FME HighEFF

Centre for an Energy Efficient and Competitive Industry for the Future

Deliverable D2.1_2018.02
From theoretical analysis to 3D-printed HX models

Delivery date: 2018-12-17

Organisation name of lead beneficiary for this deliverable:

SINTEF ER

HighEFF- Centre for an Energy Efficient and Competitive Industry for the Future is one of Norway's Centre for Environment-friendly Energy Research (FME). Project co-funded by the Research Council of Norway and Industry partners. Host institution is SINTEF Energi AS.		
Dissemination Level		
PU	Public	X
RE	Restricted to a group specified by the consortium	

Deliverable number:	D2.1_2012.02
ISBN number:	
Deliverable title:	From theoretical analysis to 3D-printed HX models
Work package:	WP2.1 Heat Exchanges
Deliverable type:	MEMO
Lead participant:	SINTEF ER

Quality Assurance, status of deliverable		
Action	Performed by	Date
Verified (WP leader)	Geir Skaugen	19.12.2018
Reviewed (RA leader)	Armin Hafner	20.12.2018
Approved (dependent on nature of deliverable)*)		

*) *The quality assurance and approval of HighEFF deliverables and publications have to follow the established procedure. The procedure can be found in the HighEFF eRoom in the folder "Administrative > Procedures".*

Authors		
Author(s) Name	Organisation	E-mail address
Geir Skaugen	SINTEF Energy Research	Geir.skaugen@sintef.no
Andreas Bolstad	SINTEF Energy Research – Summer intern	

Abstract

The work on 3D-printing was continued from 2017 as summer internship work. The result from this work was to streamline the process form heat exchanger geometry optimisation to generate control files for on-screen visualisation and 3D-editing and finally 3D-printing.

The main outcome from this was to do the visualization scripting using Python and to couple this result to Paraview, a well known visualisation tookit often used with CFD.

The summer internship report is included as is.

Table of Contents

Visualisering og 3D-printing av varmevekslermodeller

Andreas Bolstad

16. august 2018

Innhold

1. Introduksjon

Sommerprosjektet er en videreføring av fjorårets sommerprosjekt «Fra data til fysisk modell». Begge prosjektene omhandlet visualisering og 3D-printing av varmevekslermodeller. Formålet er blant annet å kunne se hvordan varmvekslerne ser ut og om kompliserte geometrier er mulige å produsere ved hjelp av 3D-printing. Utgangspunktet for prosjektet er noen modeller laget i «Visualization Toolkit» (VTK) med Tcl som programmeringsspråk, hvor geometrien er beskrevet av parametre som beregnes i simuleringer man finner i kodebasene FlexHX og FlexCS.

Simon Høgåsen, fjorårets sommerforsker, viste at 3D-printing av modeller ikke er vanskelig i teorien. Dessverre var systemet for visualisering med VTK i FlexHX vanskelig å sette seg inn i og tungt å bruke. Jeg har jobbet med å forenkle og forbedre systemet framfor å utvikle nye modeller og mer funksjonalitet. Resultatet av arbeidet er bedre støttede verktøy og mye mer oversiktlige og brukervennlige systemer. Visualiseringsskript ble oversatt fra Tcl til Python. Installasjon av visualiseringsverktøyet VTK ble forenklet og grundig dokumentert. Det hjemmesnekrede redigeringsprogrammet for stl-filer ble erstattet av Paraview. Modulen for å eksportere 3D-figurer i VTK til stl-format ble endret slik at enorme figurer også kan eksporteres (men kun til Paraview). Muligheten for å kvitte seg helt med VTK til fordel for Paraview ble også vurdert.

2. Resultat

2.1 Installere VTK

Installasjon av VTK (Visualisation Toolkit) med msys, med støtte for Tcl og Python, kan være vanskelig og tidkrevende. VTK er et stort verktøy, men man kan kompilere en egen versjon som kun har funksjonaliteten man trenger. Dessverre kan kompilering ta mange timer. Uten god forståelse for CMAKE og VTK er det utfordrende å finne instillinger som ikke fører til kompileringskrasj og som gir rett funksjonalitet.

