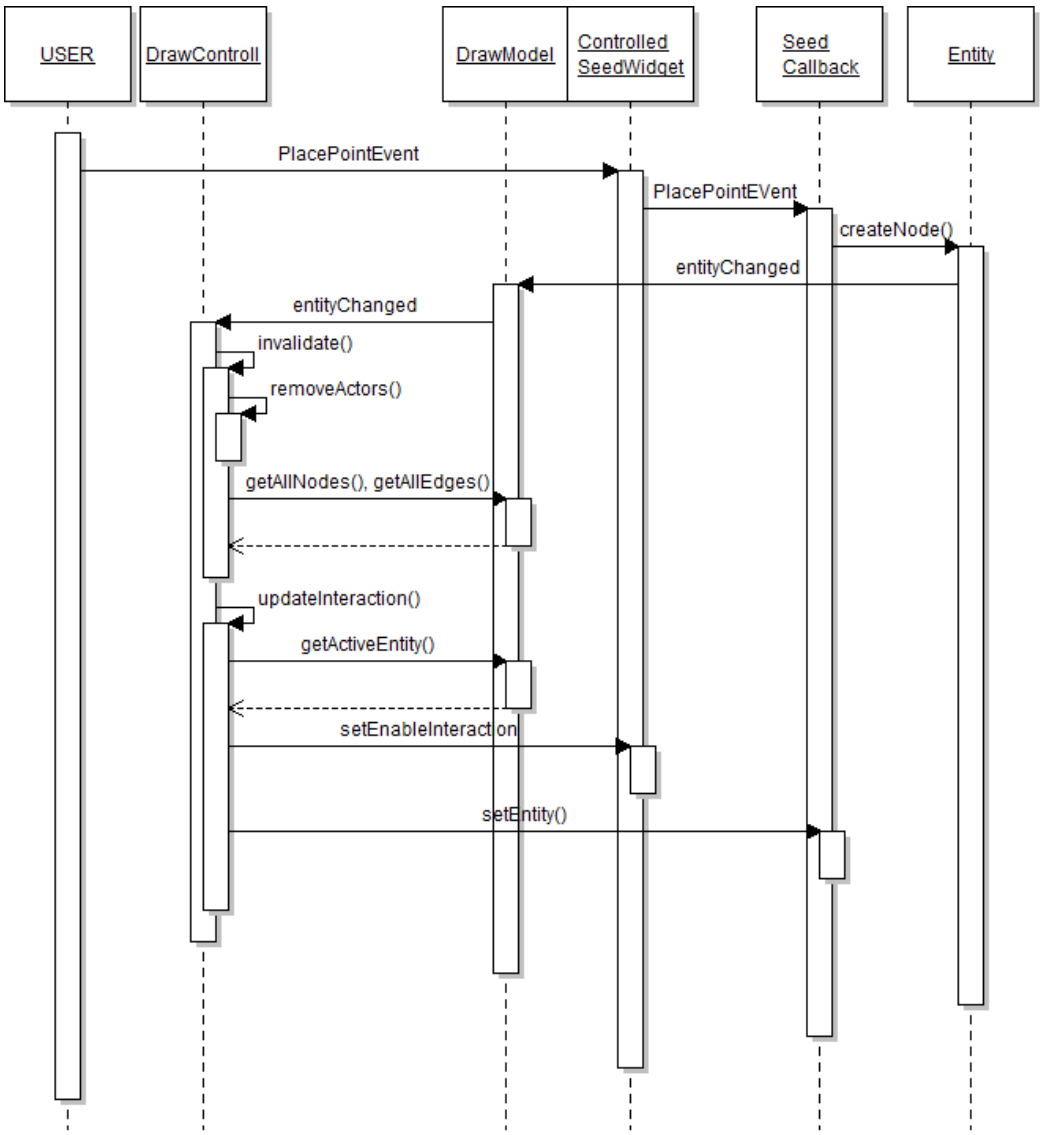
«GraphicView», og «DrawController» tegner kantene og overflatene til entitetene. En entitets overflate blir gitt en gjennomsiktighet etter om den er valgt, redigerbar, vanlig eller et hull. Til slutt blir entiteten satt i datakilden til «VTKControlledSeedWidget» slik at brukeren kan legge til noder i entiteten.

Etter at en entitet er lagt til og «VTKControlledSeedWidget» har aktivert interaksjon, går all brukerinput i grafikkvinduet direkte til den. Når en ny node legges til, kalles entiteten, og det legges til en ny node (Se figur 4.6). Deretter sender entiteten ut et signal om at data har blitt endret. Signalet er koblet til modellen og videre til «DrawController», som fører til at «GraphicView» invalideres og oppdateres.

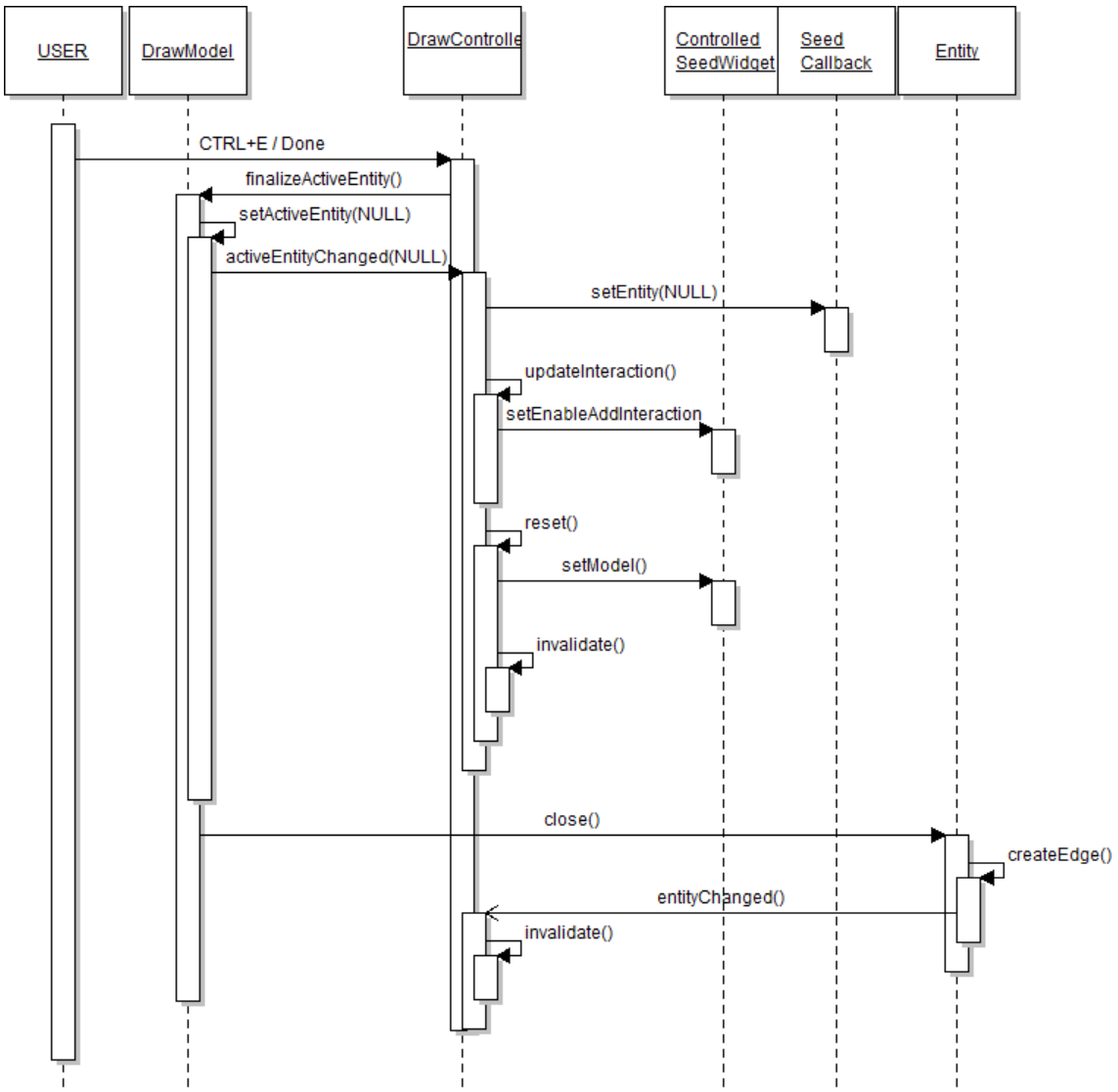
Figur 4.7 viser hendelsene når en entitet fullføres ved at brukeren trykker «Done». Det sendes da et signal til «DrawController», som kaller «finalizeActiveEntity» i modellen. Modellen sender deretter ut et signal om at det ikke er valgt noen entitet. Signalet er koblet til «DrawController», som deaktiverer interaksjonen til «VTKControlledSeedWidget», og oppdaterer modellen slik at nodene forsvinner fra skjermen. Modellen kaller deretter «finalize» på entiteten, noe som gjør at den generer den siste kanten slik at entiteten blir sammenkoblet. Til slutt invalideres grafikkvinduet og entitetene blir tegnet.



Figur 4.5: Sekvensdiagram av hendelser i systemet for operasjonen «Ny entitet». Ved denne hendelsen lager modellen en ny entitet, som blir satt som valg entitet. «DrawController» oppdaterer da grafikkvinduet, og aktiverer «VTKControlledSeedWidget» for brukerinput. Entiteten blir til slutt satt som datakilde for nodeinteraksjonen i «SeedCallback».



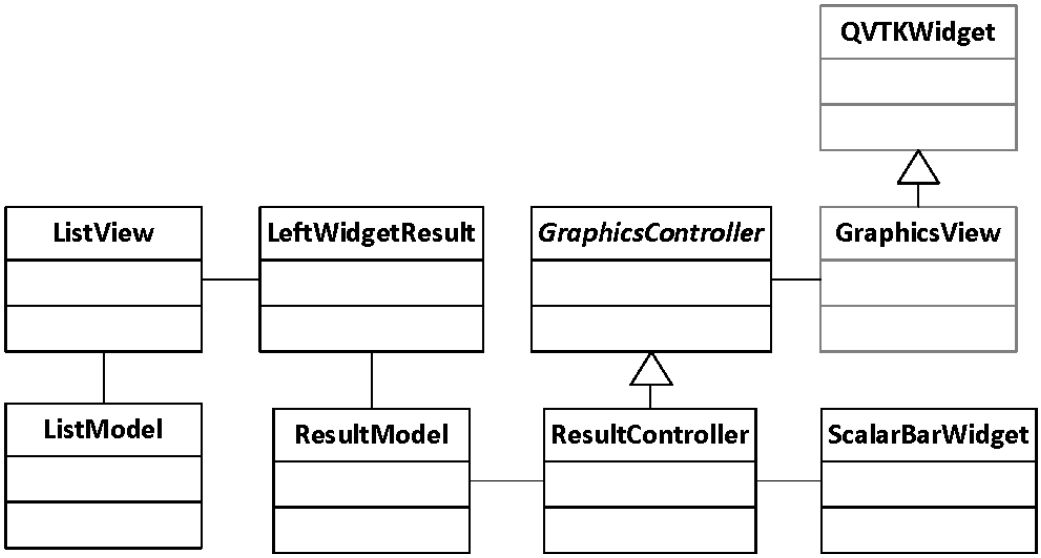
Figur 4.6: Sekvensdiagram av hendelser i systemet for operasjonen «Legg til node». Når en bruker trykker i grafikkvinduet fanges det opp av «VTKControlledSeedWidget», og videre av «vtkSeedCallBack». Det legges deretter til en node i entiteten, som sender ut et signal om at den er endret. Signalet er koblet opp til «DrawController», som oppdaterer grafikkvisningen.



Figur 4.7: Sekvensdiagram av hendelser i systemet for operasjonen «Fullfør tegning». Når brukeren trykker done, kalles modellen som ferdigstiller entiteten.

4.3.2 Resultatvisning

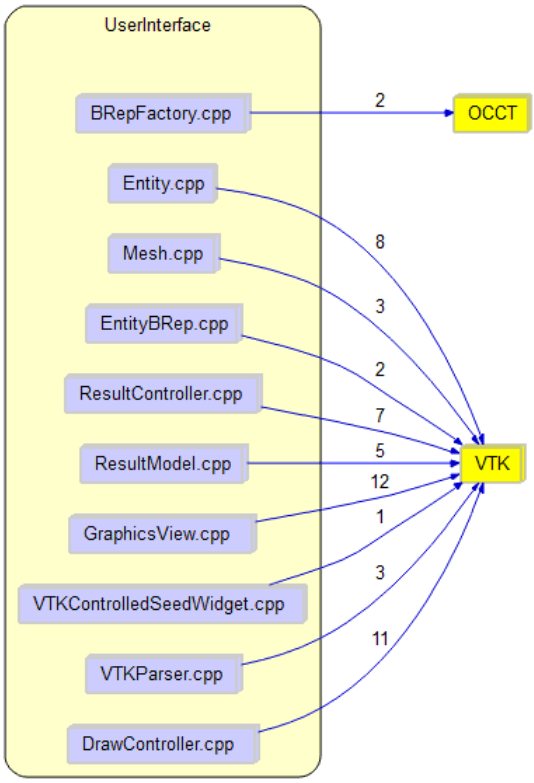
Resultatvisningen er applikasjonens postprosessor. Den viser mesh og resultater, og lar brukeren velge hvilke resultater som skal vises fra en liste. VTK er veldig god på nettopp å visualisere geometrisk data, og visningen trengte derfor ikke en omfattende implementasjon. «ResultController» benytter også de forhåndsdefinerte egenskapene i «GraphicsView» for å styre grafikkpresentasjonen. «ResultController» og «ListView» er koblet til resultatmodellen, og oppdateres automatisk når en datafil lastes inn.



Figur 4.8: Klassediagram for klasser benyttet i resultatvisning. «ResultController» oppdaterer «GraphicsView» og datamodellen. Modellen er koblet til «LeftWidgetResult» som består av en liste med resultater, som oppdateres med endringer i datamodellen. Dataobjektene er ekskludert fra dette diagrammet.

4.4 Bruk av APIer

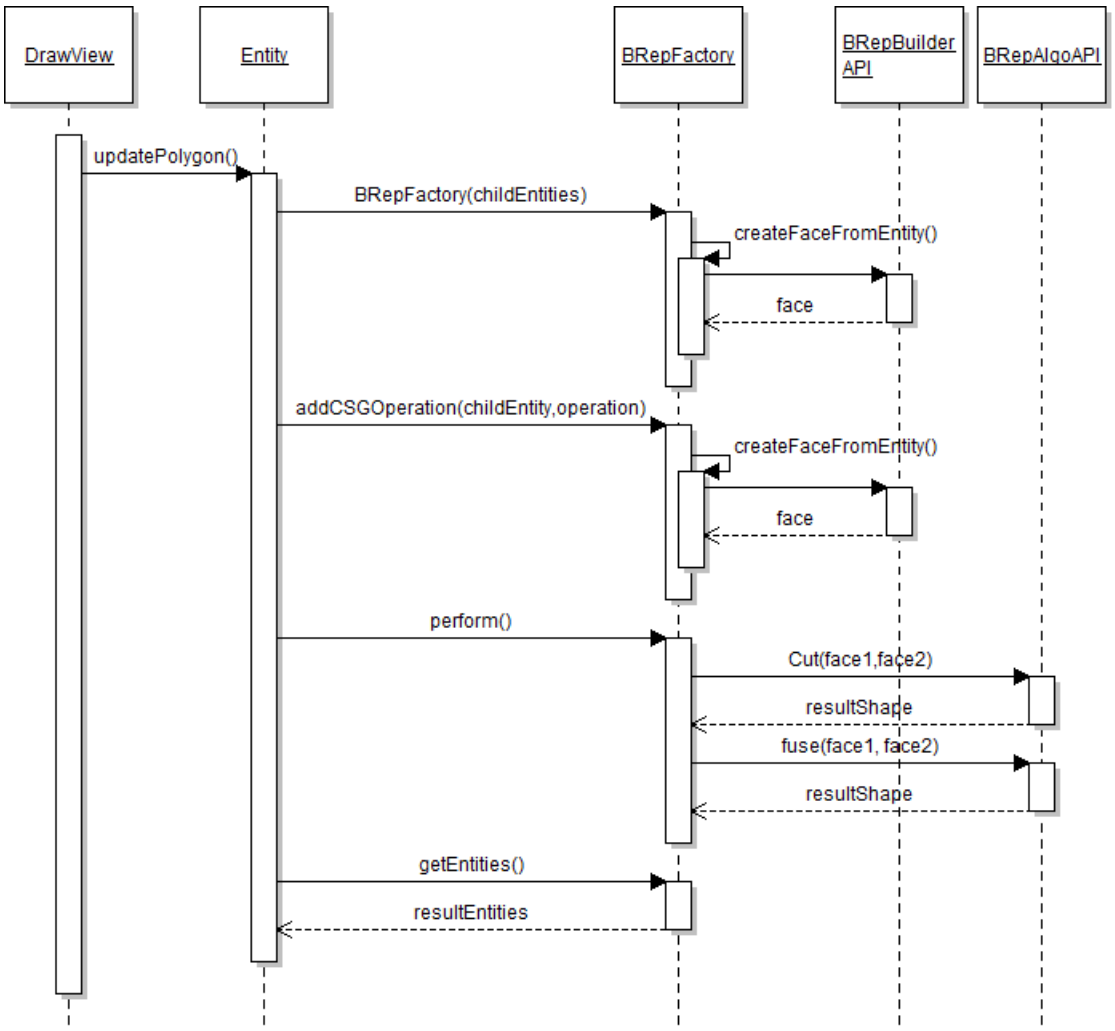
Applikasjonen er sterkt avhengige av to rammeverk, som tilfører nødvendig funksjonalitet. Rammeverkene heter OCCT og VTK, og har tidligere blitt introdusert henholdsvis i Seksjon 3.3 og 3.2. De er begge store rammeverk, og omfang av bruk og implementasjonen vil bli gjenomgått nedenfor.



Figur 4.9: Bruk av APIer i applikasjonen. OCCT kalles kun gjennom BRepFactory. VTK brukes til visualisering, og kalles fra alle klassene som enten vises på skjermen, eller styrer visningen.

4.4.1 OpenCASCADE Technology

OpenCASCADE Technology beskrevet i seksjon3.3, er koblet til applikasjonen gjennom bruk av EntityBRep (Se figur 4.9) og brukes ved oppdatering av entitetets form. BRepFactory gjør om noder og kanter i de underliggende entitetene til dataobjekter brukt i OCCT. Sekvensen starter når tegnekontrolleren invalideres, og kaller metoden «updatePolygon» på alle entiteter. EntityBRep, som er sammensatt av flere entiteter, starter da en instans av BRepFactory og sender sine underliggende entiteter og operasjoner til den. I BRepFactory blir data i entitetene konvertert til datastrukturer som brukes i OCCT. Når alle entitetene er konvertert til flater, kalles «perform» som går gjennom operasjonene og limer eller kutter flatene. Dette skjer ved at den resulterende flaten etter en operasjon brukes som input i den neste. En kutteoperasjon kan føre til at en flate blir splittet til flere mindre flater. Det fører til at BRepFactory kan inneholde en eller flere former som representerer resultatet etter operasjonene. Disse konverteres tilbake til entiteter i BRepFactory og sendes tilbake til BRepEntity, som representerer flatene som èn entitet. Det er også mulig at en kutteoperasjon resulterer i en flate med hull. I det tilfellet skiller BRepFactory ut denne flaten og definerer den i BRepEntity som et hull. Se figur 4.10 for sekvenser utført ved oppdatering av formen i BRepEntity.



Figur 4.10: Sekvensdiagram som viser bruk av OCCT for å utføre boolske operasjoner. Når en «BRepEntity» blir oppdatert sender den alle de underliggende entitetene den består av samt et sett av operasjoner til BRepFactory. BRepFactory går så gjennom entitetene og gjør dem om til OCCT-objekter og lar OCCT utføre operasjonene. Når alle operasjonene er utført bygger BRepFactory en eller flere entiteter basert på flaten returnert fra OCCT.

