Applikasjon for utmattingsberegninger av ikke-proposjonale spenningshistorier

Amund Aasen

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

Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

1 Abstract

Fatigue calculation of non-proportional stress histories is currently an ongoing research field at the Department of Engineering Design and Materials at NTNU. These calculations tend to be quite extensive with large data sets and are therefore most efficiently performed by computers. A software application to execute the computations is therefore sought after as fatigue calculation of this type has not been available for the public, but only for those with quite expensive software or for those with fundamental programming skills. There is however made several Matlab scripts that is used for inspiration regarding implementation and for validation of the results from this thesis.

The application created during the thesis is based on existing free to use open-source libraries and frameworks. A stress based critical plane approach was implemented to perform the fatigue calculations. For educational purposes a visualization of the results is presented in a 3D widget built from the Qt-framework and the Visualization Toolkit. The application is designed to input comma separated value files containing stress histories, perform the fatigue calculation and visualize the critical plane and stress histories.

The results are compared to a Matlab script from a thesis written by a fellow student and the same element is found to be critical. The critical plane is also found to be the same, and the application calculates at a more rapid pace than the Matlab script.

2 Sammendrag

Utmattelsesberegning av ikke-proporsjonal spenningshistorier er for tiden en pgende forskningsfelt ved Institutt for produktutvikling og materialer ved NTNU. Disse beregningene har en tendens til vre svrt omfattende med store datasett og er derfor mest effektivt utfrt av datamaskiner. Et program for utfre beregningene er derfor ettertraktet ettersom beregninger av denne typen har ikke vrt tilgjengelig for allmennheten, kun for de med relativt dyr programvare eller for de med grunnleggende kunnskaper om programmering. Det er imidlertid gjort flere Matlab-skript som er brukt til inspirasjon i gjennomfring av denne oppgaven og for validering av resultatene fra denne avhandlingen.

Applikasjonen utarbedet i lpet av avhandlingen er basert p eksisterende open-source-biblioteker og rammeverk. En stress basert kritiske plan metode er implementert for utfre utmattelsesberegningene. For pedagogiske forml er en visualisering av resultatene presentert i et 3D-widget bygget fra Qt-rammeverket og The Visualization Toolkit. Applikasjonen er utformet for ta inn komma separert verdi filer som inneholder spenningshistorier, utfre utmattelsesberegning og visualisere dte kritiske planet og stresshistorien

Resultatene er sammenlignet med et Matlab skript fra en avhandling skrevet av en annen student og samme element er funnet til vre kritisk. Det kritiske planet er ogs funnet vre den samme, og programmet utfrer beregningene i et raskere tempo enn matlab skriptet.

3 Acknowledgements

I would like to thank my supervisor, associate professor Bjrn Haugen, for support and advice throughout the process of writing this theis. He has been understanding and given great help in times when it was needed. I would also like to thank Simen Riiser for a great Master thesis I found to be very useful; especially the Matlab code and fatigue theory has been of big help. Thanks also to my family for keeping my spirit up during this challenging project and all my fellow classmates for a inspiring study which ended with this thesis.

NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR PRODUKTUTVIKLING
OG MATERIALER

MASTEROPPGAVE VÅR 2014
FOR
STUD.TECHN. AMUND AASEN

APPLIKASJON FOR UTMATTINGSBEREGNINGER AV IKKE-PROPOSJONALE SPENNINGSHISTORIER
Software application for computing fatigue from non-proportional stress time histories

Utmattingsberegninger basert på tidshistorier av ikke-proporsjonale spenningstilstander er et pågående forskningsfelt ved IPM. Eksisterende beregninger har i stor grad blitt utført ved bruk av «Kritiske plan» metoder implementert i Matlab.

En ønsker å utvikle en applikasjon som tilgjengelig gjør utmattingsalgoritmene både for studenter og utøvende ingeniører. Applikasjonen bør derfor være enkel å bruke, samtidig som at den kan håndtere store datamengder på en effektiv måte.

Applikasjonen bør blant annet kunne visualisere spenningstensoren, retninger på hovedspenninger, retning på det kritiske planet. Applikasjonen bør bygge på underliggende småprogram som også kan brukes i en skript-basert prosessering av tidshistorier for mange målepunkt.

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.

[illegible]
Torgeir Welo
Instituttleder

Bjørn Haugen
Bjørn Haugen
Faglærer

NTNU
Norges teknisk-
naturvitenskapelige universitet
Institutt for produktutvikling
og materialer

Contents

1 Abstract **1**

2 Sammendrag **2**

3 Acknowledgements **3**

List of Figures **8**

List of Tables **9**

4 Introduction **10**
- 4.1 Background . . . 10
- 4.2 Objective . . . 10
- 4.3 Scope . . . 10

5 Fatigue Theory **11**
- 5.1 The stress tensor . . . 11
- 5.2 Three dimensional transformation of the stress tensor . . . 13
- 5.3 Fatigue mechanisms . . . 16
- 5.4 The fatigue life . . . 17
- 5.5 Design against fatigue . . . 18
- 5.6 Proportional and non-proportional stress . . . 19
- 5.7 Multiaxial fatigue . . . 21
- 5.8 Critical Plane approach . . . 21
- 5.9 The Findley Model . . . 22
- 5.10 The principal stress . . . 23

6 Visualization **26**
- 6.1 Qt . . . 26
- 6.2 The Visualization Toolkit . . . 27
- 6.3 Integrating Qt and VTK . . . 28

7 System Description **30**
- 7.1 Software requirements description . . . 30
- 7.2 System Architecture . . . 32
- 7.3 Class Diagram . . . 34
- 7.4 Design layout . . . 36

8 Results **39**
8.1 The Application software 39
8.2 The Critical Plane approach calculation 40

9 Discussion **42**
9.1 Fatigue calculation 42
9.2 The Application 42
9.3 Accomplished requirements 43

10 Conclusion **45**

Appendices **46**

A Source Code **46**

B References **73**

List of Figures

5.1 The stress tensor 11
5.2 Plane stress transformation 13
5.3 Three dimensional stress transformation 14
5.4 Fatigue mechanism steps 17
5.5 S-N curve 19
5.6 Mohr's circle in non-proportional stress 20
5.7 CPA plane 22
5.8 Mohr's circle - 3D stress 24
6.1 Qt c++ library modules 27
6.2 VTK architecture 28
7.1 Application architecture 33
7.2 Application class diagram 34
7.3 Sphere with polar coordinates 37
8.1 Final Application 39

List of Tables

5.1 Transformation matrix constants 16
7.1 Software requirements specification 31
7.2 main.cpp functions and classes 34
7.3 CPA_Findley.cpp functions and classes 35
7.4 calculation.cpp functions and classes 36
8.1 CPA calculations 40
9.1 Accomplished requirements 44

4 Introduction

4.1 Background

Fatigue calculation of non-proportional stress histories is an ongoing research area at the Department for Engineering Design and Materials at NTNU. When working with this topic contrary to traditional fatigue analysis the approach is different as the non-proportional multiaxial loadings make traditional fatigue approaches invalid. Also, because of large data sets and relatively advanced calculations a computers computation power become a necessity.

Although methods and formulas already exist for handling the problem there is no convenient way to perform them without a Matlab script or other open-source or free software. For engineers who do not have fundamental programming skills or do not have access to Matlab, performing the necessary calculation is therefore difficult or at best extremely time consuming when done by hand.

4.2 Objective

The objective of the thesis is to create an application to perform fatigue calculations of non-proportional stress histories. The application should be available for public use, in industry and should be usable as a tool in education.

4.3 Scope

A critical plane approach method will be implemented as part of a software application to perform a fatigue analysis of potentially large stress histories. The application will then visualise different aspects of the result such as the stress tensor, the principal stress and the critical plane. The calculations will be compared to previous analytical results from the same data set and manual calculations for validation.

5 Fatigue Theory

A material is weakened when it is subjected to repeatedly applied loads. The damage which occur because of these loads will eventually change the quality of the material and in some cases the damage will lead to failure. Fatigue theory explain the process of material and structural damage caused by the applied loads.

5.1 The stress tensor

Any structural component of any sort and material can be exposed to a complicated stress and tension environment. Stresses have directions and when multiple stresses are working on one component it can sometimes be difficult to describe the nature of the component, more significantly how it will respond to the forces of its environment. By looking at an element, Figure 1), of a component it is by definition exposed to nine stresses, more specifically three normal stresses and six shear stresses.

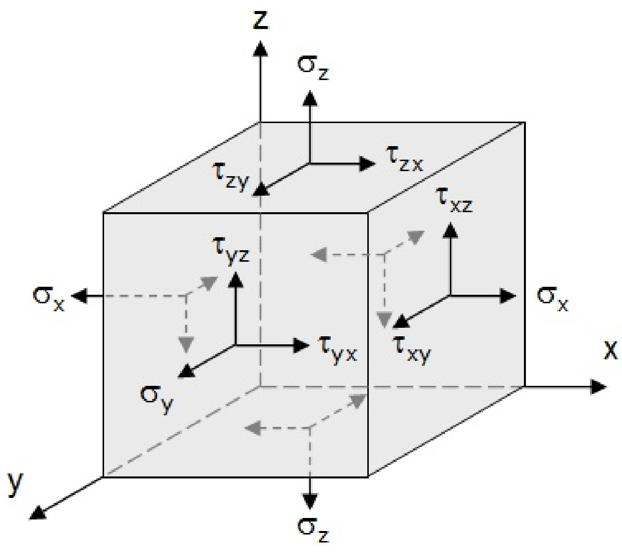


Figure 5.1: Showing the nine stresses working on an element.

The nine stresses are often described in a three by three matrix, known as the stress tensor, Equation 1 [2, p.55].

$$\sigma_{ij} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \quad (1)$$

In the equation above σ_{ij} represents the stress tensor, σ_x, σ_y and σ_z the normal stresses and the remaining variables represents the shear stresses. The symbols i and j specify which plane they work along the directions of the stresses. For example is τ_{xy} a shear stress working along the x-axis and in the y direction.

For an element to stay in one place at an static equilibrium, the shear stresses working in opposite directions must have the same value, otherwise the element would rotate. This simplifies the picture as we now only have three shear stresses to use for calculations. When by definition $\tau_{xy} = \tau_{yx}$, $\tau_{xz} = \tau_{zx}$ and $\tau_{yz} = \tau_{zy}$, the stress tensor is reduced to a matrix as shown below.

$$\sigma_i j = \begin{bmatrix} \sigma_x & \tau_x y & \tau_x z \\ - & \sigma_y & \tau_y z \\ - & - & \sigma_z \end{bmatrix} \quad (2)$$

The directions these stresses work in are defined by the given coordinate system, but this is not always the directions that are of interest. If one for example would want to calculate the largest stress working on a element it is not likely that it would have the same direction and value as is already defined by the coordinate system. It is therefore sometimes necessary to transform the stresses so that they point in other directions.

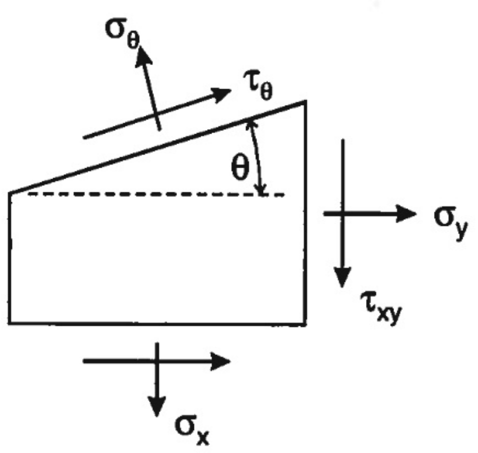


Figure 5.2: Plane stress. The stress is transformed with the angle θ.

In plane stress, when the stresses on one plane by definition is zero, it is straight forward to see how the directions of the stresses can transform using an angle θ, resulting in a new normal stress, σ_θ, and a new shear stress, τ_θ. For the element to reach equilibrium, the new stresses must offset the three stresses from before the transformation, σ_x, σ_y and τ_{xy}. We can now describe the new stresses with the given equations [8, p. 3]:

$$\sigma_\theta = \frac{\sigma_x + \sigma_y}{2} + \frac{\sigma_x + \sigma_y}{2} \times cos(2\theta) + \tau_{xy} \times sin(2\theta) \tag{3}$$

$$\tau_\theta = \frac{\sigma_x - \sigma_y}{2} \times sin(2\theta) - \tau_{xy} \times cos(2\theta) \tag{4}$$

5.2 Three dimensional transformation of the stress tensor

Plane stress, as described in the previous chapter, is a two dimensional state as the stresses on one plane are zero. The calculations needed to transform these stresses are also quite trivial as shown in equation 3 and 4 with the angle, θ. The three dimensional case occur when there are stresses working on all three planes formed by the coordination system. To describe these transformation it must be taken in to consideration that there now are six stresses to be offset from the original coordination system, and that these are working on three planes. Therefore one more angle, ϕ, is needed to describe the new stresses. In figure 3, the transformed coordinate system is described after two rotations with axes x', y' and z'. [8, p. 4-5]

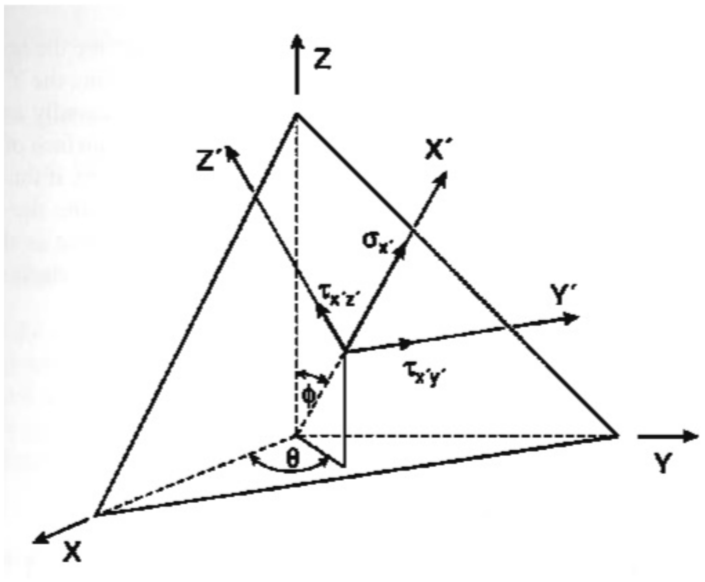


Figure 5.3: Three dimensional stress transformation with the angles θ and *phi*.

