NTNU
Det skapende universitet

Modellering og elementmetode beregning av tverrsnittsparamtere for komplekse bjelkerverrsnitt

Fredrik Nordmoen

Master i ingeniørvitenskap og IKT
Innlevert: februar 2015
Hovedveileder: Bjørn Haugen, IPM

Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR PRODUKTUTVIKLING
OG MATERIALER

MASTEROPPGAVE HØST 2014

FOR

STUD.TECHN. FREDRIK NORDMOEN

MODELLERING OG ELEMENTMETODE BEREGNING AV TVERRSNITTSPARAMETRE FOR KOMPLEKSE BJELKETVERRSNITT
Modelling and finite element computations of cross-section parameters for complex beam sections

Tverrsnittsdata for komplekse tverrsnitt 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 videreutvikle funksjonaliteten i en applikasjon som beregner tverrsnittsparametre for komplekse tverrsnitt ved hjelp av elementmetoden.
Oppgaven tar særlig sikte på å utvikle ny funksjonalitet for modellering og resultatpresentasjon av tynnveggede tverrsnitt og kompositt matriale.

Applikasjonen bør også kunne beregne 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. Applikasjonen bør bygge på enkle småprogrammer som lar seg benytte effektivt i skript dersom en skal utføre beregningene for mange forskjellige tverrsnitt.

Oppgaven har fokus på brukergrensesnittet av applikasjonen. Grensesnittet mellom den numeriske koden og det grafiske brukergrensesnittet må defineres med tanke på 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.

Kontaktperson:

Torgeir Welo
Instituttleder

Bjørn Haugen
Faglærer

NTNU
Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

Abstract

Cross-sectional properties for complex cross sections are difficult to calculate analytically, dynamic calculations using beam models are often used as a substitute, and good results are dependent on correct cross-sectional data. The application is a tool to draw good cross section for use in such analyzes.

The assigned task is further development of an application that is supposed to be used in education at the Institute of Design and Materials, it is therefore emphasized to create a simple and efficient user interface. The application should also be used in industrial applications; simplification of the user interface cannot be at the expense of functionality.

The application has complete cross-platform support and is made using the frameworks; Qt for UI, OpenCASCADE for manipulating models, gmsh for meshing and The Visualization Toolkit for visualization.

The report is primarily intended as a reference and a guidance in new development of the application as well as a detailed review of how to set up project on new machines, but there is also a review of the design and development process as well as accounts of choices made during the process.

Sammendrag

Tverrsnittsdata for komplekse tverrsnitt er vanskelig å beregne analytisk, dynamiske beregninger med bjelkemodeller benyttes ofte som en erstatning, og gode resultater er avhengig av korrekte tverrsnittsdata. Applikasjonen skal være et verktøy for å lage gode tverrsnitt til bruk i slike analyser.

Oppgaven er en videreutvikling av en applikasjon som skal brukes i undervisning ved Institutt for produktutvikling og materialer, det er derfor lagt vekt på å lage enkle og effektive brukergrensesnitt. Applikasjonen skal også kunne brukes i industrielle sammenhenger, forenklingen av brukergrensesnitt kan derfor ikke gå på bekostning av funksjonalitet.

Applikasjonen har fullstendig kryssplattformstøtte og er laget ved hjelp av rammeverkene Qt for brukergrensesnitt, OpenCASCADE for manipulering av modeller, gmsh for meshing og The Visualization Toolkit for visualisering.

Rapporten er hovedsaklig ment som et oppslagsverk og veiledning ved videreutvikling av programmet samt en detaljert gjennomgang av hvordan prosjektet settes opp på nye maskiner, men det er også en gjennomgang av design- og utviklingsprosessen samt redegjørelser for valg tatt underveis i prosessen.

Forord

Denne masteroppgaven ble utført ved Norges teknisk-naturvitenskapelige universitet (NTNU), Institutt for produktutvikling og materialer (IPM) i perioden september 2014 - februar 2015.

Masteroppgaven er en av mange oppgaver i et langsgående prosjekt som kommer fra IPMs ønske om å lage en modularisert programpakke med mulighet for beregning av komplekse tverrsnitt. Grunnfunksjonaliteten skulle tilsvare det i programmet «CrossX», men med mulighet til også å utføre analyser på tverrsnitt som inneholder kompositttmaterialer.

Denne oppgaven er en fortsettelse på en masteroppgave utført av Bjørnar Askevold ved samme institutt, våren 2014.

Arbeidet har vært utført under veiledning fra førsteamanuensis Bjørn Haugen, som jeg vil takke for kyndig veiledning i løpet av prosessen. Jeg vil også takke Michael Shilling i Nexiles for lån av kontor og utstyr.

Trondheim, 10. februar 2015

Fredrik Nordmoen

Innhold

Figurer

Tabeller

Ordliste

BSD Berkeley Software Distribuion License.

C++ Programmeringsspråket som brukes i dette prosjektet..

CCCP Complex Crossection Calculation Program, arbeidstittel for denne applikasjonen..

dialog Et mindre vindu som dukker opp underveis i applikasjonen..

DoxyGen Program brukt for å generere hjelpedokumentasjon fra eksisterende kildekode..

entitet Betegnelsen på en fullført geometrisk figur som en del av illustrasjonen av et tverrsnitt..

garbage collection En innebygd funksjon i programmer for å frigjøre ubrukt plass minne av seg selv..

Git Et distribuert versjonskontrollsystem..

GitHub En gratis leverandør av Git og tilknyttede tjenester på nettet..

gmsh Rammeverk for meshing.

GPL General Public License.

IPM Institutt for produktutvikling og materialer.

kontekstmeny Menyen som dukker opp ved høyreklikk i programmet..

KT Institutt for konstruksjonsteknikk.

LaTeX Program for typesetting og dokumentproduksjon brukt i denne oppgaven..

LGPL Lesser General Public License.

Mesher Modulen hvor en flate gjøres om til en samling homogene elementer..

MVC Model-View-Controller, en arkitektur som beskriver krav og rollefordeling til ulike klasser i applikasjoner basert på brukergrensesnitt..

OCCT OpenCASCADE Technologies, et rammeverk for manipulasjon av 3D-modeller.

Qt Rammeverk for brukergrensesnitt på tvers av operativsystemer..

Solver Modulen hvor alt av beregninger skjer..

SourceTree Program brukt for håndtering av en kodebase mot Git..

VTK The Visualisation Toolkit, et rammeverk for visualisering..

Kapittel 1

Innledning

Oppgaveteksten forteller at det skal videreutvikles et brukergrensesnitt, og tar utgangspunkt i eksisterende programvare utviklet i en tidligere masteroppgave av Bjørnar Askevold[2]. Oppgaven legger opp til en utvikling av nye funksjoner som skal utvide det eksisterende brukergrensesnittet. Rapporten tar for seg problemer ved overtagelse av andres kodebase samt utviklingen og valgene som er tatt i forbindelse med denne.

Det finnes mange oppfatninger av hva et brukergrensesnitt er og skal være. I all hovedsak er et brukergrensesnitt noe som gjør et menneske i stand til å interagere med maskiner, og et godt brukergrensesnitt gjør at mennesker gjør dette på en intuitiv måte.

På grunn av applikasjonens oppbygning brer arbeidsoppgavene til «brukergrensesnitt» seg noe utover det som ansees for å være vanlig i programvareutvikling. Dette kommer av at modulen «brukergrensesnitt» er ansvarlig for å knytte alle de andre modulene sammen.

Hovedansvarsområdene til brukergrensesnittet vil være:

Preprosessor Verktøy for tegning, manipulering og beskrivelser av materialer og grensesnitt for bruk i beregningen.

Postprosessor Visualisering resultatene som en 3D-modell med mulighet for visning av forskyvninger og forskjellige verdier som fargeplott, og valg av forskjellige datasett.

Eksekvering Eksport og import av datasett samt det å kjøre de forskjellige modulene som er nødvendig for å få et resultat.

1.1 Bakgrunn

I utgangspunktet er det ikke en mangel på programmer som kan brukes til å gjøre beregninger på krefter i tverrsnitt som driver utviklingen av dette programmet. Ved undervisning ønsker IPM å bruke programmer for å visualisere og å gjøre det enklere for studentene å forstå hva som skjer, og derfor trengs det mindre programmer som ikke krever store mengder arbeid for å få ut enkel data om tverrsnittet.

Institutt for konstruksjonsteknikk (KT) har utviklet programmet CrossX for å gjøre noe tilsvarende, CrossX mangler dog støtte for kompositttmaterialer og grensesnittet oppleves noe gammeldags og vanskelig i bruk.

Programmet i denne oppgaven er derfor tenkt å ha samme kjernefunksjonalitet som CrossX, men utvidet med støtte for kompositttmaterialer og enklere brukergrensesnitt.

Videre er det tenkt at programmets beregningsmodul i fremtiden skal kunne brukes i profesjonell skala, og derfor skal programmet være delt opp i flere separate deler slik at arbeidet kan kjøres i bulk og gjerne parallelliseres.

1.2 Avgrensninger

For å begrense omfanget av oppgaven er det forsøkt å kun fokusere på arbeid som direkte påvirker brukeren ved design og beregning av tynnveggede tverrsnitt.

Arbeid med kompositttmaterialer er begrenset til det konseptuelle nivået av utviklingen, da arbeidet med implementasjon og filformater ansees å være for omfattende til å være en del av denne oppgaven.

Kapittel 2

Programmet

Oppgaven definerer en fortsettelse på programvare utviklet tidligere masteroppgaver, og derfor også eksisterende programvare. Programmet ble tildelt arbeidstittelen Complex Crossection Calculation Program (CCCP), og dette vil benyttes videre i oppgaven.

Programmets oppbygning og utviklingsmetode har ikke endret seg stort i denne versjonen av prosjektet, det vil derfor være likheter i både teori og bakgrunn i denne rapporten, og rapporten dette prosjektet bygger videre på. Se referanse [2].

2.1 Historikk

Programmet har vært jobbet på i flere omganger og med noe variasjon i deltagere.

2.1.1 Prosjektoppgaver høsten 2013

Høsten 2013 ble det utført tre prosjektoppgaver som et samarbeid mellom Bjørnar Askevold, Kristian Strømstad og forfatteren, Fredrik Nordmoen. Disse oppgavene var det innledende arbeidet til CCCP. Oppgavene gikk ut på å få konvertert en geometrisk figur til et mesh av triangelelementer, regne ut tyngdesenter, skjærsenter og prinsipalakser ved hjelp av nevnte mesh og deretter vise frem resultatene i et enkelt brukergrensesnitt.

Det ble utformet et eget filformat for å overføre resultater fra beregninger til visualisering.

En av forutsetningene var at programmet skulle utvikles i sin helhet i C++, men komplikasjoner mellom VTK 6.0 og Qt 5 førte til at visualiseringen ble implementert i Java.

2.1.2 Masteroppgaver våren 2014

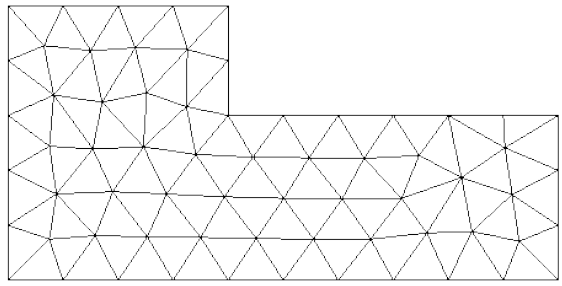
Bjørnar Askevold og Kristian Strømstad fortsatte begge med utviklingen av CCCP på sine masteroppgaver våren 2014. Kristian Strømstad fortsatte med videreutvikling av beregningsmotoren[3] mens Bjørnar Askevold implementerte og videreutviklet grensesnittet i C++[2].

2.2 Oppbygning

Programmet er delt opp i tre deler, hvor alle delene kan kjøres som uavhengige applikasjoner.

2.2.1 Mesher

Prinsippet i med elementmetoden er at enhver geometrisk figur kan regnes på med et ved å redusere figuren ned til en samling av enklere figurer som kan regnes på med standardformler. Elementene bygges opp av enten endimensjonale bjelker, todimensjonale flater eller tredimensjonale volumelementer. Se figur 2.1 for eksempel på et todimensjonalt «mesh».



Figur 2.1: Eksempel på et «mesh» hvor en L-formet flate er redusert til en serie todimensjonale trekantflater.

Hvert element i «meshet» er bygget opp av noder og kanter, hvor hver kant utgjør en kobling mellom to noder og er det som beskriver «meshets» egenskaper. Styrken i hver kant beregnes ut fra elementets materialegenskaper og området elementet dekker.

Prosjektet har tidligere kun støttet todimensjonale trekantelementer med tre noder, like de i figur 2.1, programmet skal nå utvides til å støtte endimensjonale bjelkeelementer med to noder.

2.2.2 Solver

«Solveren» er modulen som står for all matematikk og beregning. «Solveren» bruker «meshet» og materialdata, samt en ytre påvirkning som kan være gitt ved punktlast, vridning eller moment. Resultatet gis i enhetskrefter, da dette gjør at resultatet lett anvendelig og det kan enkelt konverteres til ønsket formål i etterkant.

2.2.3 Brukergrensesnitt

Brukergrensesnittet inneholder funksjonene til det som tradisjonelt sett omtales som «preprosessor» og «postprosessor». Preprosessoren lar brukeren importere tverrsnitt fra støttede filformater (vedlegg B), eller tegne et eget tverrsnitt, det er også her materialegenskaper tilegnes tverrsnittet. Det er også planlagt muligheten for å sette sammen egenkomponerte kompositttmaterialer.

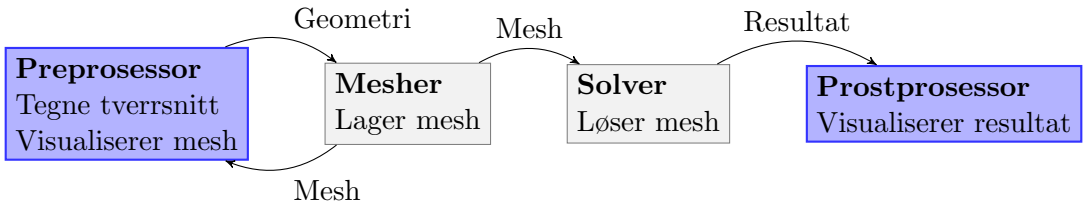
Postprosessoren leser resultatene fra «solveren» og visualiserer de de forskjellige verdiene i både 2D og 3D.

Brukergrensesnittet står også for å gi riktig informasjon til de to andre modulene i prosjektet, samt kjøringen av dem.

2.2.4 Modulenes sammensetning

Brukerens vei gjennom programmet er gjort så strømlinjeformet som mulig, og det er vanskelig å vite at det blir brukt tre forskjellige programmer hvis det ikke er kjent på forhånd.

Flyten i et normalt bruksscenario er illustrert i figur 2.2 og vil være omtrent som dette; brukeren tegner et tverrsnitt i brukergrensesnittet,



Figur 2.2: Arbeidsflyten og sammenhengen mellom modulene. Arbeidsflyten starter med å tegne tverrsnittet, tverrsnittet «meshes» og kommer i retur for visualisering, når «meshet» er godkjent blir det beregnet i «solver» og vist i postprosessor. De uthevede modulene er omfattet av denne oppgaven. [2]

setter materialer og trykker på «Mesh». Geometridata sendes da fra brukergrensesnittet til «mesheren» «gmsh» som tar seg av «meshingen», hvor resultatet returneres tilbake til brukergrensesnittet for kontroll av brukeren. I det brukeren sier seg fornøyd med «meshet» startes «solveren» av brukergrensesnittet med samme «mesh» som utgangspunkt og resultatene derfra leses tilbake i brukergrensesnittet.

Kapittel 3

Avhengigheter

Når det skal lages et såpass omfattende program er det naturlig å benytte seg av andre prosjekter med åpen kildekode eller fri bruk for å drastisk kutte ned på både arbeidsmengde og feilkilder. I dette programmet er det brukt Qt for å ha grensesnitt med kryssplattformstøtte, VTK til visualisering av tegninger og resultater og OCCT for å kunne utføre operasjoner på tegnede elementer.

3.1 The Visualization Toolkit (VTK)

Brukergrensesnittets hovedoppgaver er å visualisere tegninger og resultater, disse vil hovedsaklig eksistere kun i 2D-plan, men erfaring tilsier at det er veldig informativt å visualisere verdier med grafer og diagrammer i tre dimensjoner.

VTK er et «open source»-prosjekt som ble startet i 1993 av tre forskere i GE Corporate som skrev kjernekoden som en del av en opplæring i 3D-visualisering hvor de ville ha et system som ikke var begrenset av lisenser og patenter, kjernen ble deretter videreutviklet av blant andre GE Medical for bruk i medisinsk forskning. VTK har i senere tid blitt akseptert og implementert og videreutviklet i mange forskjellige bransjer, hvor spesielt olje og gass og informasjonsvisualisering har bidratt til utviklingen. [4]

Utbredelsen blant forskjellige fagfelt har gitt VTK en voldsom allsidighet med evnen til å lese store mengder forskjellige filformater, mange presentasjonsmuligheter og støtte for flere programmeringsspråk.

Mulighetene kommer også med en pris, det er flere lag med abstraksjoner mellom kjernefunksjonene og det som brukes og dette gjør at man må ha ganske inngående kjennskap til rammeverket før man har mulighet til å gjøre individuelle tilpasninger og nye funksjoner.

3.2 Qt

For å oppfylle kravet om kryssplattformstøtte for dette programmet er det veldig arbeidsbesparende å benytte et rammeverk for å automatisk generere plattformspesifikke definisjoner av grensesnittet, slik at man ikke trenger å vedlikeholde flere versjoner av disse filene som i all hovedsak beskriver det samme.

Det finnes mange rammeverk som kan gjøre akkurat denne jobben, men hovedgrunnen til valget av Qt kommer fra VTK. ParaView er visualiseringsprogrammet som utvikles av prosjektet bak VTK, og ParaView benytter seg av Qt for sitt brukergrensesnitt. VTK har derfor en veldig god integrasjon mot Qt innebygd fra før, og dette forenkler mye for oss. Qt ser også ut til å ha en økende popularitet blant andre utviklere av «Open Source»-prosjekter, dette gir en økt trygghet i forventet levetid og vedlikehold av Qt.

Fordeler med Qt er en enkel form for «garbage collection» i form av at det kan defineres «foreldre-barn»-relasjoner mellom objekter, og alle barn slettes automatisk når en forelder slettes. Qt kommer også med en funksjon for hendelsesdrevet dataoverføring kalt «Signals & Slots», denne må brukes for å knytte knapper og menyer til den underliggende koden. «Signals & Slots» brukes med stort hell også for å sende hendelser mellom forskjellige klasser i kjernekoden.

Ved kompilering konverteres all Qt-kode til C++ før den igjen kompileres til maskinkode som kan kjøres, dette gir større «overhead» enn om prosjektet skulle vært skrevet utelukkende i C++. Det vurderes dit at fordelene med god margin overskrider ulempene, og Qt skal derfor kunne benyttes i stor grad og uten bekymring for kompleksitet der dette finnes hensiktsmessig.

3.3 OpenCASCADE Technology (OCCT)

Biblioteket OpenCASCADE er voldsomt stort og er bygget opp på samme måte som dette programmet, med flere selvstendige moduler som har forskjellig bruksområder. Det er til å begynne med implementert en veldig liten del av verktøyene i OCCT, og det er en del overlapp mellom hva OCCT og VTK tilbyr av tjenester. Modulariteten gjør likevel at en kan plukke litt etter behov fra OCCT, og baktanken med å velge et såpass stort bibliotek er at det burde kunne dekke de fleste av behovene i fremtiden.

Implementasjon av funksjoner fra OCCT er mer tungvint enn det er å bruke VTK, det er derfor anbefalt at det sjekkes grundig om det eksisterer funksjonalitet fra VTK før en gir seg ut på å skrive en implementasjon fra OCCT. Det betyr ikke at en må unngå å bruke OCCT hvis det er til hjelp.

Installasjon av OpenCASCADE krever at bibliotekene «Tcl/Tk» og «FreeType» alt er installert på maskinen.