Konvertering av datastrukturer

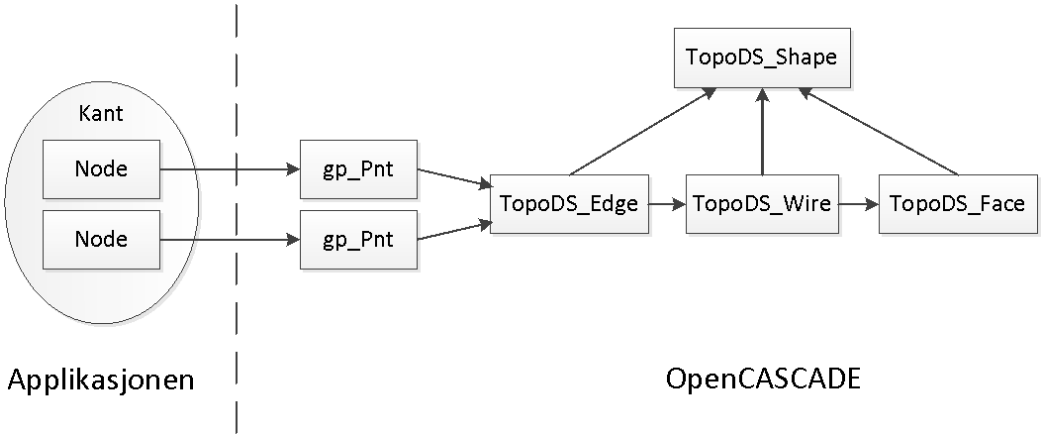
Konverteringen av datastrukturer skjer i BRepFactory, ved hjelp av OCCTs BRepBuilderAPI. BRepFactory går gjennom alle kanter i entitetene og henter ut nodene. Nodene gjøres så om til objektet gp_Pnt i OCCT og legges til i en TopoDS_Edge. Når alle kantene i en entitet er konvertert, legges de sammen til en TopoDS_Wire. TopoDS_Wire blir deretter brukt til å definere TopoDS_Face. Både TopoDS_Edge, TopoDS_Wire og TopoDS_Face arver fra TopoDS_Shape, som gjør at hver av dem kan brukes alene som en form. Se figur 4.11.

For å tillate å lage mer komplekse former har en TopoDS_Shape mulighet for å bestå av flere former. Dette er nyttig for OCCT og helt nødvendig for effektivitet i 3D-modellering, men viste seg å skape problemer for bruk i denne applikasjonen. Resultatet etter en limeoperasjon skulle kun ha vært en TopoDS_Face, men var i stedet en TopoDS_Shape bestående av tre TopoDS_Face. Dette ble løst ved gå gjennom kantene i de tre formene og kun velge de ytterste kantene. Geometriske operasjoner i OCCT har vist seg å enkelte ganger gjøre feilberegninger, men på grunn av tidsbegrensningen var det ikke mulig å lokalisere årsaken.

4.4.2 The Visualization Toolkit

The Visualization Toolkit er integrert med applikasjonen gjennom en egen «Widget» i Qt-Designer, og applikasjonen er helt avhengig av VTK. All interaksjon skjer rundt «QVTKWidget», og den blir brukt til å både tegne figurer samt å visualisere mesh og resultater.

VTK var et klart valg når det kommer til visualisering av resultater. Den gjør de fleste beregninger selv og krever lite kode for å presentere de geometriske datafilene brukt i denne applikasjonen. Den tar for seg lesingen av vtk-filer, og vet hvordan den skal håndtere disse. Til brukerinput var det derimot litt mer problematisk. VTK har støtte for input i form av «vtkSeedWidget», men den viste seg å være begrenset og det var nødvendig å legge til noe funksjonalitet for å kunne bruke denne. Til tross for den økte kompleksiteten ved bruk av VTK til brukerinput kan det i fremtiden være en fordel fordi det relativt lett kan legges til verktøy for å tegne for eksempel kurver og ellipser.



Figur 4.11: For å bruke algoritmene i OCCT må datastrukturene i applikasjonen konverteres. Det gjøres kantvis for hver entitet, og en entitet tilsvarer et TopoDS_Face. Først konverteres nodene i hver kant til gp_Pnt og legges til i en TopoDS_Edge. Flere TopoDS_Edge inngår i en TopoDS_Wire, som igjen kan brukes for å definere et TopoDS_Face. Både TopoDS_Edge, TopoDS_Wire og TopoDS_Face er typer av TopoDS_Shape.

4.5 Datautveksling

Datautveklsing med de andre prosessene skjer gjennom lagring til filer. Data skrives til fil av brukergrensesnittet, som også åpner filen i enten mesher eller solver. Datautvekslingen skjer i klassen CModel, hovedmodellen i applikasjonen. CModel styrer de andre prosessene gjennom ExternalProcessHandler, og er ansvarlig for å hente ut data fra DrawModel og ResultModel og sende dataen til en passende filskriverklasse. Filskriverklassene er strukturert slik at de arver fra en abstrakt klasse. Den abstrakte klassen tar seg av opprettelse av filen og inneholder funksjoner som forenkler lagring av data. Tanken bak dette er å forenkle implementasjon av flere filformater ved at nye filskrivere i størst mulig grad kun skal inneholde definisjon av det nye filformatet.

4.5.1 Filformater

Applikasjonen benytter tre filformat spesifisert i appendiks B for å lagre data og for å utveksle informasjon mellom de ulike prosessene. To av dem baserer seg på filformat definert av Gmsh, og inneholder henholdsvis Geometri- og

meshdata. Det siste inneholder resultater og følger formatering definert i VTK Legacy polygondata.

4.5.1.1 Geo

Geometridata i applikasjonen lagres i et format med navn geo. Formatet leses av mesher og er Gmsh sitt standardformat for geometridata. Geoformatet har støtter for flere definisjoner av geometriske figurer, noe som gjør lagringen av dem kompakt. Grunnet applikasjonens bruk av Gmsh som kun mesher, benyttes kun et fåtall av disse. I stedet lagres det meste av dataene som noder og kanter. Ved senere anledning kan det være aktuelt å benytte flere av de geometriske definisjonene i filen for å lagre mer nøyaktige geometridata. Se appendiks B.1 for spesifikasjon av formatet brukt i datautvekslingen.

4.5.1.2 Msh

Meshdata fra Gmsh lagres i et format kalt msh, og leses av både solver og denne applikasjonen. Det er Gmsh sitt standardformat for meshdata, og var derfor et naturlig valg også for programpakken. Msh-filformatet bruker et strukturbasert format med mulighet for å sende tilleggsvariabler som materialspesifikasjoner i figurfilene. Det er ikke mye brukt utenfor Gmsh, men filformatet dekker godt kravene definert så langt. Ved bruk av en annen mesher kan det være verdt å også vurdere andre formater for lagring av meshdata. Delen av Mshformatet brukt i programpakken er spesifisert i appendiks B.2.

4.5.1.3 VTK

Resultatdata i programpakken lagres i et format beskrevet av VTK. VTK har flere formater som alle er laget med tanke på visualisering av resultatdata. I programpakken ble VTK Legacy - PolygonData valgt fordi det er strukturbasert og mellomromsseparert. I utgangspunktet kan det virke korttenkt å gå for et format stemplet som utdatert, men på grunn av mellomromsseparasjonen er det mye lettere for mennesker å feilsøke enn XML. Det skal legges til at denne applikasjonen støtter både VTK sine nye og gamle formater for dataset, og er derfor uberørt av eventuelle senere valg om å gå for et nyere VTK-format. Solveren og datautvekslingen med den støtter kun VTK Legacy, og den vil trenge en liten utvidelse. Se appendiks B.3 for spesifikasjon av VTK Legacy.

4.6 Mindre Implementasjoner

Det har blitt implementert en rekke mindre funksjoner, som løser problemer i fremvisningen og gir brukeren flere verktøy å jobbe med. Nedenfor følger en forklaring av noen av de viktigste mindre implementasjonene.

4.6.1 Dybdeplassering og figurer med hull

Det viste seg tidlig at det var nødvendig å ta i bruk z-aksen(dybde) for å kunne implementere ønsket funksjonalitet. Tegnevisningen i programmet bruker gjennomsiktige entiteter, og lar brukeren velge figurer og noder direkte i vinduet. I startet var det bare noder (seeds) som lå i et øvre plan. Etter hvert som entiteter fikk egenskaper som gjennomsiktighet og hull, viste det seg at VTK ikke klarte å regne ut gjennomsiktighet av flere «vtkActor» i samme plan. For å kunne gjennomføre det var det nødvendig å separere alle entitene fra hverandre langs z-aksen.

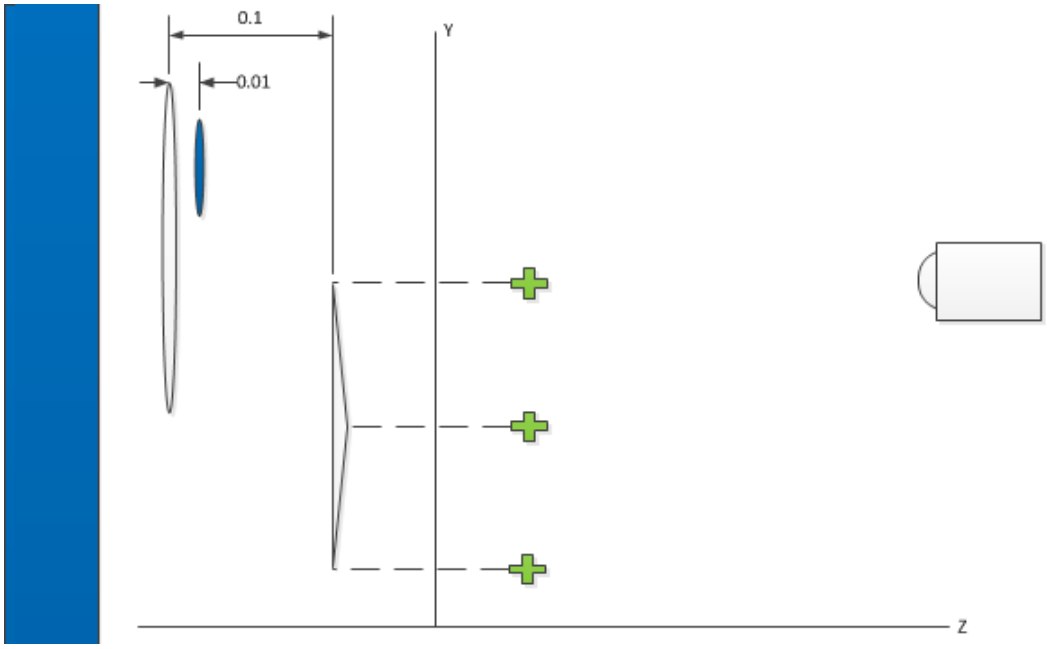
Hver entitet får angitt et tall når det blir laget som representerer rekkefølgen de skal tegnes langs z-aksen. Avstanden mellom hver entitet er satt til 0.1.

Det viste seg også at «vtkPolygon», som er brukt til å vise figurene, ikke støtter polygoner med hull. En flate med hull består derfor av to entiteter - en vanlig entitet og en hull-entitet. Se figur4.12. Avstanden fra en entitet og mellom hver hull-entitet er satt til 0.01. Det gir totalt plass til 10 hull i hver entitet. Disse avstandene er definert statisk nå, men kan endres eller regnes ut dynamisk basert på antall hull. Det er brukt et kamera uten perspektiv når tegnemodus er aktivt, og den totale avstanden disse forskyvningene utgjør langs z-aksen har derfor ingen påvirkning på utseendet.

4.6.2 Entitetsoperasjoner

Det er laget verktøy for enkle boolske operasjoner på entitetene. Disse verktøyene krever at brukeren gjør flere handlinger sekvensielt for å aktiveres. Det kreves derfor et system som holder orden på hvilke valg som er gjort og rekkefølgen disse er foretatt i. Dette er implementert gjennom «EntityOperation». Klassen har som oppgave å følge med på klikk foretatt i «GraphicsView» når tegnemodus er aktivert.

Operasjonskontrolleren startes ved klikk på en entitet. Deretter lyttes det etter enten et operasjonsvalg eller klikk på en annen entitet. Operasjonen blir utført når både operasjonsvalg og klikk på en annen entitet er foretatt. Et klikk i «GraphicsView» som ikke treffer en entitet nullstiller operasjonskontrolleren.



Figur 4.12: Illustrerer bruk av dybde, med bakgrunnen i blå til venstre, og kameraet helt til høyre. «Seeds»(Grønne kryss) ligger på et eget nivå på positiv side av z-aksen. Alle entitetene er separert med 0.1, og legges bakover langs z-aksen mot bakgrunnen. Hull(Farget blå) er separert med 0.01 mellom hverandre og fra entiteten de hører til. De er farget med samme farge som bakgrunnen og gir dermed en illusjon av at entiteten har hull.

Kapittel 5

Brukergrensesnitt

Brukergrensesnittet er den flaten brukeren ser av programmet. Kvaliteten er helt avgjørende for at brukeren skal være villig til å bruke programmet, og brukervennligheten kan være avgjørende for om et program blir populært eller glemt. Brukervennlighet har derfor vært en vesentlig del av planleggingen av brukergrensesnittet. Arbeidet har gått på vurdering av brukervennlighet, interaksjonen i de ulike visningsmodusene og den totale arbeidsflyten fra modellering av tverrsnitt til ferdig resultat. Kapittelet vil først ta for seg generelle retningslinjer for godt design av brukergrensesnitt, for så å gå inn på hvordan disse er implementert i applikasjonen. Til slutt er det gjort en brukertest for å dokumentere brukernes oppfattelse av brukergrensesnittet, og for å kommentere de problemene som testen avdekket.

5.1 Brukervennlighet

Det å lage et brukergrensesnitt som er forståelig og effektivt for mennesker med ulik erfaring, har flere ganger gjennom tidene vist seg å være vanskelig. Mennesker har forskjellig oppfatning av begreper, og noe som er en opplagt kontroll i et grensesnitt av en bruker kan være avhengig av beskrivelse for å kunne forstås av en annen. Brukervennlighet defineres som "*the effectiveness, efficiency, and satisfaction with which specified users achieve specified goals in particular environments*" fra ISO9241[1]. I korte trekk betyr det at det er umulig å avgjøre brukervennlighet uten å teste det på alle brukere som kan tenkes å bruke programmet. Det innebærer også at en utvikler sjeldent klarer å lage et godt brukergrensesnitt uten tilbakemelding fra brukerne.

Det finnes heldigvis retningslinjer fra tidligere interaksjonsdesign som gir en god indikasjon på hvordan brukergrensesnitt bør utformes. Retningslinjene beskriver blant annet menneskelig psykologi, og hvordan man på best mulig måte kan utnytte de begrepene enhver bruker allerede har kjennskap til fra den virkelige verden. Det finnes blant annet flere standarder for brukergrensesnitt. Apple var tidlig ute med å ta i bruk grensesnitt basert på vinduer, noe som senere ble akseptert som en standard etter at det populære operativsystemet til Microsoft Windows ble utbredt. Det finnes videre mange slike standarder som er allment kjent, men som det likevel ikke er gitt at en bruker kjenner til. I tillegg til disse standardene har blant annet Don Norman gjort en stor jobb med å definere grunnleggende krav for et godt brukergrensesnitt.

5.1.1 Don Normans prinsipper

Don Normans prinsipper stammer fra hans bok «The Design of Everyday Things» fra 1988, men de er fremdeles høyst aktuelle. Grunnen til det er at mange av standardene for grensesnitt som har vist seg å være suksessfulle, nettopp følger disse prinsippene.

Visibility

Tilgjengelige handlinger skal være lett synlige, og systemets tilstand skal kunne vurderes etter en kjapp observasjon. Dette impliserer at elementer bør være organisert etter samme relevans, og mye brukte handlinger bør være godt synlig. Synlighetsproblemer oppstår når menyen ikke viser tydelig nok hvilke handlinger som er tilgjengelig, eller hvis det er for mange handlinger vist samtidig. Det er derfor veldig viktig at antall valg for hver meny er begrenset.

Affordance

Begrepet “Affordance” kommer fra menneskers oppfatning av hva en fysisk ting kan gjøre. Et dørhåndtak er et slikt eksempel. I overført betydning til brukergrensesnitt, gir knapper uttrykk for å kunne trykkes ved at de har en 3D-stil og fremheves fra resten av grensesnittet. “Tabs” er også en affordance ved at de tydelig viser aktivt modus i programmet.

Constraints

Begrensning i hva som skal kunne gjøres er et av de viktigste prinsippene, og krever at antall mulig valg til enhver tid skal holdes til et minimum. Om en knapp ikke kan brukes i en tilstand, skal den gi tydelig indikasjon på at den er deaktivert.

Mapping

Mapping omtaler utseende av en kontroll og hvordan den er koblet til handlingen den utfører. Gode ikoner kan gi brukeren umiddelbar forståelse av hva en kontroll gjør uten å måtte trenge å lese en tekstbeskrivelse.

Feedback

Det er viktig å informere brukeren om at handlingen har blitt registrert. En knapp indikerer at den har blitt trykket. En handling som er ventet å ta tid bør for eksempel signaliseres med statusoppdatering for å gi brukeren tilbakemelding om at den utføres.

5.1.2 Gestaltprinsipper

Gestaltprinsipper stammer fra persepsjonpsykologiske prinsipper, og omhandler brukerens oppmerksomhet og hvordan den kan ledes mot riktig operasjon. Det er en viktig faktor for å forbedre synlighet, som beskrevet i et av Don Normans prinsipper. Det finnes flere prinsipper, som gruppering, bruk av linjer samt elementer med likhet i farge og form. Ved bruk av gruppering vil like kontroller være i nærheten av hverandre, noe som gjør det lettere for brukeren å avgjøre om det er riktig gruppe. Likhet i form og farge er et annet verktøy som kan brukes i tillegg til grupperinger for ytterlige å fremheve relasjoner. Bruk av linjer gir både struktur og hjelper brukeren med å navigere grensesnittet.

5.2 Menyer

Programmet består av flere menyer kategorisert etter klart definerte oppgaver i applikasjonen. Navigasjonen mellom oppgavene utføres ved hjelp av «tabs», som gir et tydelig inntrykk av modus i programmet. Tabs slik de er brukt i applikasjonen gir både «Affordance» og «Constraints», fra seksjon 5.1.1.

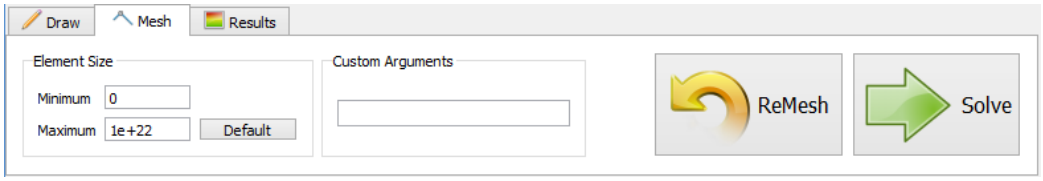
Det første modus som brukeren møter, er modelleringsvisningen med verktøy for å lage, endre og slette standardentiteter. Kontrollene er gruppert etter hvilke typer entiteter som kan lages i en egen gruppe kalt «Create figures». Tilsvarende er entitetesoperasjoner samlet i en gruppe kalt «Operations». I tillegg til grupperinger er fargen blå brukt for alle knapper som har med å lage entiteter.

Til høyre i menyen er det en knapp for å generere et mesh av geometrien. Den grønne pilen indikerer tydelig «Neste steg» og har også en tekst for å informere om hva det neste steget er. Grønnfargen og størrelsen gjør at den skiller seg fra de andre operasjonene, og den er alltid helt til høyre og synlig så fremt det finnes et neste steg. Figur 5.1 viser verktøy for modellering.

Figur 5.1: Meny med valg for å modellering. Knapper for å lage ulike standardformer samt å slette eller fullføre en form.

I meshvisningen finner brukeren felter for å endre standardkonfigurasjonen til mesher. Se figur 5.2. Som standard kan man konfigurere minste og største mulige elementstørrelse samt velge å sende egne argumenter. Sistnevnte kan være aktuelt for avanserte brukere med kjennskap til Gmsh. Kontrollene i denne modusen er foreløpig alle ganske avanserte, og hadde ikke trengt å være så synlig i henhold til Don Normans «synlighetsprinsipp». Kontrollere for mesh er likevel tildelt en egen tab fordi det ventes at senere versjoner skal ha flere implementerte egenskaper for meshgenerering. Det finnes to knapper, som begge er godt synlige. Den ene regenererer meshet gitt at innstillingene er endret og er symbolisert med en kurvet pil. En slik kurvet pil er mye brukt i

programvare for å oppdatere informasjon, og indikerer at man kommer tilbake til samme sted. Den andre pilen tar brukeren til neste modus, og er beskrevet tidligere.