Mathematically a transformation matrix, T, can be used to decide the new values for the resulting transformed stress tensor. By solving the equation [3, p. 190]

$$\sigma'_{ij} = T \cdot \sigma_{ij} \cdot T^t \tag{5}$$

the new transformed stress tensor is calculated. T is defined from the coordination system around the axes.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = T \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6}$$

As three dimensional transformation includes rotation about two axes there must exist two transformation matrices. The product of these two matrices give the total transformation matrix. If seen as two individual rotations, one can say that the first rotation with angle *theta* happen around the z-axis and that the second rotation with angle *phi* is around the y-axis. In each case the axis of rotation remains and so does the value along this axis, meaning that in the first case $z_1 = z$ and in the second case $y_1 = y$. This now gives us the two following transformation matrices

$$T_1 = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$T_2 = \begin{bmatrix} sin\phi & 0 & cos\phi \\ 0 & 1 & 0 \\ -cos\phi & 0 & sin\phi \end{bmatrix} \tag{8}$$

The total transformation matrix is a product of the two transformation matrices.

$$T = \begin{bmatrix} cos\theta \cdot sin\phi & sin\theta \cdot sin\phi & cos\phi \\ -sin\theta & cos\theta & 0 \\ -cos\phi \cdot cos\theta & -sin\theta \cdot cos\phi & sin\phi \end{bmatrix} \tag{9}$$

To make use of equation 5 without the transposed transformation matrix the right hand side of equation 4 can be calculated and one end up with one transformation matrix. Further this can be simplified by eliminating τ_{xy} and τ_{yz}, stresses which correspond to other stresses.

The result of these operations become a rather large matrix expression

$$\begin{bmatrix} \sigma'_x \\ \sigma'_y \\ \sigma'_x \\ \tau'_{xy} \\ \tau'_{xz} \\ \tau'_{yz} \end{bmatrix} = \begin{bmatrix} a_{11}^2 & a_{12}^2 & a_{13}^2 & 2a_{11}a_{12} & 2a_{11}a_{13} & 2a_{13}a_{12} \\ a_{21}^2 & a_{22}^2 & a_{23}^2 & 2a_{21}a_{22} & 2a_{21}a_{23} & 2a_{23}a_{22} \\ a_{1}^2 & a_{32}^2 & a_{33}^2 & 2a_{31}a_{32} & 2a_{31}a_{33} & 2a_{33}a_{32} \\ a_{11}a_{21} & a_{12}a_{22} & a_{13}a_{23} & (a_{11}a_{22} + a_{12}a_{21}) & (a_{13}a_{21} + a_{11}a_{23}) & (a_{12}a_{23} + a_{13}a_{22}) \\ a_{11}a_{31} & a_{12}a_{32} & a_{13}a_{33} & (a_{11}a_{32} + a_{12}a_{31}) & (a_{13}a_{31} + a_{11}a_{33}) & (a_{13}a_{32} + a_{12}a_{33}) \\ a_{21}a_{31} & a_{22}a_{32} & a_{23}a_{33} & (a_{21}a_{32} + a_{22}a_{31}) & (a_{23}a_{31} + a_{21}a_{33}) & (a_{22}a_{33} + a_{23}a_{32}) \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_x \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{bmatrix} \tag{10}$$

where the a values are defined as Table 5.1 shows below.

$$
\begin{aligned}
a_{11} &= cos\theta \cdot sin\phi \\
a_{12} &= sin\theta \cdot sin\phi \\
a_{13} &= cos\theta \\
a_{21} &= -sin\theta \\
a_{22} &= cos\theta \\
a_{23} &= 0 \\
a_{31} &= -cos\theta \cdot cos\phi \\
a_{32} &= -sin\theta \cdot cos\phi \\
a_{33} &= sin\phi
\end{aligned}
$$

Table 5.1: The a values for the transformation matrix

Now, even though the expressions are quite long, transforming a stress tensor in three dimensions become straight forward. All that is needed are the values of the original stress tensor and the angles, ϕ and θ.

5.3 Fatigue mechanisms

As mentioned in the introduction to this section, structural failure often come from a change in the quality of the material. This however is not always a rapid process, but can gradually take place over a longer time period.

Failures due to cracking appear in four steps [7, p.58]: 1) Crack nucleation 2) Stage I crack-growth 3) Stage II crack-growth 4) Ultimate ductile failure. The nucleation step is where the crack initiate. In metals, strain hardening and oxidation can occur on new surfaces caused by loading slips on specifically oriented slip planes and the metal can become brittle. As slip appear and as the load cycles continue the intrusion become a stage I crack.

Stage I cracks propagate over a few grains on the plane of maximum shear and globally in the plane normal to the principle tensile stress. When the cracks become large enough and reach a critical state, the structure will fracture as it reaches the Ultimate failure step [11, p.58].

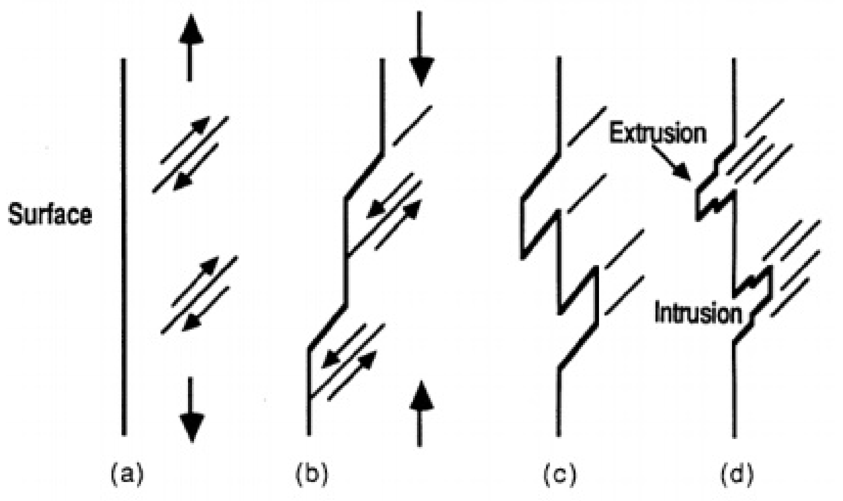


Figure 5.4: The figure shows the different stages regarding cracking. (a) Crack nucleation (b) Stage I crack-growth (c) Stage II crack-growth (d) Ultimate ductile failure.

5.4 The fatigue life

It is normal to divide fatigue into high cycle and low cycle fatigue. The classification is relevant as it defines some differences between how eventually failure, caused by fatigue, happens over a time and cycle span. Low cycle fatigue is often known as less than 10^3 cycles [5, p. 392], and high cycle is thereby known as more.

In other words, when mainly elastic strain and some plastic strain occur microscopically over many cycles the fatigue is high-cycle. If the nominal strain is of a high plastic manner, the fatigue is low-cycle. High plastic strain, meaning that the strain is much higher than the material's yield strain, obviously has a higher impact on the material and therefore the number of cycles is less than for the high-cycle fatigue.

Fatigue life, N_t describes how many cycles the structure can take before it reaches failure. As N_t becomes large enough the fatigue is described as high cycle fatigue. To calculate the expected lifetime of a structure, N_t is a good benchmark. When looking at the equations of elastic and plastic strain amplitudes, ϵ_{ae} and ϵ_{ap} we can calculate the transition fatigue life, and embrace the fact that both high-cycle and low-cycle strain contribute to ending the final ultimate failure.

$$\epsilon_{ae} = \frac{\sigma'_f}{E}(2N_f)^b \tag{11}$$

$$\epsilon_{ap} = \frac{\sigma'_f}{E}(2N_f)^e \tag{12}$$

When $\epsilon_{ae} = \epsilon_{ap}$, N_t is given by

$$N_t = \frac{1}{2}(\frac{\sigma'_f}{\epsilon'_f E})^{\overline{(c-b)}^{\,1}} \tag{13}$$

σ'_f, the fatigue strength of the material
ϵ'_f, the fatigue ductility coefficient
b, the fatigue strength exponent
c, the fatigue ductility strength
E, the modulus of elasticity

5.5 Design against fatigue

The effort many have made in research on fatigue and failure is with the purpose of stopping it from happen or to control it as well as possible. Popular tools that are used to design against fatigue are strain-based and stress-based approaches and the fracture mechanics approach.

The strain-based approach is a method that is appropriate when the cyclic load is below and up to N_t. When the cyclic loading reaches beyond the transition fatigue life the stress-based approach is often more useful. The difference between the two, if they both are applied normally, is that the strain-based approach analyses growing cracks by methods of fracture mechanics.

S-N curves are plots that describe how a sample of a material reacts to subjected large cyclic stress. If the number of cycles is increased it is because of a decrease in stress. A logarithmic straight line approximation is often used if the amount of test data is large enough.

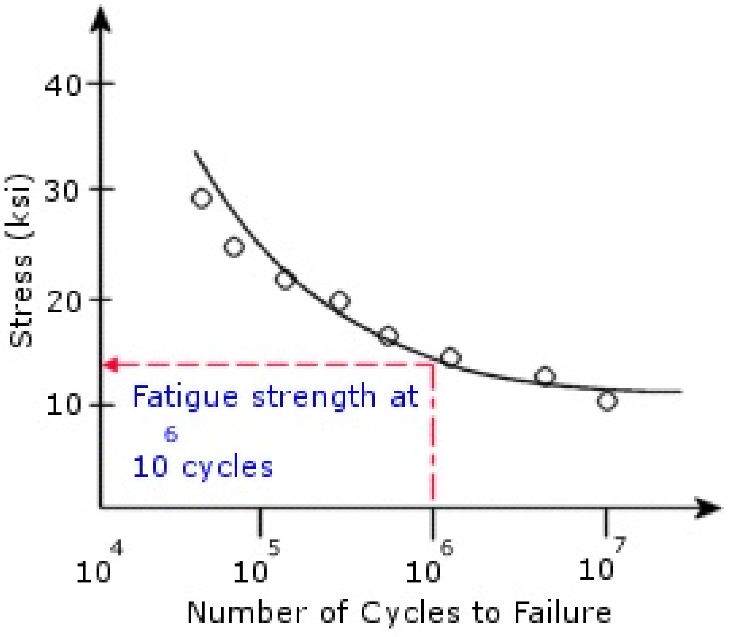


Figure 5.5: An example of a S-N curve. As the number of cycles increase the stress value must decrease. [9]

5.6 Proportional and non-proportional stress

The principal stress changes when a component is subjected to constant cyclic loading. What is important to consider is not just the size of the stress factor, but also the direction. When the principal stress direction remains constant, the component is subjected to proportional loading. If the direction changes, the loading is of a non-proportional nature. These are important aspects to consider when dealing with a multiaxial loading state as it can lead to a complicated state where standard S-N curves and stress-strain calculations are inefficient.

Mohr's circle is often used to describe the stress state of a component.

Figure 3 and 4 shows how Mohr's circle changes axes because of the non-proportional stress.

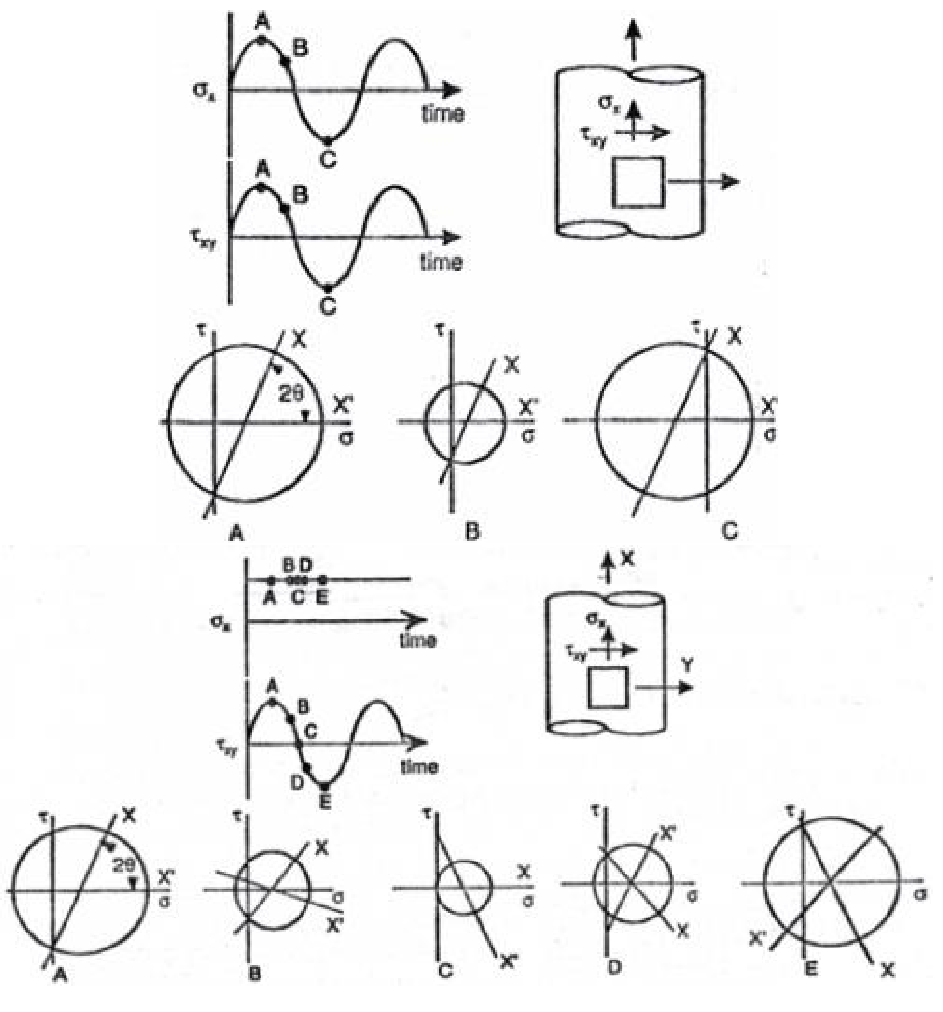


Figure 5.6: Mohr's cycle for non-proportional stress [6]

5.7 Multiaxial fatigue

Multiaxial states, states where stress occur on several planes and directions, are common and necessary in some structures for it to fulfil its means. Examples are tensile bars, crankshaft and other shafts that transmit torque. However, testing of multiaxial states have not been as well documented over the years as uniaxial loading. And when there is need for an analysis of multiaxial fatigue a uniaxial method has often been carried out to give a pinpoint of how the component will behave and eventually fail.