3.4 gmsh

I prosjektet er gmsh benyttet til all generering av mesh fra geometrien som beskriver tverrsnittsdataen. gmsh er også et velkjent verktøy i modelleringsverden, og støtter omtrent like mange filformater som VTK.

gmsh har god støtte for å ta mot kommandoer fra kommandolinje når det gjelder spesifisering av det resulterende meshet, på grunn av dette er det mulig å kjøre meshingen som en ekstern prosess som frikobles helt fra grensesnittet. Frikoblingen støtter ønsket om å gjøre programvaren så modulær som mulig, samt at en står friere til valg av mesher dersom det skulle være behov for noen spesialfunksjoner som gjør at en må bruke et annet bibliotek.

3.5 Nytt utviklingsmiljø

Blant avhengighetene er det flere som igjen avhenger av hverandre, og dette legger føringer på rekkefølgen ting kan kompileres og installeres når dette prosjektet skal settes opp på en ny maskin. Installasjonsinstruksjonene er laget for 64-bit Windows 8.1, men tilsvarende programvare eksisterer

til alle operativsystemer og valgmulighetene er noe flere dersom det skal installeres på 32-bits plattform.

3.5.1 Nødvendige nedlastinger

VTK Nyeste versjon 6.1.0 er brukt uten problemer, og alle nyere varianter skal fungere smertefritt. [5]

Visual Studio Her må det tas et valg på versjon, og det lønner seg sjelden å benytte en helt ny versjon her, da vi trenger et tilpasset bygg av Qt. Brukte selv Visual Studio 2013 fremfor 2014 av denne grunn.

Qt Spesifikt er OpenGL-varianten som er brukt i prosjektet. Her må versjonen støtte VTK (VTK 6.1.0 introduserte støtte for Qt 5) og byggversjonen må være tilpasset Visual Studio, det ble derfor versjon 5.3, bygg: `msvc2013_64_opengl`. Last også ned Visual Studio Add-in for Qt5 fra samme sted (1.2.4). [6]

CMake Versjon er ikke av betydning (3.0.2). [7]

OCCT OpenCASCADE byr på litt mer motstand. Her må man først registrere seg på nettsidene for å laste ned kildekoden, da den prekompilerte installasjonen generelt ligger fire år bak. [8]

Tcl/Tk OCCT trenger Tcl 8.5 eller 8.6 for å kunne kompileres, har her brukt ActiveTcl 8.6.2 for å forenkle installasjonsprosessen. [9]

FreeType OCCT trenger FreeType versjon 2.5.3 for å kunne kompileres. [10]

Gmsh All meshing av tverrsnitt går gjennom Gmsh, versjon er ikke av betydning (2.8.5) [11]

Windows SDK FreeType trenger DLL-filer inkludert i Windows SDK. [12]

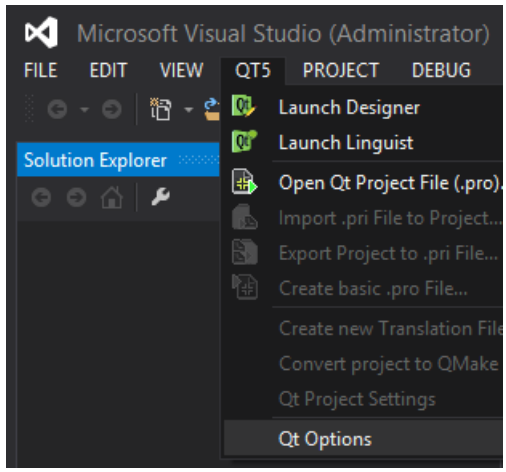
3.5.2 Installasjon

Programmer som kan installeres direkte fra medfølgende installer

- Visual Studio

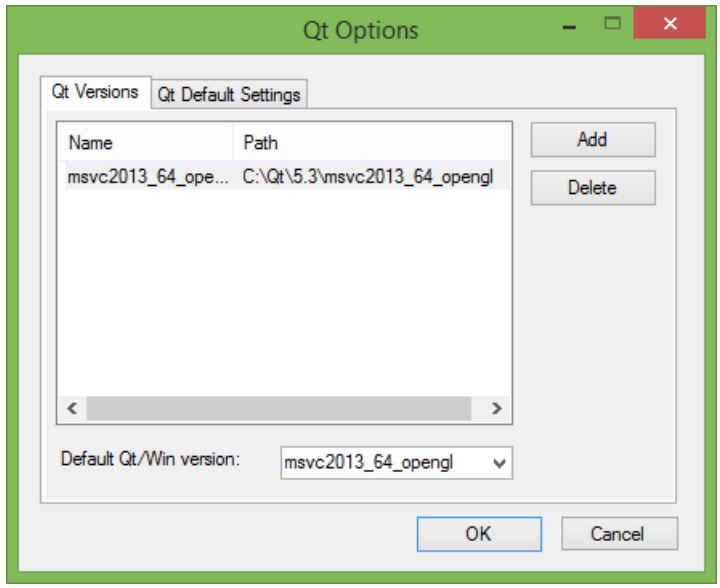
- Ot
- Qt Add-In for Visual Studio
- CMake, sørg for å få med `cmake-gui`
- ActiveTcl
- Windows SDK

Første gang Visual Studio åpnes etter at «Qt Add-In for Visual Studio» er installert er det nødvendig å sette opp hvilke installasjoner av Qt som skal benyttes. Det gjøres under toppmenyvalget «QT5».



Figur 3.1: Visual Studio: Meny for innstillinger av «Qt Add-In for Visual Studio».

Klikk på «Add» og velg mappen Qt er installert i, sett også denne versjonen som standard i rullegardinmenyen under.



Figur 3.2: Visual Studio: Innstillinger av «Qt Add-In for Visual Studio».

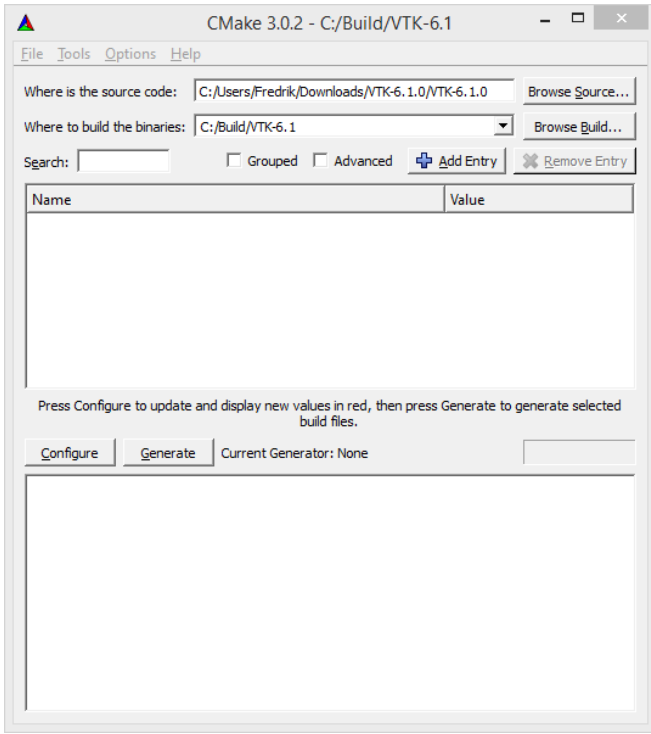
3.5.3 Kompilering

Kompilering av programmene følger alle samme prosedyren. Først brukes CMake til å lage prosjektet, dette tar ofte et par-tre runder for å få satt de riktige flaggene. Deretter genereres det en prosjektfil som kan åpnes i Visual Studio, etter en runde med kompilering her skal man sitte igjen med biblioteker, header- og dll-filer i en valgt mappe.

Kompileringen må gjøres i rekkefølgen de er listet opp.

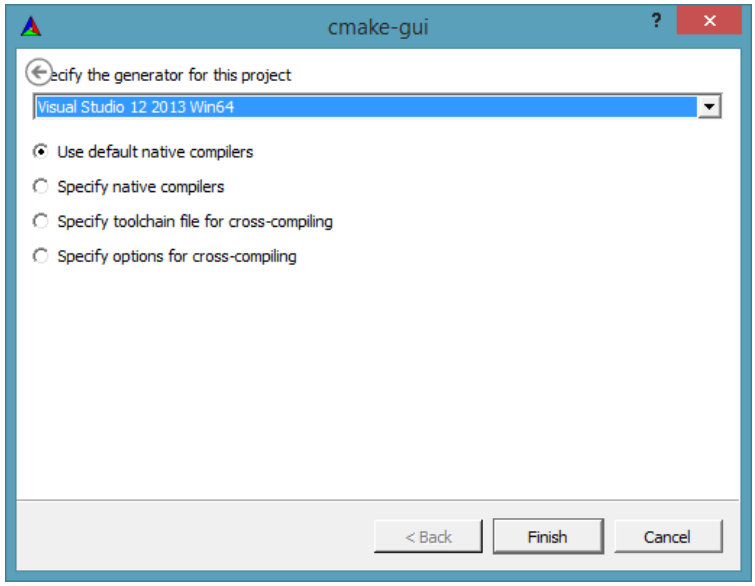
VTK

Det første som må gjøres er å starte opp CMake og gi to input plassering av kildekode og ønsket plassering av binærdata. Ingen av delene er veldig viktige å ta vare på, men jeg liker å samle alle binærdata i en egen «Builds»-mappe.



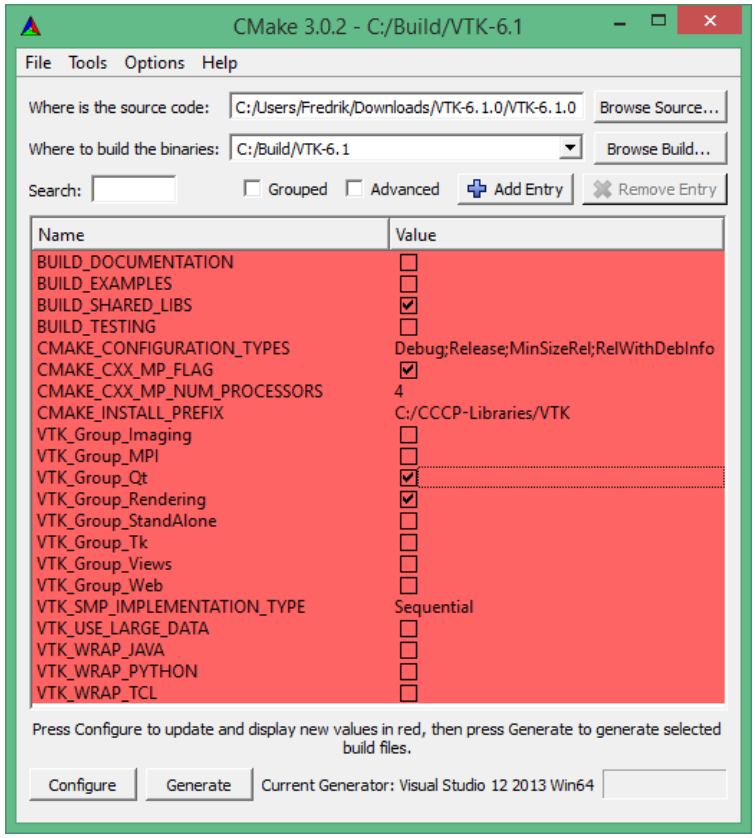
Figur 3.3: VTK: Første input i CMake, plassering av kildekode og ønsket plassering av binærdata.

Trykk på «Configure», og første gang må det velges ønsket kompilator. Velg din variant av operativsystem og Visual Studio. Obs: «Visual Studio 10» er ikke det samme som «Visual Studio 2010».



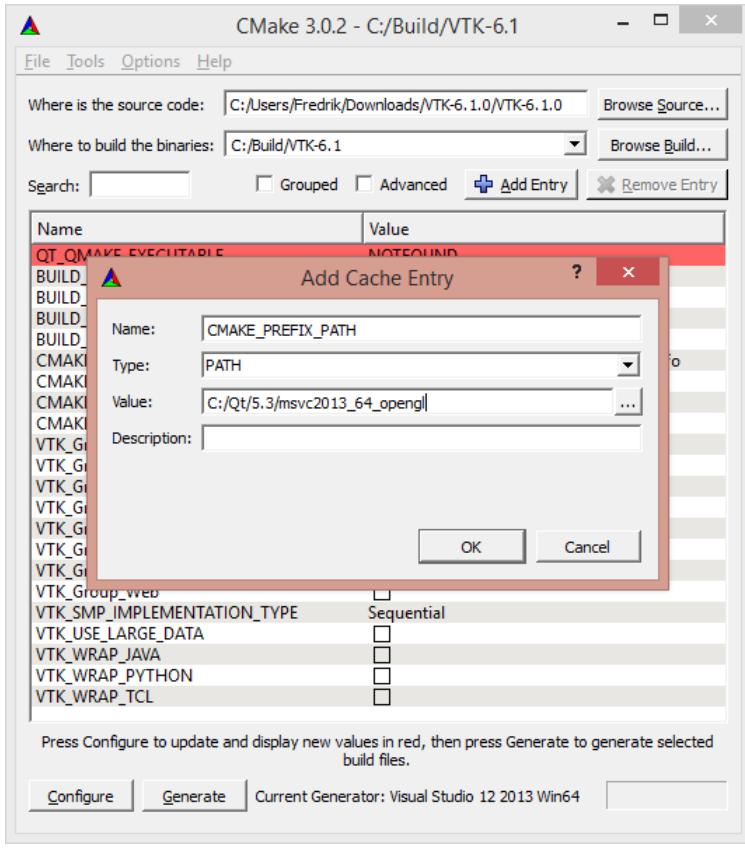
Figur 3.4: VTK: Valg av kompilator i CMake.

Dette steget kan ta ganske lang tid, men når CMake har fått kjørt gjennom kildekoden en runde er det klart for å sette variabler for å tilpasse denne installasjonen. For å spare tid velges ikke `BUILD_EXAMPLES` eller `BUILD_TESTING`, sett `CMAKE_INSTALL_PREFIX` til å peke på mappen VTK skal installeres til og husk å huke av for `VTK_Group_Qt`. Trykk «Configure».



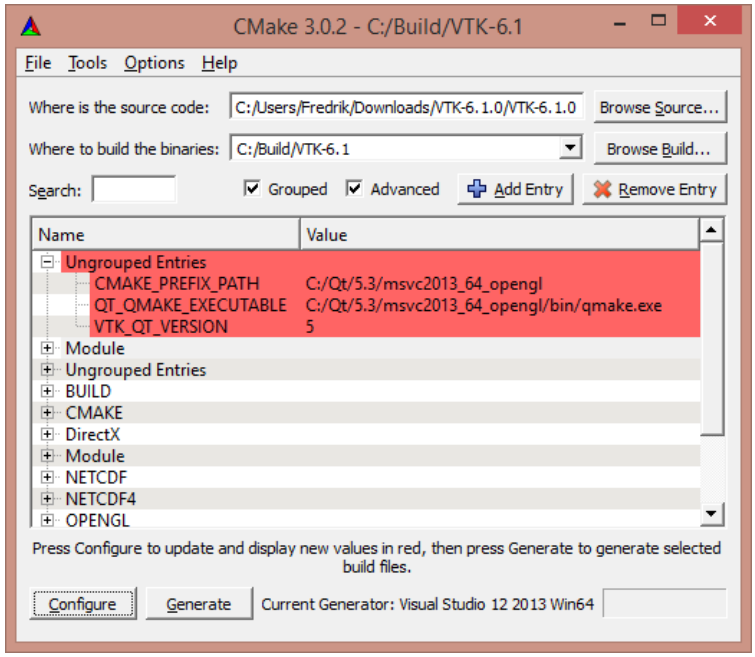
Figur 3.5: VTK: Første runde med variabler i CMake.

Trykk på «Add Entry» for å legge til en egendefinert variabel med navn «CMAKE_PREFIX_PATH», type «PATH» og velg installasjonsstien til Qt som ble installert i 3.5.2.



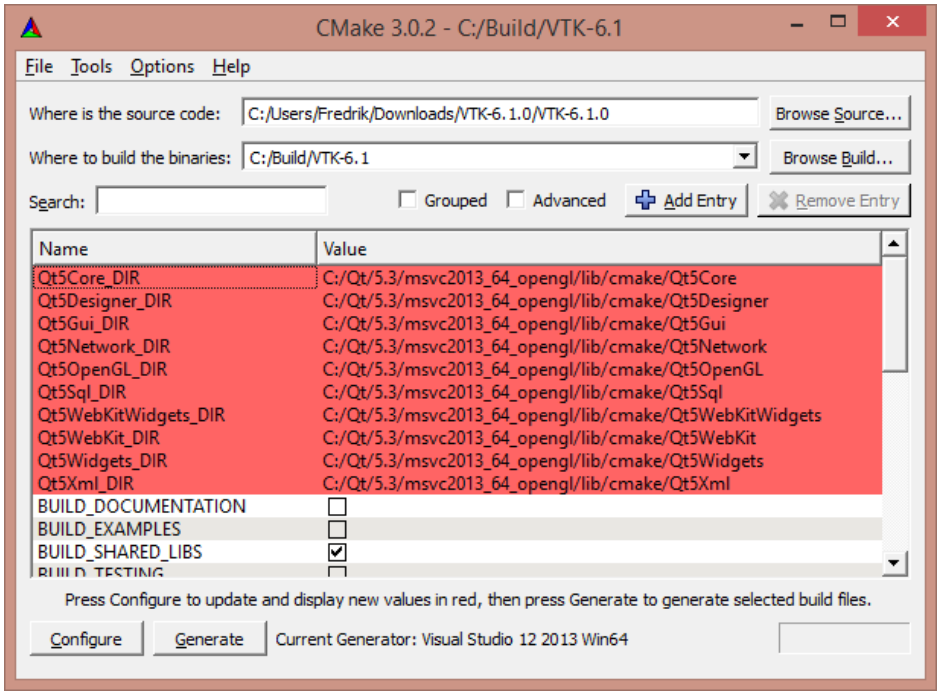
Figur 3.6: VTK: Legger til en egendefinert variabel i CMake for å forenkle andre runde.

Her må man finne frem til `QT_QMAKE_EXECUTABLE` selv, det er filen «qmake.exe» som finnes i Qt-mappen. For å finne alternativet `VTK_QT_VERSION` må det først hukes av for «Advanced». Trykk «Configure».



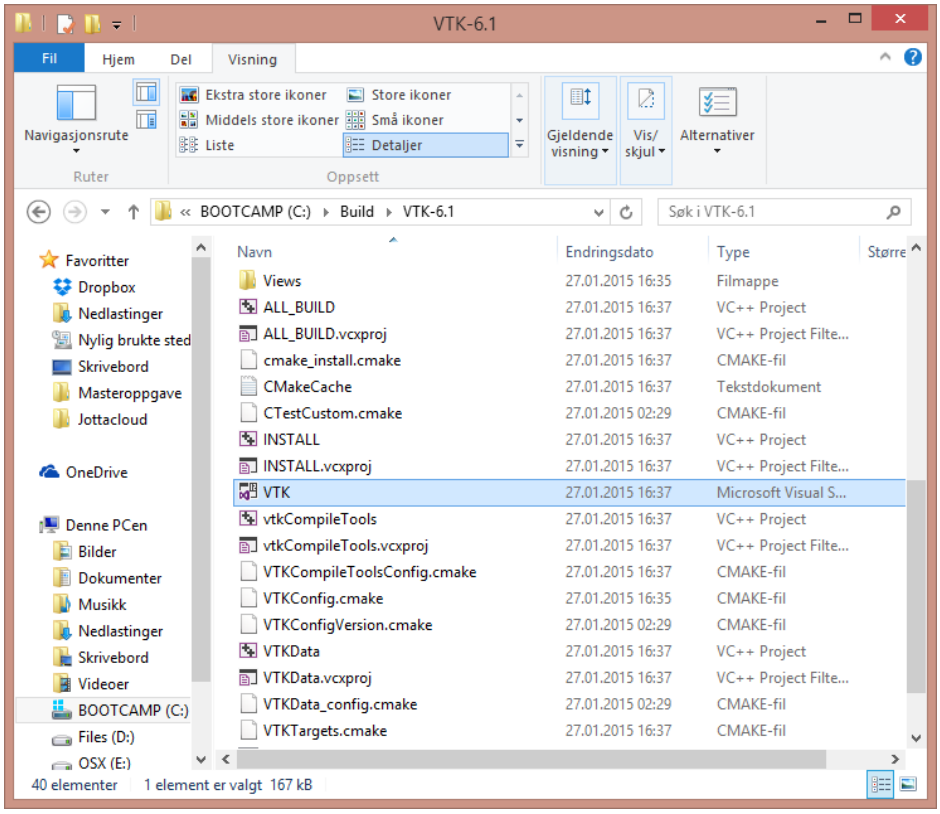
Figur 3.7: VTK: Resten av CMake-variablene i andre runde.