Figur 5.2: Meny med konfigurasjon av mesh. Den gir mulighet for å velge minste og største mulige elementstørrelse i meshet, i tillegg til å kunne skrive inn egne kommandoer til mesher

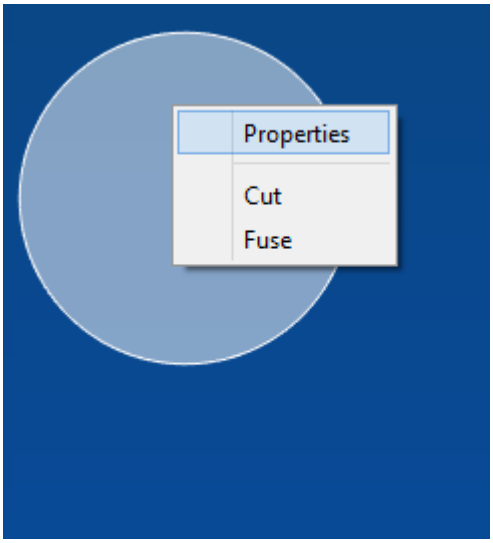
Figur 5.3: Meny med valg for resultatvisning. Den gir brukeren mulighet til å blant annet velge om mesh skal være synlig over resultatene.

Den siste modus gir mulighetene for å vise eller skjule mesh- og resultatfigurer. I motsetning til de andre menyene finnes det ikke noe neste steg, og menyen har derfor ingen pil for neste modus. Se figur 5.3.

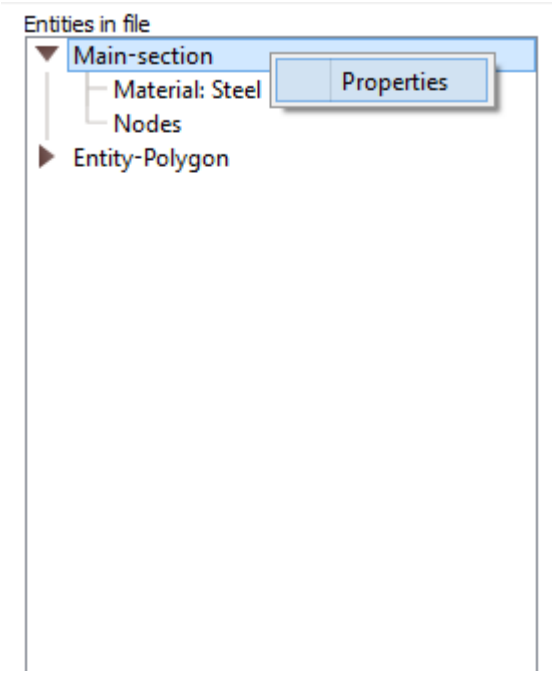
Bruken av tabs for å skifte mellom modus bygger på en ganske annen tankegang for design enn for eksempel CrossX. I CrossX har alle verktøy en fast plass og vises konstant. De blir deaktivert hvis de ikke kan brukes i en tilstand, men de er fremdeles i menyen. Fordelen med å bruke modusbaserte menyer, er at det er mer plass og at menyene blir mer oversiktlig. Ryddigheten gjør det lettere å finne fram selv om man ikke er kjent med programmet fra før. Tilnærmingen med bruk av tabs er ikke ny. Bruken i applikasjonen skiller seg likevel litt fra andre programmer ved at den aktive taben begrenser de tilgjengelige verktøyene og følger brukerens posisjon i arbeidsprosessen definert i Figur 1.1.

5.2.1 Kontekstmeny

I tegnemodus finnes det også flere valgbare kontekstmenyer. Kontekstmenyer vises ikke fast, men følger brukerens posisjon og åpnes der brukeren trenger slik funksjonalitet. Innholdet i en kontekstmeny endrer seg etter hvilke muligheter som finnes for det valgte elementet, og den hentes fram ved å høyreklikke på et element. Så langt er det bare implementert kontekstmenyer for entiteter i grafikkvinduet og i treet med entiteter, henholdsvis figur 5.4 og 5.5. Kontekstmenyen er brukt i grafikkvinduet for å spesifisere egenskaper på en entitet, i tillegg til å utføre boolske operasjoner. Egenskapene til en entitet vises i et eget dialogvindu.



Figur 5.4: Kontekstmeny for grafikkvindu. Menyen dukker opp når brukeren høyreklikker på en entitet i grafikkvinduet. Fra denne menyen kan brukeren utføre boolske operasjoner og åpne dialogvinduet med entitetsegenskaper.



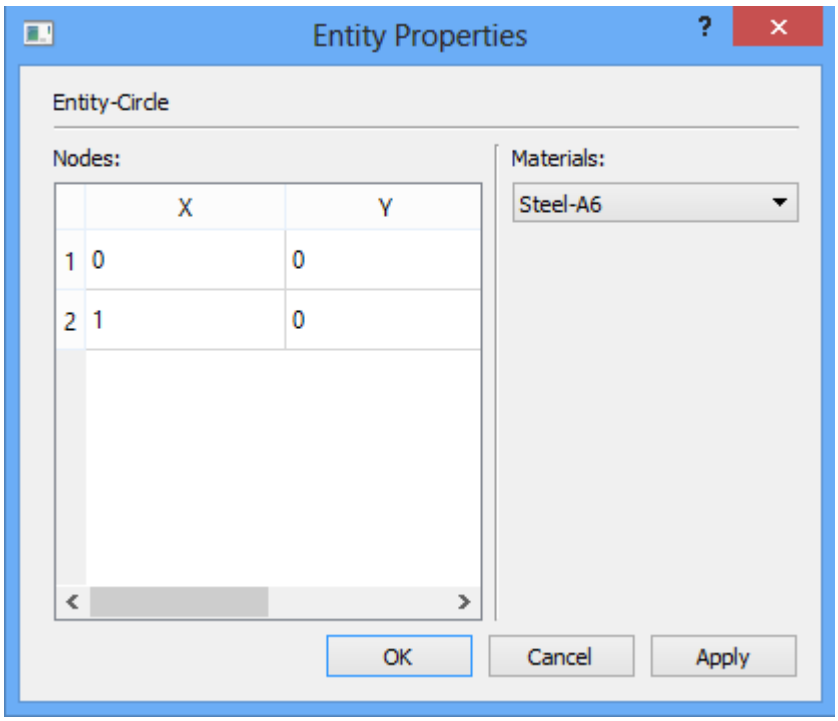
Figur 5.5: Kontekstmeny for entitet-treet. Menyen dukker opp når brukeren høyreklikker på en entitet i treet med entiteter. Den gir kun mulighet for å vise dialogvinduet med egenskaper for entiteten.

5.2.2 Dialogvindu

Dialogvinduer blir ofte brukt til å vise innstillinger for et program eller egenskaper for enkelte deler av programmet. Man kan anta at de fleste brukere er godt innforstått med hvordan man håndterer et dialogvindu, men det innfører likevel et ekstra arbeid. Dialogvinduer egner seg derfor best hvis det som skal vises har et stort antall mulige valg som ellers hadde ført til at hovedmenyen hadde blitt rotete.

I applikasjonen brukes dialogvinduer kun til å endre egenskapene til entiteter. En entitet har mange egenskaper som skal kunne endres, og egner seg derfor godt til å vises i et eget vindu. Dialogvinduet gir mulighet for å endre nodeposisjonene for alle nodene i entitetene, og kan i fremtiden også implementere mulighet for materialvalg for entiteter. Vinduet arver knappene «OK», «Cancel» og «Apply», som er standardfunksjonalitet for dialoger

og som gir brukeren mulighet for å avgjøre om endringene skal lagres eller forkastes ved lukking av dialogvinduet.



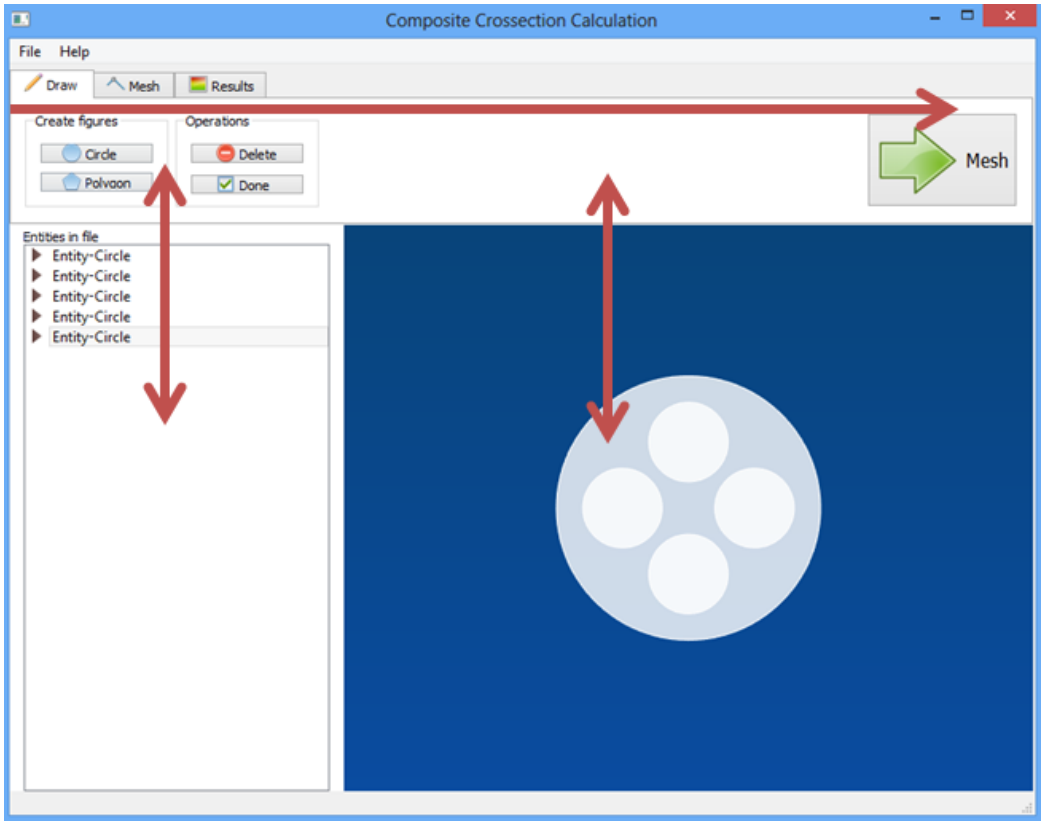
Figur 5.6: Egenskaper for entiteter vist i et eget dialogvindu. Vinduet gir mulighet for å endre posisjonen til alle nodene i entiteten. Den er også planlagt å skulle kunne brukes til å endre entitetens materiale.

5.3 Interaksjon og arbeidsflyt

Det første brukeren møter når applikasjonen åpnes er tegnemodus med menyer beskrevet i seksjon 5.2. Tegnevisningen er laget for å gi brukeren et oversiktlig grensesnitt for å tegne figurer. Verktøyene er plassert i den øverste menyen. I området nedenfor vises figurene gjennom VTK. Det er også et tre til venstre som lister opp entitetene i modellen for å gi en bedre oversikt. Tanken er at grafikkvinduet og treet i så stor grad som mulig kun skal være datafremvisninger. De supplementerer hverandre og presenterer de samme dataene i applikasjonen på ulike måter slik at brukeren lettere kan få oversikt over hvilke data som ligger i tegningen. Fraværet av verktøy i venstremenyen fører til at interaksjonen med applikasjonen skjer vertikalt innenfor hvert visningsmodus, og går mellom toppmenyen og den av visningene som til enhver tid er mest praktisk for brukeren. Se figur 5.7, 5.8 og 5.9.

Navigeringen til neste visningsmodus er tydelig vist ved en grønn pil mot høyre, som beskrevet tidligere. Når applikasjonen går til neste visningsmodus, endres aktiv tab til den til høyre for den forrige. Dette fører til at det blir en arbeidsflyt mot høyre.

Løsningen med å bruke tabs som navigasjon hjelper med å gi et tydelig begrep om hvilken arbeidsoppgave som utføres. Det brukes også for å begrense mengden verktøy som vises samtidig, slik at det kun er kontrollere som er nødvendig for at brukeren skal kunne utføre oppgaven i den respektive modusen. Dette begrenser brukerens valg, og gjør bruken mer strømlinjeformet.

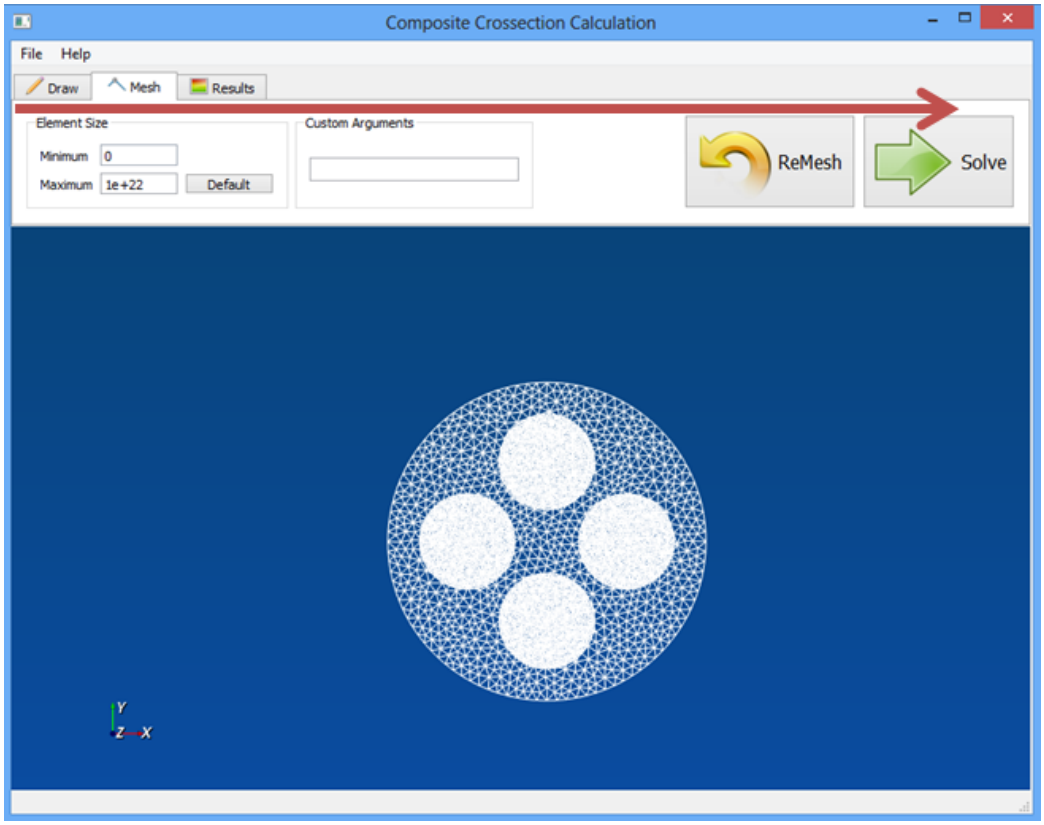


Figur 5.7: Interaksjonsflyt i tegnemodus. Alle valg ligger i toppmenyen, og brukerens bevegelser går mellom toppmenyen og et av de to visningene for entiteter. Både tre- og grafikkpresentasjonen viser de samme dataene.

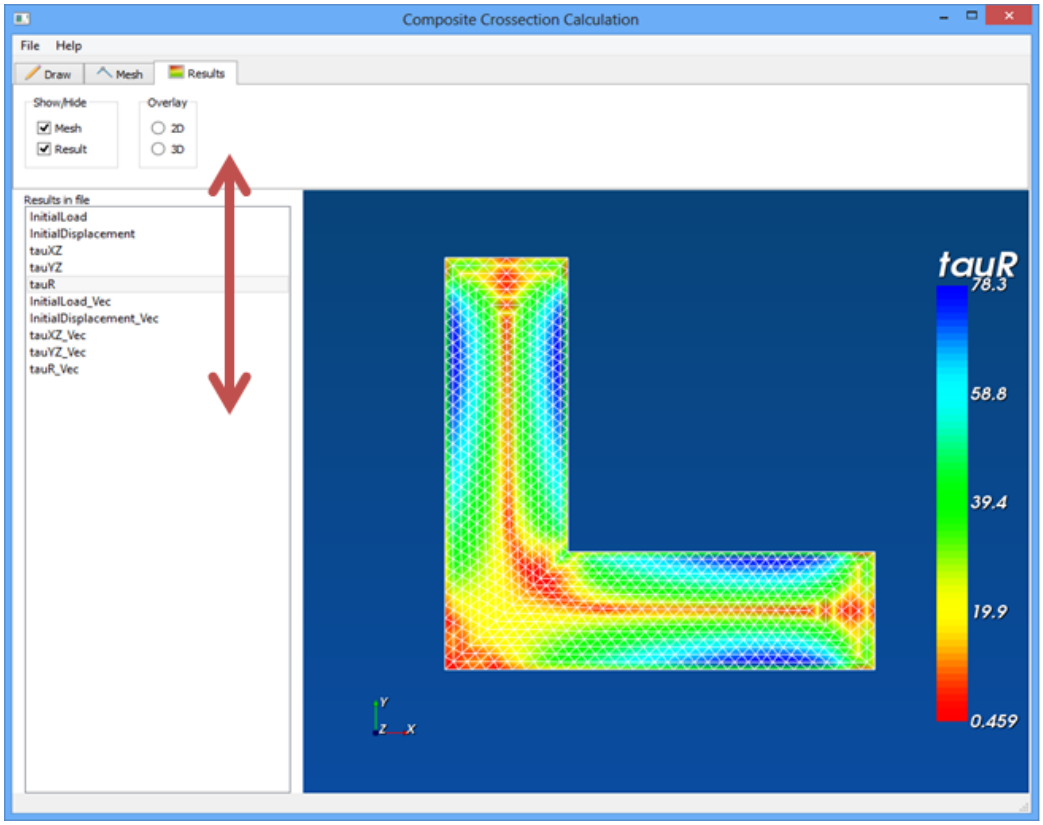
For å modellere et tverrsnitt, trykker først brukeren på en geometrisk figur. Når det er valgt, vil en entitet dukke opp som et punkt i den venstre menyen. Brukeren kan da klikke i grafikkvinduet for å legge til noder i entiteten. Nodene visualiseres med grønne kors, som brukeren kan dra dit de ønskes, og applikasjonen vil automatisk generere kanter ettersom nodene flyttes. For å fullføre entiteten trykker man på knappen «Done» eller bruker den tilsvarende tastatursnarveien. De grønne punktene som representerer nodene vil da forsvinne, og entiteten blir hvit. Nodekorsene vil komme til syne igjen hvis entiteten dobbeltklikkes. For å spesifisere posisjonen til en av nodene i entiteten, må den høyreklikkes slik at kontekstmenyen med mulighet for å

åpne dialogvinduet med egenskaper kommer frem.

Et tverrsnitt kan modelleres ved hjelp av flere geometriske entiteter, og for å kombinere dem benyttes en av de geometriske operasjonene beskrevet i seksjon 4.6.2.



Figur 5.8: Interaksjonsflyt i meshmodus. I meshvisning ser brukeren meshet av geometrien og har mulighet til å endre meshegenskaper. Den grønne pilen som indikerer neste steg, er her tydelig vist til høyre, sammen med en pil som indikerer «Oppdater».



Figur 5.9: Interaksjonsflyt i resultatmodus. Alle valg ligger i toppmenyen, og ligner mye på tegnemodus. Forskjellen er at det her velges hvilke data som skal vises i grafikkvinduet fra den venstre listen med resulater. I motsetning til modelleringsmodus, består resultater kun av èn form. Det er derfor ikke behov for å presentere dataene i en liste i tillegg til grafikkvinduet som det gjøres i tegnemodus.