Ettersom Tcl-kode har blitt oversatt til Python har installasjon blitt enklere. Pacman i msys leverer vtk forhåndskompilert med Python-wrapping og all funksjonalitet nåværende modeller benytter. En full installasjonsguide finner du i seksjon A.1. Ettersom Tcl er deprecated", altså ikke støttet i framtidige versjoner av VTK, har jeg valgt å ikke lage noen guide for hvordan man installerer VTK med Tcl-wrapping, da dette uansett ikke kan brukes i framtiden. Hvis det er et behov for å sjekke gamle tcl-modeller kan det være greit å installere en gammel versjon av VTK som er forhåndskompilert med tcl-wrapping.

2.2 Oversette Tcl til Python

Skript i Python og Tcl fungerer ganske likt. Å oversette modellene fra Tcl til Python var derfor ikke spesielt vanskelig. Det krevde likevel noen dager da de fleste kodelinjene ikke lot seg oversette direkte. I tillegg har flere modeller blitt forenklet, forkortet og utnyttet innebygd VTK-funksjonalitet bedre.

Det er en flere grunner til at Tcl fases ut til fordel for Python. Python har mye mer lettlest syntaks, mye større funksjonalitet, og det finnes alle slags nyttige moduler/bibliotek lett tilgjengelig. Jeg vil anbefale på det sterkeste å bli kjent med det svært kraftige numeriske biblioteket numpy. Algoritmene er skrevet i Fortran, så det kjører svært raskt. Hvordan man kan bruke numpy med VTK finnes det flere guider for på internett.

For å forstå modellenes Python-kode skal det være nok med kjennskap til grunnleggende python, VTK i Python-format og tkinter. Det er en stor fordel å være godt kjent med «dictionaries», da disse er mye brukt. Kode med bruk av tkinter er dessverre vanskelig å lese, men man trenger heldigvis ikke forstå

alt for å bruke det. Modellene er skrevet slik at de fungerer i både python 2 og 3. Det er noen få men viktige forskjeller mellom de to versjonene som er greit å være kjent med. Heltallsdivisjon, xrange, tkinter vs Tkinter, for å nevne noe. VTK og Paraview bruker Python 2 som standard foreløpig. Begge versjoner støttes, men det jobbes mot å kun bruke Python 3 en gang i framtiden.

2.3 Bygge egen GUI

En GUI, ≪Graphical User Interface≫, laget med tkinter ble utviklet for modellen vtkhx. Målet var å se om det ble lettere å jobbe med de tyngste modellene hvis variabler og figurer kan endres på interaktivt. Det tok undertegnede en god del tid å lære seg tkinter skikkelig, men så snart man har kontroll på funksjonalitet og syntaks går det kjapt å utvikle en GUI. GUI-en som ble laget for vtkhx gjør det mulig å skru av og på enkelte figurer, redigere oppløsning og antall finner, og lagre modellen til stl slik du ser den. Det viktigste for å oppnå slik funksjonalitet er at hver figur er én ≪actor≫ og at pipeline (se seksjon 3.1) er intakt slik at data kan modifiseres. Generelt er dette gode retningslinjer for å lage fleksibel kodei VTK.

2.4 Legge til visualisering i FlexHX/FlexCS

Simuleringene i FlexHX/FlexCS har fått automatisert generering av visualiseringsskript som enkelt kjøres med kommandoen vtkpython ¡navn på skript¿. Resultatet er forenklet arbeidsflyt for testing og visualisering som kan kjøres uten spesiell kunnskap om VTK. Nye Python-modeller ligger i biblioteksmappa, /libs/, i FlexHX og er koblet opp til noen simuleringer i både FlexHX og FlexCS. Hvordan modellene flettes inn i FlexHX er beskrevet i seksjon A.2. Forhåpentligvis vil dette gjøre det enkelt å legge inn visualisering i alle simuleringer, og mulig å bruke for de som ikke er utviklere av visualiseringsmodellene. Systemet er ganske stort og litt uoversiktlig, så det kan være en ide dokumentere/systematisere/rydde bedre framover.