Alle variablene skal nå settes automatisk på grunn av den egendefinerte variabelen i forrige steg, dersom noe mangler og markeres med «NOT-FOUND» settes stien manuelt slik som tidligere og på samme måte som figur 3.8.



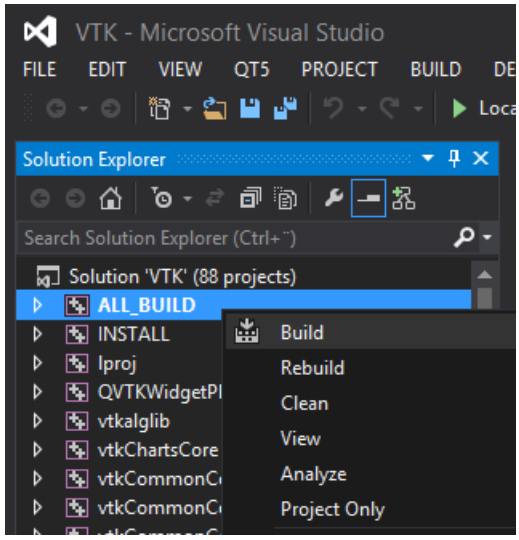
Figur 3.8: VTK: Variablene for tredje og siste runde i CMake.

Trykk «Configure» for siste gang, bekreft at det ikke er noen variabler markert i rødt, trykk deretter på «Generate». Prosjektfilene for Visual Studio genereres nå og legges i mappen valgt til binaries. I eksempelet er dette «C:\Build\VTK-6.1». I denne mappen finnes nå filen «VTK.sln», bruk denne for å åpne prosjektet i Visual Studio. Dette kan ta litt tid.



Figur 3.9: VTK: Prosjektfilen åpnes i Visual Studio for å starte kompilering.

Når prosjektet er ferdig lastet inn i Visual Studio høyreklikker man på «ALL_BUILD» og velger «Build», vær oppmerksom på at dette kan ta veldig lang tid.



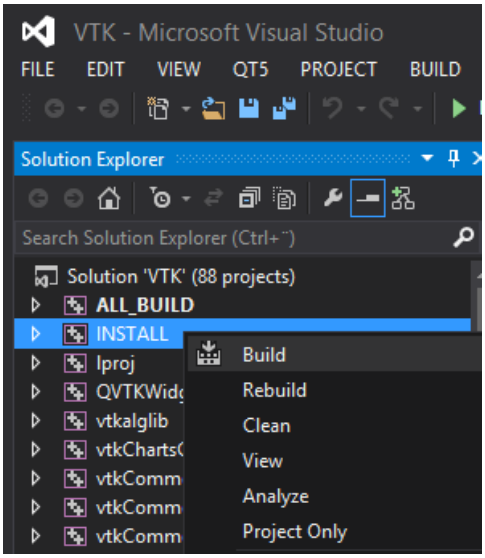
Figur 3.10: VTK: Bygg først «ALL_BUILD».

Da det flere moduler som er avhengige av at andre alt skal være bygget ferdig er det nødvendig å bygge «ALL_BUILD» flere ganger, byggingen gjentas til konsollen melder «0 failed», VTK er nå ferdig kompilert.

```
10>------ Build started: Project: vtkViewsQt, Configuration: Debug x64 ------
11>------ Build started: Project: vtkViewsGeovis, Configuration: Debug x64 -----
10>     Creating library C:/Build/VTK-6.1/lib/Debug/vtkViewsQt-6.1.lib and objec
10>  vtkViewsQt.vcxproj -> C:\Build\VTK-6.1\bin\Debug\vtkViewsQt-6.1.dll
12>------ Build started: Project: vtkGUISupportQtWebkit, Configuration: Debug x6
11>     Creating library C:/Build/VTK-6.1/lib/Debug/vtkViewsGeovis-6.1.lib and o
11>  vtkViewsGeovis.vcxproj -> C:\Build\VTK-6.1\bin\Debug\vtkViewsGeovis-6.1.dll
12>     Creating library C:/Build/VTK-6.1/lib/Debug/vtkGUISupportQtWebkit-6.1.li
12>  vtkGUISupportQtWebkit.vcxproj -> C:\Build\VTK-6.1\bin\Debug\vtkGUISupportQt
13>------ Build started: Project: ALL_BUILD, Configuration: Debug x64 ------
13>  Build all projects
========== Build: 13 succeeded, 0 failed, 73 up-to-date, 0 skipped ==========
```

Figur 3.11: VTK: Bygging ferdig når det rapporteres «0 failed».

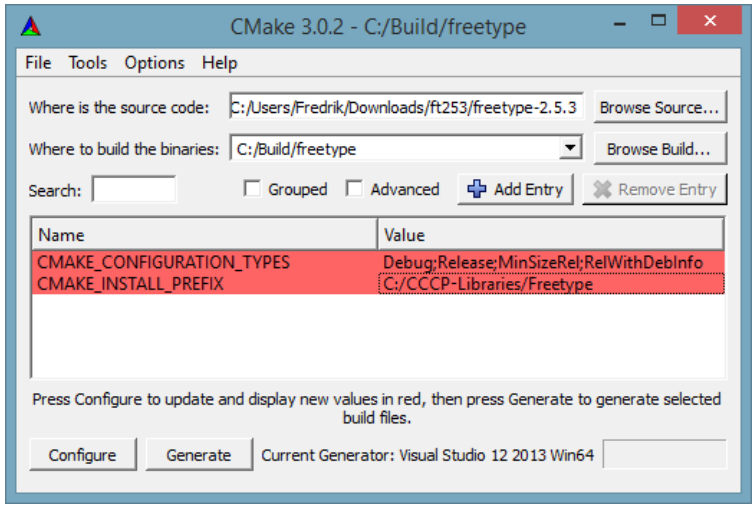
For å avslutte installasjonen av VTK skal modulen «INSTALL» bygges en gang, dette installerer alle filer til installasjonsmappen som ble definert i figur 3.5.



Figur 3.12: VTK: Bygg «INSTALL» en gang, og VTK er ferdig.

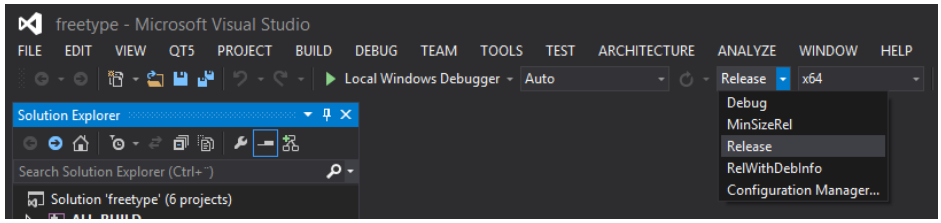
FreeType

Fremgangsmåten er mye den samme som VTK, fremgangsmåten forkortes derfor noe.



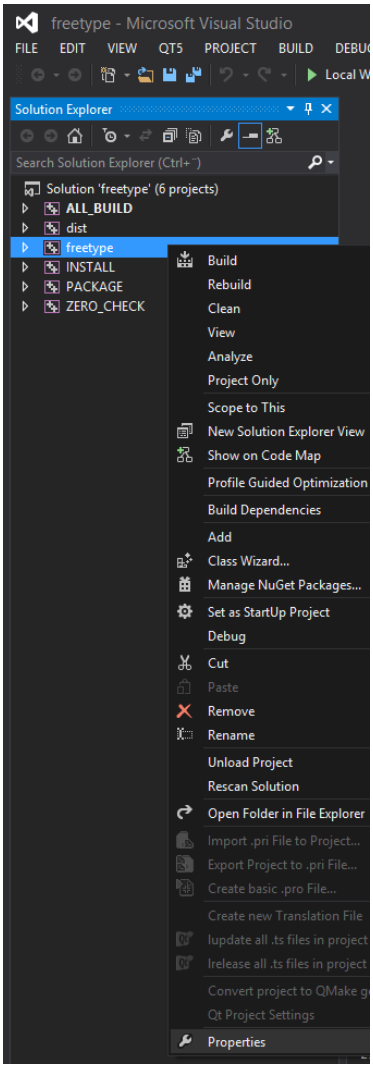
Figur 3.13: FreeType: Trenger bare en runde med konfigurasjon.

Prosjektfilen åpnes så i Visual Studio og sett byggemåte til «Release»



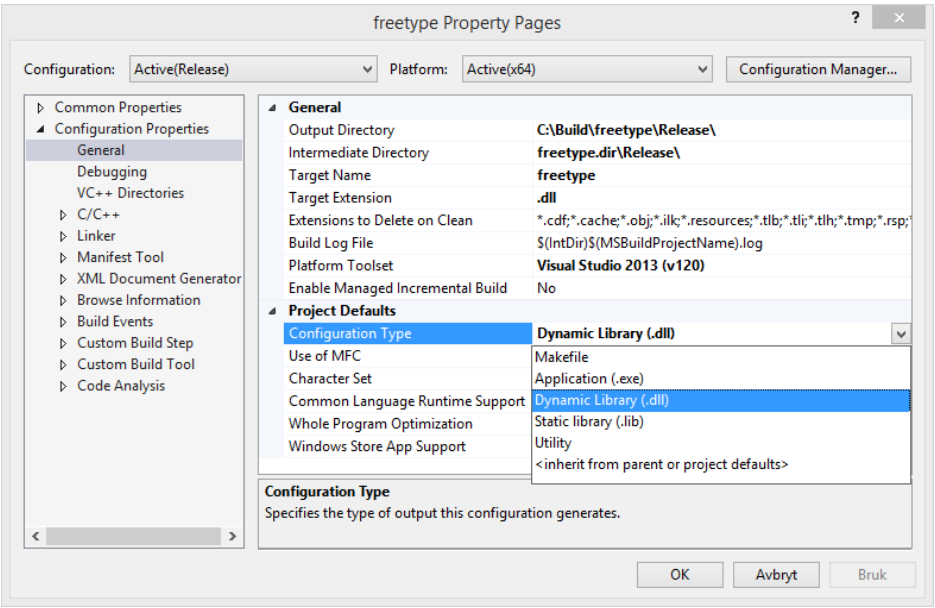
Figur 3.14: FreeType: Sett byggetype til «Release».

Installeres likt som VTK, så må det genereres en dll-fil i tillegg. Etter at «ALL_BUILD» og «INSTALL» er kjørt, åpne innstillinger for modulen «FreeType».



Figur 3.15: FreeType: Åpne innstillinger for modul «FreeType».

Sett feltene «Target Extension» og «Configuration Type» til .dll

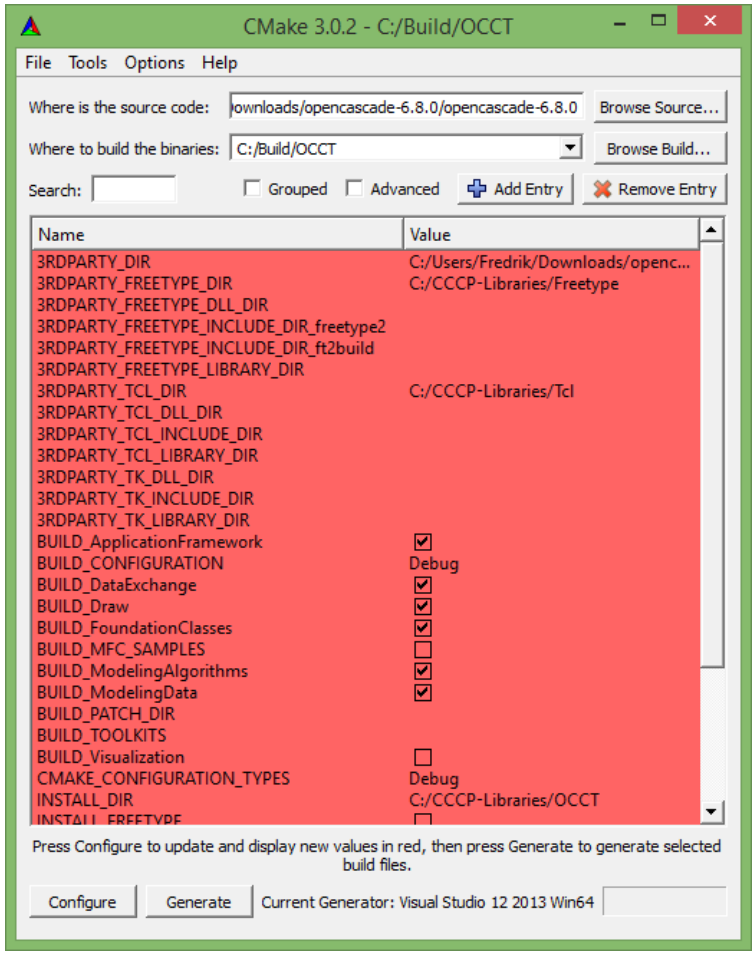


Figur 3.16: FreeType: Sett byggetype til «.dll».

Kjør så «ALL_BUILD» og «INSTALL» igjen. Filen «freetype.dll» skal nå være generert og ligge byggemappen, i dette tilfellet er det `C:\Build\freetype\Release\`. Denne dll-filen må flyttes til mappen «bin» i installasjonsmappen for FreeType, her `C:\CCCP-Libraries\FreeType\bin\`.

OpenCASCADE

Fremgangsmåten er tilnærmet identisk med VTK, med det unntak at i CMake må plasseringen til FreeType og Tcl legges til.



Figur 3.17: OpenCASCADE: Variabler satt for konfigurering av prosjekt.

3.5.4 Oppsett i Visual Studio

Når alle nødvendige biblioteker eksisterer på utviklingsmaskinen er det klart for å åpne prosjektet i Visual Studio, det er ikke gitt at miljøet er identisk med forrige utvikler og prosjektet må da settes opp på nytt selv om det importeres helt greit.

Alle innstillinger gjøres på modulen «CCCP-UserInterface». Under «Debugging» - «Environment» legges følgende tekststreng til som en del

av «PATH»:

```
PATH=$(QTDIR)\bin%3b"$(QTDIR)\bin%3b$(PATH);C:\CCCP-Libraries
\OCCT\bin;C:\CCCP-Libraries\FreeType\bin;C:\CCCP-Libraries\
gmsh
```

Under «VC++ Directories» - «Include Directories» legges følgende mapper til, de må selvfølgelig tilpasses versjon og plassering:

```
C:\CCCP-Libraries\VTK\include\vtk-6.1
C:\Qt\5.3\msvc2013_64_opengl\include
C:\CCCP-Libraries\OCCT\inc
```

Under «Linker» - «Additional Library Directories» legges følgende mapper til:

```
C:\CCCP-Libraries\VTK\lib
$(QTDIR)\lib
C:\CCCP-Libraries\OCCT\lib
```

Visual Studio er nå ferdig satt opp og klart til å kompilere og kjøre prosjektet!

3.6 Lisenser

En ting å legge merke til når det benyttes eksisterende rammeverk og andres åndsverk også fører med seg kravet om å ta hensyn til forfatterens kopibeskyttelse og pålegg om rett bruk. Hvilke lisenser ulike rammeverk har vært utgitt med har også vært med på å avgjøre hvilke rammeverk som ble inkludert.

Alle de inkluderte rammeverkene, med unntak av gmsh, er lisensiert under BSD eller LGPL. Dette gir muligheten til å lage egne implementasjoner med frihet til å lisensiere sluttproduktet slik en selv måtte ønske. gmsh er på den annen side lisensiert under GPL, dette krever at alle programmer som bruker hele eller deler av gmsh også må lisensiere sitt sluttprodukt for offentligheten med en GPL-lisens. [13]

Tredelingen av programmet, som beskrevet i kapittel 2.2 kommer til nytte også her. Oppdelingen gir at endringer gjort i på gmsh kun trenger å offentliggjøres som et selvstendig «meshing»-bibliotek, og andre deler av

programmet som i fremtiden kan gi i konkurransefortrinn fremdeles kan holdes hemmelig.

Kapittel 4

Arbeidsprosessen

Utviklingen systemer så komplekse som dette vil innebære mange forskjellige arbeidsoppgaver og problemaspekter gjennom prosessen.

4.1 Forberedelser

Siden denne oppgaven er en videreutvikling av eksisterende programvare er det slik at det også følger med en omfattende kodebase, og oppgaven med å ta over en noen andres kode uten å ha forfatteren tilgjengelig for å oppklare usikkerheter er en langtekkelig prosess.

4.1.1 Kodebasen

Årsaken til tidsforbruket ved å forstå andres kode ligger i at det programmeringsteknisk alltid finnes flere måter å løse et problem, selv om løsningen er implementert i samme programmeringsspråk. Her er det elementer som utdanning, erfaring og måten hvert individ går frem for å takle et problem som spiller inn.

Arbeidet som tidligere var blitt lagt ned i å dokumentere alle metoder ble satt veldig pris på, og gjorde det veldig mye lettere å sette seg inn i tankegangen og flyten i programmet sånn det sto ved overtagelse. Behjelpelig var det også at kildekoden var akkurat slik forrige utvikler hadde forlatt den, med utskrifter av variabler og nyttig informasjon til konsoll, og at den ikke var «ryddet opp» ved overlevering.

Det er fra starten av etablert kodekonvensjoner i prosjektet, disse forklarer grunnreglene for hvordan koden skal skrives og tar avgjørelser på punkter det ofte er uenighet om blant utviklere, se kapittel 6.1.3. Likevel ble her brukt tid på å lære hvordan kopiere «stilen» slik den var skrevet i prosjektet til nå, i håp om at dette forminsker arbeidet som må legges ned for fremtidige utviklere på dette prosjektet.

4.1.2 Oppsett av miljø

Et hull i dokumentasjonen fra tidligere var hvordan miljøet må settes opp på utviklingsmaskiner for å kunne starte med dette prosjektet. Det gikk med mye tid for å finne ut av hvordan alle de refererte bibliotekene skulle settes opp og inkluderes på rett måte for å få en smidig og smertefri kompilering.

I denne oppgaven er det forsøkt å lage en grundig gjennomgang av fremgangsmåten for å få satt opp utviklingsmiljøet på en ny maskin, det finnes i kapittel 3.5.

4.1.3 Formålsavklaring

Oppgaveteksten mangler en definisjon på når utviklingsarbeidet er «ferdig» for denne oppgaven, og det viste seg i tillegg vanskelig å definere via avgrensningene satt av forfatteren. Det ble derfor tatt en beslutning på å jobbe med én deloppgave om gangen, og nye oppgaver ble prioritert inn i køen etterhvert som de dukket opp.

4.2 Utvikling

4.2.1 Dokumentasjon

På grunn av prosjektets størrelse, og da dokumentasjonen viste seg å være såpass nyttig under oppstarten av prosjektet som ny utvikler, er det fortsatt med den gode trenden å dokumentere all kode som ble produsert i prosjektet. Til å generere dokumentasjonen brukes DoxyGen, dette programmet leser all kildekode i prosjektet, samt kommentarer i kildekoden som er innledet med spesielle tegn. Ved å holde dette til koden og i nærheten av metodene det gjelder holdes dokumentasjonen bedre oppdatert,

fordi det er lett å oppdatere kommentarene hvis en først er i gang med å endre på metoden.