Given the large database of uniaxial test data it is possible to present a good approximation for multiaxial cases. Most approaches are designed with the thought that an equivalent stress-strain approach approximation is sufficient. Examples of these methods are the maximum shear, the maximum principle, the Tresca equivalent stress and the Von Mises equivalent stress.

When considering multiaxial fatigue the are some aspects that are different from the uniaxial and needs to be taken into account. First of all there is the fact that fatigue is a directional process and that cracks grow on particular planes. Also, neither of the uniaxial approaches considers the median principle stress nor cares about the applied hydrostatic stress.

5.8 Critical Plane approach

Equivalent stress-strain approaches, as mentioned in the previous chapter, does not take into consideration that fatigue is directional and that cracking propagate along a plane. The Critical Plane approach does however exactly so [8].

The critical plane approach take into account that both the shear stress and the normal stress that are acting on a plane. Stress on a point in a component can be resolved into an infinite amount of planes passing through it. Remembering from a previous chapter about the stress tensor and how it can be transformed to describe the stress states in new planes the critical approach can be regarded as an extension.

The normal stress, σ_N, is perpendicular to the plane and the shear stress, τ, is acting on the plane. On a cubic element the normal stresses σ_X, σ_Y and

σ_Z are acting along the with six shear stresses. Considering equilibrium of the element reduces the number of stress components to six, and these are used to describe all stresses on a given plane through the point.

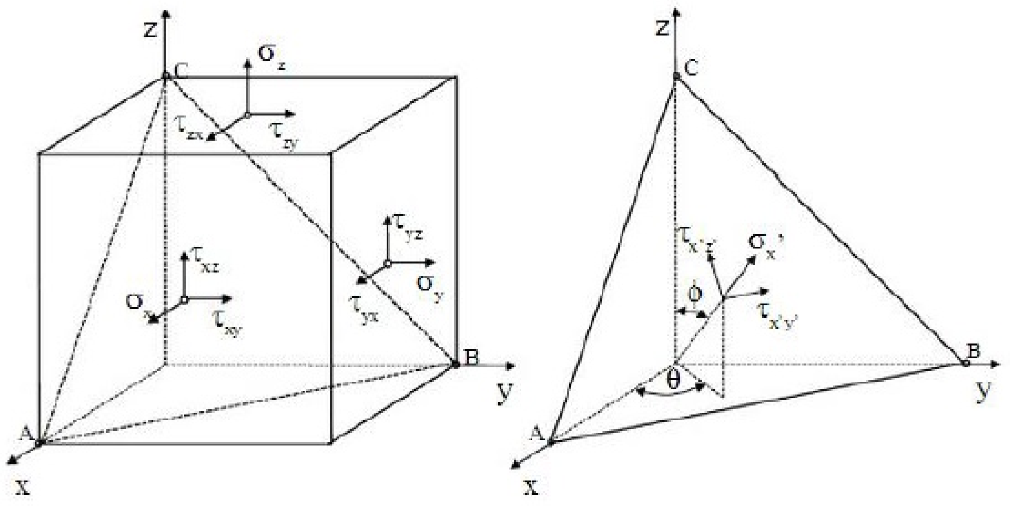


Figure 5.7: The left element is the state before the transformation. The right element is transformed with the given angles θ and ϕ and new stresses are calculated according to the new coordination system.

5.9 The Findley Model

The Findley model uses, as other critical approach methods, the matrix transformation equations described in a previous chapter to find the critical plane for shear and normal stresses. The damage criterion, which is different for each fatigue models, is the reference value which is checked against at all planes through a point. [12]

The linear influence of the normal stress on a shear plane has an alternating shear stress,$(\frac{\delta\tau}{2})$. The idea is that if

$$(\frac{\delta\tau}{2} + k\sigma_n)_{max} <= f \tag{14}$$

fatigue will not happen. In the equation above, known as the Findley model, σ_n is the normal stress and k and f are material parameters. A critical plane for crack initiation is dependent on alternating shear stress and maximum normal stress.

The left side of equation BLA OBS is known as the Findley damage parameter. When searching all the planes through a point of a stress history, the value with the highest Findley damage parameter will be the most vulnerable point on the structure, and the critical plane will appear with this value, by the angles θ and ϕ.

The k constant is normally found through experiments and fatigue tests for several stress states on the component. The parameter normally varies between 0.2 and 0.3 for ductile parameters.

5.10 The principal stress

Remembering the stress tensor from previous chapters, it is known that the state of an element of a structure is described by nine stresses, thereof three normal stresses and six shear stresses. In the case where the tensor is transformed such that the shear stresses all equal zero, the principal stress occur. In other words, at these angles there only exists normal stress which is given the name principal stress.

For the general case of plane stress the following equation give the principal stress [2, p.56]

$$\sigma_1, \sigma_2 = 0,5 \cdot (\sigma_x - \sigma_y) \pm 0,5 \cdot \sqrt{(\sigma_x - \sigma_y)^2 + 4\tau_{xy}^2} \tag{15}$$

The plane where shear stress is largest is oriented 45deg rotated relative to the largest and smallest principal stress, calculated from the formula below.

$$\tau_{max} = 0,5 \cdot \sqrt{(\sigma_x - \sigma_y)^2 + 4\tau_{xy}^2} \tag{16}$$

For each axis of the stress tensor it exist a maximum shear stress. Using Mohr's circle one can calculate the largest shear stresses using the following formulas (Norton, 2002, p. 349)

$$\tau_{13} = \frac{|\sigma_1 - \sigma_3|}{2} \tag{17}$$

$$\tau_{23} = \frac{|\sigma_2 - \sigma_1|}{2} \tag{18}$$

$$\tau_{32} = \frac{|\sigma_3 - \sigma_2|}{2} \tag{19}$$

If the circle with the largest radius is given by the principal stress σ_1 and σ_3, the largest shear stress will always be τ_{13}. An example of a Mohr's circle for three dimensional states are also given below.

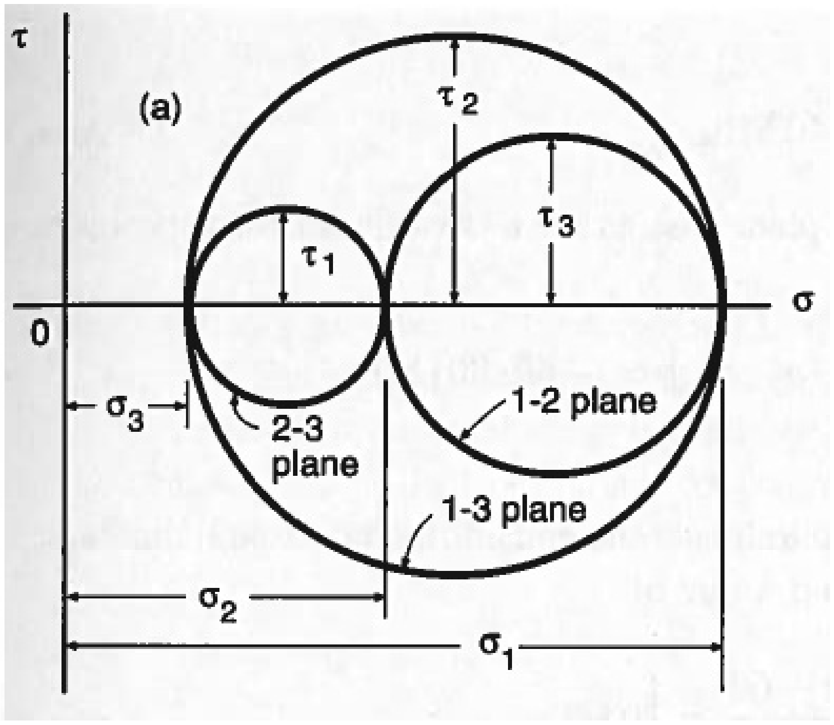


Figure 5.8: The Mohr's circle for 3D stress.

When working with the critical approach method it is clear that the stress tensor describes the stress at one point of a structure. What is worth noticing is that the principle stress of a $3x3$ matrix can be found by finding the eigenvalues of the matrix. Applying this to the stress tensor one can then find the largest principle stress at the critical plane for the critical element. The eigenvectors will give the directions for the principle stress.

6 Visualization

This thesis seek to look at the mathematical side of the critical plane approach, but for the users to get a better understanding of it a visualization is made to display the stress tensor and various data. This section will explain some of the tools used for the visualization.

6.1 Qt

Qt is a framework built for C++ programming. It is a cross-platform UI and application development framework and is growing in popularity by developers. It is mainly for developing 2D graphical interfaces, but also work well with OpenGL and thus 3D graphics its within its scope. This project mainly uses Qt for user interaction and and as a basis for visualization with the Visualization Toolkit.

The framework consist of several small and large modular libraries which form the building blocks for an application [1]. The fundamental module is the Qt Core Module which handles file IO, objects, multi-threading, plug-ins and the signal and slots communications mechanism. These are all functions which are non-graphical and thereby justifiably serve the foundation in Qts ground up architecture. For graphical purposes the GUI Module delivers among other things a set of customizable widgets, fonts, graphics effects, a 2D graphics canvas and OpenGL integration. The building blocks make user interface creation become an easy task, with few lines of code.

OpenGL is also a featured module in the Qt C++ framework library, and introduce 3D graphics classes for the Qt application. OpenGL is by itself not a permissive framework for building applications, but the widgets one can incorporate with Qt is designed so that one only need to focus on the rendering of OpenGL.

Other important modules in the framework are the 2D Graphics View for fast graphics effects in view widgets, the Multi threading feature which introduce parallel programming to the application, the Qt Script Module based on a JavaScriptCore back-end and a Networking module for client and server socket handling. There is also modules for building and handling databases, webkit integration and XML.

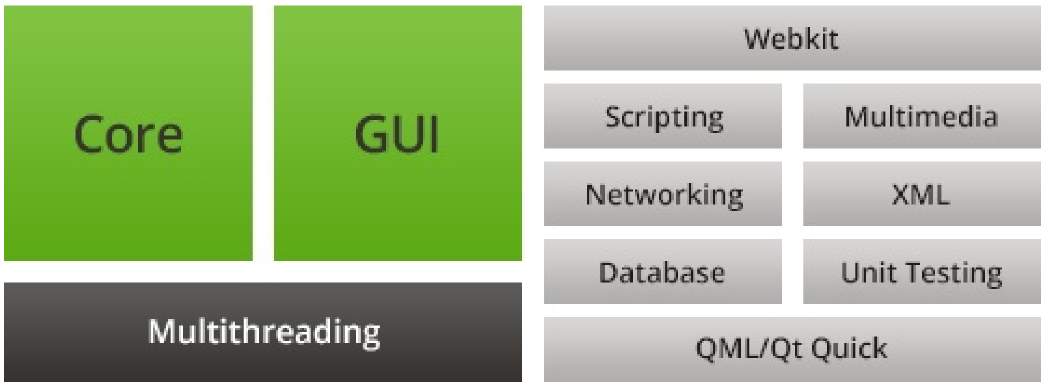


Figure 6.1: The figure shows the building blocks of the Qt C++ framework library.

When designing a Qt application there are several tools one can use for creating the layout as the Qt framework is cross-platform compatible. In this thesis native Windows 8.1 compilers and Visual Studio 2012 are used for creating the c++ application and Qt integrate without problems. Because a big part of the Qt tool-kit is the easy-to-use UI creation, a Visual Studio plug-in has been designed by Qt Digia with the entire library of for instance pre-made Qt GUI buttons and widgets. This allow for extremely efficient application design because many files are auto-generated by the plug-in.

6.2 The Visualization Toolkit

The Visualization Toolkit (VTK) is a 3D computer graphics system which consist of a C++ class library and several interface layers for other languages such Java and Python. It is open-source and freely available for public use. Kitware is the company behind the software which is mainly used for visualizing complex engineering problems, but it is also much used by people with a medical background. [10]

Modelling, image processing, volume rendering, information and scientific visualization are among the tasks VTK can perform. For this thesis the 3D computer graphics, modelling and information visualization has mainly been used to visualize polygonal data of spheres, planes and lines.

The architecture of VTK can be split into nine basic objects. At the bottom, the most significant object, is the Render Master which creates the rendering windows and coordinates methods. The Render Window is what manages the window in the display device and adds Render objects to the window. Then there is a Light object which illuminates the different Actors on display. Actors are also a basic object and defines a figure to be drawn by the renderer. The Camera object handles view positions, focal points and other camera properties. A Property object, a Mapper and a Transform objects are all Actor instances and are required for an Actor to display properly. The property object handles the actors attribute such as colouring and shading style, the mapper represent the geometric definition of a actor and give the points or cells on an actor values from a lookup table and finally the transform object specifies the position and orientation of all the other basic objects.