2.5 Eksportere figurer og klipping

For å kunne 3D-printe figurer må de eksporteres i et universelt dataformat som andre programmer støtter. Forrige sommerforsker skrev kode for eksport i stl-format. I lagringsfunksjonen velger man lag, slik at figurene (actor-ene) som lagres i samme lag blir skrevet til samme stl-fil. Hvert lag blir en egen fil.

Iløpet av dette prosjektet har koden blitt oversatt til python, men med utvidet funksjonalitet for lagring av modeller som benytter glyphing (se seksjon 3.2 om glyphing). I koden kan man velge mellom 2 alternativ. Det ene alternativet er vanlig lagring hvor hele figuren skrives til en stl-fil. Det andre alternativet er å lagre glyph-punktene i en egen vtk-fil og kun ett glyph-element til stl. Ved

store figurer må man lagre på sistnevnte måte, ellers blir resultatet enorme og ubrukelige stl-filer. Merk at vtk-format kun kan åpnes i VTK eller Paraview.

Med store og kompliserte modeller er det som regel mest aktuelt å 3D-printe et begrenset utsnitt. Dette krever klippefunksjonalitet som også vanntetter modellen etter klipping. Forrige sommerstudent lagde et eget program for klipping. Problemet er at det mangler funksjonalitet for glyphing, er lite stabilt og må manuelt oppdateres for framtidige versjoner av VTK. Ettersom Paraview har mye kraftigere funksjonalitet, er relativt lett å bruke og enkelt å installere var det logisk å forkaste det hjemmelagde systemet. Guiden i seksjon A.3 viser hvordan man åpner de eksporterte filene i Paraview og hvordan man klipper/behandler figurene for å lage et utsnitt som kan 3D-printes.

2.6 3D-printing

Det ble printet en 3D-print av «pfhelayers» som er vist i figur 2.1 for å vise at systemet med klipping i Paraview fungerer. Et materiale kalt breakaway ble såvidt testet ut mot slutten av prosjektet. Det er mye lettere å løsne enn vanlig materiale, slik at mye av problematikken med overheng kan kanskje løses. Ellers har ikke 3D-printing blitt nevneverdig utforsket iløpet av prosjektet. Fjorårets erfaringer siteres i seksjon 3.4 for å beskrive status og forbedringspotensial rundt 3D-printing.

Figur 2.1: 3D-printet utsnitt fra modellen pfhelayers.

3. Diskusjon

3.1 Pipeline

I VTK har du såkalte pipelines. Dette er et konsept det lønner seg å ha god forståelse for når man jobber med VTK. En pipeline kobler sammen ulike lag, fra rene punktdata opp til det som vises på skjerm. En pipeline i VTK ser slik ut: `punkter->geometri->filter->mapper->actor`

Hvordan man kobler pipeline i kode er ofte en utfordring. Hvert lag er koblet til hverandre ved å lage koblinger, for eksempel minMapper.setInputConnection(≪argument≫), der argumentet kan være mittFilter.GetOutputPort() eller mittFilter.GetInputConnection(1, 0). Det blir fort fristende å finne mer lettleste måter å programmere på, for eksemepl minMapper.SetInputData(≪data≫). Denne funksjonen vil jeg imidlertid sterkt fraråde noen å bruke (hvis man ikke vet nøyaktig hva man gjør) ettersom denne funksjonen ikke kobler dataene via en pipeline. Det vil si at datasettet mapperen tar inn ikke kan endres i ettertid. Mangelen på sammenkobling/≪referencing≫ gjør koden mindre fleksibel. Ved programmering i VTK lønner det seg altså å skaffe seg grunnleggende forståelse for pipleline for å spare tid på sikt.