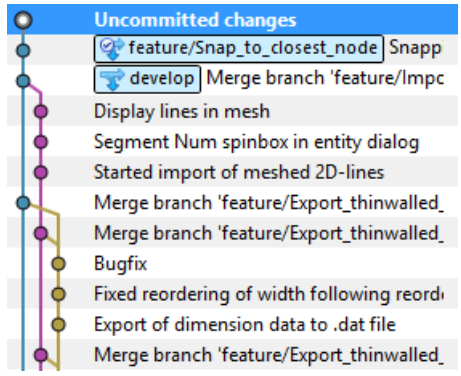
Denne informasjonen brukes til å generere en nettside hvor en kan klikke seg rundt for å finne ut mer om de forskjellige elementene i koden. DoxyGen produserer også filer i LaTex-format som kan brukes til å produsere dokumentasjon på papir, denne dokumentasjonen er lagt ved i vedlegg C.

4.2.2 Revisjonskontroll

Revisjonskontroll er en helt essensiell del av større utviklingsprosjekter hvor det er flere som jobber med utvikling av samme program. Da dette gir flere utviklere mulighet til å jobbe parallelt med samme kildekoden uten å ødelegge for hverandre. I figur 4.1 kan hver fargede «sti» være laget av forskjellige utviklere, og slås sammen med hovedstien når en funksjon er ferdig utviklet.

Systemet for revisjonskontroll gir også muligheten til å gå tilbake til gamle versjoner av kildekoden, noe som er svært fordelaktig dersom man prøver å lokalisere eller utbedre feil som har oppstått underveis.

Alle koderevisjoner lagret vet hjelp av programmet SourceTree i dette prosjektet, det er også lagt opp på en privat GitHub-konto, hvor tanken er å dele hele denne basen med ansvarlig veileder slik at fremtidige prosjekter også skal kunne nyte godt av en samlet langsgående historie.



Figur 4.1: Eksempel på stier og versjoner i Git.

4.2.3 Testdrevet utvikling

Testdrevet utvikling er en effektiv og god metode for å konsekvent skrive velfungerende kode, og det var fra tidligere lagt opp til testdrevet bruk av testdrevet utvikling via Qts opplegg for testing. Dette viste seg å ikke være så godt integrert i Visual Studio som opprinnelig ønsket, og medførte en ganske arbeidskrevende prosess for å få gjennomført testingen.

Det ble i denne omgang vurdert at testdrevet utvikling er en stor fordel, men når det følger med ekstra merarbeid utover det å lage testene i seg selv vi føre til dårlig stemning. Siden det alt finnes et fungerende program med gode visualiseringsmuligheter ble det brukt som visuell bekreftelse på at koden fungerte.

4.2.4 Brukertesting

Som en del av utviklingen av nye funksjoner og elementer i programmet har det blitt brukt brukertesting som et hjelpemiddel for å få trygghet i at utviklingen var på rett vei, det var også en god kilde til inspirasjon og nye idéer. Brukertesting har gitt verdifull innsikt i hvordan forskjellge brukere tenker, reagerer og hva de ser for seg dette programmet skal gjøre. Brukertestingen er forklart mer i kapittel 5.4.

Kapittel 5

Brukergrensesnittet

Brukergrensesnittet er essensielt for å vinne tilliten til nye brukere, samt å beholde eksisterende brukere. Det er denne delen av programmet alle brukere ser hver eneste gang de gjør noe som helst, og det er viktig at det oppleves trygt, effektivt og smidig.

Brukervennlighet har alltid vært et fokusområde for utviklingen av dette programmet, og planleggingen og testingen av de nye elementene i brukergrensesnittet har derfor vært viktig for videreutviklingen.

5.1 Brukskvalitet

Brukskvalitet er definert av *ISO 9241-11* som «i hvilken grad et system er anvendbart, effektivt og tilfredsstillende i bruk for bestemte brukere med bestemte mål i en bestemt kontekst».

Historisk sett har god brukskvalitet vært forbeholdt de systemene hvor feil bruk kunne føre til fatale konsekvenser (fly, kjernekraft og militæret). Etter hvert var det flere firmaer som så at god brukskvalitet også var anvendelig også i konsumentmarkedet, et eksempel er Apple Computer som på bakgrunn av dette kom opp med vindusbasert navigasjon på datamaskiner. Metodene de brukte for å komme opp med produktene sine var iterative designprosesser med praktisk uttesting på forsøkspersoner (brukertesting).

5.1.1 Jacob Nielsen

Jacob Nielsen fokuserer på hvordan et system oppfattes av brukeren, og hvordan det definerer kvaliteten på systemet. I sin bok «Usability Engineering» fra 1993 sier Jacob Nielsen at et system god brukskvalitet hvis systemet er:

Lett å lære Så nye brukere raskt kan produsere noe.

Effektivt Gir ekspertbrukere høy produktivitet.

Lett å huske Så sporadiske brukere kan plukke opp igjen systemet uten å måtte lære alt på nytt.

Feilfritt og feiltolerant Slik at verken brukere eller system gjør mange feil, og at disse feilene er enkle å hente seg inn fra.

Behagelig å bruke Slik at brukerene kommer tilbake uten tvang.

5.1.2 Don Norman

Don Norman så på brukskvalitet fra systemets perspektiv, og beskrev i sin bok «The Design of Everyday Things» fra 1988, prinsipper for hvordan systemer skulle designes for å gi den gode brukskvaliteten som beskrevet av Jacob Nielsen.

Consistency

Vi lærer ved å oppdage mønstre, og nye situasjoner blir mer håndterbare om vi kan bruke kjente mønstre på ukjente problemer. Derfor skal elementer som ser like ut gjøre det samme, og elementer som gjør det samme se like ut.

Inkonsistens tvinger brukerene til å memorere hvilke i tilfeller forskjellige regler gjelder, og det gir systemet lite tillit.

Visibility

Brukere oppdager hvilke funksjoner som er tilgjengelige ved å gjøre en kjapp visuell inspeksjon av systemet, alle tilgjengelige handlinger skal derfor være lett synlige, og systemets tilstand skal være kjapt observerbart.

Elementer bør også være sortert etter relevans, og elementer med samme verdi burde være plassert i nærheten av hverandre. Det er viktig å ikke overfylle menyene med elementer, det gjør dem vanskelige å finne frem i.

Affordance

«Affordance» beskriver hvordan et fysisk objekt gjør det den oppfattes å skulle gjøre. Hvor god «affordance» er en lysbryter som skrur av og på taklampen i samme rom, mens dårlig «affordance» er hvis lysbryteren åpner og lukker en dør i et annet rom. Uthevede knapper i et grensesnitt ser klikkbare ut, og har derfor god «affordance».

Mapping

Dersom man ved å se på en kontroll kan skjønne hva den gjør har den god «mapping», i et brukergrensesnitt kommer dette ofte av gode ikoner, slik at brukeren ikke trenger å lese en tekstbeskrivelse eller å prøve seg frem.

Feedback

Enhver handling utført i et grensesnitt burde også gi en tilbakemelding på at handlingen er mottatt av systemet. Dette kan gjøres ved «aktivasjons-feedback», hvor for eksempel en knapp som trykkes på endrer fasong og vise at det skjedde noe. Den andre varianten er «hendelses-feedback», her kommer tilbakemeldingen i form av at systemet skifter status i det handlingen er utført.

Constraints

Et grensesnitt må alltid være designet med begrensninger slik at systemet aldri kan gå inn i en ugyldig tilstand. «Constraints» sørger for at det aldri blir skrevet inn ugyldig data, eller at en deaktiverer knapper som ikke kan brukes.

5.1.3 Gestaltprinsippene

Øyet er veldig godt på å se mønster, og avvik fra mønster, og gestaltprinsippene stammer fra de persepsjonspsykologiske forklaringene på hvordan

vi sanser og organiserer visuelle inntrykk.[14]

Det er seks gestaltprinsipper:

- Gruppering av like elementer
- Kontinuitet og konvergens i linjer
- Mental komplettering av mønster
- Likhet i form
- Likhet i farge
- Dybdeføelse med forgrunn og bakgrunn.

Disse reglene for struktur og gruppering gjør det lettere for brukere å finne roen mens man bruker programmet, og det gjør systemet mer behagelig å bruke.

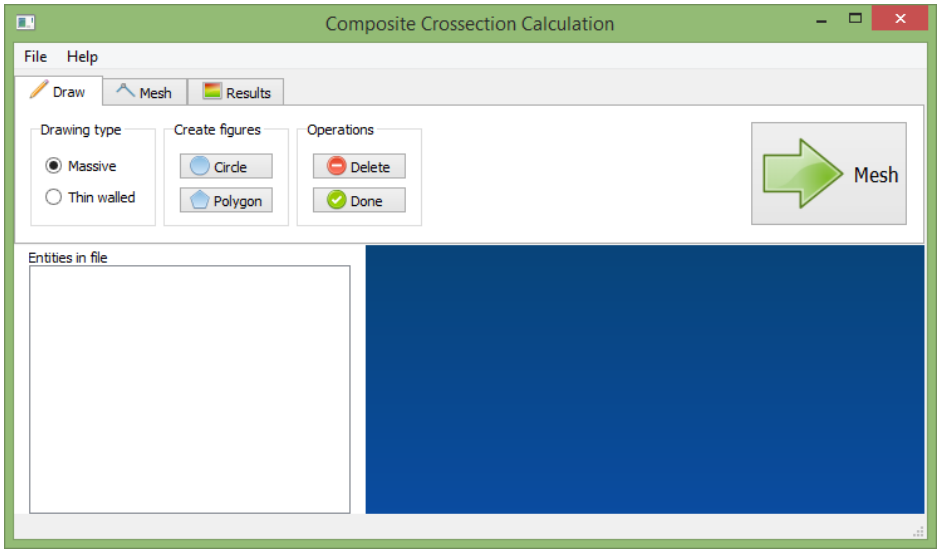
5.2 Arbeidsflyt

I hele programmet er det et gjennomgående tema at all bevegelse skjer fra venstre mot høyre for å fullføre en handling.

5.2.1 Hovedmenyer

I første modus (figur 5.1) brukeren møter er det mulighet for å tegne eller importere tverrsnittene som det skal jobbes med videre. I dagens brukstilfelle er det tenkt at størsteparten av brukeren skal tegne sitt eget tverrsnitt, filimport skjer via `File -> Open`. Knappene som kan endre tilstanden mest er fanene på toppen, disse er likevel så små at de ikke innbyr til klikking direkte, og det var heller ingen i brukertestene som prøvde å klikke på dem. Sånn de står nå fungerer forteller om veien videre i programmet.

I menyen for tegneverktøyene (figur 5.1) er knappene som hørere sammen gruppert etter tilhørighet til samme steg i prosessen, og oppfyller med det gestaltprinsippet. Flyten for tegningen av hver form går fra venstre mot høyre gjennom de tre bolkene med knapper, når tegningen er ferdig



Figur 5.1: Meny for tegnevisning, valg for tverrsnittstype, standardformer og for å slette eller fullføre en form.

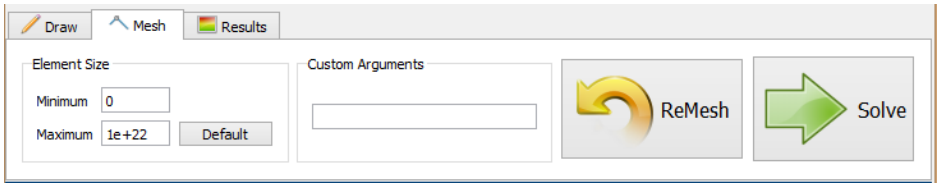
går man videre ved å trykke på den store grønne pilen til høyre i vinduet som tar deg videre til «meshing».

Hovedmenyen for «meshing» (figur 5.2) har litt andre valg enn forrige meny, da mesheren kun kan kontrolleres via kjøreparametere og tekststrenger, men likevel føles menyen kjent.

Det så sende inn parametre via tekststreng til gmsh krever for det første at brukeren har kjennskap til gmsh fra før, og for det andre at brukeren vet at det er gmsh som brukes til «meshing». Valgene her er for de aller mest avanserte brukerene, men for dem kan det være en veldig god funksjon å ha tilgjengelig. Det burde likevel informeres bedre om hva som kan sendes inn via tekstboksen.

Valget er nå enten å kjøre genereringen av «meshet» en gang til, eller å fortsette videre til resultatvisningen. Dette valget tas med knappene «ReMesh» (gul) eller «Solve» (grønn), den store grønne pilen signaliserer tydelig at brukeren er fornøyd med «meshet» sånn det ser ut. «ReMesh»-knappen er det motsetningen, både er den gul og peker i motsatt retning av definert «fremover» i programmet, det er ingen tvil om at den blir der

den er.



Figur 5.2: Meny for «meshing», valg for største og minste elementstørrelse og et felt for å sende med egne komandoer til «mesher».

Menyen i resultatvisningen (figur 5.3) ligner mer på tegnevisningen, og her velges det hva slags presentasjon som ønskes av resultatene. Bruk av standardelementer som checkbokser og radioknapper som oppfører seg etter standarden gir klare forventninger om bruk, og oppfyller dem.

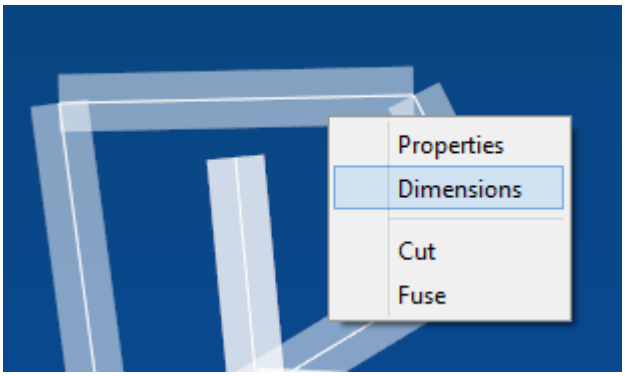
Figur 5.3: Meny for resultatvisning, valg for visning av «mesh» og/eller resultat, standardformer og for å slette eller fullføre en form.

Bruken av fanene i hovedmenyen gjør bruken av programmet renere og mer oppgavefokusert, det er mindre unødvendige valg å vurdere til enhver tid. Det gjør det også lettere å finne frem til menyene enhver tid, noe som harmonerer godt med Don Normans «Visibility» og Jacob Nielsens «Lett å lære» og «Lett å huske».

5.2.2 Dialogvindu

Dialogvinduer blir ofte brukt for å abstrahere vekk spesialiserte valg som bare er aktuelle for deler av programmet fra hovedvinduet. Hvis det høyreklikkes på en tynnvegget entitet har kontekstmenyen ett alternativ mer enn vanlig, «Dimension», se figur 5.4.

Ved valg av dette alternativet får man opp et dialogvindu tilpasset til å behandle tynnveggede entiteter, på kantnivå. Som vist i figur 5.5 det kun



Figur 5.4: Kontekstmeny for tynnveggede entiteter, alternativet «Dimensions» dukker kun opp ved høyreklikk på en tynnvegget entitet.

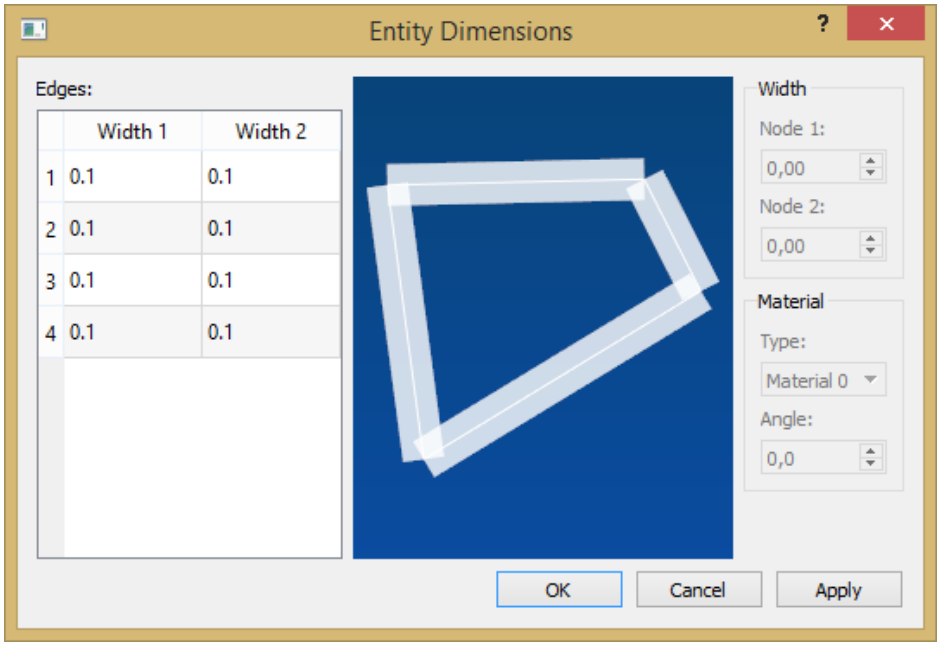
er den valgte entiteten som dukker opp i visualiseringsvinduet i midten.

Bevegelsen her er at det først velges kanten som skal endres på i menyen til venstre, deretter settes parametrene i på høyresiden av dialogvinduet. Her er er det ekte «Avbryt»-funksjonalitet som forkaster alle endringer, og visualiseringen oppdateres direkte ved endring av parametre. Visualiseringen kom opp etter at brukere ikke alltid fikk resultatet de forventet ved bruk av dette dialogvinduet, tryggheten en sanntidsoppdatert visualisering ga brukerene gjorde slik at ingen tvilte på at de gjorde det riktig ved bruk av dette vinduet senere.

5.3 Tastatursnarveier

Tastatursnarveier er funksjonaliteten for elitebrukerne som ønsker å være ekstra effektive, dette ønsker vi å tilby, og det passer rett inn i Jacob Nielsens krav om «effektivitet». Siden tastatursnarveiene kun eksisterer for ekspertbrukerne er det viktig at all funksjonalitet også finnes visuelt i grensesnittet. Et annet punkt å vurdere er hva snarveiene skal være, de burde være lette å huske slik at de nye brukeren har mulighet til å huske dem, men samtidig raske nok til at ekspertbrukerne bruker dem.

Snarveiene er nå `[Ctrl]` pluss en bokstav som assosieres med operasjonen som skal utføres, for å hjelpe hukommelsen. Tastatursnarveiene er listet opp i tabell 5.1.



Figur 5.5: Dialogvindu for valg av tykkelse og material per kant i en form.

Snarvei	Operasjon
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/Slett

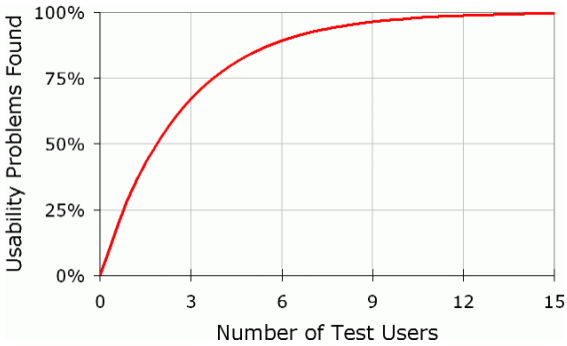
Tabell 5.1: Liste over alle tastatursnarveier implementert i programmet.

5.4 Brukertesting

Som et ledd av utviklingen var det ønskelig å gjennomføre brukertester av grensesnittet. Personer har varierende inntrykk av hvordan en grensesnitt

skal brukes første gang de ser det, og disse er viktige å kartlegge for å lage et grensesnitt som passer så mange som mulig. Det er lett å tro at ens egen oppfattelse er «riktigst» og «den beste», for alle som har denne følelsen anbefales det å teste sitt syn ved å gjennomføre et par brukertester.

Det ble gjennomført fem brukertester som alle fulgte oppskriften i tabell 5.2. Valget av akkurat fem brukere virker kanskje tilfeldig, men i løpet av en serie med brukertester er det ofte de samme problemene dukker opp flere ganger. Ved test av akkurat fem brukere dukker det tilfredsstillende mange problemer opp (se figur 5.6), men med minimal bruk av tid.[1]



Figur 5.6: Andel problemer med brukskvalitet funnet ved et gitt antall brukertester.[1]

Viktige resultater fra brukertester var at brukerene brukte lang tid på å finne riktig kontekstmeny for å sette tykkelse og andre parametre på entitetene i tverrsnittet. Denne menyen burde være tilgjengelig fra et annet sted, sammen med andre vinduer som nå er tilgjengelig kun fra kontekstmenyen ved høyreklikk.