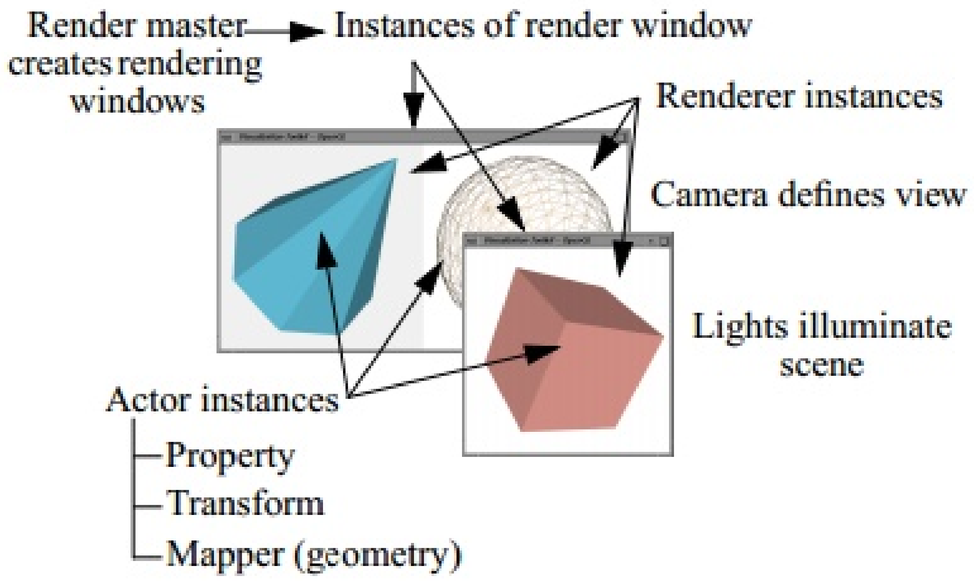


Figure 6.2: This figure display the architecture of the Visualization Toolkit.

6.3 Integrating Qt and VTK

There are several guides on the Internet on how to integrate the two frameworks, so that they together can form a powerful tool for 3D visualization application development. As this thesis focus both on visualizing a scientific result as much as the science behind it, Visual Studio 2012 was needed as a tool in combination with QT and VTK. Integrating the two c++ librarires with Visual Studio was done by building them both from source using the CMake program. Then after installing some necessities such as Python and a WebKit package, the Qt plug-in in Visual Studio integrated with the VTK widget built from source. This widget, the QVTKWidget, is what truly combines Qt and VTK as the input can be handled by both parties. The Qt plug-in also come with a Qt Designer where a drag and drop function which means that using the QVTKWidget is as efficient as any other Qt widget.

7 System Description

Programming a software application requires a structured plan. This section will deal with the architecture of the application, the design and other aspects which are relevant.

7.1 Software requirements description

When creating a software application it is important to thoroughly think about what tasks the software should be able to perform and how. Often this information is given by a employee or third party, in the case of this thesis it is given partly by the project description, but mainly it has been up to the author to decide.

When looking at the objectives for this thesis one can grasp a basic view of the major tasks the software need to perform. It is however necessary to take a closer look at what should be required of the software.

There are several templates one can use to create a software requirements description, but since no two projects are the same it is made one specifically for this problem. The main focus is on what properties the software should have, functional and non-functional. It is not an explicitly detailed specification, but then again it should fit the size of the application and do for further planning.

Name	Type	Description	Solution
Performance	Non-functionality	The software should perform at a decent speed. Can be compared to similar Matlab-software.	Framework, Eigen
Matrix transformations	Functionality	For calculations necessary to do the critical approach method. A framework that works well with matrices is wanted.	Eigen,

Principle stress calculations	Functionality	Calculating the principal stress for 3D systems can be a challenge. A framework which can handle this is wanted.	Eigen
Calculate angles and planes	Functionality	To find the critical plane it is important that the framework used can supports calculation with angles.	Eigen, c++
Visualize 3D objects	Functionality	The software should be able to visualize a sphere with colours, planes and data attributes.	VTK
Different input formats	Functionality	The software should be able to handle different input formats such as excel files and csv. Data sets could come from several sources.	Recognition of input format from file type.
User interference	Functionality	The user should be able to choose which stress to be shown - shear or normal.	Qt and VTK

Table 7.1: Software requirements specification

7.2 System Architecture

The use of Qt and VTK make this system relatively simple in a terms of system architecture. A standard main() c++ function initiates a QApplication, which is a Qt object and the foundation of the entire application. Then a QMainWindow instance is created which is what will bring some context to the QApplication. From here all VTK objects and classes are initiated and the Critical Plane Approach is calculated to give the visualization some data. The VTK objects are then sent to a QVTKWidget, which is a Qt made widget built especially to display VTK objects. Finally the widget and other UI objects from the QMainWindow are sent back to the QApplication.

From figure 11 it is possible to recognize the VTK objects from the theory section of this thesis, as you can see how the mapper, object and actor is added to the render. The Qt framework provide the remaining parts for VTK to work properly with the QVTKWidget. The Critical Plane Approach calculation is not given a large role in this architecture, but it can however be located in the function doCaluclation(). As this is just a function, although important to the application, it is not a crucial part for the surroundings. In fact if the function was to be replaced by another which return the same type of object it would not matter for the application to work. It is therefore fair to say that the mathematical part of this application is heavily separated from the visualization part, which is more complex.

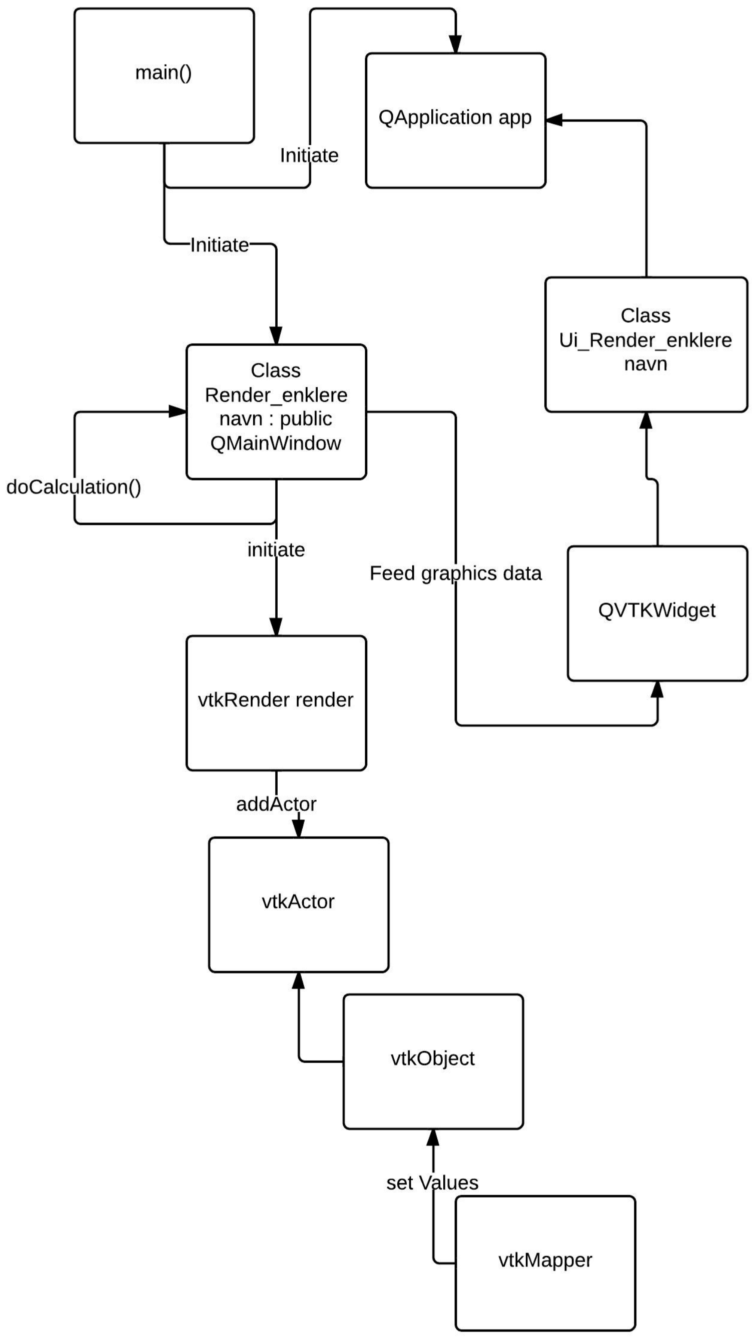


Figure 7.1: This figure display the architecture of the application - how the communication flow between different classes.

7.3 Class Diagram

Many of the classes in this thesis is auto generated by Qt or VTK and most of them serve the purpose of visualizing the output from the Critical Plane Approach calculation. This class diagram therefore leave the auto generated classes aside an focus on those which give this application a purpose.

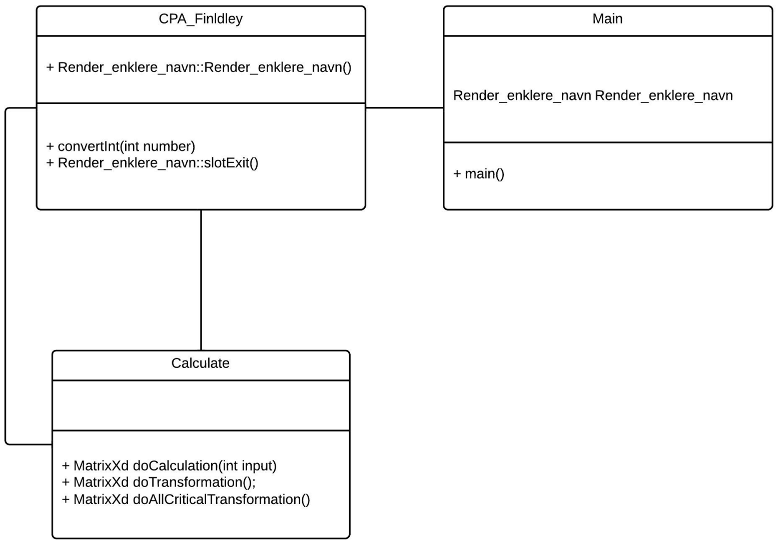

CPA_Finldley
+ Render_enklere_navn::Render_enklere_navn()
+ convertInt(int number)
+ Render_enklere_navn::slotExit()
Main
Render_enklere_navn Render_enklere_navn
+ main()
Calculate
+ MatrixXd doCalculation(int input)
+ MatrixXd doTransformation();
+ MatrixXd doAllCriticalTransformation()

Figure 7.2: This figure display the most important classes and functions of the application.

To better explain the classes a table has been made to describe the output and input variables, and what the main tasks are.

File name	function name	Input/Output	Description
main.cpp	main()	Inputt: - Output: -	The standard c++ function which starts when the executable file is started. Initializes a Qt application and starts the rendering.

Table 7.2: main.cpp functions and classes

CPA_Findley .cpp	Render_enklere _navn() - constructor	Constructs a Qt application	This is where all graphics are initiated and handled. All VTK objects can be found here.
CPA_Findley .cpp	convertInt()	Input: int number Output: string - converted integer	A function which converts an integer to a string. Used to display number as text in the widget.
CPA_Findley .cpp	slotsExit()	void	A Qt slot function which makes sure the application closes when the Exit-button is pressed.

Table 7.3: CPA_Findley.cpp functions and classes

calculation.cp p	doCalculation()	Input: int input = 1 outputs MatrixXd criticalStressTensor, input = 2 outputs MatrixXd criticalPlane, input = 3 outputs principalStress. Also creates a txt file with detailed critical stress tensor information.	All main calculations are done here, also all communication between the render_enklere_navn.cpp file and this file is done through this function.

calculation.cpp	doTransformation()	Input: MatrixXd stressTensor, double ThettaAngle, double PhiAngle, int elementIndex Output: MatrixXd transformedStressTensor	This function transforms a chosen stress tensor from the data set with the angles thetta and phi. Used for validation and easier output for the doCalculation() function.
calculation.cpp	doAllCriticalTransformation()	Input: MatrixXd stressTensor, MatrixXd MatrixofAngles, int elementIndex, int numberOfIntervals Ouput: MatrixXd criticalStressAllAngles	This function delivers a combination of transformed stress values for all angles from the wanted element.

Table 7.4: calculation.cpp functions and classes

7.4 Design layout

Displaying the scientific calculation from a large set of data offer a challenge. The fundamental idea of the Critical Plane Approach, which researches the entire data set for for all possible planes does however provide a solution.

By creating a sphere to display the stress components of the tensor one should be able to understand the data set output on a figure. As all planes are investigated, this naturally create a sphere where each point on the sphere has a transformed value of the stress component. By giving these values color from a look-up-table, the visualization is rather understandable.