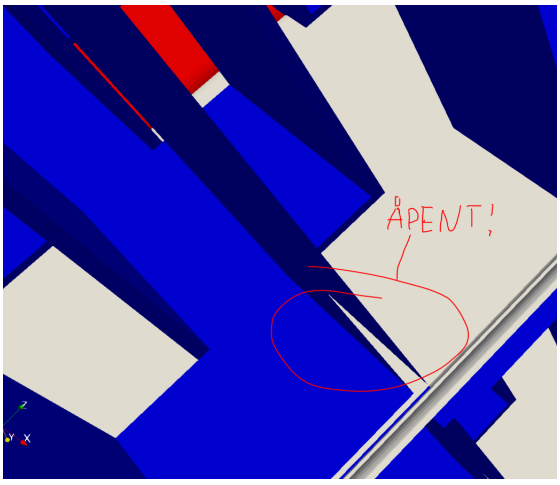
3.2 Glyphing og god bruk av filter

For å kunne bruke VTK effektivt bør man kjenne til hvilke filter som fins. Filter hjelper med redigere objekter effektivt, for eksempel omdanne en linje til et rør med 3 korte kodelinjer. Det er fort gjort å streve lenge med å lage objekter manuelt selv om det allerede finnes et filter som gjør jobben. Ikke gjør det.

Det viktigste filteret for store visualiseringer er glyphing. Uten glyphing vil visualiseringer med mange punkter bli fryktelig trege eller ubrukelige. Et glyph-filter har 2 inputs. En kildefigur, f.eks. en kule, og et sett med punkter. Output blir da en figur med en kopi av kulen i hvert spesifiserte punkt. Glyphing er altså kopiering av figurer. Helst bør du bruke vtkGlyph3DMapper(), altså en mapper og ikke filter. Glyph-filteret er mye tregere enn mapperen (bruker CPU i stedet for GPU). Filteret og mapperen fungerer nesten helt likt ellers.

Selvlagd geometri med polydata kan gi figurer som ikke vanntett. Dette kan skyldes at en flate mangler, som kan skje hvis punkter ikke er koblet riktig sammen. Eksempel på et objekt som ikke er vanntett ser du i figur 3.1. Lukket

klipping og 3D-printing krever vanntette objekter. 3D-builder eller tilsvarende programmer kan vanntette objektene for deg, men det beste er å lage vanntette modeller i utgangspunktet. Det burde gå fint med polydata så lenge man kobler sammen punktene på rett måte og sjekker vanntetthet med klipping.



Figur 3.1: Visualisering av pfhelayers etter klipping uten lukking. Resultatet er en «åpen», ikke vanntett, figur.

3.3 Paraview kan og burde erstatte VTK

Paraview tilfører mye nyttig funksjonalitet som VTK ikke har. Titalls timer av prosjekttid har allerede gått til å utvikle funksjonalitet i VTK som er innebygd i Paraview. Argumentene for å skifte til Paraview er mange.

En utfordring med å skifte verktøy er at eksisterende VTK-kode må skrives om til Paraview-kode. I teorien burde ikke denne overgangen være stor, ettersom Paraview er bygd på VTK. Det vil bli færre linjer kode, ettersom fokuset blir utelukkende på modellene og ikke hvordan man får dem opp på skjerm.