5.4 Tastatursnarveier

Tastatursnarveier er spesielt viktig for at viderekommende brukere skal kunne arbeide effektivt. De fleste designvalgene så langt går på å lage et design egnet for at nye brukere lett skal kunne forstå bruken. Viderekommende brukere som er vant til valgene og arbeidsprosessen, ønsker ofte å kunne utføre dem hurtigere. Nedsiden med disse snarveiene er at brukeren må husker disse, og det bør derfor ikke være den eneste muligheten for å utføre en operasjon. Snarveiene må i stedet være et ekstra lag med valgfri interaksjon slik at man ikke er avhengig av dem for å kunne utføre applikasjonen oppgaver. For å appellere mot viderekommende brukere var det også et ønske om å støtte tastatursnarveier. Det er implementert snarveier for ny fil, åpne fil, bytte mellom visningsmodusene samt fullføre eller avbryte tegninger. Det var planlagt å bruke ENTER-tasten for å fullføre tegning, men det gikk ikke på grunn av begrensninger i VTK. I stedet ble snarveien for å fullføre tegning CTRL+E - slik den også er i Gmsh. En komplett oversikt over tastaturkombinasjoner og samsvarende utført operasjon vises i tabell 5.1.

Snarvei	Operasjon utført
CTRL+ N	Ny tegning
CTRL+ O	Åpne fil
CTRL+ E	Fullfør tegning
CTRL+ D	Gå til tegnemodus
CTRL+ M	Gå til meshmodus
CTRL+ R	Gå til resultatmodus
ESCAPE	Avbryt operasjon/Slett

Tabell 5.1: Tastatursnarveier implementert i applikasjonen for å øke hastigheten for viderekommende brukere. Snarveier er implementert for å bytte mellom visningsmodus, filbehandling og enkelte mye brukte tegneoperasjoner.

5.5 Brukertest

Brukervennlighet oppnås gjennom en iterativ prosess. Interaksjonsdesignere starter først med design av grensesnitt med bruk av blant annet standarder og retningslinjene beskrevet i seksjon 5.1.1. Når grensesnittet er på et brukbart nivå kan brukervennligheten verifiseres av faktiske brukere. En brukervennlighetstest vil nesten alltid avdekke problemer og forbedringer som ikke designere har tenkt på. Resultatet blir derfor ofte bedre hvis designerne tilpasser brukergrensesnittet underveis, og involverer brukerne og tar i mot deres tilbakemeldinger. Det er foretatt en brukervennlighetstest for å avdekke problemer og eventuelle forbedringer.

5.5.1 Utførelse

Testen ble utført individuelt av fem personer. De fikk utdelt et ark med oppgaver som skulle utføres. Se tabell 5.2. Oppgavene var abstrakte og inneholdt ingen annen informasjon enn hva brukeren skulle prøve å oppnå. Det ble ikke gitt noe introduksjon til de ulike modusene i brukergrensesnittet, og heller ikke noe informasjon om hvilke verktøy som var tilgjengelige. Oppgavene ble utført uten hjelp, og brukeren fikk først veiledning etter å ha gitt opp. Etter hver oppgave ble tidsbruken notert, og brukeren ble spurt om det var vanskelig. Hvis brukeren hadde problemer ble det spurt om hvilken kontroll de så etter. Alle brukerene ble også spurt om de hadde noen ytterligere forslag til forbedringer. Oppgavene genererte mange tilbakemeldinger som delvis omhandlet de samme problemene, og det ble derfor ikke ansett som nødvendig med flere testpersoner i denne omgang.

5.5.2 Resultat

Brukertesten avdekket fort manglende eller lite logisk funksjonalitet. Den første brukeren brukte fire minutter før han ga opp å fullføre tegning. Problemene var så kritiske for brukervennligheten at det ikke var noe annet valg enn å legge til denne funksjonaliteten før applikasjonen ble testet på de resterende brukerne. Etter å ha lagt til den manglende funksjonaliteten var det ingen brukere som ga opp på noen oppgaver. Alle oppgavene ble utført på to minutter eller mindre, bortsett fra oppgaven som testet operasjonen for geometriske funksjoner, noe som tok 5 minutter. Oppgaven tok lenger tid enn

Oppgaver	Beskrivelse
Lage et tverrsnitt 1	Brukeren skal lage en vilkårlig form, for så å slette den.
Lage et tverrsnitt 2	Brukeren skal lage et sirkulært tverrsnitt, sentrert i origo med radius 1.
Lage et tverrsnitt 3	Brukeren skal lage et rektangel med en depresjon (sprekk) i form av en halvsirkel, plassert på en av langsidene.
Analysere et tverrsnitt	Brukeren skal modellere et tverrsnitt av en I-profil, generere mesh, og velge resultat nr. 2.
Analysere generert mesh	Brukeren skal åpne filen «brukertest.msh» og analysere den.

Tabell 5.2: Oppgaver benyttet for å evaluere brukervennlighet til brukergrensesnittet. De er i størst mulig grad utformet slik at brukeren tvinges til å teste funksjonalitet i programmet uten å fortelle hvordan det skal gjøres.

de andre fordi brukerne brukte lang tid på å finne kontektsmenyen, og opplevde problemer med å utføre operasjonene i riktig rekkefølge. For geometriske operasjoner var det flere forslag til forbedringer som gikk fra bruk av ikoner i kontekstmenyen til å la operasjonene åpne et eget vindu for å tydeligere vise rekkefølgen av valgte entiteter. Det var også uttrykt ønske om en tekstbar nederst i hovedvinduet, som hinter om at det går an å høyreklikke på figurer. Generelt for tegning var det et ønske om å vise origo samt akser for å gi brukeren en bedre følelse av arbeidsområdet. Det ble også savnet funksjonalitet for å legge til noder i dialogvinduet for en entitet. Selv om det ikke var noe direkte problem kom det også forslag om å støtte tegning av linjer med «Snap to grid» funksjonalitet.

5.5.3 Evaluering

Resultatet av brukertesten bekreftet mange av tankene rundt designet av brukergrensesnittet. De fleste forstod arbeidsflyten samt mapping av tabs for å vise stegene i prosessen. Etter at et tverrsnitt var ferdig tegnet brukte de fleste mindre enn et minutt på å komme frem til resultatet. Moduset med det største forbedringspotensialet var tegnevisningen. Enkelte av problemene er allerede godt kjent, men behøver funksjonalitet som er til dels tidkrevende å implementere. Angrefunksjonalitet ville antakeligvis ha redusert tiden betraktelig for alle brukerne når de utførte testen av de geometriske operasjonene. Det skal ikke legges skjul på at det også kom flere nye, gode synspunkter. Disse ble utbedret umiddelbart der tiden tillot det, og der det ikke var mulig har de blitt foreslått under videre arbeid.

Under testen ble det avdekket flere feil som førte til at programmet kræsjet. Dette er i all hovedsak grunnet i at brukerne prøver seg frem og utfører operasjoner i sekvenser som er tilsynelatende lite logiske. Det er en helt naturlig måte å lære seg programmet på, men feilene er svært vanskelig å finne fordi utvikleren kjenner programmet for godt og ikke tenker på å bruke operasjonene slik. Testen forbedret dermed stabiliteten ved å utbedre alvorlige feil, som ellers kanskje ikke hadde blitt funnet.

Kapittel 6

Konklusjon og videre arbeid

6.1 Konklusjon

I denne oppgaven har flere rammeverk blitt studert og implementert i et brukergrensesnitt for beregninger med elementmetoden. Brukergrensesnittet inneholder i nåværende tilstand funksjonalitet for både modellering av tverrsnitt og presentering av resultater. Det er koblet opp mot mesher og solver, og utveksler data automatisk med disse.

Grafikkrammeverket VTK har blitt gjennomgått, tilpasset og integrert i applikasjonen. Nodeinteraksjon med bruk av VTK ble ikke like elegant som ønsket på grunn av begrensninger i VTK, og det oppstår enkelte feilkalkulasjoner som er begrenset til noen operasjoner utført med OCCT. Bortsett fra det virker rammeverkene svært godt. De er implementert på en lite inntrengende måte som bidrar til å holde applikasjonens arkitektur på et oversiktlig nivå.

Arbeidet med brukervennlighet har vært en kontinuerlig prosess der det har blitt sett på hvilke oppgaver brukeren ønsker å utføre, og så blitt tilpasset deretter. Det ble utført en brukertest på slutten av arbeidet for å kartlegge brukervennligheten. Tilbakemeldingene fra testen var konstruktive og avdekket noen problemer med brukergrensesnittet samtidig som de i stor grad bekreftet at implementert funksjonalitet virket som det skulle.

I sin nåværende tilstand av modellering- og visningsfunksjonalitet skal ap-

plikasjonen kunne brukes til å modellere og beregne tverrsnittdata. Arbeidet nedlagt i oppbygning og dokumentasjon av koden skal kunne gjøre kodebasen til en plattform det er mulig å bygge videre på.

6.2 Videre arbeid

Det er flere muligheter for videre utvikling av applikasjonen. Brukervennligheten kan ytterligere forbedres ved å lage flere dialogvinduer og en gjennomgående statusbar som informerer brukeren om systemets tilstand og eventuelt gir hint om tilgjengelige handlinger. Det er et ønske om å gjøre valg av geometriske operasjoner mer strukturert, noe som kanskje kan løses ved å ta i bruk et eget dialogvindu for operasjoner. Det er også ønsket mulighet for «Snap to grid» i tegnevisningen, og mulighet for å legge til en node på en gitt posisjon direkte i entitetsdialogen uten bruk av mus.

Angrefunksjonalitet i tegnevisningen er også en vesentlig funksjonalitet for å øke brukervennlighet. Det vil gi brukeren en enkel utvei ved feil, og er så kjent blant brukere at det vil øke både brukerglede og produktiviteten betraktelig. Ved implementasjon av en slik angrefunksjonalitet vil det være nødvendig å først definere angrefunksjonens omfang, for deretter å lage støtte for det i de berørte klassene. Implementasjon av en angrefunksjonalitet kan derfor være ganske omfattende.

Feilberegningene i rammeverket OCCT bør bli sett nærmere på, og det er her også stort potensial for å implementere flere verktøy fra OCCT. En del brukere er kjent med mange verktøy fra andre programmer, og et større antall verktøy vil kunne øke produktiviten deres. Eksempler på ønsket verktøy er flere geometriske operasjoner samt mulighet for å kopiere en entitet, alene eller langs et mønster, kjent som «Array».

Det er planlagt støtte for tverrsnitt bestående av kompositttmaterialer i fremtiden. I deler av grensesnittet er det allerede lagt inn støtte for å tildele hver enkel entitet et materiale, men det gjenstår å lage et grensesnitt for å velge og definere avanserte materialer samt et system for å håndtere de ulike materialene. Dette arbeidet bør koordineres med utvikling av støtte for kompositttmaterialer i solver.

Bibliografi

[1] Ergonomics of human-system interaction - part 210: Human-centred design for interactive systems, 2009. ISO 9241-210.

[2] Gnu lesser general public licence. http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_Licensel, September 2013.

[3] Msh-ascii-file-format. http://www.geuz.org/gmsh/doc/texinfo/gmsh.html#MSH-ASCII-file-format, September 2013.

[4] Brukergrensesnitt. http://snl.no/brukergrensesnitt%2FIT, Mai 2014.

[5] Download qt. http://qt-project.org/downloads, Jan 2014.

[6] Gnu general public licence. https://www.gnu.org/copyleft/gpl.html, Mai 2014.

[7] Visual studio add-in for qt5. http://download.qt-project.org/official_releases/vsaddin/qt-vs-addin-1.2.3-opensource.exe, Feb 2014.

[8] Windows software development kit (sdk) for windows 8k. http://msdn.microsoft.com/en-us/windows/desktop/hh852363.aspx, Jan 2014.

[9] Kolbein Bell, Oddmund V. Bleie, and Lars Wollebaek. *CrossX User's Manual*. NTNU, 2000.

[10] Christophe Geuzaine and Jean-Francois Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://geuz.org/gmsh/, September 2013. Version 2.8.3.

[11] Norman Herr. Summary of don norman's design principles. http://www.csun.edu/science/courses/671/bibliography/preece.html, May 2013.

[12] Kitware. Cmake. http://cmake.org/cmake/resources/software.html.

[13] Kitware. The visualisation toolkit. http://www.vtk.org/VTK/resources/software.html.

[14] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[15] Ben Schneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson, Addison Wesley, 2008.

[16] Dag Svanaes. *Oppsummering TDT4180 - Menneske Maskin Interaksjon*.

Appendiks

Tillegg A

Lisenser

Det finnes flere populære lisenser innen fri kode, blant annet BSD, LGPL og GPL. BSD «Berkeley Software Distribution» er den mest åpne lisensen. Koden skrevet under en slik lisens kan endres og distribueres fritt. GPL «General Public License» er også åpen. Den er et resultat av at utviklere så seg lei på at fri kode ble benyttet i kommersiell programvare uten at samfunnet fikk noe tilbake. Den er derfor bygget på en visjon om at åpen kode skal forbli åpen, og at prosjekter som benytter åpen kode også må frigi koden. Dette tvinger en kommersiell aktør som benytter et slikt rammeverk til å frigjøre koden. Denne lisensen utgjør derfor et helt klart hinder for kommersiell programvare. Den nyere LGPL «Lesser General Public Licence» løser dette på en elegant måte. LGPL tillater å bruke rammeverk gitt at det distribueres som et eget bibliotek og at koden kobles til prosjektet ved «dynamisk linking». I Windows betyr det at rammeverket må distribueres sammen med applikasjonen som såkalte DLL-filer. Følges dette krever LGPL kun at endringer gjort i rammeverkets egen kode må offentliggjøres.

A.1 Implikasjoner for prosjektet

Applikasjonen benytter seg av rammeverk og programvare lisensiert under alle tre lisensener. VTK er lisensiert under BSD. Qt og OCCT er lisensiert under LGPL. Gmsh er lisensiert under GPL.

Lisensene er fulgt ved at Qt og OCCT er kompilert separat og linket dynamisk til applikasjonen. Bruken av Gmsh er også innenfor lisensen ved at

den distribueres som et eget program og kun behandles som en ekstern prosess av applikasjonen. Det er verdt å merke seg at en tettere integrasjon av Gmsh, ved å bruke den som et rammeverk vil medføre at prosjektet blir åpen kildekode.

Tillegg B

Spesifikasjon av filer

B.1 Spesifikasjon av geo-filer

Geometriske data som er sent til GMsh er lagret i Gmsh sitt eget format. For å generere et mesh trenger Gmsh overflater. Disse er definert med punkter som holder posisjonen, som igjen er koblet sammen av linjer. Tilslutt er en «line loop» definert av et sett linjer. På denne loopen kan en overflate defineres. I tilfellet med en figur med et hull, vil overflaten gå mellom to «line loops», der den ene går med klokken og den andre går mot.

```
Point(Nummer) = {x, y, z, 1};
...

Line(Nummer) = {Point1, Point2};
...

Line Loop(Antall linjer + Nummer) = {Line1, Line2, Line3, .., ..};
..

Plane Surface(Antall linjer + Nummer) = {LineLoop1, LineLoop2};
..
```

Det er verdt å merke seg at nummereringen av «Line Loops» starter på det totale antallet linjer, og at en overflate kan ha en eller to «Line Loops».

B.2 Spesifikasjon av msh-filer

Datafilen for meshet er et tabseparert tekstformat som er satt sammen av blokker. Under følger definisjonen av datafeltene som denne applikasjonen leser av formatet.

```
$MeshFormat
[versjon] [Tekst/Binær] [datastørrelse]
$EndMeshFormat
$Nodes
[Antall noder]
[Nodenummer] [x] [y] [z]
...
$EndNodes
$Elements
[Antall elementer]
[Elementnummer] [Type] [Antall Tilleggsdata] <Tilleggsdata>... [Node1]
[Node2] [Node3]
...
$EndElements
```

Alle blokkene skal ha start og sluttkode. Første blokk definerer formatet, med versjon, type, og nøyaktighet.

Deretter følger antallet noder og alle nodene, og tilsvarende for elementer. Det er verdt å merke seg at nummereringen alltid starter fra 1 og at applikasjonen i nåværende tilstand ignorerer tilleggsdata og kun leser elementer av «Type 2», altså trekanter.

B.3 Spesifikasjon av vtk-filer

Applikasjonen benytter seg av «vtkDataSetReader», noe som gjør at den vil kunne lese alle dataformater støttet av vtk. Dataformatet definert for utveksling av data mellom denne applikasjonen og solveren er «vtkPolygonData». Dette er et tabseparert tekstformat, som er lett å lese og feilsøke.

```
# vtk DataFile Version 2.0
[Kommentar]
ASCII
DATASET
POLYDATA
POINTS [Antall punkter] [Datatype]
p0x p0y p0z
p1x p1y p1z
...
p(n-1)x p(n-1)y p(n-1)z

POLYGONS [Antall polygoner] size
AntallPolygonPunkter0, i0, j0, k0, ...
AntallPolygonPunkter1, i1, j1, k1, ...
...
AntallPolygonPunkter(n-1), i(n-1), j(n-1), k(n-1), ...

POINT_DATA [Antall punkter]
SCALARS [Resultatnavn] [Datatype]
LOOKUP_TABLE default
p0
p1
...
p(n-1)

VECTORS [Resultatnavn] [Datatype]
LOOKUP_TABLE default
v0x v0y v0z
v1x v1y v1z
...
v(n-1)x v(n-1)y v(n-1)z
```

Tillegg C

Kompilering av VTK med Qt

C.1 Nødvendige programmer

- Qt 5.1.1

  (QT5.1.1 ble brukt i dette eksempelet, men senere versjoner skal også virke)

  Last ned Qt[5] og Qt-Add on for Visual Studio 2012[7]

- Cmake

  Last ned CMake[12]

- VTK 6.1

  Last ned VTK-6.1.0.rc2.zip (Må kompileres)[13]

- Windows 8 developer kit

  Last ned Window SDK[8]

C.2 Kompilere VTK

Bruk Cmake for å lage VTK prosjektet i Visual Studio med Qt 5.
Kompiler først prosjektet «ALL_BUILD» i VTK.sl med VS2012. Kompiler deretter prosjektet «Install». For å kunne kompilere «Install» må Visual Studio kjøres som administrator.
VTK med biblioteker, header, og dll filer skal nå ligge i en mappe kalt VTK under programfiler.

OBS: Qt5 prosjekt bør lages i Visual Studio fremfor Qt-Creator. Qt Creator hadde problemer med å vise QVTKWidget, og det var også problemer med å inkludere VTK-headere

C.3 Opprette prosjekt

Prosjektopprettelse i Visual Studio for VTK-Qt

Ny Qt5 applikasjon (Husk å inkludere OpenGL)

Det er nå et Qt-prosjekt i Visual Studio. Prosjektet må deretter videre konfigureres for å kompilere med VTK. Dette skjer i *Project Settings*, som dukker opp når man høyreklikker på prosjektet

Inkluder VTK-header

Project -> VS++ Directory -> Include Directory.
Include Directory må inneholde en stien til VTK innstalasjonen, som standard ligger i *C:\Program Files\VTK\include\vtk-6.1.*

Inkluder VTK-biblioteker

Legg til under Projekt -> Linker -> Additional Library Directory
C:\Program Files\VTK\bin;
C:\Program Files\VTK\lib;
Uten dette vil det oppstå en mengde linkerfeil fordi kompilatoren ikke finner det faktiske rammeverket. Feilmeldingene vises ikke før under kompilering. Hvis det fremdeles oppstår linkerfeil; legg til alle navnene på bibliotekene som er brukt. Disse legges til på bunnen av listen i
Project -> Linker -> Additional Dependencies.

Det som er standard dependencies i et Qt5 prosjekt, og allerede skal være der er:
qtmain.lib;Qt5Core.lib;Qt5Gui.lib;Qt5OpenGL.lib;opengl32.lib;glu32.lib;Qt5Widgets.lib

Legg til følgende øverst i klassen med main-metoden hvis det oppstår en feilmelding med «override PolydataMapper.h» under lasting av symboler:
#define vtkRenderingCore_AUTOINIT 4(vtkInteractionStyle,vtkRenderingFreeType, vtkRenderingFreeTypeOpenGL,vtkRenderingOpenGL)
#define vtkRenderingVolume_AUTOINIT 1(vtkRenderingVolumeOpenGL)

Tillegg D

Kodekonvensjoner og kodekvalitet

D.1 Konvensjoner

Hva man velger som kodekonvensjon har ikke så mye å si så lenge man holder seg tro til den. I denne applikasjonen har det blitt valgt å følge kodekonvensjonene der alle navn er skrevet med camel case, og konstanter kun er skrevet med store bokstaver. Det vil si at både variabler, funksjoner og klasser er skrevet i camel case. Det som derimot skiller navngivingen av klasser fra funksjoner og variabler, er at klasser har stor bokstav for hvert ord, mens navnene til variabler og funksjoner starter med liten bokstav. Alle funksjoner skal ha en kommentar som beskriver funksjonens innhold, og det skal helst være brukt kommentarkode som gjør at doxygen kan hente ut innholdet.