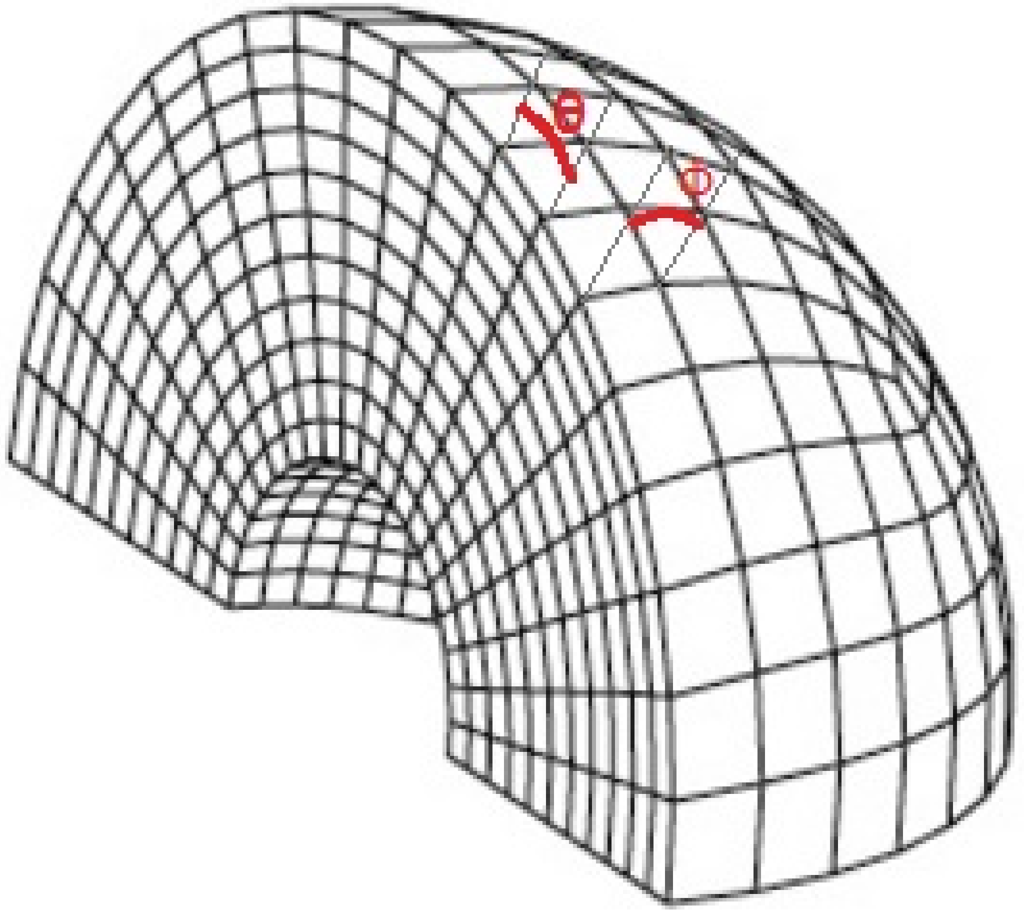


Figure 7.3: A sphere with the steps theta and phi.

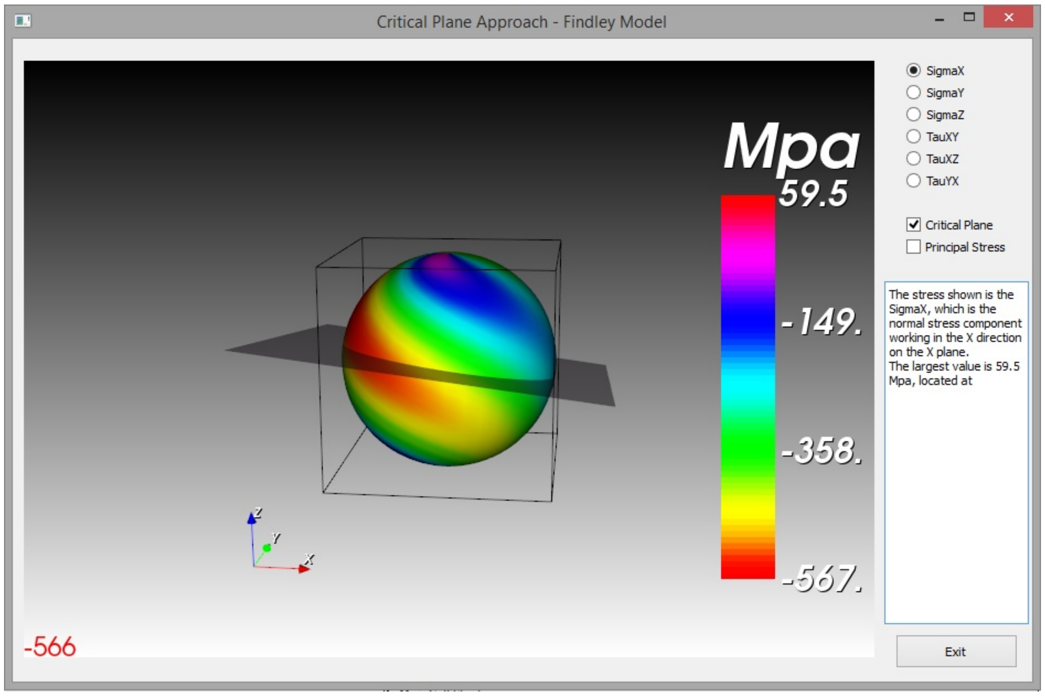
The figure above illustrates the idea. Where each line meet a point is formed with two specific angles θ and ϕ. As each of the angles increment they move to a new point to create a new plane where the stress tensor component is tested. Finally when all points are investigated a complete sphere with unique data is created.

The angles which finally become the critical plane is also of interest. As these angles are given by the Critical Plane Approach it is a matter of creating a plane which intersect he sphere at the correct angles.

8 Results

8.1 The Application software

From the software requirements description it was created a perspective of what functions and properties the application should have. The result is a simple application where most of these functions and properties are realized. The Appendix A show the source code for this application. There is also a digital attachment to this thesis with the application software.



[✓ SigmaX] SigmaY SigmaZ TauXY TauXZ TauYX

[✓ Critical Plane] Principal Stress

The stress shown is the SigmaX, which is the normal stress component working in the X direction on the X plane.
The largest value is 59.5 Mpa, located at

Figure 8.1: The final appearance of the application.

In the center of the application is the sphere, which can be rotaded with a touchpad or computer mouse when dragged. Each point on the sphere represent one value of a the stress component for the angles which create this point. It is of course the stress tensor components of the critical element which is at display. For interpreting the colors mapped on the sphere a lookup table bar is showed on the right hand side of the widget.
On the right hand side of the application window it is possible to change the desired input values for the sphere to another stress component. There is

also a toggle function for hiding or showing the critical plane.
The critical plane is shown by the the gray plane passing through the sphere. The plane is fixed and rotate with the sphere as it the mouse is clicked and dragged.
The text-box try to explain what the viewers see in the widget, an provide some explanatory data such as the angles of the critical plane. Finally there is an axis-system, showing the x, y and z-direction and cube edges around the sphere for better visualization.

8.2 The Critical Plane approach calculation

The critical plane approach demand a large dataset for it to serve a purpose. The data set used to test the application is retrieved from a previously written thesis by Simen Riiser at NTNU [4]. The test data was in his thesis used to calculate the expected life span of a coupler, made from a ductile material. The thesis also used the critical plane approach with Findleys method to investigate the data set. In addition a Excel script was created in this thesis for manual calculation with given angles.

Program	Critical Element	Findely Criterion	Critical Theta	Critical Phi
Matlab	9509	251,089	90	100
CPA Application	9509	251,09	90	100
Excel	-	251,0088599	90	100

Table 8.1: Displaying the calculated results from the application compared to a matlab script

As the table above show, the values from all three scripts are the same. Of course, in the Excel file only the angles vary and the stress tensor is collected from the data set used in the matlab script and the application. That the Findley criterion in reality is the same value is however a reassuring as all calculations in the three programs are completely separated. That the critical element in both the matlab script and the application is the same, as well as the angles and the Findely criterion, is the true test. The excel file does however function as a good test for the fundamental matrix transformation mathematics.

In addition to the Findely criterion and the critical element, the principal stress has been calculated. This is not visualized in the final application, but the application write this information to a txt file for engineers to investigate.

9 Discussion

9.1 Fatigue calculation

This thesis set out to calculate the fatigue of non-proportional stress histories. The information the application processes are most likely simulations of structures of a material with a calculated ductile parameter, k for use in the Findely model. The application created, successfully analyse the data set and find the element where the most critical stresses occur at a given plane. What is important to remember however is that there are more aspects to take into consideration when using these findings to calculate presumed life span or similar. First of all it is not always likely that the structure will experience stresses on this plane, second the simulations used to create the data set does not necessarily integrate all the necessary material properties to calculate the correct stresses. In other words, this application must be seen as a tool, a part of a larger picture for finding the weakest point on a structure.

Even though the results from this calculation is compared to other similar calculations and found to be consistent, the amount of test data is in reality too small for any conclusions to be drawn about the reliability of the calculations. There has only been one data set available for testing the application, but to further investigate the liability of this software, more tests must be undertaken.

9.2 The Application

Building the application based on Qt and VTK result in a solid environment for the fatigue calculation and visualization. Integrating the two is from the beginning not a hassle free task, as it there are a total of three environment that is supposed to work together. It should be recommended for any other programmer that wishes to set up a similar environment to allocate a good amount of time for this process. However, when the environment is functioning, it provides an excellent tool for GUI creation and 3D graphics visualization taken into consideration the pre-made buttons and window element of the Qt alongside the substantial amount of 3D objects available from the VTK library.

This application uses a sphere to visualize the critical stress tensor for the critical element. The critical plane is also displayed at the correct angels. How this come across to the users of this application is not determined. It is reasonable to believe that the amount of data provided by the application is not enough to fully understand the complex underlying calculations. First of all the sphere is not a intuitive way of relating to a stress tensor component. It is first when the user understand the underlying mathematics, how each point on the sphere is represented by a transformed stress value for the given plane, that it become intuitive.

A visualization of the principle stress may help the users gain a clearer view of what the calculations result to. Other items that might increase the usability of this application are more data from the initial stress analysis. The data set used in this thesis is a history made of only two stress tensors. If the application had more stress tensors from an element it would be possible to see how the stress on the element changed over time, which certainly would be beneficial for educational purposes.

9.3 Accomplished requirements

Previously in this thesis there was created a table describing the requirements of the software. This section will determine how and whether they are fulfilled.

Name	Type	Description	Status
Performance	Non-functionality	The software should perform at a decent speed. Can be compared to similar Matlab-software.	The application is fast, but not tested properly.
Matrix transformations	Functionality	For calculations necessary to do the critical approach method. A framework that works well with matrices is wanted.	Accomplished. Use of the Eigen c++ library.

Principle stress calculations	Functionality	Calculating the principal stress for 3D systems can be a challenge. A framework which can handle this is wanted.	Accomplished.
Calculate angles and planes	Functionality	To find the critical plane it is important that the framework used can supports calculation with angles.	Accomplished.
Visualize 3D objects	Functionality	The software should be able to visualize a sphere with colours, planes and data attributes.	Partly accomplished. Some objects are missing.
Different input formats	Functionality	The software should be able to handle different input formats such as excel files and csv. Data sets could come from several sources.	Not accomplished.
User interference	Functionality	The user should be able to choose which stress to be shown - shear or normal.	Accomplished.

Table 9.1: Accomplished software requirements

10 Conclusion

The work completed in this thesis has resulted an application for fatigue calculation of non-proportional stress histories and a visualization of the result. A critical plane approach, the Findley model, was implemented for the fatigue calculation and the open-source libraries Qt and Visualization Toolkit was used for building the application.

An objective for this thesis was to make it free to use for the public, for people in industry and for educational purposes. As all code is written from open-source libraries or created separately this objective must be seen as accomplished. If the application is of any value to potential users is yet to be determined, but hopefully it is so.

Concerns regarding the quality of this application is however present. Calculations performed on the data set prove to be consistent with other results from matlab-scripts and excel files, but as there is only one data set available for testing, this application should not be considered fully tested.

The application visualize the transformed stress tensor components, the critical plane, and a look up table as the main components. The principal stress is calculated and written to a text file for users to view and use for further calculation. A Findley damage parameter for the critical element is calculated from the data set and the result correspond to a Matlab-script based on the same methods. The critical plane also correspond to these calculations.

Appendices

A Source Code

Listing 1: main.cpp file

```
#include "cpa_findley.h"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
  // QT Stuff
  QApplication app( argc, argv );

  Render_enklere_navn Render_enklere_navn;
  Render_enklere_navn.show();

  return app.exec();
}
```