Brukerene savnet en illustrasjon for å skille kantene fra hverandre og nodene fra hverandre i dimensjonsdialogen (se figur 5.5), her burde det vært visning av «Edge 1», «Edge 2» og så videre for alle kanter samt visning av «Node 1» og «Node 2» for kanten som alt er valgt, slik at brukeren slipper å prøve seg frem. Dette var med i utkastet til dialogen, men ble lagt vekk av hensyn til utviklingstid.

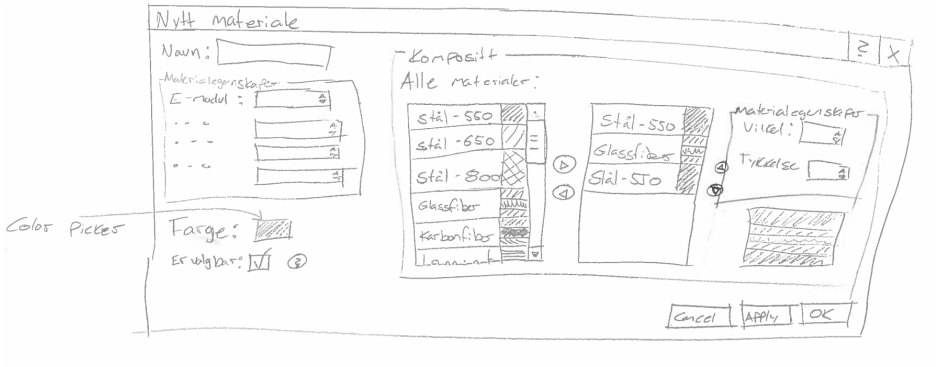
En mindre viktig utbedring ved dimensjonsdialogen er muligheten til å velge kant ved å klikke i selve visualiseringen.

Oppgave	Beskrivelse
Lag et massivt tverrsnitt	Lag et massivt tverrsnitt med vilkårlig form, som deretter slettes
Lag et tynnvegget tverrsnitt	Lag et lukket tynnvegget tverrsnitt med vilkårlig form, lag et åpent tverrsnitt med vilkårlig form
Tilpasse tverrsnitt	Endre veggtykkelsen på det lukkede tverrsnittet til 0,4, sett materialet på halvparten av veggene i det åpne tverrsnittet til «Material 3»
Generere mesh	Generer et mesh fra tegnede tverrsnitt, stemmer meshet med forventningene?

Tabell 5.2: Liste oppgaver som ble gjennomført under brukertest av programmet.

5.4.1 Materialdesigner

I arbeidet med en modul for design av materialer har brukeren stått i fokus, det har vært tegnet flere skisser det har vært utført enkle brukertester på gjennom prosessen for å få et godt sluttprodukt.

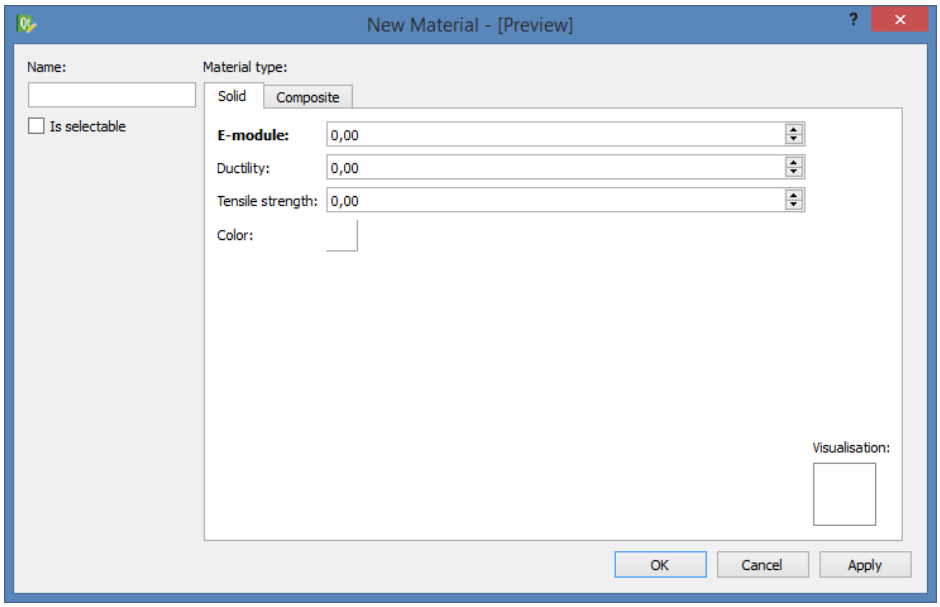


Figur 5.7: Siste skisse i prosessen for å lage et nytt brukergrensesnitt til materialdesigner.

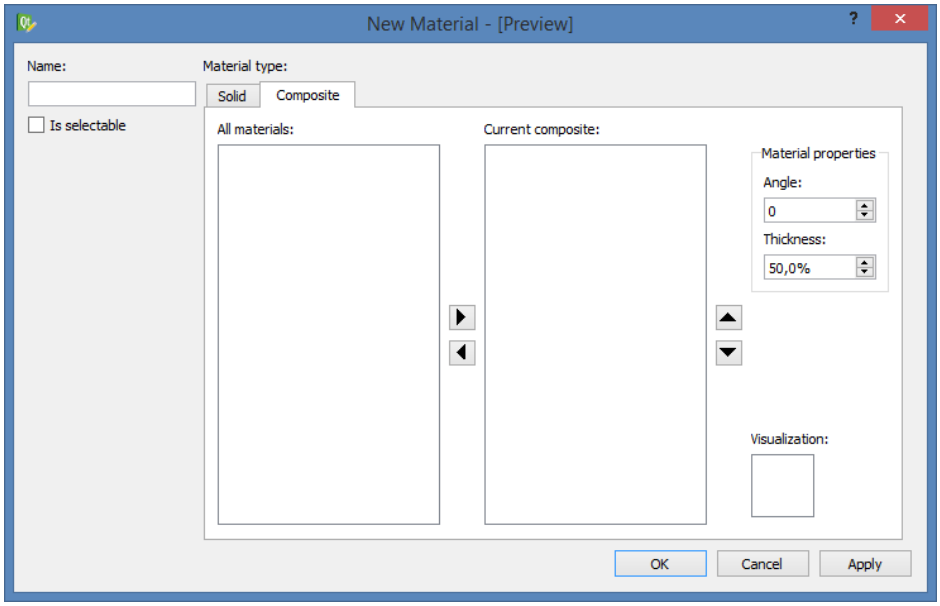
En funksjon som er viktig å nevne med materialdesigneren er den gjennomgående visualiseringen, dette er en samling farger og mønstre som genereres automatisk ut fra informasjon gitt av brukeren om de forskjellige lag og deres farge og vinkel på kompositter. Det er tenkt at denne visualiseringsboksen skal gå igjen i alle opplistinger av materialer, samt fargeleggingen av tverrsnitt i tegnevinduet.

På skissen i figur 5.7 ble det gjennomført brukertester. Brukeren fikk i oppgave å lage flere nye materialer ut fra en liste med både relevante og urelevante spesifikasjoner. Et viktig resultat av brukertesten var problemer med å skille feltene som var relevante for kompositttmaterialer fra feltene som kun var relevante for homogene materialer. Dette ble ble oppdatert i designprototypen vist i figur 5.8 og 5.9.

Disse figurene er bilder tatt fra forhåndsvisningen i Qt, hvor endelig designprototype ble laget for å se hvordan elementene faktisk ville se ut. Disse modellene ble ikke brukt til brukertesting da det ikke er mulig å fylle dem med testdata, og det tar mye lengre tid å få frem en visjon dersom ting skal se «helt riktig» ut. Skisser som er litt skjeve og tegnet med blyant sørger også for at testobjektet fokuserer på tanken bak og interaksjonen, mer enn hvordan designet ser ut og hvilke farger som brukes hvor.



Figur 5.8: Prototype av materialdesigneren laget i Qt, fra dette vinduet kan brukeren lage homogene materialer. Det er satt av mye plass til å legge inn flere materialegenskaper.



Figur 5.9: Prototype av materialdesigneren laget i Qt, fra dette vinduet kan brukeren sette sammen flere av de eksisterende materialene til et nytt kompositttmateriale, samt sette kompositttspesifikke egenskaper.

Kapittel 6

Videreutvikling av programmet

6.1 Arvede krav

Ved overtagelse av et kodeprosjekt er det alltid valg som er tatt på forhånd, og som det ikke lenger er mulig eller ønskelig å gjøre noe med.

6.1.1 Programmeringsspråk

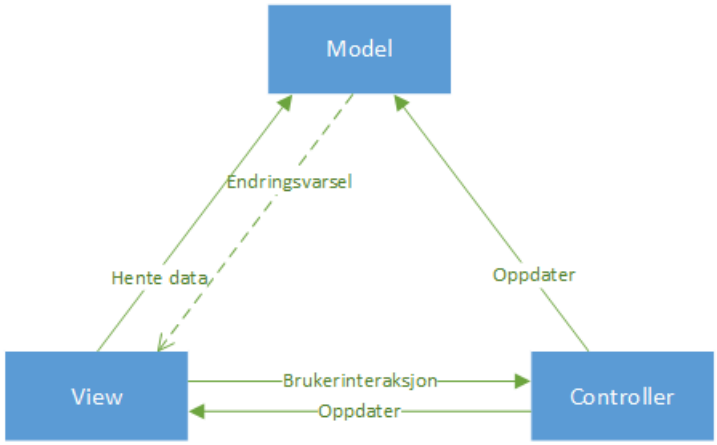
Programmeringsspråket er C++, som følge av et ønske om kryssplattformstøtte og ønsket om å bruke samme programmeringsspråk som «solveren».

6.1.2 Systemarkitektur

Prosjektet bruker Model-View-Controller (MVC)-arkitekturen, et naturlig valg for interaksjonsbaserte programmer slik at en kan skille design fra logikk. I MVC plasseres utseende og design i klassen «View», og være så lite påvirket av den andre klassene som mulig. Data lagres i klassen «Model», og «Controller» kobler de to andre klassene sammen og er klassen som implementeres i resten av programmet. Se figur 6.1 for en illustrasjon.

«Controlleren» inneholder logikken som reagerer på hendelser fra resten av systemet, og oppdaterer «Model» og «View» dersom en handling utføres. Det er i denne situasjonen hvor «Signals & Slots»-funksjonaliteten fra Qt kommer til sin rett når den sørger for at endringer i «Controlleren»

automatisk propageres videre til både «View» og «Model» der dette er nødvendig. Dette verktøyet gjør at alle klasser kan bli observert og være observatør.



Figur 6.1: Sammenhengen mellom komponentene i MVC, «Model» holder på data for applikasjonen, «View» behandler og definerer visning av data mot bruker og «Controller» er kontaktpunktet mot resten av programmet og sørger for å oppdatere «M» og «V» ved endringer.

6.1.3 Kodekonvensjoner

Siden det alt er definert kodekonvensjoner for prosjektet, er de fulgt etter beste evne nå også. Kovensjonene er som følger

Type	Navnekonvensjon
Variabelnavn	Camel case
Funksjonsnavn	Camel case
Klassenavn	Pascal case
Konstanter	Store bokstaver

Tabell 6.1: Navngivingskonvensjoner i dette prosjektet, se tabell 6.2 for beskrivelser.

Det er også krav til at alle funksjoner skal ha en kommentar i DoxyGen-format som beskriver funksjonens innhold.

Navn	Eksempel	Beskrivelse
Camel case	camelCase	Ordet begynner med liten bokstav. Alle påfølgende ord starter med stor bokstav
Pascal case	PascalCase	Alle ord starter med stor bokstav
Store bokstaver	KONSTANTER	Alt skrives med stor bokstav.

Tabell 6.2: De forskjellige navnekonvensjonene brukt i programmering.

6.2 Utviklingsoppgaver

6.2.1 Tynnvegget tegneverktøy

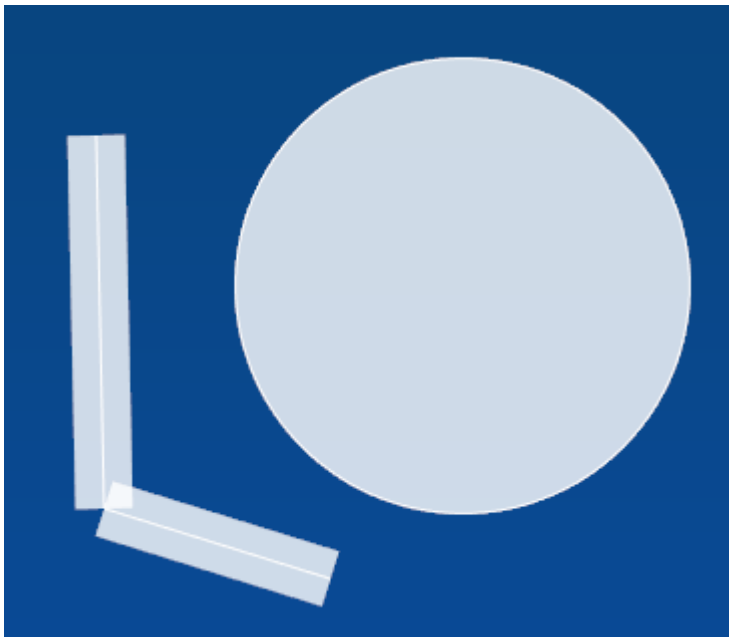
Originalt var det kun mulig å tegne massive tverrsnitt i preprosessoren, og som hovedfokus ved oppgaven ble dette tatt tak i først. Det viste seg fort at de tegnede elementene ikke egentlig var definert som massive figurer, men de var hule former som ble fylt med en fyllfarge under tegning i grensesnittet.

Ved å innføre en identifikator på typen tverrsnitt hver enkelt enhet kunne også tegningen av fyllet gjøres på de riktige tegningene. For å tegne tynnveggede entiteter riktig ble fyllfunksjonen gjenbrukt og videreutviklet slik at den også kunne illustrere en veggtykkelse slik den ble definert av brukeren, se figur 6.2. Det ble med dett unngått å måtte lage egne tegneverktøy eller visualiseringsmoduler for å håndtere tynnveggede elementer.

6.2.2 Dimensjonsdialog

Med introduksjonen av tynnveggede elementer er det også et behov for å sette verdier for valg som kun er relevante for disse. Vinduet er illustrert i figur 6.3.

Bruker velger først kanten som skal endres i listen til venstre, endrer så parametre for tykkelse i hver endre på høyresiden. Det er også mulighet



Figur 6.2: Metoden for å tegne fyllet i massive tverrsnitt ble gjenbrukt til å tegne veggtykkelse for tynnveggede elementer.

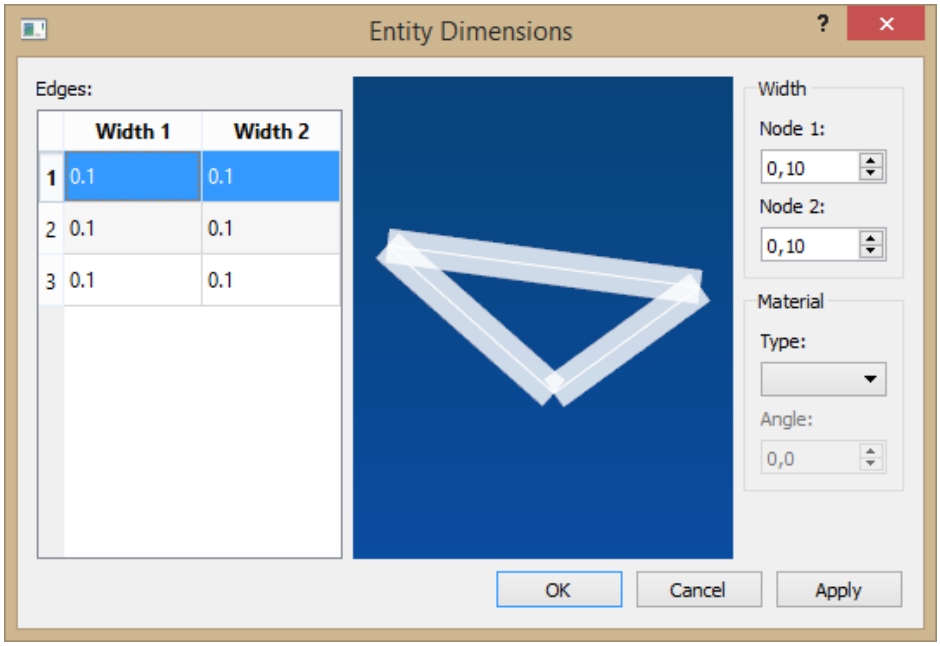
for å sette materialer på hver enkelt kant, og kompositttmaterialer som er avhengige av en vinkel åpner også for å sette det.

Alle endringer er fullstendig reversible ved å benytte «Cancel» eller lukke vinduet.

EntityGraphic

For å gi brukeren mer tilfredsstillende opplevelse ved bruk av dialogvinduet var det behov for en forhåndsvisning av endringene som ble utført på entiteten. Det blå visualiseringsvinduet i midten av dialogen fra figur 6.3 er implementert via «EntityGraphicController», en klasse som ble laget for å kjapt visualisere én entitet, det er en forenklet versjon som også arver fra den originale «GraphicController» som driver visualiseringen i alle deler av programmet.

Implementasjonen av «EntityGraphicModel» og «EntityGraphicCon-



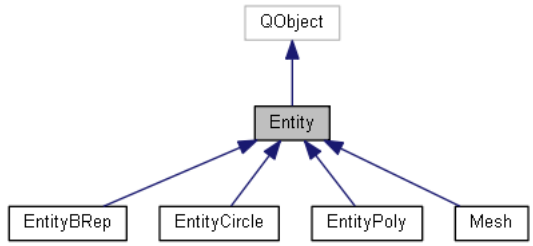
Figur 6.3: Dialogvinduet for valg relatert til tynnveggede elementer.

troller» er gjort så generisk og ryddig som mulig, med tanke på gjenbruk i fremtiden. Med denne klassen er det enkelt å putte inn en visualisering av en enkelt entitet for å gi bruker bedre opplevelse av interaksjon.

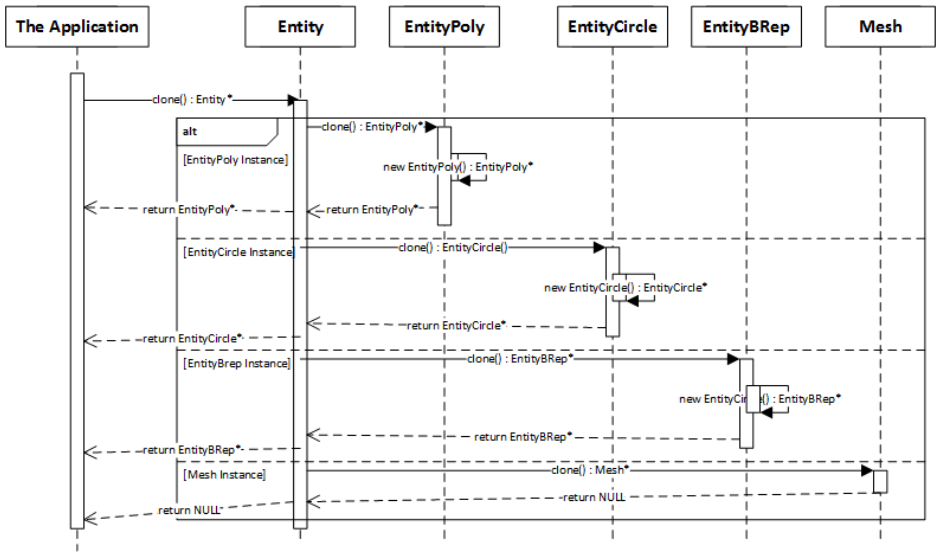
Refaktorering av Entity

For å gi dimensjonsdialogen mulighet vise frem endringene som blir gjort, og samtidig kunne avbryte endringene uten å lagre dem var det behov for å kopiere klassene som beskriver formenene som tegnes, klassene er alle av typen `Entity` og `Entity` arver fra `QObject`, se figur 6.4.