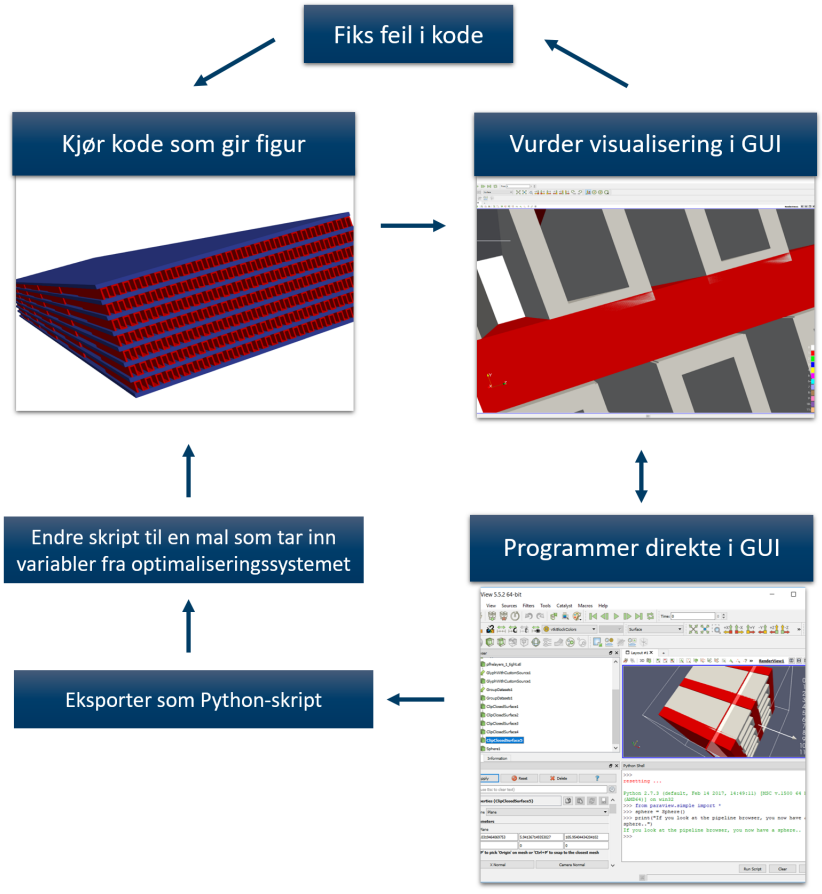
I tillegg til fordelene som beskrives i figur 3.2 er det verdt å nevne at Paraview er betydelig lettere å installere enn VTK, og det er ikke noen vanskeligheter med «PATH» eller konflikter med andre program, for eksempel diverse python-installasjoner. Blir «FPS» for lav (oppdateringsfrekvens til visualisering) på grunn av tunge modeller kan den miste interaktiviteten. I vtkhx ble det forsøkt å innføre LODactorer som reduserer detaljer for å øke «FPS» mens man forflytter bildet. I Paraview genereres LODactorer automatisk hvis «FPS» blir for lav. Dette er bare noe av det Paraview fikser for deg. Over tid vil det altså lønne seg å gjøre den nødvendig innsatsen som trengs for å erstatte VTK med Paraview.

Nå kan "alle" lage visualiseringsskript

I mitt arbeid har jeg vært nødt til å benytte et veletablert, men relativt lavnivå (tungt å bruke) visualiseringsverktøy. Jeg har jobbet for at nyere og bedre verktøy kan benyttes i stedet. Det vil spare tid og gi bedre resultater. Her er en oppsummering av fordeler med det "høynivå" visualiseringsverktøyet Paraview sammenlignet med det "lavnivå" VTK som jeg bruker nå:

- Brukervennlig og kraftig GUI, Graphical User Interface
- Enkel grafisk programmering som effektiviserer testing
- Mer intuitiv syntaks og færre linjer kode å skrive
- Vanntett klippefunksjonalitet som er perfekt for 3D-printing
- Server-rendering, innebygd Python-skall, scene-eksportering og mye mer…

Det tar lite tid å komme i gang. Så hvis du har interesse for å visualisere dine data ville jeg prøvd Paraview, f.eks. med denne arbeidsflyten:



Figur 3.2: Utdrag fra poster til sommerforskerkonferanse. Omhandler fordeler med Paraview mot ren VTK.