Listing 2: calculation.cpp file

```
#include <iostream>
#include <fstream>
#include <Eigen/Dense>
#include <Eigen/Eigenvalues>
#include <math.h>
#include <algorithm> // for copying
#include <locale>
#include "calculation.h"

#define PI 3.14159265

using namespace std;
using Eigen::MatrixXd;
using Eigen::EigenSolver;

template <class T>
T fromstring ( std::string s )
{
  T result;
  std::stringstream str;
  str << s;
  str >> result;
  return result;
}

class WithComma: public numpunct<char> // class for decimal
    numbers with comma
{
  protected: char do_decimal_point() const { return ','; } //
      override the function that gives the decimal separator
};


MatrixXd doCalculation(int input)
{

  MatrixXd test(2,2);

  for(int ti=0; ti<2; ti++){
    for(int tj=0; tj<2; tj++){
      test(ti,tj) =ti+tj;
    }
  }
//
```

```
    /////////////////////////////////////////////////////////////////////////////////
//////Dealing with the angles and creating a set of angles
    and planes that will be investigated////
//
    /////////////////////////////////////////////////////////////////////////////////


double degreeInterval = 5.0; // starting from 0 to 180 with
    an interval of 5 degrees
double radianInterval = (PI/180)*degreeInterval;

int numbIntervals = ceil((PI+radianInterval)/radianInterval);
     //How many intervals??

MatrixXd intervalMatrix(1,numbIntervals); // the matrix which
     contains the radian values for each step
  intervalMatrix(0,0) = 0.0;
for(int i=0; i<numbIntervals-1; i++){
  intervalMatrix(0,i+1) = intervalMatrix(0,i) +
      radianInterval;
}

MatrixXd matrixTetta(1, numbIntervals*numbIntervals); //All
    Tetta angles
MatrixXd matrixPhi(1, numbIntervals*numbIntervals); //All phi
     angles

int k=0;
for(int i=0; i<numbIntervals*numbIntervals; i++){
  if(k==numbIntervals){
    k=0;
  }
  matrixTetta(0,i) = intervalMatrix(i/numbIntervals); //
      filling the matrix with tetta angles
  matrixPhi(0,i) = intervalMatrix(0,k); // filling the matrix
       with phi angles
  k++;
}

MatrixXd anglesMatrix(2,numbIntervals*numbIntervals); //
    (0,0...n) for Tettas, (1,0...n) for Phis

// Finally the wanted set of angles in anglesMatrix
```

```
for (int j=0; j<numbIntervals*numbIntervals; j++){
  anglesMatrix(0,j) = matrixTetta(0,j);
  anglesMatrix(1,j) = matrixPhi(0,j);

}


/////////////////////////////////////////////////////
//// Dealing with the input in this section ///////
/////////////////////////////////////////////////////



locale myloc(  locale(), new WithComma);// Own numeric facet
string readFromfile;
ifstream r_file_number("OperationTest.csv");
ifstream r_file("OperationTest.csv");
int numberoflines=0;
stringstream *temp;
double n;

//How many lines are there in the input file to be read?
while (getline(r_file_number, readFromfile, ';')) {
    numberoflines++;
}
std::cout << "numberOfLines␣" << numberoflines << "\n" << std
   ::endl;
std::cout << "(numberoflines-1)/6␣" << (numberoflines-1)/6 <<
    "\n" <<  std::endl;


int numbCols = 6; //one column for each stress component
int numbRows = ((numberoflines-1)/6); // on row for each
   stress tensor


// Puting all the stresshistories from the the excel file
   into a matrix where each row represents one stresstensor
// The first number on each line, the element number, in the
   csv-file is negelected
/*
The format of inputData:
inputData(i,0) = sigmaX
inputData(i,1) = sigmaY
inputData(i,2) = sigmaZ
inputData(i,3) = tauXY
inputData(i,4) = tauXZ
inputData(i,5) = tauYZ
......
*/

MatrixXd inputData(numbRows,numbCols);
int count=0;
int numberOfLinesAfter=0;
while (getline(r_file, readFromfile, ';')) {
  if(numberOfLinesAfter == 0){ //Skipping the first line in
     every row
    numberOfLinesAfter++;
    continue;
  }
  temp = new stringstream(readFromfile);
  temp -> imbue(myloc);
  temp->operator >> (n);
  inputData(numberOfLinesAfter-1, count) = n;
  count++;
  if(count==6){
    count=0;
    numberOfLinesAfter++;
  }
}


  // Just to make sure the format is correct... : printing to
      a file example.txt
  ofstream myfile ("inputFile.txt");
  if (myfile.is_open())
  {
  for(int wR=0; wR<numbRows; wR++){
    for(int wC=0;wC<numbCols;wC++){
      myfile << inputData(wR,wC);
      myfile << "\n";
    }
        myfile << "newline␣\n";
  }

    myfile.close();
  }

//
   ////////////////////////////////////////////////////////////////////////////
```

```
///////// Dealing with the transformation matrix (a-values)
   //////////////////
//
   //////////////////////////////////////////////////////////////////////////////

    MatrixXd a11(1, numbIntervals*numbIntervals); //
       calculate the a values
  MatrixXd a12(1, numbIntervals*numbIntervals);
  MatrixXd a13(1, numbIntervals*numbIntervals);
  MatrixXd a21(1, numbIntervals*numbIntervals);
  MatrixXd a22(1, numbIntervals*numbIntervals);
  MatrixXd a23(1, numbIntervals*numbIntervals);
  MatrixXd a31(1, numbIntervals*numbIntervals);
  MatrixXd a32(1, numbIntervals*numbIntervals);
  MatrixXd a33(1, numbIntervals*numbIntervals);

  MatrixXd stressTensor(2,6);
  double sigmaXTransf, sigmaYTransf, sigmaZTransf,
     tauXYTransf, tauXZTransf, tauYZTransf;


  MatrixXd transformedStressTensor(2,6);
  double normalmax = 0.0;
  double skjerXYmax = 0.0;
  double skjerXYmin = 0.0;
  double skjerviddeXY = 0.0;
  double skjerXZmax = 0.0;
  double skjerXZmin = 0.0;
  double skjerviddeXZ = 0.0;
  double resSkjervidde = 0.0;

  double findleyCriterion = 0.0;
  double findleyFactor=0.1362;

  double critTetta = 0.0;
  double critPhi = 0.0;
  double findleyCriterionLast = 0.0;
  double finalFindley = 0.0;
  double critnormalmax= 0.0;
  double critskjerXYmax = 0.0;
  double critskjerXYmin = 0.0;
  double critskjerXZmax = 0.0;
  double critskjerXZmin = 0.0;
```

```
int nextline=0;
int which=0;

//findleys(numbRows, 4) isn't really used yet.. It's
   supposed to be a "sum up" of all the elements with
   critical findley values and angles.
MatrixXd findleys(numbRows, 4);
int progress =2;

while(nextline < numbRows){ //For each element, not each
   stress tensor

  // find the 2*six next stresses and put them in two
     tensors   !!! OBS - not the general case - needs
     fixing
  for(int is=0; is<12; is++)
  {
    if(is<6){
    stressTensor(0,is)=inputData(nextline,is);
    }
    else{
      stressTensor(1,is-6)=inputData(nextline+1,is-6);
    }
  }

//For each set of possible angles from the anglesMatrix, to
    calulcate the transformation matrix
for(int ia=0; ia<numbIntervals*numbIntervals; ia++){

  a11(0,ia) = cos(anglesMatrix(0,ia))*sin(anglesMatrix(1,ia
     ));
  a12(0,ia) = sin(anglesMatrix(0,ia))*sin(anglesMatrix(1,ia
     ));
  a13(0,ia) = cos(anglesMatrix(1,ia));
  a21(0,ia) = -sin(anglesMatrix(0,ia));
  a22(0,ia) = cos(anglesMatrix(0,ia));
  a23(0,ia) = 0;
  a31(0,ia) = -cos(anglesMatrix(0,ia))*cos(anglesMatrix(1,
     ia));
  a32(0,ia) = -sin(anglesMatrix(0,ia))*cos(anglesMatrix(1,
     ia));
  a33(0,ia) = sin(anglesMatrix(1,ia));

  // for each stress history - !!!OBS not the general case
```

```
   - needs fixing
for(int ie=0; ie<2; ie++)
{

transformedStressTensor(ie,0) =
a11(0,ia)*a11(0,ia)*stressTensor(ie,0) +
a12(0,ia)*a12(0,ia)*stressTensor(ie,1) +
a13(0,ia)*a13(0,ia)*stressTensor(ie,2) +
2*a11(0,ia)*a12(0,ia)*stressTensor(ie,3) +
2*a11(0,ia)*a13(0,ia)*stressTensor(ie,4) +
2*a13(0,ia)*a12(0,ia)*stressTensor(ie,5);

transformedStressTensor(ie,1) =
a21(0,ia)*a21(0,ia)*stressTensor(ie,0) +
a22(0,ia)*a22(0,ia)*stressTensor(ie,1) +
a23(0,ia)*a23(0,ia)*stressTensor(ie,2) +
2*a21(0,ia)*a22(0,ia)*stressTensor(ie,3) +
2*a21(0,ia)*a23(0,ia)*stressTensor(ie,4) +
2*a23(0,ia)*a22(0,ia)*stressTensor(ie,5);

transformedStressTensor(ie,2) =
a31(0,ia)*a31(0,ia)*stressTensor(ie,0) +
a32(0,ia)*a32(0,ia)*stressTensor(ie,1) +
a33(0,ia)*a33(0,ia)*stressTensor(ie,2) +
2*a31(0,ia)*a32(0,ia)*stressTensor(ie,3) +
2*a31(0,ia)*a33(0,ia)*stressTensor(ie,4) +
2*a33(0,ia)*a32(0,ia)*stressTensor(ie,5);;

transformedStressTensor(ie,3) =
a11(0,ia)*a21(0,ia)*stressTensor(ie,0) +
a12(0,ia)*a22(0,ia)*stressTensor(ie,1) +
a13(0,ia)*a23(0,ia)*stressTensor(ie,2) +
(a11(0,ia)*a22(0,ia) + a12(0,ia)*a21(0,ia))*stressTensor(
   ie,3) +
(a13(0,ia)*a21(0,ia) + a11(0,ia)*a23(0,ia))*stressTensor(
   ie,4) +
(a12(0,ia)*a23(0,ia) + a13(0,ia)*a22(0,ia))*stressTensor(
   ie,5);

transformedStressTensor(ie,4) =
a11(0,ia)*a31(0,ia)*stressTensor(ie,0) +
a12(0,ia)*a32(0,ia)*stressTensor(ie,1) +
a13(0,ia)*a33(0,ia)*stressTensor(ie,2) +
(a11(0,ia)*a32(0,ia) + a12(0,ia)*a31(0,ia))*stressTensor(
   ie,3) +
(a13(0,ia)*a31(0,ia) + a11(0,ia)*a33(0,ia))*stressTensor(
   ie,4) +
(a13(0,ia)*a32(0,ia) + a12(0,ia)*a33(0,ia))*stressTensor(
   ie,5);;

transformedStressTensor(ie,5) =
a21(0,ia)*a31(0,ia)*stressTensor(ie,0) +
a22(0,ia)*a32(0,ia)*stressTensor(ie,1) +
a23(0,ia)*a33(0,ia)*stressTensor(ie,2) +
(a21(0,ia)*a32(0,ia) + a22(0,ia)*a31(0,ia))*stressTensor(
   ie,3) +
(a23(0,ia)*a31(0,ia) + a21(0,ia)*a33(0,ia))*stressTensor(
   ie,4) +
(a22(0,ia)*a33(0,ia) + a23(0,ia)*a32(0,ia))*stressTensor(
   ie,5);


    // just to make it a bit easier...
sigmaXTransf = transformedStressTensor(ie,0);
sigmaYTransf = transformedStressTensor(ie,1);
sigmaZTransf = transformedStressTensor(ie,2);
tauXYTransf = transformedStressTensor(ie,3);
tauXZTransf = transformedStressTensor(ie,4);
tauYZTransf = transformedStressTensor(ie,5);

      //Finding largest normal stress and shear stresses
         for the findley criterion calculation
if(ie != 0)
      {
        if(sigmaXTransf > normalmax){
          normalmax = sigmaXTransf;

        }
          if(tauXYTransf > skjerXYmax){
          skjerXYmax = tauXYTransf;
        }
          if(tauXYTransf < skjerXYmin){
          skjerXYmin = tauXYTransf;
        }
          if(tauXZTransf > skjerXZmax){
          skjerXZmax = tauXZTransf;
        }
          if(tauXZTransf < skjerXZmin){
          skjerXZmin = tauXZTransf;
        }
      } else
      {
        normalmax= sigmaXTransf;
        skjerXYmax = tauXYTransf;
        skjerXYmin = tauXYTransf;
        skjerXZmax = tauXZTransf;
        skjerXZmin = tauXZTransf;
      }

      // Finding max and min shear range
      skjerviddeXY = skjerXYmax-skjerXYmin;
      skjerviddeXZ = skjerXZmax-skjerXZmin;

      // The resulting shear range
      resSkjervidde = sqrt((skjerviddeXY*skjerviddeXY)+(
         skjerviddeXZ*skjerviddeXZ));

      // The findley criterion for the given element
      findleyCriterion = ((0.5*resSkjervidde)+(
         findleyFactor*normalmax));

      // Deciding which criterion is largest of the
         stress tensors for the element
      // Also finding the angles tetta and phi at the
         largest criterion
      if(ie != 0)
      {
        if(findleyCriterion > findleyCriterionLast)
        {
          critPhi = anglesMatrix(1,ia);
          critTetta = anglesMatrix(0,ia);
          finalFindley = findleyCriterion;
          normalmax= sigmaXTransf;
          critskjerXYmax = skjerXYmax;
          critskjerXYmin = skjerXYmin;
          critskjerXZmax = skjerXZmax;
          critskjerXZmin = skjerXZmin;
          which = nextline;

        }
      findleyCriterionLast = finalFindley;

      }
}
```

```
    // This isn't really used...
       findleys(nextline,0) = nextline;
       findleys(nextline,1) = findleyCriterion;
       findleys(nextline,2) = critTetta;
       findleys(nextline,3) = critPhi;

  }
    nextline = nextline+2; // OBS!! Not the general case -
        needs some fixing

    /// This isn't really used either...
    if(nextline == progress*2){
      //std::cout << "nextline = " << nextline << std::endl;
      progress= progress + 2;
    }
}
  // At last! The largest findley criterion is found and the
      critical angles (and plane) with it.
  std::cout << "Final␣Findley␣criterion␣=␣" << finalFindley
      << "\n" << std::endl;
  std::cout << "Final␣phi␣angle␣=␣" << critPhi*(180/PI) << "\
      n" << std::endl;

  std::cout << "Final␣tetta␣angle␣=␣" << critTetta*(180/PI)
      << "\n" << std::endl;
  std::cout << "Final␣element␣(line␣nr..␣)␣=␣" << which << "\
      n" << std::endl;

  MatrixXd crticalStressTensorAllAngles(numbIntervals*
      numbIntervals, 6);
  crticalStressTensorAllAngles = doAllCriticalTransformation(
      inputData, anglesMatrix, which,numbIntervals);

 ofstream myfile2 ("critStress1.txt");
  if (myfile2.is_open())
  {
  //  for(int wi=0; wi<crticalStressTensorAllAngles.innerSize
      (); wi++){
  //     myfile2 << crticalStressTensorAllAngles(wi,0);
  //     myfile2 << "\n";


  //  }
    myfile2 << "The␣Final␣Findley␣element:␣\n␣\n";
    myfile2 << "PHI:␣"  << critPhi*(180/PI) << "\n";
    myfile2 << "THETTA:␣" << critTetta*(180/PI) << "\n";
    myfile2 << "The␣findley␣riterion␣Value:␣" << finalFindley
        << "\n␣\n";
    myfile2 << "the␣tensor␣history␣1:␣␣\n" << inputData(which
        , 0) << "␣" << inputData(which, 1) << "␣" << inputData
        (which, 2) << "␣" << inputData(which, 3) << "␣" <<
        inputData(which, 4) << "␣" << inputData(which, 5) << "
       \n";
    myfile2 << "the␣tensor␣histpry␣2:␣␣\n" << inputData(which
       +1, 0) << "␣" << inputData(which+1, 1) << "␣" <<
        inputData(which+1, 2) << "␣" << inputData(which+1, 3)
        << "␣" << inputData(which+1, 4) << "␣" << inputData(
       which+1, 5) << "\n";
    myfile2 << "the␣transformed␣stresses:␣\n";
    myfile2 << "ONE:␣\n";
    myfile2 << doTransformation(inputData, critTetta, critPhi
        , which) << "\n";
    myfile2 << "TWO:␣\n";
    myfile2 << doTransformation(inputData, critTetta, critPhi
        , which+1);
    myfile2 << "critvalues␣\n:␣" ;
    myfile2 << "critskjerXYmax␣=␣" << critskjerXYmax << "\n";
    myfile2 << "critskjerXYmin␣=␣" << critskjerXYmin << "\n";
    myfile2 << "critskjerXZmax␣=␣" << critskjerXZmax << "\n";
    myfile2 << "critskjerXZmin␣=␣" << critskjerXZmin << "\n";
    myfile2 << "skjervidde␣=␣" << sqrt((critskjerXYmax-
        critskjerXYmin)*(critskjerXYmax-critskjerXYmin) + (
        critskjerXZmax-critskjerXZmin)*(critskjerXZmax-
        critskjerXZmin)) << "\n";
    myfile2 << "The␣crit␣=␣" << 0.5*sqrt((critskjerXYmax-
        critskjerXYmin)*(critskjerXYmax-critskjerXYmin) + (
        critskjerXZmax-critskjerXZmin)*(critskjerXZmax-
        critskjerXZmin))+findleyFactor*normalmax;

  //myfile << "min: " << min << "\n";
  //myfile << "max: " << max << "\n";
  myfile2.close();
  }

  std::cout << "Remsa␣=␣ferdig!␣\n" << std::endl;


  MatrixXd criticalStressTensor(3,3);
  MatrixXd transformedCriticalTensor(1,6);
```

```
  transformedCriticalTensor = doTransformation(inputData,
     critTetta, critPhi, which);

  criticalStressTensor(0,0) = transformedCriticalTensor(0,0);
  criticalStressTensor(1,1) = transformedCriticalTensor(0,1);
  criticalStressTensor(2,2) = transformedCriticalTensor(0,2);
  criticalStressTensor(1,0) = transformedCriticalTensor(0,3);
  criticalStressTensor(0,1) = transformedCriticalTensor(0,3);
  criticalStressTensor(2,0) = transformedCriticalTensor(0,4);
  criticalStressTensor(0,2) = transformedCriticalTensor(0,4);
  criticalStressTensor(1,2) = transformedCriticalTensor(0,5);
  criticalStressTensor(2,1) = transformedCriticalTensor(0,5);


  EigenSolver<MatrixXd> es(criticalStressTensor);

  std::cout << "The␣eigenvalues␣of␣criticalStressTensor␣are:"
      << endl << es.eigenvalues() << endl;
  std::cout << "The␣matrix␣of␣eigenvectors,␣V,␣is:" << endl
     << es.eigenvectors() << endl << endl;



  std::cout << "Final␣element␣=␣\n" << criticalStressTensor
     << "\n" << std::endl;
  std::cout << "\n" << std::endl;

  if(input == 1){
    return  crticalStressTensorAllAngles;
  }
  else{
    return criticalStressTensor; // and eigenvalues... need
        to combine these into one matrix
  }

 // system("pause");
}

//
// doTransformation does the transformation of a stress
    tensor matrix (1,6) with given angles tetta and phi
// Used for transforming the critical element found and for
    the visualization part of the application
//
```

```
MatrixXd doTransformation(MatrixXd stressTensor, double tetta
    , double phi, int which){

MatrixXd transformedMatrix(1,6);

MatrixXd stressTensor2 (1, 6);

for(int i=0; i<6; i++){
  stressTensor2(0,i) = stressTensor(which, i);

}

double a11 = cos(tetta)*sin(phi);
double a12 = sin(tetta)*sin(phi);
double a13 = cos(phi);
double a21 = -sin(tetta);
double a22 = cos(tetta);
double a23 = 0;
double a31 = -cos(tetta)*cos(phi);
double a32 = -sin(tetta)*cos(phi);
double a33 = sin(phi);

transformedMatrix(0,0) =
  a11*a11*stressTensor2(0,0) +
  a12*a12*stressTensor2(0,1) +
  a13*a13*stressTensor2(0,2) +
  2*a11*a12*stressTensor2(0,3) +
  2*a11*a13*stressTensor2(0,4) +
  2*a13*a12*stressTensor2(0,5);
transformedMatrix(0,1) =
  a21*a21*stressTensor2(0,0) +
  a22*a22*stressTensor2(0,1) +
  a23*a23*stressTensor2(0,2) +
  2*a21*a22*stressTensor2(0,3) +
  2*a21*a23*stressTensor2(0,4) +
  2*a23*a22*stressTensor2(0,5);
transformedMatrix(0,2) =
  a31*a31*stressTensor2(0,0) +
  a32*a32*stressTensor2(0,1) +
  a33*a33*stressTensor2(0,2) +
  2*a31*a32*stressTensor2(0,3) +
  2*a31*a33*stressTensor2(0,4) +
  2*a33*a32*stressTensor2(0,5);;
transformedMatrix(0,3) =
  a11*a21*stressTensor2(0,0) +
  a12*a22*stressTensor2(0,1) +
  a13*a23*stressTensor2(0,2) +
  (a11*a22 + a12*a21)*stressTensor2(0,3) +
  (a13*a21 + a11*a23)*stressTensor2(0,4) +
  (a12*a23 + a13*a22)*stressTensor2(0,5);
transformedMatrix(0,4) =
  a11*a31*stressTensor2(0,0) +
  a12*a32*stressTensor2(0,1) +
  a13*a33*stressTensor2(0,2) +
  (a11*a32 + a12*a31)*stressTensor2(0,3) +
  (a13*a31 + a11*a33)*stressTensor2(0,4) +
  (a32*a32 + a12*a33)*stressTensor2(0,5);;
transformedMatrix(0,5) =
  a21*a31*stressTensor2(0,0) +
  a22*a32*stressTensor2(0,1) +
  a23*a33*stressTensor2(0,2) +
  (a21*a32 + a22*a31)*stressTensor2(0,3) +
  (a23*a31 + a21*a33)*stressTensor2(0,4) +
  (a22*a33 + a23*a32)*stressTensor2(0,5);


  return transformedMatrix;
}



MatrixXd doAllCriticalTransformation(MatrixXd stressTensor,
    MatrixXd anglesMatrix, int which, int numbIntervals){

  MatrixXd transformedStressTensor(numbIntervals*
     numbIntervals,6);

  MatrixXd a11(1, numbIntervals*numbIntervals); // calculate
     the a values
  MatrixXd a12(1, numbIntervals*numbIntervals);
  MatrixXd a13(1, numbIntervals*numbIntervals);
  MatrixXd a21(1, numbIntervals*numbIntervals);
  MatrixXd a22(1, numbIntervals*numbIntervals);
  MatrixXd a23(1, numbIntervals*numbIntervals);
  MatrixXd a31(1, numbIntervals*numbIntervals);
  MatrixXd a32(1, numbIntervals*numbIntervals);
  MatrixXd a33(1, numbIntervals*numbIntervals);

    for(int ia=0; ia<numbIntervals*numbIntervals; ia++){
```

```
a11(0,ia) = cos(anglesMatrix(0,ia))*sin(anglesMatrix(1,
    ia));
a12(0,ia) = sin(anglesMatrix(0,ia))*sin(anglesMatrix(1,
    ia));
a13(0,ia) = cos(anglesMatrix(1,ia));
a21(0,ia) = -sin(anglesMatrix(0,ia));
a22(0,ia) = cos(anglesMatrix(0,ia));
a23(0,ia) = 0;
a31(0,ia) = -cos(anglesMatrix(0,ia))*cos(anglesMatrix
    (1,ia));
a32(0,ia) = -sin(anglesMatrix(0,ia))*cos(anglesMatrix
    (1,ia));
a33(0,ia) = sin(anglesMatrix(1,ia));

  transformedStressTensor(ia,0) =
  a11(0,ia)*a11(0,ia)*stressTensor(which,0) +
  a12(0,ia)*a12(0,ia)*stressTensor(which,1) +
  a13(0,ia)*a13(0,ia)*stressTensor(which,2) +
  2*a11(0,ia)*a12(0,ia)*stressTensor(which,3) +
  2*a11(0,ia)*a13(0,ia)*stressTensor(which,4) +
  2*a13(0,ia)*a12(0,ia)*stressTensor(which,5);
  transformedStressTensor(ia,1) =
  a21(0,ia)*a21(0,ia)*stressTensor(which,0) +
  a22(0,ia)*a22(0,ia)*stressTensor(which,1) +
  a23(0,ia)*a23(0,ia)*stressTensor(which,2) +
  2*a21(0,ia)*a22(0,ia)*stressTensor(which,3) +
  2*a21(0,ia)*a23(0,ia)*stressTensor(which,4) +
  2*a23(0,ia)*a22(0,ia)*stressTensor(which,5);
  transformedStressTensor(ia,2) =
  a31(0,ia)*a31(0,ia)*stressTensor(which,0) +
  a32(0,ia)*a32(0,ia)*stressTensor(which,1) +
  a33(0,ia)*a33(0,ia)*stressTensor(which,2) +
  2*a31(0,ia)*a32(0,ia)*stressTensor(which,3) +
  2*a31(0,ia)*a33(0,ia)*stressTensor(which,4) +
  2*a33(0,ia)*a32(0,ia)*stressTensor(which,5);;
  transformedStressTensor(ia,3) =
  a11(0,ia)*a21(0,ia)*stressTensor(which,0) +
  a12(0,ia)*a22(0,ia)*stressTensor(which,1) +
  a13(0,ia)*a23(0,ia)*stressTensor(which,2) +
  (a11(0,ia)*a22(0,ia) + a12(0,ia)*a21(0,ia))*
     stressTensor(which,3) +
  (a13(0,ia)*a21(0,ia) + a11(0,ia)*a23(0,ia))*
     stressTensor(which,4) +
  (a12(0,ia)*a23(0,ia) + a13(0,ia)*a22(0,ia))*
```

```
           stressTensor(which,5);
        transformedStressTensor(ia,4) =
        a11(0,ia)*a31(0,ia)*stressTensor(which,0) +
        a12(0,ia)*a32(0,ia)*stressTensor(which,1) +
        a13(0,ia)*a33(0,ia)*stressTensor(which,2) +
        (a11(0,ia)*a32(0,ia) + a12(0,ia)*a31(0,ia))*
           stressTensor(which,3) +
        (a13(0,ia)*a31(0,ia) + a11(0,ia)*a33(0,ia))*
           stressTensor(which,4) +
        (a13(0,ia)*a32(0,ia) + a12(0,ia)*a33(0,ia))*
           stressTensor(which,5);;
        transformedStressTensor(ia,5) =
        a21(0,ia)*a31(0,ia)*stressTensor(which,0) +
        a22(0,ia)*a32(0,ia)*stressTensor(which,1) +
        a23(0,ia)*a33(0,ia)*stressTensor(which,2) +
        (a21(0,ia)*a32(0,ia) + a22(0,ia)*a31(0,ia))*
           stressTensor(which,3) +
        (a23(0,ia)*a31(0,ia) + a21(0,ia)*a33(0,ia))*
           stressTensor(which,4) +
        (a22(0,ia)*a33(0,ia) + a23(0,ia)*a32(0,ia))*
           stressTensor(which,5);


    }

    return transformedStressTensor;
}



//
   //////////////////////////////////////////////////////////////////////////////

////////////////////////////////////**Pensjonert Kode
   **////////////////////////////////////
//
   //////////////////////////////////////////////////////////////////////////////

/*


/////
//////// Code for solving a transformation of a matrix with
   angles tetta and phi
////
```

```
// angles for transformation
int tettaDeg = 10;
int phiDeg = 20;

//Degrees to radians
double tetta = tettaDeg*PI/180;
double phi = phiDeg*PI/180;

// Constants for transformation matrix
double a11 = cos(tetta)*sin(phi);
double a12 = sin(tetta)*sin(phi);
double a13 = cos(phi);
double a21 = -sin(tetta);
double a22 = cos(tetta);
double a23 = 0;
double a31 = -cos(tetta)*cos(phi);
double a32 = -sin(tetta)*cos(phi);
double a33 = sin(phi);

std::cout << "a11 = "  << a11 << "\n" << std::endl;
std::cout << "a12 = "  << a12 << "\n" << std::endl;
std::cout << "a13 = "  << a13 << "\n" << std::endl;
std::cout << "a21 = "  << a21 << "\n" << std::endl;
std::cout << "a22 = "  << a22 << "\n" << std::endl;
std::cout << "a23 = "  << a23 << "\n" << std::endl;
std::cout << "a31 = "  << a31 << "\n" << std::endl;
std::cout << "a32 = "  << a32 << "\n" << std::endl;
std::cout << "a33 = "  << a33 << "\n" << std::endl;

//The stress tensor to be transformed

stressTensor(0,0) = 30;
stressTensor(0,1) = 60;
stressTensor(0,2) = 0;
stressTensor(0,3) = 10;
stressTensor(0,4) = 0;
stressTensor(0,5) = 0;


double sigmaXTransf, sigmaYTransf, sigmaZTransf, tauXYTransf,
     tauXZTransf, tauYXTransf;

sigmaXTransf = transformedStressTensor(0,0);
sigmaYTransf = transformedStressTensor(0,1);
```

```
sigmaZTransf = transformedStressTensor(0,2);
tauXYTransf = transformedStressTensor(0,3);
tauXZTransf = transformedStressTensor(0,4);
tauYZTransf = transformedStressTensor(0,5);



std::cout << "\nThe transformed stress tensor: " << std::endl
   ;
std::cout << transformedStressTensor << std::endl;

*/


/*
  // Just to make sure the format is correct... : printing to
       a file example.txt
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
  for(int wR=0; wR<numbRows; wR++){
    for(int wC=0;wC<numbCols;wC++){
      myfile << inputData(wR,wC);
      myfile << "\n";
    }
        myfile << "newline \n";
  }

    myfile.close();
  }

// double numbStories = numberoflines/numberOfLinesAfter;
   TESINGTESING
// std::cout << "numberoflines = " << numberoflines << "\n"
   << std::endl;
// std::cout << "numberOfLinesAfter = " << numberOfLinesAfter
    << "\n" << std::endl;
// std::cout << "numbStories should be 2 = " << numbStories
   << "\n" << std::endl;
*/
```