`QObject` kommer fra Qt-biblioteket og må inkluderes for å benytte «garbage collection» (se kapittel 3.2). Det viser seg at klasser av typen `QObject` ikke kan støtter kopieres på vanlig måte med innebygde metoder i C++. Alle konkrete klasser skrives da om for å støtte kloning av seg selv via konstruktør, men siden klassene kun refereres til som den abstrakte `Entity` var det nødvendig med en polymorf klonefunksjon.



Figur 6.4: Illustrert arv av `Entity`.



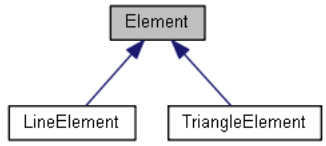
Figur 6.5: Sekvensdiagram for gjennomføring av `Entity:clone()`.

6.2.3 Visning av tynnveggede resultater

Forhåndsvisning av resultatet etter at geometrien var «meshet» førte til at .msh-parseren måtte utvides til å støtte todimensjonale elementer. Dette førte igjen til en utvidelse og refaktorering av klassen `Element`, som holder på geometrien som skal vises fra «meshet».

`Element` ble gjort abstrakt, og delvis refaktorert til `TriangleElement` som holder på trekantede flateelementer. Det ble også opprettet klassen `LineElement` som holder på data for endimensjonale bjelkeelementer, il-

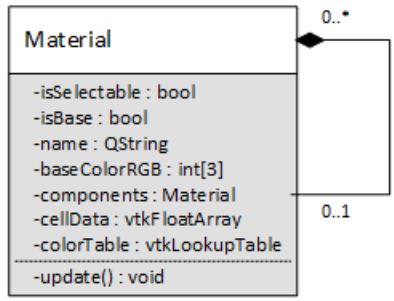
lustrert i figur 6.6. Årsaken til at det ble valgt en struktur med arv og abstrakt hovedklasse var for å tilrettelegge for fremtidige utvidelser, da gmsh har 33 forskjellige datatyper som kan være inkludert i .msh-filen. [15]



Figur 6.6: Illustrert arv av `Element`.

6.2.4 Datastruktur for materialer

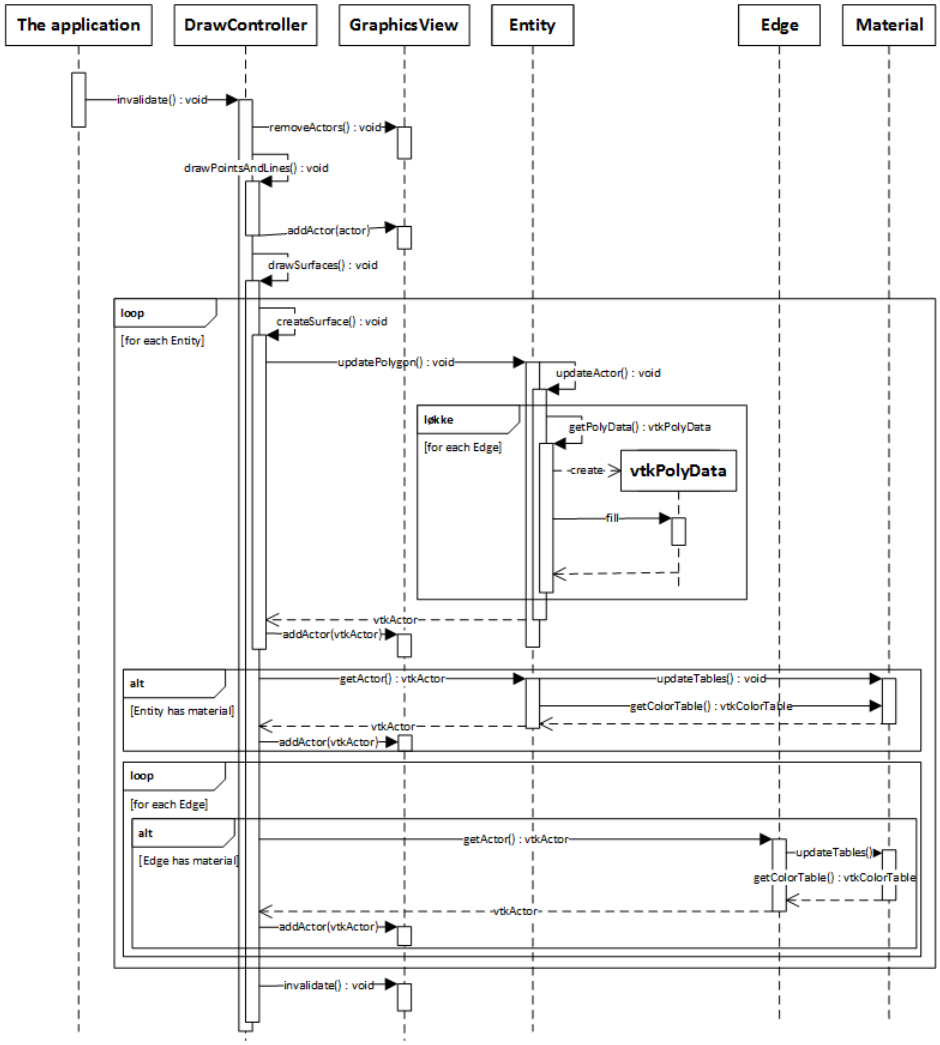
Etter at brukergrensesnittet for materialer ble designet og brukertestet så ble modellering og implementering av datastruktur for materialene gjennomført. Klassediagrammet er vist i figur 6.7.



Figur 6.7: Detaljert klassediagram for `Material`.

Tanken er at kompositttmaterialer skal beskrives ved å sette sammen to eller flere eksisterende materialer for å lage et nytt materiale, pekere til disse materialene er listet opp i `Material.components`, hvor rekkefølgen er relevant og materialer kan gjentas flere ganger. Alle materialer kan tildeles en farge, og denne skal brukes til å generere en visualisering av materialet slik at det er lett å kjenne igjen. `Material.cellData` og `Material.colorTable` hjelper begge til med visualiseringen ved å levere

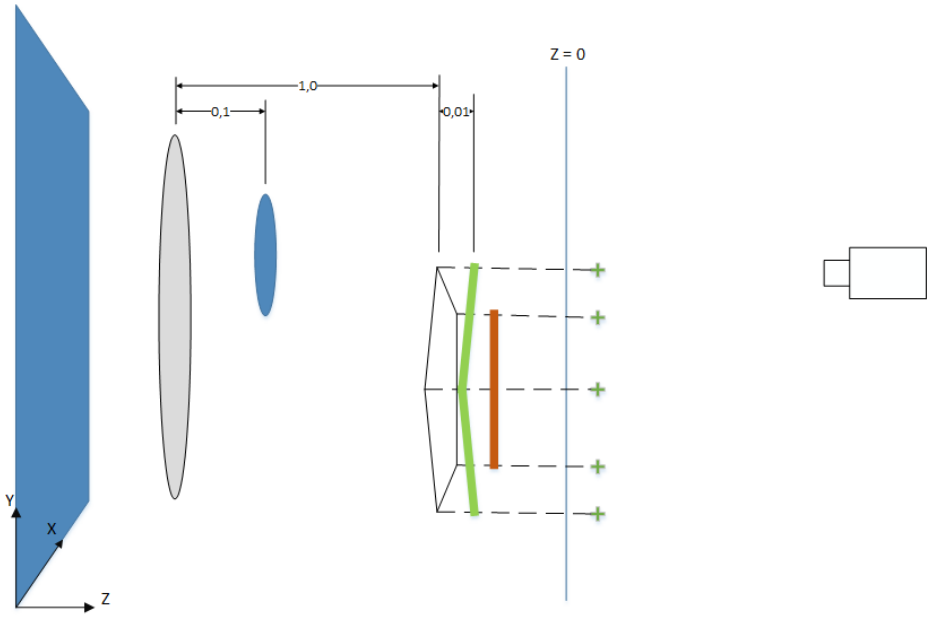
informasjonen som skal til for å fargelegge en «vtkActor». Se sekvensdiagram i figur 6.8.



Figur 6.8: Sekvensdiagram for alle hendelser utført ved tegning av `Element` og `Edge`, med og uten tildelt `Material`.

En hver entitet som lages i programmet tegnes på skjermen som en «vtkActor», men det er slik at VTK ikke støtter at en «vtkActor» har

flere elementer med helt ulike farger, og dette er noe som vil oppstå siden det er lagt opp til hver kant skal kunne ha forskjellige materialer. Ved tegning av en kant som har tildelt et materiale må det tegnes en egen «vtkActor» som et lag over hver kant for å illustrere kompositttmaterialet. Denne teknikken er alt brukt for å visualisere hull i tegnede overflater og fungerer godt, i figur 6.9 er det laget et utkast til hvordan plasseringen av elementer skal fordeles langs z-aksen.



Figur 6.9: Forslag til bruk av dybde under visualisering i applikasjonen ved implementering av materialer, for å vise informasjon som er mer kompleks enn det som støttes av VTK.
Fra venstre mot høyre kommer først den blå bakgrunnen. En massiv sirkel med et hull (blå), alle hull ligger 0,1 nærmere kamera enn entiteten og har samme farge som bakgrunnen. En tynnvegget entitet, entiteter er adskilt med 1,0 mellom hver. Materialer den tynnveggede entiteten består av (grønn og oransje). På positiv side av z-aksen ligger «seeds» (grønne kryss).

Fordelingen av elementer i gitte intervaller av z-aksen legger føringer på antall elementer som kan legges til av hver type. I tabell 6.3 finnes maksimalt antall av hvert element som kan benyttes. Det er knyttet en

tvil til om flere «vtkActors» kan plasseres på samme z-verdi uten å skape konflikter i visualiseringen, hvis viser seg at alle «vtkActors» må fordeles kan det benyttes så mange desimaler som nødvendig, slik som alternativ 2 i tabell 6.3.

Element	Z-nivå	Maks antall
Entitet	1,0	∞
Hull	,1	9
Materiale (alt. 1)	,01	9
Materiale (alt. 2)	,01-,0..01	∞

Tabell 6.3: Antall tilgjengelige plasser for elementer av hver type med en videreføring av dagens plassering av elementer på z-aksen. To alternativer for materialer, hvor alt. 2 kan tas i bruk ved behov.

6.2.5 Snapping av noder

Et viktig bruksscenario ved tynnveggede er gjenbruk av eksisterende noder, det vil si at to forskjellige elementer i tegningen av tverrsnittet skal kunne bruke samme node. For å tilrettelegge for dette ved tegning av tverrsnittet ble det implementert en enkel form for «snapping», ved at en eksisterende node ble gjenbrukt dersom avstand i både X- og Y-retning var under en gitt terskelverdi.

Funksjonen fungerer som planlagt, er implementert i «DrawModel» og kalles fra «vtkSeedCallback». Implementasjonen har dog noen svakheter, for å finne nærmeste node søkes det gjennom alle noder i tegningen hver gang det plasseres ut en ny node. Dette går fort utover effektiviteten og sinker til slutt programmet, spesielt ved tegning av sirkler med mange linjestykker.

Kapittel 7

Konklusjon og videre arbeid

7.1 Konklusjon

Oppgaven har vært en videreføring av et tidligere prosjekt, og har spesielt i tidlig fase båret preg av det med mye arbeid for å sette opp prosjektet på en fungerende måte på ny maskin. Ett av målene med denne rapporten har derfor vært å minimere tidsbruken for fremtidige fortsettelser av dette prosjektet.

Erfaringer sa fort at det lønner seg med god planlegging og gjennomgang før det utvikles store tillegg til programmet, arbeidet med «materialdesigneren» har derfor gått flere runder med brukertester og skisser før endelig resultat ble fastslått.

Programmet har fått gjennomgående støtte for tynnveggede bjelkeelementer fra tegneverktøy til «mesher» og innlesing av mesh til både forhåndsvisning og resultatvisning. Dette arbeidet har også medført noen strukturelle endringer i kjernekoden, disse større endringene menes å være til fordel for fremtidig arbeid og utvikling av prosjektet.

7.2 Videre arbeid

Når det kommer til videreutvikling av applikasjonen er det naturlig å anta at videre arbeid i lang tid fremover vil være å legge til nye hovedfunksjonaliteter. En viktig funksjonalitet vil være å implementere materialverktøyet det er gjort forundersøkelser på og skisser eksisterer, dette vil nok være

et såpass stort prosjekt at det må defineres i en oppgavetekst før en begir seg ut på det.

Mindre, men viktige endringer for å forbedre brukeropplevelse og gi stor verdi i følge brukertester er å legge til identifikasjon av hvilken kant som er valgt og hvilken node som er 1 og 2 i «dimensjonsdialogen». Det er også ønskelig at musepekeren illustrere at systemet er opptatt når «mesher» og «solver» kjører, dette er mest utpreget på trege systemer hvor disse operasjonene kan ta lengre tid.

Forslag til refaktorering er at entiteter burde fortelle kontekstmenyer hvilke valg de har når de klikkes på, slik koden er lagt opp i dag må dette hardkodes inn i hver enkelt listetype. Dette er en større oppgave, men likevel burde dette prioriteres før det er for sent, for å gjøre fremtidig utvikl mer smidig.

Til slutt anbefales «Videre arbeid» fra rapporten til B. Askevold[2], her er det flere gode kandidater til videre arbeid som ikke ble gjennomført i denne oppgaven.

Bibliografi

[1] J. Nielsen, Why You Only Need to Test with 5 Users, NNgroup (des 2014).
URL http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[2] B. Askevold, Brukergrensesnitt for beregning av tverssnittsdata for komplekse bjelketverrsnitt, Masteroppgave, Norges teknisk-naturvitenskapelige universitet (jun 2014).
URL https://daim.idi.ntnu.no/masteroppgave?id=11656

[3] K. Strømstad, Applikasjon for beregning av tverrsnittsparametre for komplekse tverrsnitt, Master's thesis, Norges teknisk-naturvitenskapelige universitet (jun 2014).
URL https://daim.idi.ntnu.no/masteroppgave?id=11626

[4] Kitware Inc., VTK - About (jan 2015).
URL http://www.vtk.org/VTK/project/about.html

[5] Kitware Inc., VTK - Latest build (sep 2014).
URL http://www.vtk.org/VTK/resources/software.html#latestcand

[6] The Qt Company, Qt Open Source Download (sep 2014).
URL http://www.qt.io/download-open-source/

[7] CMake, CMake - Download (sep 2014).
URL `http://www.cmake.org/download/`

[8] OPEN CASCADE, OCCT Download (sep 2014).
URL `http://www.opencascade.org/getocc/download/loadocc/`

[9] ActiveState Software Inc., Download Tcl: ActiveTcl community edition (sep 2014).
URL `http://www.activestate.com/activetcl/downloads`

[10] The FreeType Project, Download FreeType 2.5.3 (sep 2014).
URL `http://sourceforge.net/projects/freetype/files/freetype2/2.5.3/`

[11] C. Geuzaine, J.-F. Remacle, Gmsh - Download (sep 2014).
URL `http://geuz.org/gmsh/#Download`

[12] Microsoft, Windows Software Development Kit (sep 2014).
URL `https://msdn.microsoft.com/en-us/windows/desktop/bg162891.aspx`

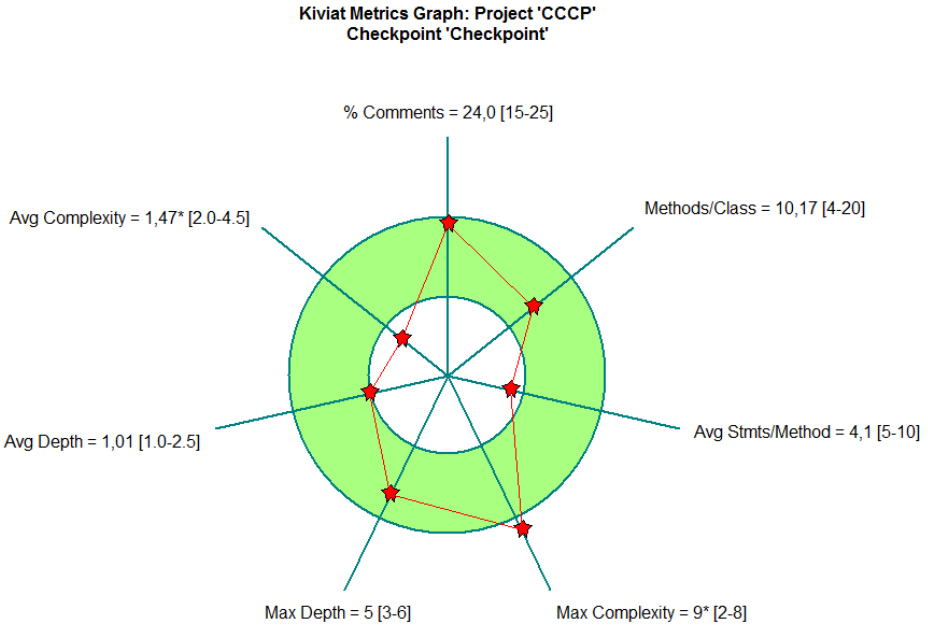
[13] Open Source Initiative, Open Source Licenses (jan 2015).
URL `http://opensource.org/licenses`

[14] T. A. Øritsland, Gestaltprinsipper og Layout av Grafiske Brukergrensesnitt (jan 2015).
URL `http://nettkringkasting.no/~wictor/typografi/Layout_typografi/F7GestaltLayout.pdf`

[15] C. Geuzaine, J.-F. Remacle, Gmsh Manual (nov 2014).
URL `http://geuz.org/gmsh/doc/texinfo/gmsh.html`

Tillegg A

Kodekvalitet



Figur A.1: Kodeanalyse fra programmet SourceMonitor, med måltall i et Kiviatdiagram.

Ut fra analysen kan vi se at programmet i snitt har for lite kompleksitet, dette betyr at det finnes for mange klasser i forhold til antall kall i

programmet. Dette antas å ligge i at programmet fremdeles er i en oppstartfase, og det er bygget en del skall det er tenkt at fremtidig arbeid skal fylle ut og gjøre mer komplekse.

Det kommenteres mye, og det er med vilje. Det er et sterkt ønske om å kunne overføre prosjektet så smidig som mulig fra en oppgave til den neste. Det skal sies at det nok også er noen klasser som er kommentert ut av testårsaker som burde vært ryddet opp i og fjernet.

Videre er det for lite kall per metode også, det følger samme årsak som kompleksiteten. Likevel er scores det utenfor anbefalt område på «Maks kompleksitet», dette er kun ett tilfelle og det er i «VTKControlledSeedWidget», her er det kode som kaller både VTK og CCCP og det er ikke overraskende at denne er kompleks, likevel burde denne vurderes å refaktoreres til flere klasser i fremtiden.

Tillegg B

Filtyper

B.1 .geo

.geo-filen inneholder geometrisk data som sendes til gmsh, i gmshs eget format. gmsh mesher overflater (Plane Surface), en overflate må være definert av én eller to løkker (Line Loop), én dersom den er hel og to dersom den har et hull, her må den ene løkken gå med klokken og den andre mot. En løkke er definert av en ubegrenset mengde linjer (Line), og en linje er definert av to punkter (Point).

Listing B.1: Definisjon av .geo

```
Point(nummer) = {x, y, z, 1};
...
Line(nummer) = {point1, point2};
...
Line Loop(antall Line + nummer) = {line1, line2, line3, ...}
...
Plane Surface(antall Line + nummer) = {lineloop1, lineloop2}
```

OBS: Mens «Point» og «Line» kan ha samme nummer, så fortsetter Line Loops-nummereringen der «Line» slutter. Det er likevel ikke slik at alle linjer må være en del av en «Line Loop», tynnveggede åpne elementer er for eksempel kun beskrevet med «Point» og «Line».

B.2 .dat

.dat-filen ble puttet inn i prosjektet da det var et behov for å beskrive tegningene i større grad enn det .geo-filen hadde støtte for som standard. Deling av tegninger med andre tegneprogrammer som støtter samme format er en viktig funksjonalitet, og det ble derfor besluttet å holde .geo-filen fri for egne utvidelser. .dat-filen er derfor tenkt som en type samlefil for alt av ekstrafunksjonalitet som kun eksisterer i dette prosjektet.