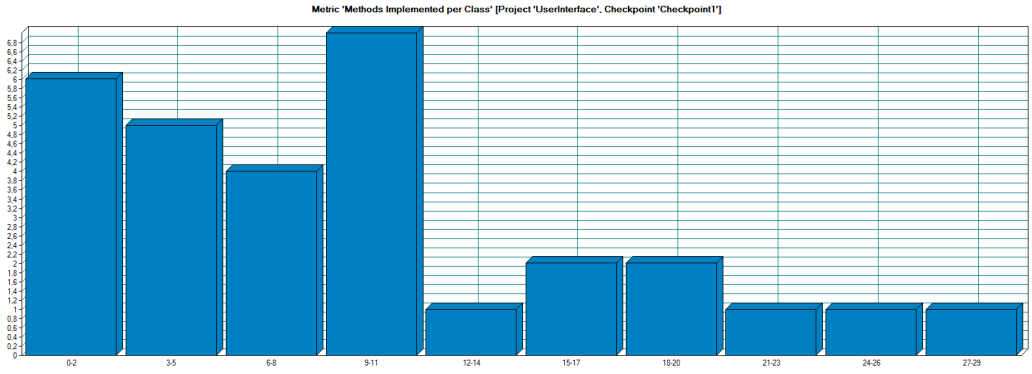
Navn	Beskrivelse
camelCase	Navn start med liten bokstav. Alle påfølgende ord starter med stor bokstav
PascalCase	Alle ord starter med stor bokstav
KONSTANTER	Alle bokstaver i stor bokstav

Tabell D.1: Oversikt over navnkonvensjoner brukt for klasser, funksjoner og variabler

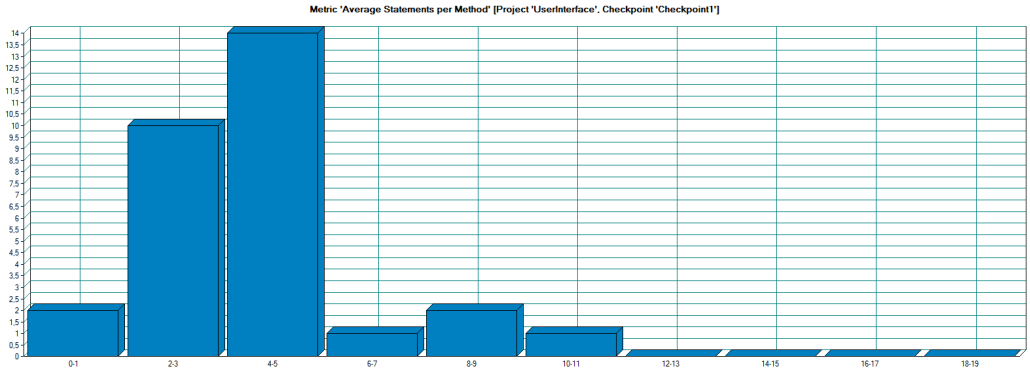
D.2 Kodekvalitet

Kodekvalitet er et resultat av flere faktorer. En av dem er lengde på klasser og funksjoner. I følge Clean Code[14] er lesbarheten til en klasse best hvis den kun har et ansvar. Man skal altså kunne si hva en klasse er ansvarlig for med et ord. Klasser bør være så små som mulig, og helst ikke ha mer enn 30 funksjoner. Tilsvarende skal en funksjon også helst være begrenset til kun å utføre en oppgave.

Den andre faktoren er god navngivning. Klasser og funksjoner må ha beskrivende navn, lange hvis det er nødvendig. Et kort funksjon med et beskrivende navn gjør leseren mindre avhengig av å lese ekstra dokumentasjon for å forstå innholdet. Det er vanskelig å måle kvaliteten på navngivningen, men det finnes flere programmer med mulighet for å analysere størrelsen av funksjoner og klasser. Figur D.1 er produsert med programmet «Source Monitor» og viser antallet funksjoner i hver klasse. Den viser at de fleste klassene har mindre enn 11 funksjoner, og det finnes ingen med mer enn 30 funksjoner. Anbefalt verdi i «Source Monitor» er under 25 funksjoner i hver klasse, så også her ligger koden for det meste innenfor grensen. Figur D.2 viser antall uttrykk per funksjon, og viser at alle funksjonene har færre enn 10 uttrykk. Anbefalt verdi fra «Source Monitor» ligger mellom 5 og 10 uttrykk, men retningslinjene definert i Clean Code er åpne for at det også er positivt med færre uttrykk.



Figur D.1: Figuren viser distribusjonen av klasser med antall funksjoner i klassen langs x-aksen. Anbefalt verdi er mindre enn 25 funksjoner.



Figur D.2: Figuren viser antall uttrykk i hver funksjon, med antall funksjoner langs x-aksen. Anbefalt verdi er mindre enn 10 .

Tillegg E

Prosjektets konfigurasjon og oppbygning

E.1 Prosjektkonfigurasjon

Applikasjonen er avhengig av at utvikleren har flere biblioteker innstallert på maskinen. Utvikleren vil trenge rammeverkene Qt, VTK og OCCT. De har alle headerfiler, biblioteksfiler og DLL-filer. Visual Studio må vite hvor den finner disse filene. Det spesifiseres for hvert rammeverk i prosjektets egenskaper (se Appendiks C.3 for veiledning for konfigurering av rammeverket VTK).

For å kjøre applikasjonen trenger programmet DLL-filene. Det kan løses på to måter. Det letteste er å legge filene i den samme mappen som den kompilerte applikasjonen. Hvis filene skal ligge i en annen mappe må denne adressen til mappen legges inn som en PATH variabel på systemet.

E.1.1 Kompilerte biblioteker

De kompilerte bibliotekene som applikasjonen er avhengig av er samlet i to zippede filer. Linker til filene vil være å finne i README i prosjektmappen.

Utviklerversjon Inneholder alle filer i rammeverkene brukt for å kompilere og kjøre applikasjonen.

Brukerversjon Inneholder kun filene nødvendig for å kjøre applikasjonen. Den inneholder derfor ikke filene nødvendig for å endre programmet.

Filene inneholder kun nødvendige rammeverk for å kompilere og kjøre programmet. Mesher og solver er to separate programmer som må lastes ned og legges sammen med applikasjonen. Det er gjort på denne måten fordi det antas at de nevnte programmene kommer til å bli oppdatert mye hyppigere enn bibliotekene.

E.2 Prosjektstruktur

Kodebasen har blitt kategorisert etter klassenes funksjon og bruksområde. Det er gjort for å enklere kunne navigere i klassene, og det har blitt gjort ved bruk av filter i Visual Studio. Klassene er delt opp i kategoriene DataObject, In/Out, Models, Utilities og ViewController.

DO Filteret inneholder dataobjekter. Klasser i dette filteret er ment som dataholdere for objekter. De ligner en modell, men i motsetning til en modell har disse klassene utallige instanser. Noder, kanter, elementer og entiteter er typiske dataobjekter for prosjeketet.

IO Filteret inneholder klasser brukt til Inn/Ut operasjoner - altså lesing og skriving av data til filer. I filteret finner man parsere for de ulike filformatene som leser filene og lager dataobjekter av informasjonen. Filskrivere for de ulike filformatene støttet ligger også i dette filteret.

Models Filteret inneholder Datamodellene. Det er disse som holder på alle dataobjektene og manipulerer dem. Det finnes tre modeller for hele applikasjonen; DrawModel, ResultModel og CModel.

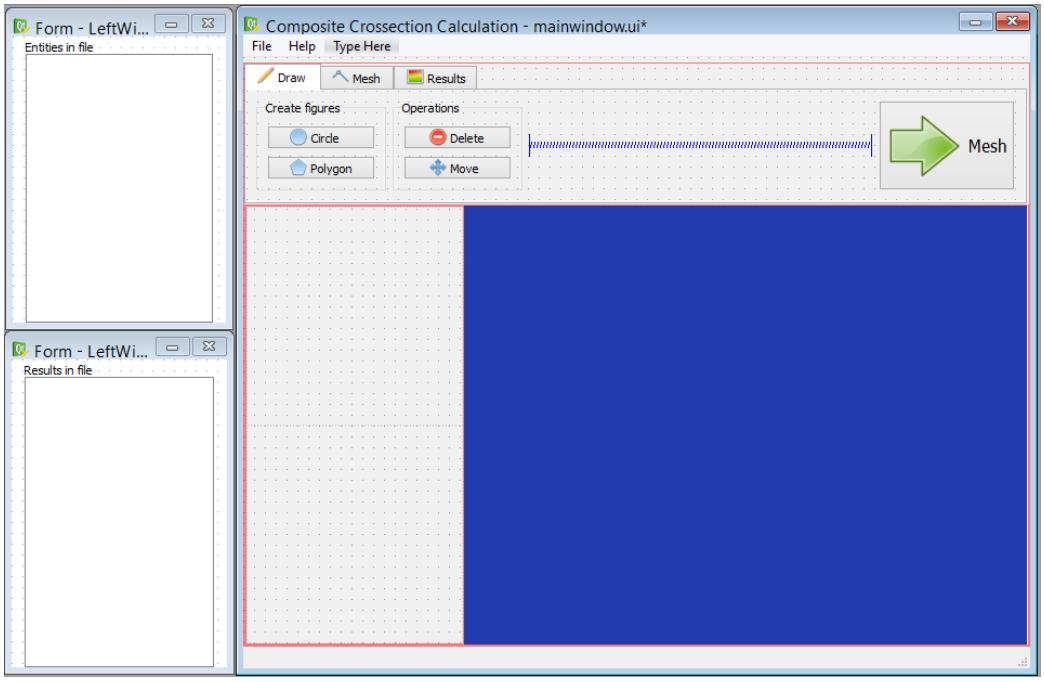
Utilities Filteret inneholder verktøyklasser. Dette er klasser som ofte er ekstrahert ut av enten en modell eller kontroller for å gjøre dem mer lesbare. Klassene inneholder logikk eller definisjoner, og skal kun ha et ansvarsområde. EntityOperations, ExternalProcessHandler, ShortcutManager og BRepFactory er eksempler på slike verktøyklasser.

ViewController ViewController er delt opp i flere underkategorier ettersom hvilken del av brukergrensesnittet de styrer. Det er for å skille mellom klasser brukt i tegnevisning og resultatvisning. Kontekstmenyer og dialoger er også samlet i hver sine filtere. ContextMenu inneholder

kode for å styre kontekstmenyene, mens Dialogs inneholder kode for å sette opp og kontrollere dialogvinduet. Begge disse har fått egne filtere fordi de ikke nødvendigvis er bundet til et visningmodus, samtidig som at det er antatt at antallet kommer til å øke etterhvert i utviklingen.

E.3 Brukergrensesnittets oppbygning

Brukergrensesnittet er bygget opp med Qt-Designer ved hjelp av grafiske elementer kalt «widget» som plasseres i vinduet. Elementene holdes på plass av en «layout» som spesifiserer hvordan elementene skal ligge i forhold til hverandre. Det er mulig å bruke vertikal, horisontal, grid og form layout.
Fordelen med bruk av widgets er at brukergrensesnittet kan modulariseres. Modularisering reduserer blant annet utviklingstiden fordi det åpner for gjenbruk av design. Designet kan også bli ryddigere fordi kompliserte deler kan ekstraheres og designes som egne filer. Sistnevnte er gjort for entitetstreet og resultatlisten. Når applikasjonen er i tegnevisningen fyller widgeten med entitetstreet hele venstremenyen, men blir erstattet av resultatwidgeten i resultatvisningen. Tilnærmingen med widgets gjør det mye klarere å designe innholdet i menyene som skiftes ut fordi de designes i hvert sitt vindu i Qt-Designer(Se figur E.1).
Det er brukt en «Spacer» i toppmenyen for å gjøre at knappene holder seg langs kantene når størrelsen på programvinduet endres.



Figur E.1: Skjermbilde av brukergrensesnittets oppbygning i Qt-Designer. Venstremenyene er skilt ut som to egne «Widgets» for å enklere kunne designes, og ligger referert hovedvinduet. De ligger lagret som egne filer, og designes i hvert sitt vindu i Qt-Designer.

Tillegg F

Doxygendokumentasjon

CCCP-UserInterface

Generated by Doxygen 1.8.6

Tue Jun 10 2014 10:51:53

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

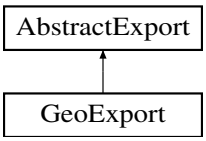
Class Documentation

3.1 AbstractExport Class Reference

Abstract class providing a general interface for data extracted from the draw model.

`#include <AbstractExport.h>`

Inheritance diagram for AbstractExport:



Public Member Functions

- **AbstractExport** (DrawModel ∗)
- bool open (std::string)
- void save ()

Protected Member Functions

- int findGlobalNodeNumber (Node ∗)

Protected Attributes

- std::ofstream file

  File stream with data written to file.
- std::vector< Node ∗ > globalNodes

  All nodes in model reenumerated with a global value.
- std::vector< Edge ∗ > globalEdges

  All edges in model reenumerated with a global value.
- std::vector< std::vector< int > > lineloops

  Line loops/wires in model.
- std::vector< int > surfaces

  Surfaces in the model.
- std::map< Node ∗, int > nodeLookupTable

  Table containing mapping a node reference to its global number.

3.1.1 Detailed Description

Abstract class providing a general interface for data extracted from the draw model.

It is done this way to make extended file format support easier.

It exports all nodes and edges present in the model. The edges and nodes are numbered within each node, and the class reenumerate these into a "global" system.

3.1.2 Member Function Documentation

3.1.2.1 int AbstractExport::findGlobalNodeNumber (Node * *node*) [protected]

Find the global node number given a node reference, and an already computed map with node references and global number.

3.1.2.2 bool AbstractExport::open (std::string *fileName*)

Opens an output file stream to the given filename. Everything sent to this file stream is saved to the file.

3.1.2.3 void AbstractExport::save ( )

Close the opened file stream, releasing the write lock.

The documentation for this class was generated from the following files:

- UserInterface/AbstractExport.h
- UserInterface/AbstractExport.cpp

3.2 BRepFactory Class Reference

An adapter connecting the application with openCASCADE, used when boolean operations are executed.

`#include <BRepFactory.h>`

Public Member Functions

- BRepFactory (Entity *)
- void addCSGOperation (Entity *, int)
- void perform ()
- std::list< Entity * > getEntities ()
- std::vector< Edge * > getEdges ()

Public Attributes

- std::vector< TopoDS_Shape > faces

  Vector holding one or more faces to be performed operations on.
- std::vector< int > operations

  vector holding one or more operations to be performed.
- TopoDS_Shape currentFace

  Shape class holding the current faces. Cut operations might create more than one face.

Static Public Attributes

- static const int **SUBTRACT** = 0
- static const int **ADD** = 1

3.2.1 Detailed Description

An adapter connecting the application with openCASCADE, used when boolean operations are executed.

It takes two entities and an operation and returns one or several new entities. The class is only used inside Entity↩ BRep when updating resultEntities.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BRepFactory::BRepFactory (Entity * *entity*)

Adding the main entity on which operations are performed.

3.2.3 Member Function Documentation

3.2.3.1 void BRepFactory::addCSGOperation (Entity * *entity*, int *operation*)

Adds secondary entity and operation to the list of entities that are used to either add or cut the main entity. Nothing is done before perform() is called.

3.2.3.2 std::vector< Edge * > BRepFactory::getEdges ( )

Gets all edges after the boolean operation. Only used in tests to verify result before data is exported to entities.

Returns

vector<Edge*> Edges in face after boolean operation

3.2.3.3 std::list< Entity * > BRepFactory::getEntities ( )

Converting the currentFace to entities. It loops through the wires

Returns

list<Entity*> Containing entities resulting from the boolean operations

3.2.3.4 void BRepFactory::perform ( )

Loops through faces and either adds or subtracts the face from the currentFace

The documentation for this class was generated from the following files:

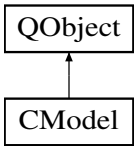
- UserInterface/BRepFactory.h
- UserInterface/BRepFactory.cpp

3.3 CModel Class Reference

The main model used in the application.

`#include <CModel.h>`

Inheritance diagram for CModel:



Public Slots

- void setDataModelListeners ()
- void setMeshOutOfSync ()
- void setResultOutOfSync ()
- void executeMeshProcess ()
- void executeSolveProcess ()
- void moveToTemporaryFolder (QString filename, QString tmpFileEnding)
- void openMSHFile (QString)
- void openVTKFile (QString)
- void resetMeshOptions ()

Signals

- void newDrawModel (DrawModel ∗)

  Emitted when a new draw model is created.
- void drawMode ()

  Emitted when the UI mode is changed to draw mode.
- void meshMode ()

  Emitted when the UI mode is changed to mesh mode.
- void resultMode ()

  Emitted when the UI mode is changed to result mode.
- void uiModeChanged ()

  Emitted whenever the UI mode is changed.

Public Member Functions

- **CModel** (QObject ∗parent=0)
- int getActiveUIMode ()
- void setUIMode (int)
- void reset ()
- void exportDataModel (QString)
- QStringList getMeshOptions ()
- void setMinElementSize (QString)
- void setMaxElementSize (QString)
- void setCustomArguments (QString)
- QString getMinElementSize ()
- QString getMaxElementSize ()

Public Attributes

- QString fileName

  Current fileName opened.
- QString **tmpPath**
- DrawModel * drawModel

  Data model for drawing and storing entities.
- ResultModel * resultModel

  Data model holding mesh and result data.
- bool meshOutOfSync

  The validity of the mesh. False whenever changes are done to entities after meshing.
- bool **resultOutOfSync**

Static Public Attributes

- static const int **DRAWMODE** = 0
- static const int **MESHMODE** = 1
- static const int **RESULTMODE** = 2

3.3.1 Detailed Description

The main model used in the application.

3.3.2 Member Function Documentation

3.3.2.1 void CModel::executeMeshProcess ( ) [slot]

Starts the meshing. It creates an external process manager to controll the mesh process and exports the draw data model to a temporary file "tmp/tmp_file.geo". The external process handler monitors the external process and emits a signal when it is done.

3.3.2.2 void CModel::executeSolveProcess ( ) [slot]

Starts the solver. It creates an external process manager to controll the solver process and passes the mesh file name to the solver. The external process handler monitors the external process and emits a signal when it is done.

3.3.2.3 void CModel::exportDataModel (QString *fileName*)

Creates an instance of GeoExport which saves the geometry data in drawModel to a file

3.3.2.4 int CModel::getActiveUIMode ( )

Returns

integer representing the current UI mode

3.3.2.5 QString CModel::getMaxElementSize ( )

Returns

QString Maximum element size used in meshing

3.3.2.6 QStringList CModel::getMeshOptions ( )

Creates a QStringList based on the mesh settings configured in text boxes, in the UI.

Returns

QStringList with arguments to be passed to the mesh application

3.3.2.7 QString CModel::getMinElementSize ( )

Returns

QString Minimum element size used in meshing

3.3.2.8 void CModel::moveToTemporaryFolder (QString *filename,* QString *tmpFileEnding*) [slot]

Moves a file from a path to the temporary folder with the given file type ending

3.3.2.9 void CModel::openMSHFile (QString *fileName*) [slot]

Opens and loads a mesh file using MSHParser. The parser is set as a mesh data source in the result model.

3.3.2.10 void CModel::openVTKFile (QString *fileName*) [slot]

Opens and loads a result file using a VTKParser. The data is set as a result data soure in the result model.

3.3.2.11 void CModel::reset ( )

Deletes the data models and creates new ones.

3.3.2.12 void CModel::resetMeshOptions ( ) [slot]

Sets the mesh to outofsync and reverts the mesh options to default values defined in this function.

Default values are: MinElement: 0, MaxElement: 1e+22

3.3.2.13 void CModel::setCustomArguments (QString *customArgs*)

Sets the model's custom arguments used in meshing

3.3.2.14 void CModel::setDataModelListeners ( ) [slot]

Connects the data model signals to this model.