Listing 3: main.cpp file

```
#define vtkRenderingCore_AUTOINIT 4(vtkInteractionStyle,
   vtkRenderingFreeType,vtkRenderingFreeTypeOpenGL,
   vtkRenderingOpenGL)
#define vtkRenderingVolume_AUTOINIT 1(
   vtkRenderingVolumeOpenGL)

#include "cpa_findley.h"
#include "calculation.h"

// This is included here because it is forward declared in
// RenderWindowUISingleInheritance.h
#include "ui_render_enklere_navn.h"

#include <vtkArrowSource.h>
#include <stdlib.h>
#include <vtkPolyDataMapper.h>
#include <vtkAxesActor.h>
#include <vtkPolyData.h>
#include <vtkDataSet.h>
#include <vtkDataArray.h>
#include <vtkPoints.h>
#include <vtkPolyData.h>
#include <vtkPointData.h>
#include <vtkDoubleArray.h>
#include <vtkFloatArray.h>
#include <vtkTextActor.h>
#include <vtkTextProperty.h>
#include <vtkDataSetAttributes.h>
#include <vtkDataSetMapper.h>
#include <vtkPolyDataAlgorithm.h>
#include <vtkRenderer.h>
#include <vtkProperty.h>
#include <vtkPoints.h>
#include <vtkRenderWindow.h>
#include <vtkSphereSource.h>
#include <vtkCellArray.h>
#include <vtkSmartPointer.h>
#include <vtkUnsignedCharArray.h>
#include <vtkGlyph3D.h>
#include <vtkFloatArray.h>
#include <vtkOrientationMarkerWidget.h>
#include <vtkTransform.h>
#include <vtkCellLocator.h>
#include <vtkKdTreePointLocator.h>
```

```
#include <vtkLineSource.h>
#include <stdio.h>
#include <string>
#include <sstream>
#include <vtkPlaneSource.h>
#include <vtkCubeSource.h>
#include <vtkScalarBarActor.h>
#include <vtkExtractEdges.h>
#include <vtkLookupTable.h>
#include <Eigen/Dense>

using Eigen::MatrixXf;
using namespace std;

std::string convertInt(int number);

// Constructor
Render_enklere_navn::Render_enklere_navn()
{

        MatrixXd stressTensorsAll;

        stressTensorsAll = doCalculation(1);

        double min = 0.0;
        double max = 0.0;
        double tempMax = 0.0;
        double tempMin = 0.0;

        for(int i=0; i<stressTensorsAll.innerSize(); i++){
                if(i==0){
                        min = stressTensorsAll(i,0);
                        max = stressTensorsAll(i,0);
                }
                if(stressTensorsAll(i,0) > tempMax){
                        max = stressTensorsAll(i,0);
                        tempMax = max;
                }
                if(stressTensorsAll(i,0) < tempMin){
                        min = stressTensorsAll(i,0);
                        tempMin = min;
                }
        }
```

```
      ofstream myfile ("example.txt");
if (myfile.is_open())
{
        for(int wR=0; wR<stressTensorsAll.innerSize(); wR
           ++){
                myfile << stressTensorsAll(wR,0);
                      myfile << "\n";
              }

      myfile << "min:␣" << min << "\n";
      myfile << "max:␣" << max << "\n";
      myfile.close();
}

      this->ui = new Ui_Render_enklere_navn;
      this->ui->setupUi(this);


  // Create a sphere
      vtkSmartPointer<vtkSphereSource> sphere =
         vtkSmartPointer<vtkSphereSource>::New();
      sphere->SetCenter(0,0,0);
      sphere->SetThetaResolution(38);
      sphere->SetPhiResolution(38);
      sphere->SetRadius(1);
      sphere->Update();




      // Create scalar data to associate with the vertices
         of the sphere
      int numPts = sphere->GetOutput()->GetPoints()->
         GetNumberOfPoints();


      MatrixXf sigmaXvalues(1, numPts);
      for(int j=0; j<numPts; j++){
        sigmaXvalues(0, j) = stressTensorsAll(j,0);

      }

      vtkSmartPointer<vtkFloatArray> scalars =
         vtkSmartPointer<vtkFloatArray>::New();
      scalars->SetNumberOfValues( numPts );
```

```
        for( int i = 0; i < numPts; ++i )
        {
                scalars->SetValue(i,sigmaXvalues(0,i));
    }
        vtkSmartPointer<vtkPolyData> poly = vtkSmartPointer<
           vtkPolyData>::New();
        poly->DeepCopy(sphere->GetOutput());
        poly->GetPointData()->SetScalars(scalars);

        vtkSmartPointer<vtkPolyDataMapper> mapper =
           vtkSmartPointer<vtkPolyDataMapper>::New();
#if VTK_MAJOR_VERSION <= 5
  mapper->SetInput(poly);
#else
  mapper->SetInputData(poly);
#endif
         mapper->ScalarVisibilityOn();
         mapper->SetScalarRange(min,max);
        mapper->SetScalarModeToUsePointData();
        mapper->SetColorModeToMapScalars();

        vtkSmartPointer<vtkActor> actor = vtkSmartPointer<
           vtkActor>::New();
        actor->SetMapper(mapper);

        vtkSmartPointer<vtkScalarBarActor> scalarBar =
           vtkSmartPointer<vtkScalarBarActor>::New();
        scalarBar->SetLookupTable(mapper->GetLookupTable());
        scalarBar->SetTitle("lut");
        scalarBar->SetNumberOfLabels(4);

        // Create a lookup table to share between the mapper
           and the scalarbar
        vtkSmartPointer<vtkLookupTable> hueLut =
           vtkSmartPointer<vtkLookupTable>::New();
        hueLut->SetTableRange (min, max);
        hueLut->SetHueRange (0, 1);
        hueLut->SetSaturationRange (1, 1);
        hueLut->SetValueRange (1, 1);
        hueLut->Build();

        mapper->SetLookupTable( hueLut );
        scalarBar->SetLookupTable( hueLut );
        vtkSmartPointer<vtkCubeSource> box = vtkSmartPointer<
           vtkCubeSource>::New();
        box->SetXLength(2);
        box->SetYLength(2);
        box->SetZLength(2);

    vtkSmartPointer<vtkExtractEdges> edges = vtkSmartPointer<
       vtkExtractEdges>::New();
    edges->SetInputConnection(box->GetOutputPort());

    vtkSmartPointer<vtkPolyDataMapper> cubeMapper =
       vtkSmartPointer<vtkPolyDataMapper>::New();
    cubeMapper->SetInputConnection(edges->GetOutputPort());

    vtkSmartPointer<vtkActor> cubeEdgeActor = vtkSmartPointer
       <vtkActor>::New();
    cubeEdgeActor->SetMapper(cubeMapper);
    cubeEdgeActor->GetProperty()->SetColor(0,0,0);
    cubeEdgeActor->GetProperty()->SetAmbient(1);


        vtkSmartPointer<vtkPlaneSource> planeSource =
           vtkSmartPointer<vtkPlaneSource>::New();
        planeSource-> SetNormal(-tan(1.5708),-tan(2.094),1.0)
           ;
        //planeSource-> SetOrigin(3.0, 0.0, 0.0);
        planeSource-> SetPoint1(3.0, 0.0, 0.0 );
        planeSource-> SetPoint2(0.0, 3.0, 0.0 );
        planeSource->SetCenter(0,0,0);
        planeSource-> SetXResolution(10);
        planeSource-> SetYResolution(10);
        planeSource-> Update();

        // Create a mapper and actor.
        vtkSmartPointer<vtkPolyDataMapper> planeSourceMapper
           = vtkSmartPointer<vtkPolyDataMapper>::New();
        planeSourceMapper->SetInputData(planeSource->
           GetOutput());

        vtkSmartPointer<vtkActor> planeSourceActor =
           vtkSmartPointer<vtkActor>::New();
        planeSourceActor->SetMapper(planeSourceMapper);
        planeSourceActor->GetProperty()->SetColor(0.5, 0.3,
           0.5);
        planeSourceActor->GetProperty()->SetOpacity(0.5);
        vtkSmartPointer<vtkTransform> transform =
           vtkSmartPointer<vtkTransform>::New();
        transform->Translate(-1.5, -1.5, -1.5);

        vtkSmartPointer<vtkAxesActor> axes = vtkSmartPointer<
           vtkAxesActor>::New();

        // The axes are positioned with a user transform
        axes->SetUserTransform(transform);
        axes->SetTotalLength(0.5, 0.5, 0.5);

        //Number of points on the sphere
        int numberofpoints = (int) sphere->GetOutput()->
           GetNumberOfPoints();

        std::string numb = convertInt(min); //
           stressTensorsAll.outerSize()
        const char * c = numb.c_str();

        vtkSmartPointer<vtkTextActor> textActor =
           vtkSmartPointer<vtkTextActor>::New();
        textActor->GetTextProperty()->SetFontSize ( 24 );
        textActor->SetPosition2 ( 5, 30 );
        textActor->SetInput(c);
        textActor->GetTextProperty()->SetColor ( 1.0,0.0,0.0
           );

        // VTK Renderer
        vtkSmartPointer<vtkRenderer> renderer =
           vtkSmartPointer<vtkRenderer>::New();
        /*
    // Setup the text and add it to the window
        vtkSmartPointer<vtkTextActor> textActor =
           vtkSmartPointer<vtkTextActor>::New();
        textActor->GetTextProperty()->SetFontSize ( 24 );
        textActor->SetPosition2 ( 5, 30 );
        textActor->SetInput(c);
        textActor->GetTextProperty()->SetColor ( 1.0,0.0,0.0
           );

    // Setup the text and add it to the window 2
        vtkSmartPointer<vtkTextActor> textActor2 =
           vtkSmartPointer<vtkTextActor>::New();
        textActor2->GetTextProperty()->SetFontSize ( 24 );
```

```
        textActor2->SetPosition(5, 20);
        textActor2->SetInput(c3);
        textActor2->GetTextProperty()->SetColor ( 1.0,0.0,0.0
            );


        */

        // VTK/Qt wedded
        this->ui->qvtkWidget->GetRenderWindow()->AddRenderer(
           renderer);

        renderer->GradientBackgroundOn();
        renderer->SetBackground(1,1,1);
        renderer->SetBackground2(0,0,0);
        renderer->AddActor(actor);
        renderer->AddActor(cubeEdgeActor);
        renderer->AddActor(axes);
        renderer->AddActor(planeSourceActor);
        renderer->AddActor2D(scalarBar);
        renderer->AddActor2D(textActor);


        // Set up action signals and slots
        connect(this->ui->actionExit, SIGNAL(triggered()),
           this, SLOT(slotExit()));

}

void Render_enklere_navn::slotExit()
{
  qApp->exit();
}

std::string convertInt(int number)
{
   std::stringstream ss;//create a stringstream
   ss << number;//add number to the stream
   return ss.str();//return a string with the contents of the
       stream
}



  /*
```

```
      **********************************************************
      */
 /****************PENSJONERTE DELER
     **************************/
/************************************************************
    */


  /*vtkSmartPointer<vtkCellLocator> cellLocator =
     vtkSmartPointer<vtkCellLocator>::New();
  cellLocator->SetDataSet(sphereSource->GetOutput());
  cellLocator->BuildLocator();

  double testPoint[3] = {2.0, 2.0, 0.0};

  //Find the closest points to TestPoint
  double closestPoint[3];//the coordinates of the closest
     point will be returned here
  double closestPointDist2; //the squared distance to the
     closest point will be returned here
  vtkIdType cellId; //the cell id of the cell containing the
     closest point will be returned here
  int subId; //this is rarely used (in triangle strips only,
     I believe)
  cellLocator->FindClosestPoint(testPoint, closestPoint,
     cellId, subId, closestPointDist2);

  int cel = (int) cellId;
  std::string numbCellId = convertInt(cel);
   const char * c3 = numbCellId.c_str();
   */
```