Til nå er det kun bredden på tynnveggede linjestykker som beskrives i denne filen, men intensjonen er at også materialvalg og materialspesifikasjon skal havne her. Formatet er tenkt å være så likt som .geo-filen, dette gjør det lettere å forstå sammenhengen og at de kommer fra samme sted i prosessen. .dat-filen er også helt avhengig av å leses sammen med en .geo- eller en .msh-fil da .dat-filen kun beskriver ekstra informasjon om de allerede definerte geometriene.

Listing B.2: Definisjon av .dat

```
Width(Linjenummer) = {Tykkelse Node1, Tykkelse Node2};
```

«Linjenummer» refererer til linjen definert med det samme nummer i .geo-filen, og tykkelsen i desimaler gjelder tykkelsen ved nodene i rekkefølgen de er oppgitt i definisjonen av linjestykket i .geo-filen. Se følgende eksempel for en bedre forklaring.

Figur B.1: Eksempel 1: Tegningen er grunnlaget for filene B.3 og B.4.

Listing B.3: Eksempel 1 - .geo

```
Point(1) = {0.0, 0.0, 0.0, 1};
Point(2) = {0.0, 0.5, 0.0, 1};
Point(3) = {0.5, 0.5, 0.0, 1};
Line(1) = {1, 2};
Line(2) = {2, 3};
```

Listing B.4: Eksempel 1 - .dat

```
Width(1) = {0.2, 0.4};
Width(2) = {0.1, 0.2};
```

B.3 .msh

Datafilen for meshet er tabseparert og satt sammen av blokker.

Listing B.5: Definisjon av .msh

```
$MeshFormat
[version] [text/binary] [size]
$EndMeshFormat
$Nodes
[number of nodes]
[node number] [x] [y] [z]
...
$EndNodes
$Elements
[number of elements]
[element number] [type] [number of additional fields] <additional fields>
        [node1] [node2] [node3]
...
$EndElements
```

Alle blokker må startes og avsluttes med «$Navn» ... «$EndNavn». OBS: Nummereringen fra gmsh er her 1-indeksert og innlesingen av filen i preprosessoren ignorerer elementer som ikke er av type «1» eller «2», henholdsvis bjelkeelementer og trekanter, alle unntatt siste felt med tilleggsdata ignoreres også, dette er antatt til å være original id fra .geo-filen

B.4 .vtk

Til innlesing av .vtk-filen benytter postprosessoren seg av «vtkDataSetReader», noe som gjør den i stand til å lese alle dataformater støttet av VTK. Dataformatet brukt for overføring av data mellom solveren og postprosessoren er «vtkPolygonData». Dette er et tabseparert tekstformat, som er valgt for sin lesbarhet og enkle feilsøking.

B.5 .msh

Datafilen for meshet er tabseparert og satt sammen av blokker.