3.3.2.15 void CModel::setMaxElementSize (QString *maxElementSize*)

Sets the model's maximum element size for meshing

3.3.2.16 void CModel::setMeshOutOfSync ( ) [slot]

Set the mesh state to out of sync with the geometry. This function is called whenever an entity is added, changed or deleted from the draw model.

3.3.2.17 void CModel::setMinElementSize (QString *minElementSize*)

Sets the model's minimum element size for meshing

3.3.2.18 void CModel::setResultOutOfSync ( ) [slot]

Set the result state to out of sync with the mesh. This function is called whenever a new mesh is loaded or genereated.

3.3.2.19 void CModel::setUIMode (int *mode*)

Sets the current UI mode and emits signal based on the given UI mode.

The documentation for this class was generated from the following files:

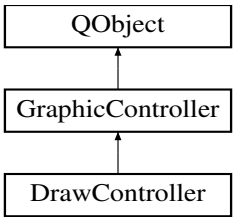
- UserInterface/CModel.h
- UserInterface/CModel.cpp

3.4 DrawController Class Reference

A controller changing the style of GraphicsView, and lets the user draw shapes to the program.

`#include <DrawController.h>`

Inheritance diagram for DrawController:



Public Slots

- void setActiveEntity (Entity ∗)
- void reset ()
- void setModel (DrawModel ∗DrawModel)
- void doneEvent ()
- void cancelEvent ()
- void actorClicked (vtkSmartPointer< vtkActor >)
- void actorDoubleClicked (vtkSmartPointer< vtkActor >)

Public Member Functions

- **DrawController** (DrawModel ∗, QWidget ∗, QObject ∗parent=0)
- void connectView ()
- void invalidate ()
- void drawPointsAndLines (std::vector< Node ∗ >, std::vector< Edge ∗ >)
- void updateInteraction (Entity ∗)
- void drawSurfaces ()
- void createSurface (Entity ∗)

Public Attributes

- DrawModel * **model**

3.4.1 Detailed Description

A controller changing the style of GraphicsView, and lets the user draw shapes to the program.

It sets up Graphicsview to use 2D, and adds a seedWidget to the GraphicsView

3.4.2 Member Function Documentation

3.4.2.1 void DrawController::actorClicked (vtkSmartPointer< vtkActor > *pickedActor*) [slot]

Find the active entity based on the actor selected.

3.4.2.2 void DrawController::actorDoubleClicked (vtkSmartPointer< vtkActor > *pickedActor*) [slot]

Find the active entity based on the actor selected, and unfinalizes it. The seeds are then shown in the seed widget, and the entity is editable.

3.4.2.3 void DrawController::cancelEvent ( ) [slot]

Deletes the active entity

3.4.2.4 void DrawController::connectView ( )

Connect the view to this controller, so the controller can handle the neccessary user interactions applied to the view.

3.4.2.5 void DrawController::createSurface (Entity * *entity*)

Create and add a surface actor given an entity. It also handles the color and visibility of entities, given their state. (Active, regular, hole or invisible)

3.4.2.6 void DrawController::doneEvent ( ) [slot]

Done event will cause the view to finalized the active entity. Finalizing the active entity will close it if not closed and remove the seeds from the view.

3.4.2.7 void DrawController::drawPointsAndLines (std::vector< Node * > *nodes*, std::vector< Edge * > *edges*)

Draws all the nodes and edges present in the draw model.

3.4.2.8 void DrawController::drawSurfaces ( )

Create surfaces for all closed entities in draw model.

3.4.2.9 void DrawController::invalidate ( )

Resets the GraphicsView and draws lines, points and surfaces.

3.4.2.10 void DrawController::reset ( ) [slot]

Invalidates the view and sets the seed widgets data model to the selected entity if the active entity is not finalized.

3.4.2.11 void DrawController::setActiveEntity (Entity * *activeEntity*) [slot]

Updates the callback for the seedwidget, and update the seedwidgets interaction mode. Then the view is reset.

3.4.2.12 void DrawController::setModel (DrawModel * *DrawModel*) [slot]

Sets the active draw model used to draw entities, and connects the model to the controller.

3.4.2.13 void DrawController::updateInteraction (Entity * *activeEntity*)

Updates the seed widget interaction mode. Used when a new entity is selected to Enable/disable "add seed" feature. Enabled if entity is selected, and the entity is closed. Otherwise disabled.

The documentation for this class was generated from the following files:

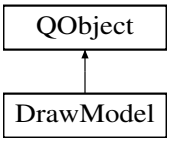
- UserInterface/DrawController.h
- UserInterface/DrawController.cpp

3.5 DrawModel Class Reference

Data model containing entity data and functions to manipulate entity data.

`#include <DrawModel.h>`

Inheritance diagram for DrawModel:



Public Slots

- void uiModeChanged ()
- void setEntityChanged (Entity *)
- void finalizeActiveEntity ()
- void setActiveEntityNULL ()
- void executeEntityOperation (EntityOperation *eOperation)
- void showProperties (Entity *)

Signals

- void entityCreated (Entity *)

  Emitted when a entity is created and added to the model.
- void entityChanged (Entity *)

  Emitted when a entity in the model is changed.
- void entityDeleted (Entity *)

Emitted when a entity in the model is removed.

- void activeEntityChanged (Entity *)

  Emitted when the active/selected entity is changed.

Public Member Functions

- **DrawModel** (QObject *parent)
- std::vector< Node * > getAllNodes ()
- std::vector< Edge * > getAllEdges ()
- void createEntityPolygon ()
- void createEntityCircle ()
- void addEntity (Entity *)
- void setActiveInput (Entity *)
- void deleteActiveEntity ()
- void deleteEntity (Entity *)
- void removeEntity (Entity *)
- bool isActiveEntity (Entity *)
- bool isNotActiveEntity (Entity *)

Public Attributes

- Entity * activeEntity

  Reference to the selected entity. It is active for input. NULL if none selected.

- bool activeEntityFinalized

  State of active entity. The seed widget is shown if false.

- std::list< Entity * > entities

  Entities in model.

3.5.1 Detailed Description

Data model containing entity data and functions to manipulate entity data.

It also keeps track of the selected entity. Because the model manipulates the entity data, it also supply and implements the tools made for drawing sections. (Entity operations)

3.5.2 Member Function Documentation

3.5.2.1 void DrawModel::addEntity (Entity * *entity*)

Adds the given entity to the list of entities and connects the entity signals to the model. It also assigns the entity a depth level to which it is rendered. The added entity is always set as the selected entity.

3.5.2.2 void DrawModel::createEntityCircle ( )

Creates an empty entity of type circle and adds the entity to the model

3.5.2.3 void DrawModel::createEntityPolygon ( )

Creates an empty entity of type polygon and adds the entity to the model

3.5.2.4 void DrawModel::deleteActiveEntity ( )

Deletes the active/selected entity.

3.5.2.5 void DrawModel::deleteEntity (Entity * *entity*)

Removes the given entity from the model and deletes it.

3.5.2.6 void DrawModel::executeEntityOperation (EntityOperation * *eOperation*) [slot]

Listener for entity operations. Called if the correct sequence is completed. It creates an entity boundary representation based on data extracted from the entity operation.

The entity boundary representation is added to the model, and the entities now part of the new entity is removed from the model.

3.5.2.7 void DrawModel::finalizeActiveEntity ( ) [slot]

Function closes active entity and unselects it.

3.5.2.8 std::vector< Edge * > DrawModel::getAllEdges ( )

Loops trough all entities creating a vector of all the edges

Returns

vector<Node*> All edges in entities

3.5.2.9 std::vector< Node * > DrawModel::getAllNodes ( )

Loops trough all entities creating a vector of all the nodes

Returns

vector<Node*> All nodes in entities

3.5.2.10 bool DrawModel::isActiveEntity (Entity * *entity*)

Checks wether the given entity is selected.

Returns

bool True if active entity

3.5.2.11 bool DrawModel::isNotActiveEntity (Entity * *entity*)

Checks wether the given entity is not selected.

Returns

bool True if NOT active entity

3.5.2.12 void DrawModel::removeEntity (Entity * *entity*)

Removes the given entity from the model and updates the entity depth levels of the remaining entities.

3.5.2.13 void DrawModel::setActiveEntityNULL ( ) [slot]

Unselects any active/selected entity

3.5.2.14 void DrawModel::setActiveInput (Entity * *entity*)

Sets the active/selected entity to the given entity.

3.5.2.15 void DrawModel::setEntityChanged (Entity * *entity*) [slot]

Listens for changes in any entity and fire entity changed signal

3.5.2.16 void DrawModel::showProperties (Entity * *entity*) [slot]

Display property window for an entity. The entity to display is either given by an argument or by the active entity set in model.

3.5.2.17 void DrawModel::uiModeChanged ( ) [slot]

Called whenever UI mode changes. It unselects any selected entity and deletes all unfinished(not closed) entities.

The documentation for this class was generated from the following files:

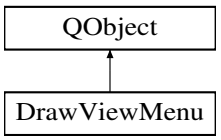
- UserInterface/DrawModel.h
- UserInterface/DrawModel.cpp

3.6 DrawViewMenu Class Reference

Context menu shown in the GraphcsView in draw mode after the user right click on an entity.

`#include <DrawViewMenu.h>`

Inheritance diagram for DrawViewMenu:



Public Slots

- void ShowContextMenu (const QPoint &pos)

Signals

- void newAction (int operation)

  Emitted when an operation is selected.
- void propertiesSelected (Entity ∗)

  Emitted when properties is selected.

Public Member Functions

- **DrawViewMenu** (QWidget ∗, EntityOperation ∗, DrawModel ∗, QObject ∗parent)

3.6.1 Detailed Description

Context menu shown in the GraphcsView in draw mode after the user right click on an entity.

3.6.2 Member Function Documentation

3.6.2.1 void DrawViewMenu::ShowContextMenu (const QPoint & *pos*) [slot]

Adds possible actions to the menu and displays it at the given postion.

The documentation for this class was generated from the following files:

- UserInterface/DrawViewMenu.h
- UserInterface/DrawViewMenu.cpp

3.7 Edge Class Reference

A data object holding edge data.

`#include <Edge.h>`

Public Member Functions

- **Edge** (Node ∗node1, Node ∗node2)

Public Attributes

- Node ∗ n1

  Node 1 in edge.
- Node ∗ n2

  Node 2 in edge.

3.7.1 Detailed Description

A data object holding edge data.

The documentation for this class was generated from the following files:

- UserInterface/Edge.h
- UserInterface/Edge.cpp

3.8 Element Class Reference

A data object contains element data and relevant functions.

`#include <Element.h>`

Public Member Functions

- void calculateArea ()
- void calculateAreaCentre ()
- double getSecondAreaMomentX (double)
- double getSecondAreaMomentY (double)
- double getSecondAreaMomentProduct (double, double)

Public Attributes

- Node * **i**
- Node * **j**
- Node * **k**
- double area

  Elements total area.
- double material

  Young's modulus for the element.
- double Iex

  Elements second area moment x.
- double Iey

  Elements second area moment y.
- double Iexy

  Elements second area moment product.
- double **xI1**
- double **xI2**
- double **xI3**
- double **yI1**
- double **yI2**
- double **yI3**
- double **ecx**
- double ecy

  Elements area centre.
- double **dx**
- double dy

  Distance between mesh- and node area centre.
- double **x1**
- double **x2**
- double x3

  Elements nodepositions in x-directions.
- double **y1**
- double **y2**
- double y3

  Elements nodepositions in x-directions.

3.8.1 Detailed Description

A data object contains element data and relevant functions.

3.8.2 Member Function Documentation

3.8.2.1 void Element::calculateArea ( )

Calculates the area of the element.

$A_e = \frac{1}{2} \cdot |J|$

3.8.2.2 void Element::calculateAreaCentre ( )

Calculates the area centre of the element.

$ec_x = \frac{x1+x2+x3}{3}$

$ec_y = \frac{y1+y2+y3}{3}$

3.8.2.3 double Element::getSecondAreaMomentProduct (double *Ax*, double *Ay*)

Calculating the product of moment of inertia

$dx_e = Ac_x - Ac_{ex}$

$dy_e = Ac_y - Ac_{ey}$

$I_{xy} = \sum (I_{xy_e} + dx_e * dy_e * A_e)$

3.8.2.4 double Element::getSecondAreaMomentX (double *Ay*)

Calculates the second moment of area for the element.

Parameters

Ay	Area centre of the mesh

$dy_e = A_y - Ac_{ey}$

$I_x = \sum (I_{x_e} + dy_e^2 * A_e)$

3.8.2.5 double Element::getSecondAreaMomentY (double *Ax*)

Calculates the second moment of area for the element.

Parameters

Ax	Area centre of the mesh

$dx_e = A_x - Ac_{ex}$

$I_y = \sum (I_{y_e} + dx_e^2 * A_e)$

The documentation for this class was generated from the following files:

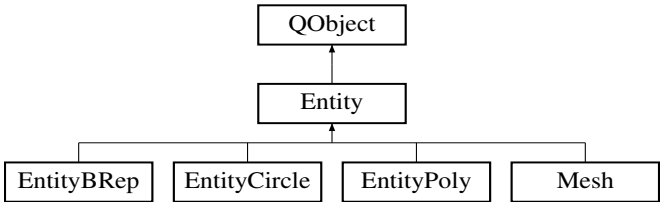
- UserInterface/Element.h
- UserInterface/Element.cpp

3.9 Entity Class Reference

An abstract class containing methods and datafields required by all types of entities.

`#include <Entity.h>`

Inheritance diagram for Entity:



Public Member Functions

- **Entity** (QObject ∗parent)
- void setName (std::string)
- void setType (int)
- virtual void changeNode (double pos[], int)=0
- virtual void createNode (double pos[], int)=0
- virtual std::vector< Node ∗ > getSeeds ()=0
- virtual void updatePolygon ()
- void close ()
- bool isClosed ()
- bool isHole ()

Public Attributes

- int type

  Type of element. Polygon, Circle, Rectangle or shapes determined by boundary operations.
- std::string name

  Name of entity.
- double material

  Material.
- std::vector< Node ∗ > seeds

  Tangible nodes representing the entity.
- std::list< Node ∗ > nodes

  All nodes in entity.
- std::list< Edge ∗ > edges

  All edges in entity.

Static Public Attributes

- static const int **POLYGON** = 0
- static const int **CIRCLE** = 1
- static const int **RECTANGLE** = 2
- static const int **BREP** = 3

3.9.1 Detailed Description

An abstract class containing methods and datafields required by all types of entities.

It holds information regarding shapes displayed on screen. Nodes, edges and as well as functionality common to all types of shapes.

3.9.2 Member Function Documentation

3.9.2.1 virtual void Entity::changeNode (double *pos[],* int) [pure virtual]

Abstract function required by all entities in order to be editied by the seed widget. When an entity is chosen, the entity is set as the seed widget's callback and this method is called.

Implemented in EntityBRep, Mesh, EntityCircle, and EntityPoly.

3.9.2.2 void Entity::close ( )

Closes the entity if not already closed. It adds an edge from the last node to the first, and sets isClosed = true.

3.9.2.3 virtual void Entity::createNode (double *pos[],* int) [pure virtual]

Abstract function required by all entities in order to be created. When an empty entity is created, the entity is set as the seed widget's callback and this method is called.

Implemented in EntityBRep, EntityPoly, Mesh, and EntityCircle.

3.9.2.4 virtual std::vector<Node*> Entity::getSeeds ( ) [pure virtual]

Abstract function required by all entities in order to be drawn in the seed widget. The seeds are a sub set of nodes which can be moved.

Implemented in EntityBRep, EntityPoly, EntityCircle, and Mesh.

3.9.2.5 bool Entity::isClosed ( )

Returns true if the entity is closed. False otherwise

3.9.2.6 bool Entity::isHole ( )

Returns wether the entity is a hole or not. This property is only set to true if a boundary operation has been executed.

3.9.2.7 void Entity::setName (std::string *name*)

Sets the name of the entity

3.9.2.8 void Entity::setType (int *type*)

Sets the entitytype

3.9.2.9 void Entity::updatePolygon ( ) [virtual]

Abstract function required by all entities in order to be displayed by the GraphicView Updates the PolyData and the actor.

Reimplemented in Mesh, and EntityBRep.

The documentation for this class was generated from the following files:

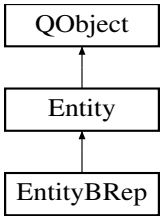
- UserInterface/Entity.h
- UserInterface/Entity.cpp

3.10 EntityBRep Class Reference

A data object used to hold entity data made up of several other entities.

`#include <EntityBRep.h>`

Inheritance diagram for EntityBRep:



Public Slots

- void setRequireUpdate ()

Public Member Functions

- **EntityBRep** (Entity ∗, Entity ∗, int, QObject ∗parent=0)
- bool safeConvertToEditableEntity ()
- void updatePolygon ()
- vtkSmartPointer< vtkPolyData > getPolyData ()
- void changeNode (double pos[], int)
- void createNode (double pos[], int)

  Implemented and ignored entity functionality.
- std::vector< Node ∗ > getSeeds ()

  Implemented and ignored entity functionality.

Public Attributes

- bool **updateRequired**
- std::vector< Entity ∗ > childEntities

  Child entities. The entities whos boundaries make this entity.
- std::vector< int > operations

  Operations executed on the child entities.
- std::list< Entity ∗ > resultEntities

  Result entities. A boundary represented entity can consist of multiple entities if a operation splits one of the child entities.
- std::list< Entity ∗ > resultHoleEntities

  Result entities which are holes in the BRep entity.

Static Public Attributes

- static const int **SUBTRACT** = 0
- static const int **ADD** = 1

3.10.1 Detailed Description

A data object used to hold entity data made up of several other entities.

Entity boundary representation is created by boolean operations on multiple entities. The entities making up the boundary represented entity is added as child entities. It keep track of the child entitites's state and execute the boolean operation again if they've changed since last update.

3.10.2 Member Function Documentation

3.10.2.1 void EntityBRep::changeNode (double *pos[]*, int) [inline], [virtual]

Abstract function required by all entities in order to be editied by the seed widget. When an entity is chosen, the entity is set as the seed widget's callback and this method is called.

Implements Entity.

3.10.2.2 vtkSmartPointer< vtkPolyData > EntityBRep::getPolyData ( )

Creates a polygon for each result entity.

3.10.2.3 std::vector< Node * > EntityBRep::getSeeds ( ) [virtual]

Implemented and ignored entity functionality.

If seeds from childEntities are returned here, the seed references will point to this entity.

Returns

Always a empty vector of seeds.

Implements Entity.

3.10.2.4 bool EntityBRep::safeConvertToEditableEntity ( )

Verify that the entity has the properties required to be converted to a regular entity.

Returns

True if BRep entity safely can be converted. False otherwise

3.10.2.5 void EntityBRep::setRequireUpdate ( ) [slot]

Force the boolean operation to be redone if child entities change

3.10.2.6 void EntityBRep::updatePolygon ( ) [virtual]

Overridden Entity function to enable the entity to also update the child entities. The BRep entity actor is based on polygons retrieved from the result entities. It is done this way to use one clickable actor for the whole boundary representation.

Reimplemented from Entity.

The documentation for this class was generated from the following files:

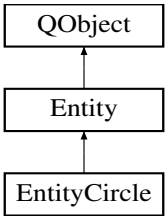
- UserInterface/EntityBRep.h
- UserInterface/EntityBRep.cpp

3.11 EntityCircle Class Reference

An type of entity that represents a circle shape. It extends the abstract class entity, which hold most of the data.

`#include <EntityCircle.h>`

Inheritance diagram for EntityCircle:



Public Member Functions

- **EntityCircle** (QObject *parent=0)
- void createNode (double[], int)
- void changeNode (double[], int)
- std::vector< Node * > getSeeds ()
- void createCircleBorder ()
- Node * getCentre ()
- Node * getEdge ()
- double getRadius ()

Additional Inherited Members

3.11.1 Detailed Description

An type of entity that represents a circle shape. It extends the abstract class entity, which hold most of the data.

3.11.2 Member Function Documentation

3.11.2.1 void EntityCircle::changeNode (double *pos[]*, int *handle*) [virtual]

Changes the node and updates the circle edges.

Implements Entity.