B References

[1] Qt Digia. Qt digia qt framework c++ modular library. http://qt.digia.com/Product/Qt-Framework/modular-library/, January 2014.

[2] Thaulow C. & Valberg H. *TMM4140 Plastisk deformasjon og brudd.* Norges Teknisk Naturvitenskapelige Universitet, Trondheim, Norway, 2012.

[3] Lee Y-L. & Barkey M.E. & Kang H-T. *Metall Fatigue Analysis Handbook: Practical problem-solving techniques for computer aided engineering.* Elsivier, -, 2011.

[4] kelsrud Riiser S. *Fatigue analysis and testing of SX-22 couplers.* Norges Teknisk Naturvitenskapelige Universitet, -, 2013.

[5] Dowling N.E. *Mechanical Behavior of Materials (Third Edition).* Pearson Education International, -, 2007.

[6] University of Waterloo. Fatigue stress analysis lab. https://uwaterloo.ca/fatigue-stress-analysis-lab/research-areas/energy-based-fatigue-life-model-proportional-and., January 2014.

[7] R.H.M.B.Y.-L.Lee. *Fatigue testing and analysis: Theory and Practice. Volume 1.* Elsivier, -, 2005.

[8] G.B. Socie D.F. & Marquis. *Multiaxial Fatigue.* SAE International, -, 2000.

[9] Solidworks. S-n curve. http://help.solidworks.com/2012/English/SolidWorks/cworks/S-NCurve.htm, January 2014.

[10] Visualization Toolkit. About visualization toolkit. http://www.vtk.org/VTK/project/about.html, January 2014.

[11] Wood W. *Recent observations on fatigue fracture metals. Volume 1.* 1958.

[12] Findley W.N. *A Theory for the effect of mean stress on fatigue of metals under combined torsion and axial load or bending.* Journal of Engineering for Industry, -, 1959.