Listing B.6: Definisjon av .vtk

```
# 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

Dokumentasjon

Automatisk generert dokumentasjon fra DoxyGen.

CCCP

0.2

Generated by Doxygen 1.8.8

Tue Feb 3 2015 01:59:11

Contents

1 Hierarchical Index

1.1 Class Hierarchy

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

2 Class Index

2.1 Class List

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

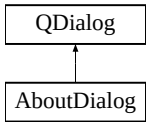
3 Class Documentation

3.1 AboutDialog Class Reference

Controller for the about dialog box used to show the history and information about this program.

`#include <AboutDialog.h>`

Inheritance diagram for AboutDialog:



Public Member Functions

- **AboutDialog** (QDialog *parent=0)

3.1.1 Detailed Description

Controller for the about dialog box used to show the history and information about this program.

For now all information is hard coded into the .ui-file, creating this controller for future expansion

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

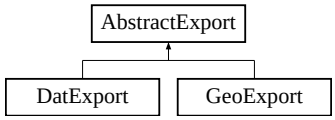
- AboutDialog.h
- AboutDialog.cpp

3.2 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.
- std::vector< Material * > usedMaterials

  All used top level materials.

3.2.1 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.

3.2.2 Member Function Documentation

3.2.2.1 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.

3.2.2.2 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.

3.2.2.3 void AbstractExport::save ( )

Close the opened file stream, releasing the write lock.

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

- AbstractExport.h
- AbstractExport.cpp

3.3 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

3.3.1 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.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 BRepFactory::BRepFactory (Entity * *entity*)

Adding the main entity on which operations are performed.

3.3.3 Member Function Documentation

3.3.3.1 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.

3.3.3.2 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

3.3.3.3 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

3.3.3.4 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:

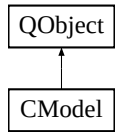
- BRepFactory.h
- BREPFactory.cpp

3.4 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 ()
- void **activeEntityChanged** (Entity *activeEntity)

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.

- void massiveEntitySelected (bool)

  Signal emitted when a massive crossection is selected (clicked).

- void thinwalledEntitySelected (bool)

  Signal emitted when a thin walled crossection is selected (clicked).

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

3.4.1 Detailed Description

The main model used in the application.

3.4.2 Member Function Documentation

3.4.2.1 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.

3.4.2.2 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.

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

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

3.4.2.4 int CModel::getActiveUIMode ( )

Returns

integer representing the current UI mode

3.4.2.5 QString CModel::getMaxElementSize ( )

Returns

QString Maximum element size used in meshing

3.4.2.6 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

3.4.2.7 QString CModel::getMinElementSize ( )

Returns

QString Minimum element size used in meshing

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

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

3.4.2.9 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.

3.4.2.10 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.

3.4.2.11 void CModel::reset ( )

Deletes the data models and creates new ones.

3.4.2.12 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

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

Sets the model's custom arguments used in meshing

3.4.2.14 void CModel::setDataModelListeners ( ) [slot]

Connects the data model signals to this model.

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

Sets the model's maximum element size for meshing

3.4.2.16 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.

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

Sets the model's minimum element size for meshing

3.4.2.18 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.

3.4.2.19 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:

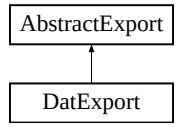
- CModel.h
- CModel.cpp

3.5 DatExport Class Reference

A file writer providing functionaltiy for writing a custom ".dat" format.

`#include <DatExport.h>`

Inheritance diagram for DatExport:



Public Member Functions

- **DatExport** (std::string, DrawModel ∗)
- void **write** ()

Additional Inherited Members

3.5.1 Detailed Description

A file writer providing functionaltiy for writing a custom ".dat" 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:

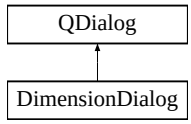
- DatExport.h
- DatExport.cpp

3.6 DimensionDialog Class Reference

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

`#include <DimensionDialog.h>`

Inheritance diagram for DimensionDialog:



Public Slots

- void accept ()
- void reject ()
- void apply ()
- void edgeSelected (QModelIndex)
- void update ()
- void propertyChanged ()

Signals

- void selectedEdgeChanged (Edge ∗)
- void close ()
- void saveEntity (Entity ∗newEntity, Entity ∗∗oldEntity)

Public Member Functions

- **DimensionDialog** (Entity ∗∗, DrawModel ∗, QDialog ∗parent=0)

Public Attributes

- DrawModel ∗ **model**
- EntityGraphicController ∗ gController

  Current active controller for VTK/GraphicsView.
- EntityGraphicModel ∗ gModel

  Model holding the data for the graphic display of edges.
- int **selectedRow**
- bool isPopulating

  When fields are being populated, ignore change events.

3.6.1 Detailed Description

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

It has a table populated with the edges of an entity. A visualisation of the given entity and highilights the selected edge. It can be used to specify material properties for entities.

3.6.2 Member Function Documentation

3.6.2.1 void DimensionDialog::accept ( ) [slot]

Applies the changes and closes the dialog window

3.6.2.2 void DimensionDialog::apply ( ) [slot]

Applies the changes to the active edge.

3.6.2.3 void DimensionDialog::close ( ) [signal]

Signal fired when a this dialog is closing

3.6.2.4 void DimensionDialog::edgeSelected (QModelIndex *index*) [slot]

Edge selected in left list

3.6.2.5 void DimensionDialog::propertyChanged ( ) [slot]

Collects data from all fields and stores them temporarily

3.6.2.6 void DimensionDialog::reject ( ) [slot]

Closes the window

3.6.2.7 void DimensionDialog::saveEntity (Entity * *newEntity,* Entity ** *oldEntity*) [signal]

Signal fired when the user wants to save the Entity

3.6.2.8 void DimensionDialog::selectedEdgeChanged (Edge *) [signal]

Signal fired when a new edge is selected

3.6.2.9 void DimensionDialog::update ( ) [slot]

Updates enabled/disabled interface features, field values and passes updated values to the graphic widget

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

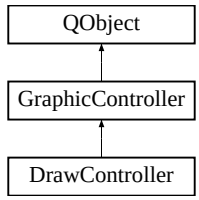
- DimensionDialog.h
- DimensionDialog.cpp

3.7 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**

3.7.1 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

3.7.2 Member Function Documentation

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

Find the active entity based on the actor selected.

3.7.2.2 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.

3.7.2.3 void DrawController::cancelEvent ( ) [slot]

Deletes the active entity

3.7.2.4 void DrawController::connectView ( )

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

3.7.2.5 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)

3.7.2.6 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.

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

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

3.7.2.8 void DrawController::drawSurfaces ( )

Create surfaces for all closed entities in draw model.

3.7.2.9 void DrawController::invalidate ( )

Resets the GraphicsView and draws lines, points and surfaces.

3.7.2.10 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.

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

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

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

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

3.7.2.13 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:

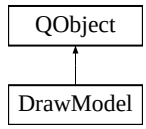
- DrawController.h
- DrawController.cpp

3.8 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 *)
- void showDimensions (Entity *)
- void setDrawType (int)
- int getActiveDrawType ()
- void dialogClosing ()
- void saveEntity (Entity *, 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.
- void drawTypeChanged (int)

  Emitted when the drawing type is changed.
- void dialogClosed ()

  Emitted when any kind of dialog is closed.

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 *)
- int findMaterialId (Material *)
- Node * findCloseNode (double[])
- void mergeEntities (Entity *, Entity *)

Public Attributes

- double snapPrecision = 0.05

  How close should two nodes be to "collide"? Triggers reuse of nodes.
- 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.
- std::list< Node > nodes

  Nodes in model, they can be shared.
- std::vector< Material * > materials

  Materials available in model.

3.8.1 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)

3.8.2 Member Function Documentation

3.8.2.1 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.

3.8.2.2 void DrawModel::createEntityCircle ( )

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

3.8.2.3 void DrawModel::createEntityPolygon ( )

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

3.8.2.4 void DrawModel::deleteActiveEntity ( )

Deletes the active/selected entity.

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

Removes the given entity from the model and deletes it.

3.8.2.6 void DrawModel::dialogClosing ( ) [slot]

Catching a closing dialog window, forwards to controller

3.8.2.7 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.

3.8.2.8 void DrawModel::finalizeActiveEntity ( ) [slot]

Function closes active entity and unselects it.

3.8.2.9 Node * DrawModel::findCloseNode (double *pos[]*)

Finds if there are any Node close enough to the given position, by a set threshold value

Returns

Node that was found or NULL if no Node within range

3.8.2.10 int DrawModel::findMaterialId (Material * *mat*)

Finds the current ID of the provided material

Returns

int id of material

3.8.2.11 int DrawModel::getActiveDrawType ( ) [slot]

Returns

integer representing the active draw type

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

Loops trough all entities creating a vector of all the edges

Returns

vector<Node*> All edges in entities

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

Loops trough all entities creating a vector of all the nodes

Returns

vector<Node*> All nodes in entities

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

Checks wether the given entity is selected.

Returns

bool True if active entity

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

Checks wether the given entity is not selected.

Returns

bool True if NOT active entity

3.8.2.16 void DrawModel::mergeEntities (Entity *, Entity *)

Adds all nodes from the first entity to the second entity and deletes the first entity

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

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

3.8.2.18 void DrawModel::saveEntity (Entity * *newEntity,* Entity ** *originalEntity*) [slot]

Saves changes to the second entity by replacing it with the first one

3.8.2.19 void DrawModel::setActiveEntityNULL ( ) [slot]

Unselects any active/selected entity

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

Sets the active/selected entity to the given entity.

3.8.2.21 void DrawModel::setDrawType (int *type*) [slot]

Sets the drawing type to the given input value

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

Listens for changes in any entity and fire entity changed signal

3.8.2.23 void DrawModel::showDimensions (Entity * *entity*) [slot]

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

3.8.2.24 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.

3.8.2.25 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:

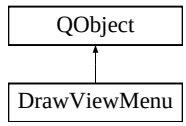
- DrawModel.h
- DrawModel.cpp

3.9 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.
- void dimensionsSelected (Entity *)

  Emitted when dimensions is selected.

Public Member Functions

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

3.9.1 Detailed Description

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

3.9.2 Member Function Documentation

3.9.2.1 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:

- DrawViewMenu.h
- DrawViewMenu.cpp

3.10 Edge Class Reference

A data object holding edge data.

`#include <Edge.h>`

Public Member Functions

- **Edge** (Node ∗, Node ∗)
- **Edge** (Node ∗, Node ∗, double, double)
- **Edge** (const Edge &)
- Edge ∗ **clone** () const
- void **setWidth** (double, double)

Public Attributes

- Node ∗ n1

  Node 1 in edge.
- Node ∗ n2

  Node 2 in edge.
- double width1

  Edge width at Node 1.
- double width2

  Edge width at Node 1.
- Material ∗ **material**

3.10.1 Detailed Description

A data object holding edge data.

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

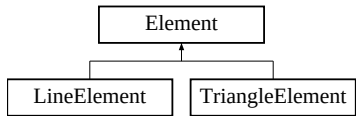
- Edge.h
- Edge.cpp

3.11 Element Class Reference

A data object contains element data and relevant functions.

`#include <Element.h>`

Inheritance diagram for Element:



Public Member Functions

- virtual void calculateArea ()=0
- virtual void calculateAreaCentre ()=0
- virtual double getSecondAreaMomentX (double)=0
- virtual double getSecondAreaMomentY (double)=0
- virtual double getSecondAreaMomentProduct (double, double)=0

Public Attributes

- int **type**
- double area

  Elements total area.
- double material

  Young's modulus for the element.
- double **ecx**
- double ecy

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

  Distance between mesh- and node area centre.
- std::vector< Node * > nodes

  List of all nodes in this element.

Static Public Attributes

- static const int **LINE** = 1
- static const int **TRIANGLE** = 2

3.11.1 Detailed Description

A data object contains element data and relevant functions.

3.11.2 Member Function Documentation

3.11.2.1 virtual void Element::calculateArea ( ) [pure virtual]

Calculates the area of the element.

Implemented in TriangleElement, and LineElement.

3.11.2.2 virtual void Element::calculateAreaCentre ( ) [pure virtual]

Calculates the area centre of the element.

Implemented in TriangleElement, and LineElement.

3.11.2.3 virtual double Element::getSecondAreaMomentProduct (double , double) [pure virtual]

Calculating the product of moment of inertia

Implemented in TriangleElement, and LineElement.

3.11.2.4 virtual double Element::getSecondAreaMomentX (double) [pure virtual]

Calculates the second moment of area for the element.

Parameters

Ay	Area centre of the mesh

Implemented in TriangleElement, and LineElement.

3.11.2.5 virtual double Element::getSecondAreaMomentY (double) [pure virtual]

Calculates the second moment of area for the element.

Parameters

Ax	Area centre of the mesh

Implemented in TriangleElement, and LineElement.

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

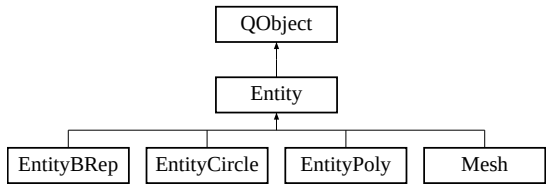
- Element.h
- Element.cpp

3.12 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)
- **Entity** (const Entity &, QObject ∗parent=0)
- virtual Entity ∗ **clone** (QObject ∗parent=0) const =0
- void setName (std::string)
- void setType (int)
- void setCrossectionType (int)

- void setActive (bool)
- virtual void changeNode (double pos[], int)=0
- virtual void createNode (double pos[], int)=0
- virtual void useSnapNode (Node *)=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.
- int **crossectionType**
- std::string name

  Name of entity.
- Material * 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
- static const int **MASSIVE** = 0
- static const int **THINWALLED** = 1

3.12.1 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.

3.12.2 Member Function Documentation

3.12.2.1 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, EntityCircle, Mesh, and EntityPoly.

3.12.2.2 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.

3.12.2.3 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.

3.12.2.4 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.

3.12.2.5 bool Entity::isClosed ( )

Returns true if the entity is closed. False otherwise

3.12.2.6 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.

3.12.2.7 void Entity::setActive (bool *active*)

Is this the active entity (currently editing)

3.12.2.8 void Entity::setCrossectionType (int *crossectionType*)

Sets the crossection type

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

Sets the name of the entity

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

Sets the entitytype

3.12.2.11 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.

3.12.2.12 virtual void Entity::useSnapNode (Node *) [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 EntityCircle, EntityBRep, EntityPoly, and Mesh.

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

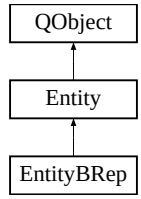
- Entity.h
- Entity.cpp

3.13 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)
- **EntityBRep** (const EntityBRep &e, QObject ∗parent)
- EntityBRep ∗ **clone** (QObject ∗parent) const
- bool safeConvertToEditableEntity ()
- void updatePolygon ()
- vtkSmartPointer< vtkPolyData > getPolyData ()
- void changeNode (double pos[], int)
- void createNode (double pos[], int)

  Implemented and ignored entity functionality.
- void useSnapNode (Node ∗)

  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

3.13.1 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.

3.13.2 Member Function Documentation

3.13.2.1 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.

3.13.2.2 vtkSmartPointer< vtkPolyData > EntityBRep::getPolyData ( )

Creates a polygon for each result entity.

3.13.2.3 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.

3.13.2.4 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

3.13.2.5 void EntityBRep::setRequireUpdate ( ) [slot]

Force the boolean operation to be redone if child entities change

3.13.2.6 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:

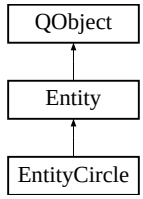
- EntityBRep.h
- EntityBRep.cpp

3.14 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)
- **EntityCircle** (const EntityCircle &e, QObject *parent)
- EntityCircle * **clone** (QObject *parent) const
- void createNode (double[], int)
- void changeNode (double[], int)
- std::vector< Node * > getSeeds ()
- void createCircleBorder ()
- Node * getCentre ()
- Node * getEdge ()
- double getRadius ()
- void useSnapNode (Node *)

Public Attributes

- int segmentNum

  Number of segments used to draw and mesh this circle.

Additional Inherited Members

3.14.1 Detailed Description

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

3.14.2 Member Function Documentation

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

Changes the node and updates the circle edges.

Implements Entity.

3.14.2.2 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;
$stepSize = 2 * \frac{\pi}{numberOfSteps}$

3.14.2.3 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.

3.14.2.4 Node * EntityCircle::getCentre ( )

Returns

the first seed representing the centre of the circle.

3.14.2.5 Node * EntityCircle::getEdge ( )

Returns

the second seed representing the position of the circles edge

3.14.2.6 double EntityCircle::getRadius ( )

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

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

Returns

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

Implements Entity.

3.14.2.8 void EntityCircle::useSnapNode (Node *) [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.

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

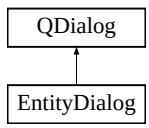
- EntityCircle.h
- EntityCircle.cpp

3.15 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 *)

3.15.1 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.

3.15.2 Member Function Documentation

3.15.2.1 void EntityDialog::accept ( ) [slot]

Applies the changes and closes the dialog window

3.15.2.2 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.

3.15.2.3 void EntityDialog::reject ( ) [slot]

Closes the window

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

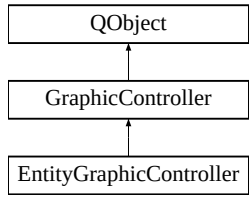
Sets the entity and populates the property window.

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

- EntityDialog.h
- EntityDialog.cpp

3.16 EntityGraphicController Class Reference

Inheritance diagram for EntityGraphicController:



Public Slots

- void **activeEdgeChanged** (Edge *)
- void **invalidate** ()

Public Member Functions

- **EntityGraphicController** (EntityGraphicModel ∗, QWidget ∗, QObject ∗parent=0)

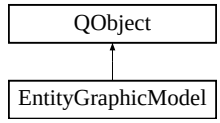
Additional Inherited Members

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

- EntityGraphicController.h
- EntityGraphicController.cpp

3.17 EntityGraphicModel Class Reference

Inheritance diagram for EntityGraphicModel:



Public Slots

- void **setSelectedEdge** (Edge ∗)

Signals

- void **selectedEdgeChanged** (Edge ∗)
- void **valuesUpdated** ()

Public Member Functions

- **EntityGraphicModel** (Entity ∗)
- std::vector< Node ∗ > **getNodes** ()
- std::vector< Edge ∗ > **getEdges** ()
- void updateValues (double, double, Material ∗)

Public Attributes

- Entity ∗ **entity**
- Edge ∗ **selectedEdge**

3.17.1 Member Function Documentation

3.17.1.1 void EntityGraphicModel::updateValues (double *width1*, double *width2*, Material ∗ *mat*)

Updates all values on the currently active edge, then triggers a redraw

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

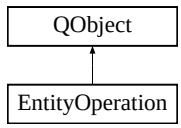
- EntityGraphicModel.h
- EntityGraphicModel.cpp

3.18 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.

3.18.1 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.

3.18.2 Member Function Documentation

3.18.2.1 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

3.18.2.2 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.

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

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

3.18.2.4 void EntityOperation::resetOperation ( )

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

3.18.2.5 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:

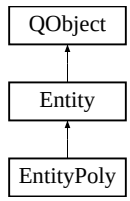
- EntityOperation.h
- EntityOperation.cpp

3.19 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)
- **EntityPoly** (const EntityPoly &e, QObject *parent)
- EntityPoly * **clone** (QObject *parent) const
- void changeNode (double pos[], int)
- void clickNode (int)
- void createNode (double pos[], int)
- void useSnapNode (Node *)
- void createEdge (Node *, Node *)
- std::vector< Node * > getSeeds ()

Additional Inherited Members

3.19.1 Detailed Description

A type of entity representing a polygon shape.

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

3.19.2 Member Function Documentation

3.19.2.1 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.

3.19.2.2 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.

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

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

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

Adds a node to the entity and emits entityChanged.

Implements Entity.

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

Returns

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

Implements Entity.

3.19.2.6 void EntityPoly::useSnapNode (Node * *snapTo*) [virtual]

Adds an existing node to the entity and emits entityChanged

Implements Entity.

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

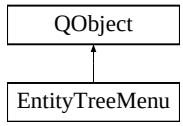
- EntityPoly.h
- EntityPoly.cpp

3.20 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**

3.20.1 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.

3.20.2 Member Function Documentation

3.20.2.1 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:

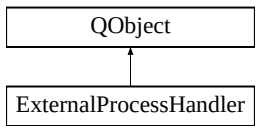
- EntityTreeMenu.h
- EntityTreeMenu.cpp

3.21 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 ()

3.21.1 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..

3.21.2 Member Function Documentation

3.21.2.1 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.

3.21.2.2 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:

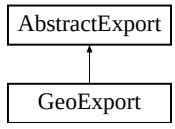
- ExternalProcessHandler.h
- ExternalProcessHandler.cpp

3.22 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

3.22.1 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:

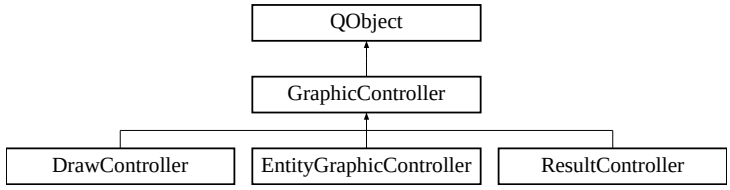
- GeoExport.h
- GeoExport.cpp

3.23 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**

3.23.1 Detailed Description

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

3.23.2 Constructor & Destructor Documentation

3.23.2.1 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:

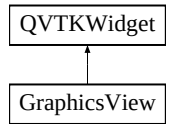
- GraphicController.h

3.24 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.

3.24.1 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.

3.24.2 Member Function Documentation

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

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

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

Adds the given renderer to the renderWindow and updates the widget

3.24.2.3 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.

3.24.2.4 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.

3.24.2.5 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)

3.24.2.6 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.

3.24.2.7 void GraphicsView::enterPressed ( ) [slot]

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

3.24.2.8 void GraphicsView::escapePressed ( ) [slot]

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

3.24.2.9 void GraphicsView::invalidate ( ) [slot]

Updates the widget

3.24.2.10 void GraphicsView::removeActors ( )

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

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

Removes the given renderer from the renderWindow and updates the widget

3.24.2.12 void GraphicsView::reset ( ) [slot]

Removes all actors and updates the widget

3.24.2.13 void GraphicsView::resetInteractor ( )

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

3.24.2.14 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.

3.24.2.15 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.

3.24.2.16 void GraphicsView::set3DMode ( )

Sets the renderers active camera to a regular camera.

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

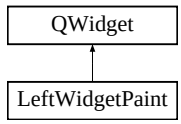
- GraphicsView.h
- GraphicsView.cpp

3.25 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 ∗)

3.25.1 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

3.25.2 Member Function Documentation

3.25.2.1 void LeftWidgetPaint::setModel (DrawModel ∗ *model*)

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

3.25.2.2 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:

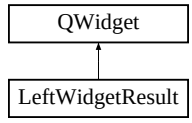
- LeftWidgetPaint.h
- LeftWidgetPaint.cpp

3.26 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)

3.26.1 Detailed Description

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

It lets the user change the active result.

3.26.2 Member Function Documentation

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

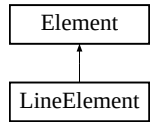
Sets the model active result name to the name of the list item.

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

- LeftWidgetResult.h
- LeftWidgetResult.cpp

3.27 LineElement Class Reference

Inheritance diagram for LineElement:



Public Member Functions

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

Public Attributes

- double **x1**
- double x2

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

  Elements nodepositions in y-directions.
- double **xl1**
- double **xl2**
- double **yl1**
- double **yl2**

Additional Inherited Members

3.27.1 Member Function Documentation

3.27.1.1 void LineElement::calculateArea ( ) [virtual]

Calculates the area of the element.
$A_e = \frac{1}{2} \cdot |J|$

Implements Element.

3.27.1.2 void LineElement::calculateAreaCentre ( ) [virtual]

Calculates the area centre of the element.
$ec_x = \frac{x1+x2+x3}{3}$
$ec_y = \frac{y1+y2+y3}{3}$

Implements Element.

3.27.1.3 double LineElement::getSecondAreaMomentProduct (double *Ax*, double *Ay*) [virtual]

Calculating the product of moment of inertia

$dx_e = Ac_x - Ac_{ex}$
$dy_e = Ac_y - Ac_{ey}$
$I_{xy} = \sum (I_{xy_e} + dx_e * dy_e * A_e)$

Implements Element.

3.27.1.4 double LineElement::getSecondAreaMomentX (double *Ay*) [virtual]

Calculates the second moment of area for the element.

Parameters

Ay	Area centre of the mesh

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

Implements Element.

3.27.1.5 double LineElement::getSecondAreaMomentY (double *Ax*) [virtual]

Calculates the second moment of area for the element.

Parameters

Ax	Area centre of the mesh

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

Implements Element.

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

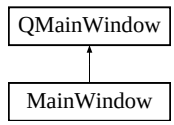
- LineElement.h
- LineElement.cpp

3.28 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)
- void setDrawTypeThin (bool)
- void setDrawTypeMassive (bool)
- void massiveEntitySelectedSlot (bool)
- void thinwalledEntitySelectedSlot (bool)
- void showAbout ()

Signals

- void meshVisible (bool)
- void resultVisible (bool)
- void drawTypeThin ()
- void drawTypeMassive ()
- void massiveEntitySelectedSig (bool)
- void thinwalledEntitySelectedSig (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.

3.28.1 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.

3.28.2 Member Function Documentation

3.28.2.1 void MainWindow::addCircle ( ) [slot]

Calls createEntityCircle in drawModel given a UI signal

3.28.2.2 void MainWindow::addPolygon ( ) [slot]

Calls createEntity in drawModel given a UI signal

3.28.2.3 void MainWindow::deleteEntity ( ) [slot]

Calls deleteActiveEntity in drawModel given a UI signal

3.28.2.4 void MainWindow::drawTypeMassive ( ) [signal]

Signal emitted when setDrawTypeFilled is called by the UI.

3.28.2.5 void MainWindow::drawTypeThin ( ) [signal]

Signal emitted when setDrawTypeThin is called by the UI.

3.28.2.6 void MainWindow::massiveEntitySelectedSig (bool) [signal]

Signal emitted when a massive crossection is selected (clicked). Used to set the correct value in the drawing type selector

3.28.2.7 void MainWindow::massiveEntitySelectedSlot (bool *selected*) [slot]

Signal emitted from "CModel" when a massive crossection is selected (clicked). Used to set the correct value in the drawing type selector

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

Signal emitted when setMeshVisible is called by the UI.

3.28.2.9 void MainWindow::newFile ( ) [slot]

Resets the models and redraws the active view

3.28.2.10 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)

3.28.2.11 void MainWindow::resetMeshOptions ( ) [slot]

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

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

Signal emitted when setResultVisible is called by the UI.

3.28.2.13 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.

3.28.2.14 void MainWindow::setCustomArguments ( ) [slot]

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

3.28.2.15 void MainWindow::setDrawMode ( ) [slot]

Sets the model to drawMode given an UI signal

3.28.2.16 void MainWindow::setDrawTypeMassive (bool *enable*) [slot]

Signal from "Draw Tab" Sets the drawing type to filled crossection mode

3.28.2.17 void MainWindow::setDrawTypeThin (bool *enable*) [slot]

Signal from "Draw Tab" Sets the drawing type to thin walled mode

3.28.2.18 void MainWindow::setMaxElementSize ( ) [slot]

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

3.28.2.19 void MainWindow::setMeshMode ( ) [slot]

Sets the model to meshMode given an UI signal

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

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

3.28.2.21 void MainWindow::setMinElementSize ( ) [slot]

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

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

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

3.28.2.23 void MainWindow::setResultMode ( ) [slot]

Sets the model to resultMode given an UI signal

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

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

3.28.2.25 void MainWindow::showAbout ( ) [slot]

Signal from mainwindow.ui, show the about window.

3.28.2.26 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)

3.28.2.27 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)

3.28.2.28 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)

3.28.2.29 void MainWindow::thinwalledEntitySelectedSig (bool) [signal]

Signal emitted when a thin walled crossection is selected (clicked). Used to set the correct value in the drawing type selector

3.28.2.30 void MainWindow::thinwalledEntitySelectedSlot (bool *selected*) [slot]

Signal emitted from "CModel" when a thin walled crossection is selected (clicked). Used to set the correct value in the drawing type selector

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

- mainwindow.h
- mainwindow.cpp

3.29 Material Class Reference

Public Member Functions

- **Material** (std::string)
- void updateTables ()

Public Attributes

- bool **isSelectable**
- bool **isBase**
- QString **name**
- int **baseColorRGB** [3]
- std::vector< Material * > **components**
- vtkSmartPointer< vtkFloatArray > cellData
- vtkSmartPointer< vtkLookupTable > colorTable

Friends

- bool **operator==** (Material m1, Material m2)

3.29.1 Member Function Documentation

3.29.1.1 void Material::updateTables ( )

Generate new values and populate cellData and ColorTable

3.29.2 Member Data Documentation

3.29.2.1 vtkSmartPointer<vtkFloatArray> Material::cellData

Collection of cells used to color a crossection made by this material

3.29.2.2 vtkSmartPointer<vtkLookupTable> Material::colorTable

Collection of colors used to color a crossection using this material

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

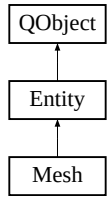
- Material.h
- Material.cpp

3.30 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)
- **Mesh** (const Mesh &e, QObject *parent)
- Entity * **clone** (QObject *parent)
- void changeNode (double pos[], int handle)

  Implemented and ignored entity functionality.
- void createNode (double pos[], int handle)

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

  Implemented and ignored entity functionality.
- std::vector< Element * > **getElements** ()
- QString getFileName ()
- int **getNumberOfNodes** ()
- int **getNumberOfElements** ()
- void updatePolygon ()
- vtkSmartPointer< vtkPolyData > getPolyData ()

Additional Inherited Members

3.30.1 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.

3.30.2 Constructor & Destructor Documentation

3.30.2.1 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.

3.30.3 Member Function Documentation

3.30.3.1 QString Mesh::getFileName ( )

Returns

QString File name of the mesh loaded.

3.30.3.2 vtkSmartPointer< vtkPolyData > Mesh::getPolyData ( )

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

3.30.3.3 void Mesh::updatePolygon ( ) [virtual]

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

Reimplemented from Entity.

3.30.3.4 void Mesh::useSnapNode (Node *) [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.

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

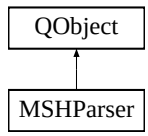
- Mesh.h
- Mesh.cpp

3.31 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)

3.31.1 Detailed Description

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

3.31.2 Constructor & Destructor Documentation

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

Parameters

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

3.31.3 Member Function Documentation

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

Returns

Element at the given position

3.31.3.2 std::vector< Element * > MSHParser::getElements ( )

Returns

vector<Element*> containing all valid elements parsed from file

3.31.3.3 std::string MSHParser::getFilename ( )

Returns

name of parsed file

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

Returns

Node at the given position

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

Returns

vector<Node> containing all nodes parsed from file

3.31.3.6 int MSHParser::getNumberOfElements ( )

Returns

number of elements

3.31.3.7 int MSHParser::getNumberOfNodes ( )

Returns

number of nodes

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

- MSHParser.h
- MSHParser.cpp

3.32 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.

3.32.1 Detailed Description

A data object storing node data.

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

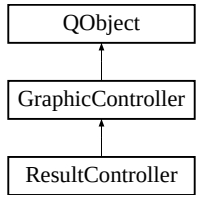
- Node.h

3.33 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

3.33.1 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.

3.33.2 Member Function Documentation

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

Not implemented functionality in this view

Implements GraphicController.

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

Not implemented functionality in this view

Implements GraphicController.

3.33.2.3 void ResultController::createMeshActor ( )

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

3.33.2.4 void ResultController::createResultActor ( )

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

3.33.2.5 void ResultController::reset ( ) [slot]

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

3.33.2.6 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:

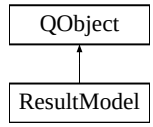
- ResultController.h
- ResultController.cpp

3.34 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.

3.34.1 Detailed Description

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

3.34.2 Member Function Documentation

3.34.2.1 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

3.34.2.2 std::string ResultModel::getActiveResultName ( )

Returns

std::string Name of the active result

3.34.2.3 vtkSmartPointer< vtkDataSetMapper > ResultModel::getMapper ( )

Returns

vtkSmartPointer<vtkDataSetMapper> Contains result data

3.34.2.4 Mesh * ResultModel::getMesh ( )

Gets the active mesh entity in the model.

Returns

Mesh*

3.34.2.5 bool ResultModel::hasMeshData ( )

Checks if the mesh entity is initialized

Returns

True if model has mesh data. False otherwise.

3.34.2.6 bool ResultModel::hasResultData ( )

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

Returns

True if model has vtkData. False otherwise.

3.34.2.7 void ResultModel::modelChanged ( ) [signal]

Fires if activeResult changes.

3.34.2.8 void ResultModel::modelDataSourceChanged ( ) [signal]

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

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

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

3.34.2.10 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.

3.34.2.11 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.

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

Sets the mesh visibility in the model and fires modelChanged

3.34.2.13 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:

- ResultModel.h
- ResultModel.cpp

3.35 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 *)

3.35.1 Detailed Description

Container for all keyboard shortcuts supported by the application.

3.35.2 Member Function Documentation

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

Sets shortcuts that connects to the main window.

3.35.2.2 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:

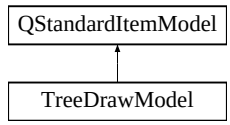
- ShortcutManager.h
- ShortcutManager.cpp

3.36 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**

3.36.1 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:

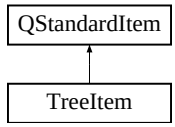
- TreeDrawModel.h
- TreeDrawModel.cpp

3.37 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

3.37.1 Detailed Description

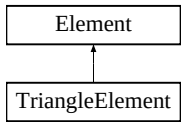
Holder for an entity shown in the TreeWidget. It holds a reference to the entity for fast selection.

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

- TreeItem.h
- TreeItem.cpp

3.38 TriangleElement Class Reference

Inheritance diagram for TriangleElement:



Public Member Functions

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

Public Attributes

- double Iex

  Elements second area moment x.
- double Iey

  Elements second area moment y.
- double Iexy

  Elements second area moment product.
- double **x1**
- double **x2**

- double x3

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

  Elements nodepositions in y-directions.
- double **xl1**
- double **xl2**
- double **xl3**
- double **yl1**
- double **yl2**
- double **yl3**

Additional Inherited Members

3.38.1 Member Function Documentation

3.38.1.1 void TriangleElement::calculateArea ( ) [virtual]

Calculates the area of the element.
$A_e = \frac{1}{2} \cdot |J|$

Implements Element.

3.38.1.2 void TriangleElement::calculateAreaCentre ( ) [virtual]

Calculates the area centre of the element.
$ec_x = \frac{x1+x2+x3}{3}$
$ec_y = \frac{y1+y2+y3}{3}$

Implements Element.

3.38.1.3 double TriangleElement::getSecondAreaMomentProduct (double *Ax*, double *Ay*) [virtual]

Calculating the product of moment of inertia

$dx_e = Ac_x - Ac_{ex}$
$dy_e = Ac_y - Ac_{ey}$
$I_{xy} = \sum (I_{xy_e} + dx_e * dy_e * A_e)$

Implements Element.

3.38.1.4 double TriangleElement::getSecondAreaMomentX (double *Ay*) [virtual]

Calculates the second moment of area for the element.

Parameters

Ay	Area centre of the mesh

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

Implements Element.

3.38.1.5 double TriangleElement::getSecondAreaMomentY (double *Ax*) [virtual]

Calculates the second moment of area for the element.

Parameters

Ax	Area centre of the mesh

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

Implements Element.

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

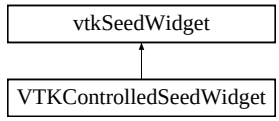
- TriangleElement.h
- TriangleElement.cpp

3.39 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 *)

3.39.1 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.

3.39.2 Member Function Documentation

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

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

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

Overrides the vtkSeedWidget's delete function.

3.39.2.3 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:

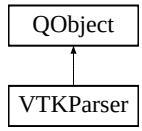
- VTKControlledSeedWidget.h
- VTKControlledSeedWidget.cpp

3.40 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**

3.40.1 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:

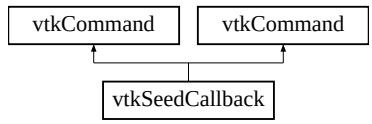
- VTKParser.h
- VTKParser.cpp

3.41 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 **setModel** (DrawModel ∗model)
- 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** ()

3.41.1 Detailed Description

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

3.41.2 Member Function Documentation

3.41.2.1 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.

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

Sets the entity the callback is doing work on.

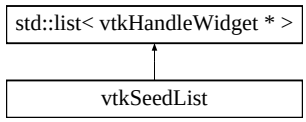
The documentation for this class was generated from the following files:

- VTKControlledSeedWidget.h
- VTKDrawCallback.h

3.42 vtkSeedList Class Reference

Data structure used for seeds in VTK.

Inheritance diagram for vtkSeedList:



3.42.1 Detailed Description

Data structure used for seeds in VTK.

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

- VTKControlledSeedWidget.cpp