3.11.2.2 void EntityCircle::createCircleBorder ( )

Clears the current nodes and edges. Calculates the circle radius by: $dx = Seed_{1x} - Seed_{2x}$

$dy = Seed_{1y} - Seed_{2y}$

Positions for the nodes along the circle is then calculated by:

$Node(i)_x = getCentre() -> x + r * cos(startAngle + stepSize * i)$,

$Node(i)_y = getCentre() -> y + r * sin(startAngle + stepSize * i)$

with step size being a size calculated by;

$stepSize = 2 * \frac{\pi}{numberOfSteps}$

3.11.2.3 void EntityCircle::createNode (double *pos[],* int *handle*) [virtual]

Creates a seed if the entity has less than two seeds.

If first seed; create seed.

If second seed; create seed and create circle edges based on the positon of the two seeds.

Implements Entity.

3.11.2.4 Node * EntityCircle::getCentre ( )

Returns

the first seed representing the centre of the circle.

3.11.2.5 Node * EntityCircle::getEdge ( )

Returns

the second seed representing the position of the circles edge

3.11.2.6 double EntityCircle::getRadius ( )

Calculates the radius of the circle, given the two seeds defining the circle.

3.11.2.7 std::vector< Node * > EntityCircle::getSeeds ( ) [virtual]

Returns

vector<Node*> containing the two seeds used to control a circle shape

Implements Entity.

The documentation for this class was generated from the following files:

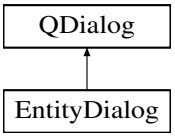
- UserInterface/EntityCircle.h
- UserInterface/EntityCircle.cpp

3.12 EntityDialog Class Reference

Controller for the propery dialog box used to spesify entity properties.

`#include <EntityDialog.h>`

Inheritance diagram for EntityDialog:



Public Slots

- void accept ()
- void reject ()
- void apply ()

Public Member Functions

- **EntityDialog** (QDialog *parent=0)
- void setDataSource (Entity *)

3.12.1 Detailed Description

Controller for the propery dialog box used to spesify entity properties.

It has a table populated with the entity'ss seed positions on x and y axes. It can also be used to spesify material properties for entities.

3.12.2 Member Function Documentation

3.12.2.1 void EntityDialog::accept ( ) [slot]

Applies the changes and closes the dialog window

3.12.2.2 void EntityDialog::apply ( ) [slot]

Function loops through the seeds and compare the entity's values with the ones in the table. If the difference is larger then a certain precision criterium, the entity is updated.

3.12.2.3 void EntityDialog::reject ( ) [slot]

Closes the window

3.12.2.4 void EntityDialog::setDataSource (Entity * *entity*)

Sets the entity and populates the property window.

The documentation for this class was generated from the following files:

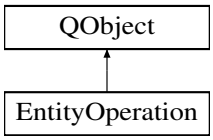
- UserInterface/EntityDialog.h
- UserInterface/EntityDialog.cpp

3.13 EntityOperation Class Reference

The class is an independent operations tracker that is connected to the model.

`#include <EntityOperation.h>`

Inheritance diagram for EntityOperation:



Public Slots

- void newSelection (Entity ∗entity)
- void setEntityOperation (int operation)
- void removeEntity (Entity ∗)

Signals

- void execute (EntityOperation ∗)

  emitted if two entities and an operation are selected

Public Member Functions

- **EntityOperation** (QObject ∗, QObject ∗parent)
- bool hasValidOperation ()
- void resetOperation ()

Public Attributes

- Entity ∗ entity1

  Reference to the first entity selected.
- Entity ∗ entity2

  Reference to the second and any later entity until operation is reset or executed.
- int operation

  Active operation selected.
- bool locked

  Operation is locked while it is executed.

3.13.1 Detailed Description

The class is an independent operations tracker that is connected to the model.

It is extracted into a separate class to better control the feature, and make use of Qt's signal/slots to be loosly coupled from the model. The class, when active listens for user selections. Entity and operation selections are saved until a click outside an entity is carried out. If two entities and an operation are selected, an execute signal is emitted, and the operation is executed by the model.

3.13.2 Member Function Documentation

3.13.2.1 bool EntityOperation::hasValidOperation ( )

Check to verify that all data is set to carry out an operation

Returns

bool. True if all data is set. False otherwise

3.13.2.2 void EntityOperation::newSelection (Entity * *entity*) [slot]

Adds the given entity to the operation if it matches the criterias. It can't be NULL (NULL resets the operation), and not already selected.

3.13.2.3 void EntityOperation::removeEntity (Entity * *entity*) [slot]

Called if an entity is deleted, and removes the entity from operation.

3.13.2.4 void EntityOperation::resetOperation ( )

Reset the operation. Sets entity references to NULL, and operation to -1.

3.13.2.5 void EntityOperation::setEntityOperation (int *operation*) [slot]

Sets the active operation to be executed if two entities are selected.

The documentation for this class was generated from the following files:

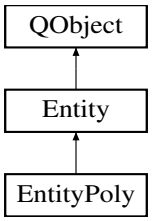
- UserInterface/EntityOperation.h
- UserInterface/EntityOperation.cpp

3.14 EntityPoly Class Reference

A type of entity representing a polygon shape.

`#include <EntityPoly.h>`

Inheritance diagram for EntityPoly:



Public Member Functions

- **EntityPoly** (QObject *parent=0)
- void changeNode (double pos[], int)
- void clickNode (int)
- void createNode (double pos[], int)
- void createEdge (Node *, Node *)
- std::vector< Node * > getSeeds ()

Additional Inherited Members

3.14.1 Detailed Description

A type of entity representing a polygon shape.

It extends the abstract class entity, which hold most of the data entity data

3.14.2 Member Function Documentation

3.14.2.1 void EntityPoly::changeNode (double *pos[],* int *handle*) [virtual]

Changes one of the entity's nodes based on the node id, and emits entityChanged

Implements Entity.

3.14.2.2 void EntityPoly::clickNode (int *handle*)

Listens for clicks on seeds in entity. Closes the entity if the first seed is clicked and the entity is not already closed.

3.14.2.3 void EntityPoly::createEdge (Node *, Node *)

Creates an edge between the two given, and adds the edge to the entity.

3.14.2.4 void EntityPoly::createNode (double *pos[],* int *handle*) [virtual]

Adds a node to the entity and emits entityChanged.

Implements Entity.

3.14.2.5 std::vector< Node * > EntityPoly::getSeeds ( ) [virtual]

Returns

vector<Node*> containing the entity's seeds (This implies all nodes for EntityPolygon)

Implements Entity.

The documentation for this class was generated from the following files:

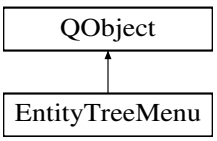
- UserInterface/EntityPoly.h
- UserInterface/EntityPoly.cpp

3.15 EntityTreeMenu Class Reference

Context menu displayed in the entity tree widget in draw mode. It is displayed when the user right click on an entity in the tree.

`#include <EntityTreeMenu.h>`

Inheritance diagram for EntityTreeMenu:



Signals

- void propertiesSelected ()

  Emitted the properties is clicked in the context menu.

Public Member Functions

- **EntityTreeMenu** (QWidget ∗, QObject ∗)
- void show (const QPoint &point)

Public Attributes

- QWidget ∗ **view**

3.15.1 Detailed Description

Context menu displayed in the entity tree widget in draw mode. It is displayed when the user right click on an entity in the tree.

3.15.2 Member Function Documentation

3.15.2.1 void EntityTreeMenu::show (const QPoint & *point*)

Adds "properties" to the menu and displays the menu at the given position

The documentation for this class was generated from the following files:

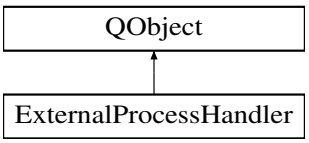
- UserInterface/EntityTreeMenu.h
- UserInterface/EntityTreeMenu.cpp

3.16 ExternalProcessHandler Class Reference

External process handler. It is extraced from CModel to make the process execution clearer.

`#include <ExternalProcessHandler.h>`

Inheritance diagram for ExternalProcessHandler:



Public Slots

- void **meshFinished** ()
- void **solveFinished** ()

Signals

- void **finished** (QString)

Public Member Functions

- **ExternalProcessHandler** (QObject *parent)
- void **setMeshProcessName** (QString)
- void **setSolveProcessName** (QString)
- void **setWorkDir** (QString)
- void **setAdditionalArguments** (QStringList)
- void execMesh ()
- void execSolve ()

3.16.1 Detailed Description

External process handler. It is extraced from CModel to make the process execution clearer.

It was originally planned to keep track of file names used for data exchange between processes. It is one of the last properties implemented, and would probably benefit from some refactoring..

3.16.2 Member Function Documentation

3.16.2.1 void ExternalProcessHandler::execMesh ( )

Adds arguments necessary to run the mesher and starts the process. It also connects a process callback in this class, emitted when the process is done.

3.16.2.2 void ExternalProcessHandler::execSolve ( )

Adds arguments necessary to run the solver and starts the process. It also connects a process callback in this class, emitted when the process is done.

The documentation for this class was generated from the following files:

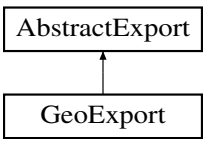
- UserInterface/ExternalProcessHandler.h
- UserInterface/ExternalProcessHandler.cpp

3.17 GeoExport Class Reference

A file writer providing functionaltiy for writing Gmsh's ".geo" format.

`#include <GeoExport.h>`

Inheritance diagram for GeoExport:



Public Member Functions

- **GeoExport** (std::string, DrawModel *)
- void **write** ()

Additional Inherited Members

3.17.1 Detailed Description

A file writer providing functionaltiy for writing Gmsh's ".geo" format.

It contains information about header file structure, but relies on AbstractExport for data export.

The documentation for this class was generated from the following files:

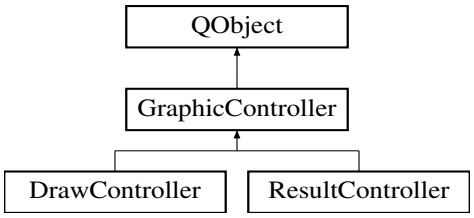
- UserInterface/GeoExport.h
- UserInterface/GeoExport.cpp

3.18 GraphicController Class Reference

An abstract class for controllers, used mostly as an interface for DrawController and ResultController.

`#include <GraphicController.h>`

Inheritance diagram for GraphicController:



Public Member Functions

- GraphicController (QWidget *view, QObject *parent)
- virtual void **actorClicked** (vtkSmartPointer< vtkActor >)=0
- virtual void **actorDoubleClicked** (vtkSmartPointer< vtkActor >)=0

Public Attributes

- GraphicsView * **gView**

3.18.1 Detailed Description

An abstract class for controllers, used mostly as an interface for DrawController and ResultController.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 GraphicController::GraphicController (QWidget * *view,* QObject * *parent*) [inline]

Resets the GraphicView when a new controller is created

The documentation for this class was generated from the following file:

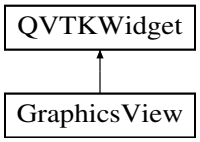
- UserInterface/GraphicController.h

3.19 GraphicsView Class Reference

A wrapper for the VTK, providing the graphics and is represents the blue field viewed in the UI.

`#include <GraphicsView.h>`

Inheritance diagram for GraphicsView:



Public Slots

- void escapePressed ()
- void enterPressed ()
- void reset ()
- void invalidate ()

Signals

- void doneEvent ()

  Closes active entity.
- void cancelEvent ()

  Deletes active entity/unfinished entity.
- void **actorClicked** (vtkSmartPointer< vtkActor >)
- void **actorDoubleClicked** (vtkSmartPointer< vtkActor >)

Public Member Functions

- **GraphicsView** (QWidget *parent)
- void set2DInteractor ()
- void resetInteractor ()
- void addRenderer (vtkSmartPointer< vtkRenderer >)
- void removeRenderer (vtkSmartPointer< vtkRenderer >)
- void createBackground ()
- vtkSmartPointer< vtkRenderer > createRenderer ()
- void removeActors ()
- void addActor (vtkSmartPointer< vtkActor > actor)
- void set2DMode ()
- void set3DMode ()
- void clickCallback ()
- void contextMenuCallback ()

Public Attributes

- double pos [2]

  Position for last click. Used to separate a double click from a regular mouse click.
- vtkSmartPointer< vtkRenderer > renderer

  Renderer used to display all entities and results.
- vtkSmartPointer< vtkCamera > cam2D

Camera configured without perspective.

- vtkSmartPointer< vtkCamera > cam3D

  Camera with perspective.

- std::vector< vtkSmartPointer < vtkActor > > actors

  All actors active in the GraphicsView.

3.19.1 Detailed Description

A wrapper for the VTK, providing the graphics and is represents the blue field viewed in the UI.

It extends QVTKWidget to ease control of 2D and 3D view mode and to better handle the actors. Its responsebility is limited to handle active actors and set up view modes like interaction style and perspective. GraphicsView is only a view controlled by two very different graphic controllers; DrawController and ResultController. This approach of only swapping controllers lets the program switch modes without reinitialize VTK on every mode change.

3.19.2 Member Function Documentation

3.19.2.1 void GraphicsView::addActor (vtkSmartPointer< vtkActor > *actor*)

Adds the given actor to the view and the vector containing all active actors.

3.19.2.2 void GraphicsView::addRenderer (vtkSmartPointer< vtkRenderer > *renderer*)

Adds the given renderer to the renderWindow and updates the widget

3.19.2.3 void GraphicsView::clickCallback ( )

Click callback for mouse clicks done inside the render window. A prop picker is used to check wether an entity actor was clicked given the event position.

Double clicks are implemented by comparing the position of the last with a new. If the distance is less than 0.001, it is a double click.

3.19.2.4 void GraphicsView::contextMenuCallback ( )

Context menu callback for mouse clicks done inside the render window. A prop picker is used to check wether an entity actor was clicked given the event position.

The interactor uses a coordinate system with origo placed in the upper left corner, unlike the contextmenu which uses the bottom left. This means the y-axes is swapped before it is passed on to draw the context menu.

3.19.2.5 void GraphicsView::createBackground ( )

Sets the number of layers/renderers possible in the renderWindow and creates a backgroundrenderer on layer 0. The background is set to a weak gradient of blue. (Default is black)

3.19.2.6 vtkSmartPointer< vtkRenderer > GraphicsView::createRenderer ( )

Creates the renderer used for presentation and assigns it to layer 1 (Background layer is 0)

Returns

vtkSmartPointer<vtkRenderer> Renderer to render entity- and result actors.

3.19.2.7 void GraphicsView::enterPressed ( ) [slot]

Creates a "done" signal based on a keyboard shortcut signal

3.19.2.8 void GraphicsView::escapePressed ( ) [slot]

Creates a cancel/delete signal based on a keyboard shortcut signal

3.19.2.9 void GraphicsView::invalidate ( ) [slot]

Updates the widget

3.19.2.10 void GraphicsView::removeActors ( )

Function loops through all actors and removes them from the view.

3.19.2.11 void GraphicsView::removeRenderer (vtkSmartPointer< vtkRenderer > *renderer*)

Removes the given renderer from the renderWindow and updates the widget

3.19.2.12 void GraphicsView::reset ( ) [slot]

Removes all actors and updates the widget

3.19.2.13 void GraphicsView::resetInteractor ( )

Sets the interactor style to trackball camera. The default interactor style.

3.19.2.14 void GraphicsView::set2DInteractor ( )

Configures the interactor style to use an image interactor style, which removes the rotation capabilities. It also adds listeners for clicks and context menus to the graphics view.

3.19.2.15 void GraphicsView::set2DMode ( )

Sets the renderers active camera to a camera configured without perspective. This is necessary in order to the depth separation of entities and seeds to work.

3.19.2.16 void GraphicsView::set3DMode ( )

Sets the renderers active camera to a regular camera.

The documentation for this class was generated from the following files:

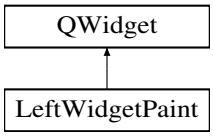
- UserInterface/GraphicsView.h
- UserInterface/GraphicsView.cpp

3.20 LeftWidgetPaint Class Reference

A controller for the tree widget showing the entities in draw model.

`#include <LeftWidgetPaint.h>`

Inheritance diagram for LeftWidgetPaint:



Public Slots

- void showPropertiesWindow ()

Public Member Functions

- **LeftWidgetPaint** (QWidget *parent=0)
- void setModel (DrawModel *)

3.20.1 Detailed Description

A controller for the tree widget showing the entities in draw model.

It lets the user select entities and edit their properties through a context menu

3.20.2 Member Function Documentation

3.20.2.1 void LeftWidgetPaint::setModel (DrawModel * *model*)

Sets the model and connect the click callback to the model.

3.20.2.2 void LeftWidgetPaint::showPropertiesWindow ( ) [slot]

Callback from the context menu. The property window is activated through the draw model, on the entity selected in the tree.

The documentation for this class was generated from the following files:

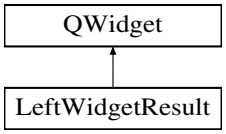
- UserInterface/LeftWidgetPaint.h
- UserInterface/LeftWidgetPaint.cpp

3.21 LeftWidgetResult Class Reference

A controller for the result widget that shows results available in the result model.

`#include <LeftWidgetResult.h>`

Inheritance diagram for LeftWidgetResult:



Public Slots

- void **setModel** (ResultModel ∗)
- void clicked (QListWidgetItem ∗)
- void **updateList** ()

Public Member Functions

- **LeftWidgetResult** (QWidget ∗parent=0)

3.21.1 Detailed Description

A controller for the result widget that shows results available in the result model.

It lets the user change the active result.

3.21.2 Member Function Documentation

3.21.2.1 void LeftWidgetResult::clicked (QListWidgetItem ∗ *current*) [slot]

Sets the model active result name to the name of the list item.

The documentation for this class was generated from the following files:

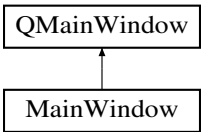
- UserInterface/LeftWidgetResult.h
- UserInterface/LeftWidgetResult.cpp

3.22 MainWindow Class Reference

A controller for the window, and handles the initialization of views and other controllers.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Public Slots

- void addPolygon ()
- void addCircle ()
- void deleteEntity ()

- void openFile ()
- void saveFile ()
- void newFile ()
- void setMode (int)
- void setDrawMode ()
- void setMeshMode ()
- void setResultMode ()
- void showDrawController ()
- void showMeshView ()
- void showResultController ()
- void setMinElementSize ()
- void setMaxElementSize ()
- void setCustomArguments ()
- void resetMeshOptions ()
- void setMeshVisible (bool)
- void setResultVisible (bool)

Signals

- void meshVisible (bool)
- void resultVisible (bool)

Public Member Functions

- **MainWindow** (QWidget *parent=0)

Public Attributes

- CModel * cModel

  Main model. Handles IO operations and contains drawModel and resultModel.
- GraphicController * gController

  Current active controller for VTK/GraphicsView.

3.22.1 Detailed Description

A controller for the window, and handles the initialization of views and other controllers.

This class controls which widgets that are being displayed and handles most of the signals from the UI.

3.22.2 Member Function Documentation

3.22.2.1 void MainWindow::addCircle ( ) [slot]

Calls createEntityCircle in drawModel given a UI signal

3.22.2.2 void MainWindow::addPolygon ( ) [slot]

Calls createEntity in drawModel given a UI signal

3.22.2.3 void MainWindow::deleteEntity ( ) [slot]

Calls deleteActiveEntity in drawModel given a UI signal

3.22.2.4 void MainWindow::meshVisible (bool) [signal]

Signal emitted when setMeshVisible is called by the UI.

3.22.2.5 void MainWindow::newFile ( ) [slot]

Resets the models and redraws the active view

3.22.2.6 void MainWindow::openFile ( ) [slot]

Opens the file dialog box for opening files. It evaluates the file extension and opens the file in the model.

Extensions supported: ".msh" - Mesh file ".vtk" - VTK file (Results)

3.22.2.7 void MainWindow::resetMeshOptions ( ) [slot]

Signal from "Mesh Tab" Resets the model's mesh options and reloads the UI mesh elements given an UI signal.

3.22.2.8 void MainWindow::resultVisible (bool) [signal]

Signal emitted when setResultVisible is called by the UI.

3.22.2.9 void MainWindow::saveFile ( ) [slot]

Opens a file dialog for saving geometry files. Only ".geo" files are supported. ExportDataModel is called on the model, which writes the data.

3.22.2.10 void MainWindow::setCustomArguments ( ) [slot]

Signal from "Mesh Tab" Sets the model's custom arguments that are passed to the mesher.

3.22.2.11 void MainWindow::setDrawMode ( ) [slot]

Sets the model to drawMode given an UI signal

3.22.2.12 void MainWindow::setMaxElementSize ( ) [slot]

Signal from "Mesh Tab" Sets the model's maximum element size used for meshing given an UI signal.

3.22.2.13 void MainWindow::setMeshMode ( ) [slot]

Sets the model to meshMode given an UI signal

3.22.2.14 void MainWindow::setMeshVisible (bool *visible*) [slot]

Signal from "Result Tab" Sets the model's visibility status for mesh and refreshes the ResultController.

3.22.2.15 void MainWindow::setMinElementSize ( ) [slot]

Signal from "Mesh Tab" Sets the model's minimum element size used for meshing given an UI signal.

3.22.2.16 void MainWindow::setMode (int *mode*) [slot]

Sets the mode in the model given a UI signal from the tabWidget

3.22.2.17 void MainWindow::setResultMode ( ) [slot]

Sets the model to resultMode given an UI signal

3.22.2.18 void MainWindow::setResultVisible (bool *visible*) [slot]

Signal from "Result Tab" Sets the model's visibility status for results and refreshes the ResultController.

3.22.2.19 void MainWindow::showDrawController ( ) [slot]

ShowDrawController hides the resultWidget and shows the paintWidget, and set the active tab to draw.

It also removes the active GraphicsController and adds a DrawController -if its not a DrawController. (Graphics↩ Controller is not to be confused with GraphicsView, which never is removed from the window)

3.22.2.20 void MainWindow::showMeshView ( ) [slot]

ShowMeshView hides the paintWidget and shows the resultWidget, and set the active tab to mesh.

It also removes the active GraphicsController and adds a ResultController -if its not a ResultController. (Graphics↩ Controller is not to be confused with GraphicsView, which never is removed from the window)

3.22.2.21 void MainWindow::showResultController ( ) [slot]

ShowMeshView hides the paintWidget and shows the resultWidget, and set the active tab to results.

It also removes the active GraphicsController and adds a ResultController -if its not a ResultController. (Graphics↩ Controller is not to be confused with GraphicsView, which never is removed from the window)

The documentation for this class was generated from the following files:

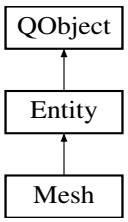
- UserInterface/mainwindow.h
- UserInterface/mainwindow.cpp

3.23 Mesh Class Reference

A data object that holds mesh data.

`#include <Mesh.h>`

Inheritance diagram for Mesh:



Public Member Functions

- **Mesh** (QObject *parent)
- Mesh (QString, std::vector< Node >, std::vector< Element >, QObject *parent)
- void changeNode (double pos[], int handle)
- void createNode (double pos[], int handle)
- std::vector< Node * > getSeeds ()
- std::vector< Element > **getElements** ()
- QString getFileName ()
- int **getNumberOfNodes** ()
- int **getNumberOfElements** ()
- void updatePolygon ()
- vtkSmartPointer< vtkPolyData > getPolyData ()

Additional Inherited Members

3.23.1 Detailed Description

A data object that holds mesh data.

It extends entity to make the drawing more streamlined, and relies upon functions in Entity as far as possible. The resemblence to Entity permits the operation of converting it to a standard entity, to further edit.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 Mesh::Mesh (QString *fileName,* std::vector< Node > *nodes,* std::vector< Element > *elements,* QObject * *parent*)

Creates a Mesh entity that is drawed in ResultController. It copies the nodes and element from the MSHParser as well as the filename.

3.23.3 Member Function Documentation

3.23.3.1 void Mesh::changeNode (double *pos[],* int) [inline], [virtual]

Abstract function required by all entities in order to be editied by the seed widget. When an entity is chosen, the entity is set as the seed widget's callback and this method is called.

Implements Entity.

3.23.3.2 void Mesh::createNode (double *pos[],* int) [inline], [virtual]

Abstract function required by all entities in order to be created. When an empty entity is created, the entity is set as the seed widget's callback and this method is called.

Implements Entity.

3.23.3.3 QString Mesh::getFileName ( )

Returns

QString File name of the mesh loaded.

3.23.3.4 vtkSmartPointer< vtkPolyData > Mesh::getPolyData ( )

Overridden function from Entity. Polydata is created based on nodes and elements instead of nodes and edges.

3.23.3.5 std::vector<Node*> Mesh::getSeeds ( ) [inline], [virtual]

Abstract function required by all entities in order to be drawn in the seed widget. The seeds are a sub set of nodes which can be moved.

Implements Entity.

3.23.3.6 void Mesh::updatePolygon ( ) [virtual]

Overridden function from Entity. UpdatePolygon calls the Mesh's getPolyData() and uses it to update the entityActor.

Reimplemented from Entity.

The documentation for this class was generated from the following files:

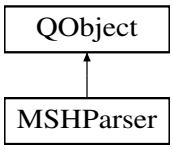
- UserInterface/Mesh.h
- UserInterface/Mesh.cpp

3.24 MSHParser Class Reference

A parser for interpreting meshes structured in the ".msh" format.

`#include <MSHParser.h>`

Inheritance diagram for MSHParser:



Public Member Functions

- MSHParser (QObject *parent, std::string)
- std::vector< Element > getElements ()
- std::vector< Node > getNodes ()
- int getNumberOfNodes ()
- int getNumberOfElements ()
- std::string getFilename ()
- Element getElementAt (int)
- Node getNodeAt (int)

3.24.1 Detailed Description

A parser for interpreting meshes structured in the ".msh" format.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 MSHParser::MSHParser (QObject * *parent*, std::string *fileName*)

Parameters

String	Filename of the file to parse. Excluding the file type (ending)

3.24.3 Member Function Documentation

3.24.3.1 Element MSHParser::getElementAt (int *e*)

Returns

Element at the given position

3.24.3.2 std::vector< Element > MSHParser::getElements ( )

Returns

vector<Element> containing all valid elements parsed from file

3.24.3.3 std::string MSHParser::getFilename ( )

Returns

name of parsed file

3.24.3.4 Node MSHParser::getNodeAt (int *n*)

Returns

Node at the given position

3.24.3.5 std::vector< Node > MSHParser::getNodes ( )

Returns

vector<Node> containing all nodes parsed from file

3.24.3.6 int MSHParser::getNumberOfElements ( )

Returns

number of elements

3.24.3.7 int MSHParser::getNumberOfNodes ( )

Returns

number of nodes

The documentation for this class was generated from the following files:

- UserInterface/MSHParser.h
- UserInterface/MSHParser.cpp

3.25 Node Class Reference

A data object storing node data.

`#include <Node.h>`

Public Member Functions

- void **setPosition** (double pos[])

Public Attributes

- int id

  Node id. Used for node ordering.
- double **x**
- double **y**
- double **z**
- int handle

  Node handle used for seed interaction.

3.25.1 Detailed Description

A data object storing node data.

The documentation for this class was generated from the following file:

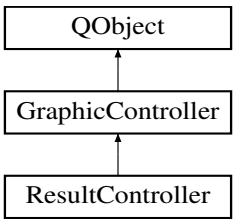
- UserInterface/Node.h

3.26 ResultController Class Reference

A controller changing the style of GraphicsView, and displays results.

`#include <ResultController.h>`

Inheritance diagram for ResultController:



Public Slots

- void reset ()
- void **invalidate** ()
- void **createScalarBarActor** ()
- void **setVisibility** ()

Public Member Functions

- **ResultController** (ResultModel *, QWidget *, QObject *parent=0)
- void setModel (ResultModel *)
- void actorClicked (vtkSmartPointer< vtkActor >)
- void actorDoubleClicked (vtkSmartPointer< vtkActor >)
- void createResultActor ()
- void createMeshActor ()

Additional Inherited Members

3.26.1 Detailed Description

A controller changing the style of GraphicsView, and displays results.

It sets up Graphicsview to use 3D, creates the color bar and the axisorientation actor.

3.26.2 Member Function Documentation

3.26.2.1 void ResultController::actorClicked (vtkSmartPointer< vtkActor >) [inline], [virtual]

Not implemented functionality in this view

Implements GraphicController.

3.26.2.2 void ResultController::actorDoubleClicked (vtkSmartPointer< vtkActor >) [inline], [virtual]

Not implemented functionality in this view

Implements GraphicController.

3.26.2.3 void ResultController::createMeshActor ( )

Creates a mesh actor if model has meshdata and resets the camera to display the actor.

3.26.2.4 void ResultController::createResultActor ( )

Creates a result actor if model has resultdata, and resets the camera to display the actor.

3.26.2.5 void ResultController::reset ( ) [slot]

Removes all active actors from the GraphicsView and creating result actor, mesh actor and scalar bar actor.

3.26.2.6 void ResultController::setModel (ResultModel * *model*)

Sets the current resultmodel used and resets the view. It also connects model signals to the view.

The documentation for this class was generated from the following files:

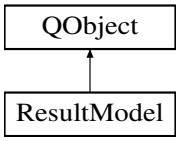
- UserInterface/ResultController.h
- UserInterface/ResultController.cpp

3.27 ResultModel Class Reference

Model which holds the data and functions to manipulate the VTK and Mesh datasets.

`#include <ResultModel.h>`

Inheritance diagram for ResultModel:



Public Slots

- void **showScalarBar** ()
- void **hideScalarBar** ()

Signals

- void modelChanged ()
- void modelDataSourceChanged ()

Public Member Functions

- **ResultModel** (QObject *parent)
- void computeResultNameAndRange ()
- vtkSmartPointer< vtkDataSetMapper > getMapper ()
- void setActiveResult (QString)
- void setDataSource (MSHParser *)
- void setDataSource (VTKParser *)
- bool hasResultData ()
- bool hasMeshData ()
- Mesh * getMesh ()
- void setMeshVisible (bool)
- void setResultVisible (bool)
- std::string getActiveResultName ()

Public Attributes

- vtkSmartPointer< vtkLookupTable > lut

  Lookup table to create a color map for a range of values.
- bool meshVisible

  Mesh actor visibility.
- bool resultVisible

  Result actor visibility.
- bool scalarBarVisible

  Scalar actor visibility.
- std::vector< std::string > resultNames

  Name of results in file.

3.27.1 Detailed Description

Model which holds the data and functions to manipulate the VTK and Mesh datasets.

3.27.2 Member Function Documentation

3.27.2.1 void ResultModel::computeResultNameAndRange ( )

Loops through the pointData arrays in the loaded dataset, and creates a list of data names. It also computes a map containing scale ranges based with the resultName as a key

3.27.2.2 std::string ResultModel::getActiveResultName ( )

Returns

std::string Name of the active result

3.27.2.3 vtkSmartPointer< vtkDataSetMapper > ResultModel::getMapper ( )

Returns

vtkSmartPointer<vtkDataSetMapper> Contains result data

3.27.2.4 Mesh * ResultModel::getMesh ( )

Gets the active mesh entity in the model.

Returns

Mesh*

3.27.2.5 bool ResultModel::hasMeshData ( )

Checks if the mesh entity is initialized

Returns

True if model has mesh data. False otherwise.

3.27.2.6 bool ResultModel::hasResultData ( )

Checks if polyData has points > 0 & lines > 0 & polygons > 0.

Returns

True if model has vtkData. False otherwise.

3.27.2.7 void ResultModel::modelChanged ( ) [signal]

Fires if activeResult changes.

3.27.2.8 void ResultModel::modelDataSourceChanged ( ) [signal]

Fires if one of the model's data sources changes.

3.27.2.9 void ResultModel::setActiveResult (QString *itemText*)

Sets the active scalar pointData given the pointData's name.

3.27.2.10 void ResultModel::setDataSource (MSHParser * *parser*)

Set the Mesh datasource and loads the data from the meshparser into the model. When done, the model fires "modelDataSourceChanged" and the view is invalidated.

3.27.2.11 void ResultModel::setDataSource (VTKParser * *parser*)

Set the Result datasource and loads the data from the vtkparser into the model. When done, the model fires "modelDataSourceChanged" and the view is invalidated.

3.27.2.12 void ResultModel::setMeshVisible (bool *visible*)

Sets the mesh visibility in the model and fires modelChanged

3.27.2.13 void ResultModel::setResultVisible (bool *visible*)

Sets the result visibility in the model and fires modelChanged

The documentation for this class was generated from the following files:

- UserInterface/ResultModel.h
- UserInterface/ResultModel.cpp

3.28 ShortcutManager Class Reference

Container for all keyboard shortcuts supported by the application.

`#include <ShortcutManager.h>`

Static Public Member Functions

- static void setMainWindow (QMainWindow *)
- static void setWidget (QWidget *)

3.28.1 Detailed Description

Container for all keyboard shortcuts supported by the application.

3.28.2 Member Function Documentation

3.28.2.1 void ShortcutManager::setMainWindow (QMainWindow * *main*) [static]

Sets shortcuts that connects to the main window.

3.28.2.2 void ShortcutManager::setWidget (QWidget * *widget*) [static]

Sets shortcuts that connects directly to the GraphicsView

The documentation for this class was generated from the following files:

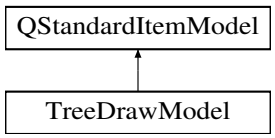
- UserInterface/ShortcutManager.h
- UserInterface/ShortcutManager.cpp

3.29 TreeDrawModel Class Reference

Tree model for Qt's tree widget. It implements functionality for entity selection in the tree, and listen for signals from the draw model.

`#include <TreeDrawModel.h>`

Inheritance diagram for TreeDrawModel:



Public Slots

- void **newEntity** (Entity *)
- void **removeEntity** (Entity *)

Signals

- void **selectionChanged** (Entity *)

Public Member Functions

- **TreeDrawModel** (DrawModel *, QObject *parent=0)
- void **addEntity** (Entity *)
- void **invalidate** ()
- void **setDataSource** (DrawModel *model)
- QList< QStandardItem * > **prepareRow** (const QString &)
- void **clicked** (const QModelIndex &)
- TreeItem * **getTreeItemFromIndex** (const QModelIndex &index)

Public Attributes

- QStandardItem * **topNode**
- DrawModel * **dataModel**
- std::map< Entity *, TreeItem * > **treeItemLookupTable**

3.29.1 Detailed Description

Tree model for Qt's tree widget. It implements functionality for entity selection in the tree, and listen for signals from the draw model.

The documentation for this class was generated from the following files:

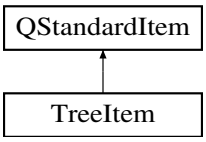
- UserInterface/TreeDrawModel.h
- UserInterface/TreeDrawModel.cpp

3.30 TreeItem Class Reference

Holder for an entity shown in the TreeWidget. It holds a reference to the entity for fast selection.

`#include <TreeItem.h>`

Inheritance diagram for TreeItem:



Public Member Functions

- **TreeItem** (const QString &, Entity *entity)

Public Attributes

- int **type**
- Entity * entityReferece

  Reference to the entity.

Static Public Attributes

- static const int **MATERIAL** = 0
- static const int **ENTITY** = 1

3.30.1 Detailed Description

Holder for an entity shown in the TreeWidget. It holds a reference to the entity for fast selection.

The documentation for this class was generated from the following files:

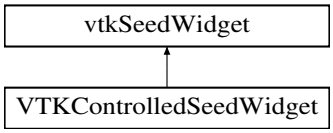
- UserInterface/TreeItem.h
- UserInterface/TreeItem.cpp

3.31 VTKControlledSeedWidget Class Reference

Wrapper for the vtkSeedWidget, providing extended control of the seed widget.

`#include <VTKControlledSeedWidget.h>`

Inheritance diagram for VTKControlledSeedWidget:



Public Member Functions

- **vtkTypeMacro** (VTKControlledSeedWidget, vtkSeedWidget)
- void **setEnableAddInteraction** (bool)
- void **setStartInteraction** ()
- void **setCallback** (vtkSmartPointer< vtkSeedCallback > scbk)
- void **AddSeed** (double, double, double)
- void setModel (Entity *)
- vtkHandleWidget * **CreateNewHandle** ()
- void **update** ()

Static Public Member Functions

- static VTKControlledSeedWidget * **New** ()
- static void **StartMoveAction** (vtkAbstractWidget *)
- static void DeleteAction (vtkAbstractWidget *)
- static void AddPointAction (vtkAbstractWidget *)

3.31.1 Detailed Description

Wrapper for the vtkSeedWidget, providing extended control of the seed widget.

This class provides an interface to control if seed creation is enabled, or if only changing seeds are allowed. vtkSeedWidget, the class it extends, adds a set of forced interactions in the constructor. The program relies heavily on the ability to enable/disable these interactions on the go, and vtkSeedWidget did not fulfill these needs in its current state.

3.31.2 Member Function Documentation

3.31.2.1 void VTKControlledSeedWidget::AddPointAction (vtkAbstractWidget * *w*) [static]

Edited and overridden VTK-code. It add the possibility of enable/disable "add seed" feature

3.31.2.2 void VTKControlledSeedWidget::DeleteAction (vtkAbstractWidget * *w*) [static]

Overrides the vtkSeedWidget's delete function.

3.31.2.3 void VTKControlledSeedWidget::setModel (Entity * *entity*)

Sets the entity containing the seeds to be displayed, and displays the seeds on the screen

The documentation for this class was generated from the following files:

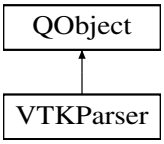
- UserInterface/VTKControlledSeedWidget.h
- UserInterface/VTKControlledSeedWidget.cpp

3.32 VTKParser Class Reference

A parser for interpreting VTK files using VTK's vtkDataSetReader, and provides a convenient way to extract the data.

`#include <VTKParser.h>`

Inheritance diagram for VTKParser:



Public Member Functions

- **VTKParser** (std::string inputFilename, QObject *parent)
- vtkSmartPointer< vtkPointData > **getPointData** ()

Public Attributes

- vtkSmartPointer< vtkDataSet > **dataSet**
- vtkSmartPointer< vtkDataSetMapper > **dataSetMapper**
- vtkSmartPointer< vtkPointData > **pd**

3.32.1 Detailed Description

A parser for interpreting VTK files using VTK's vtkDataSetReader, and provides a convenient way to extract the data.

The documentation for this class was generated from the following files:

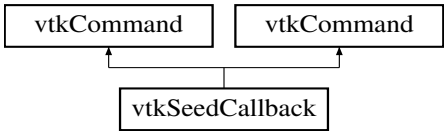
- UserInterface/VTKParser.h
- UserInterface/VTKParser.cpp

3.33 vtkSeedCallback Class Reference

Callback for seed widget. Handles creation of new seeds and seed interaction.

`#include <VTKControlledSeedWidget.h>`

Inheritance diagram for vtkSeedCallback:



Public Member Functions

- virtual void Execute (vtkObject *, unsigned long event, void *calldata)

- void setEntity (Entity *entity)
- void **SetRepresentation** (vtkSmartPointer< vtkSeedRepresentation > rep)
- void **SetRepresentation** (vtkSmartPointer< vtkSeedRepresentation >)
- virtual void **Execute** (vtkObject *, unsigned long event, void *calldata)

Static Public Member Functions

- static vtkSeedCallback * **New** ()
- static vtkSeedCallback * **New** ()

3.33.1 Detailed Description

Callback for seed widget. Handles creation of new seeds and seed interaction.

3.33.2 Member Function Documentation

3.33.2.1 virtual void vtkSeedCallback::Execute (vtkObject * , unsigned long *event*, void * *calldata*) [inline], [virtual]

Creates or changes a seed if interaction occurs. All seeds(nodes) added to entity is given an id based on the number the seed har in the widget. It is used to keep track of which node is being manipulated.

3.33.2.2 void vtkSeedCallback::setEntity (Entity * *entity*) [inline]

Sets the entity the callback is doing work on.

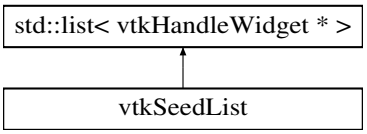
The documentation for this class was generated from the following files:

- UserInterface/VTKControlledSeedWidget.h
- UserInterface/VTKDrawCallback.h

3.34 vtkSeedList Class Reference

Data structure used for seeds in VTK.

Inheritance diagram for vtkSeedList:



3.34.1 Detailed Description

Data structure used for seeds in VTK.

The documentation for this class was generated from the following file:

- UserInterface/VTKControlledSeedWidget.cpp