3.4 Fjorårets erfaringer med 3D-printing

Denne seksjonen består hovedsakelig av et utdrag fra fjorårets rapport, «Fra data - til fysisk modell» av Simon Høgåsen. Utdraget er en god beskrivelse av hvordan 3D-printing fungerer og hva man bør jobbe med framover:

> 3D-skrivere for plast leser vanligvis et filformat som kalles «gcode» som har litt ulike implementasjoner avhengig av skriveren. Kort beskrevet er denne koden et sett med instruksjoner som forteller hvor skriverhodet skal bevege seg, hvilke temperaturer som skal settes og når det skal sendes ut plast. For å komme fra en 3D-modell til gcode, bruker man en «slicer». Det er et program som deler opp modellen i tynne lag og finner ut hvordan skriverhodet skal bevege seg for å legge plast på de rette stedene. Når lagene skrives ut på hverandre, blir resultatet en 3D-modell.
>
> 3D-skriveren som er brukt i prosjektet er en Ultimaker 3, og den anbefalte «slicer»-en til denne skriveren heter Cura. Programmet importerer 3D-modeller i et lite utvalg formater og produserer gcode-filer som senere kan leses av skriveren. Programmet har mulighet for direkte oppkobling mot skriveren over et nettverk, eller man kan overføre filene med en USB-minnepinne. Siden programmet utvikles av produsenten av 3D-skriveren, gir standardinnstillingene gode resultater. Det hadde likevel vært nyttig å sammenligne resultatet fra andre «slicer»-e. Cura fungerer godt dersom man har en modell som kan importeres og deretter skrives ut direkte, og gir god kontroll over innstillingene til skriveren. Programmet kan sette sammen flere komponenter til en modell, men egner seg ikke like godt til mer sammensatte modeller. Hvis flere komponenter skal importeres, lønner det seg at de allerede er skalert og beskrevet relativt til et felles origo. Bedre valgmuligheter for å generere støttestrukturer vil også være ønskelig, og er tilgjengelig i andre, tilsvarende programmer, men ingen av alternativene har blitt testet i dette arbeidet.
>
> 3D-modeller bør være sammenhengende og ha lite nok overheng for å gi gode resultater. Cura forventer lukkede flater med definert innside og utside, og genererer feil gcode hvis modellen har flater som ikke er det. Ved å sette innstillingen «Surface mode» til «Surface» eller «Both», og å velge «Spiralize outer surface», kan modeller som bare består av enkeltflater skrives ut ved at skriveren følger disse flatene. Ulike kombinasjoner av disse to innstillingene gir litt ulike resultater. Overheng blir generelt bedre med lavere skrivehastighet, men uten støttestrukturer er overheng en viktig begrensning. I genererte modeller som ikke er veldig presise, hadde det vært gunstig med bedre, automatisk oppretting. En vurdering av verktøy som kan gjøre dette kan være interessant.

4. Konklusjon

Det viktigste for å lage gode visualiseringer effektivt er å ikke gjenoppfinne hjulet. Sommerprosjektet har dermed hatt som fokus å teste ut forskjellige programmer for å gjøre utviklingsprosessen enklere, og dermed raskere. Bedre støttet programmeringspråk, Python, bidrar også til dette. For videre forenkling er neste steg å innføre Paraview. Dette kraftige programmet har funksjonalitet som dekker alle steg fra data til 3D-printbar modell. Slik det er nå skal systemet være forenklet nok til at alle på prosjektet kan benytte seg av visualiseringsmodellene. Med Paraview og videre forenkling blir terskelen for å lage nye modeller forhåpentligvis så lav at flere prosjektmedlemmer kan lage modellene de vil ha.

En naturlig videreføring av prosjektet vil være å...

- sette seg grundig inn i Paraview.
- gjøre vtkhx om til en komplett modell.
- rydde i FlexHX (erstatte tcl-filer, dokumentere filsystem, etc.).
- lage nye, mer komplekse modeller.
- videre utforske muligheter rundt 3D-printing.

A. Selvlagde guider

A.1 Installere VTK med Python

```
# Installing VTK with msys2 and mingw-w64 for windows 64-bit
Total installation time: 30-90 min (a good internet connection is
    useful)

Install msys2 with mingw-w64 at root
Download site and installation procedure can be found at:
https://www.msys2.org/

Update all packages like the installation manual says..

Installation of vtk and python2 with pacman:

Msys packages:
a.  pacman -S make
b.  pacman -S diffutils
c.  pacman -S vim (text editor which includes xxd)

##############################
#######  BUG FIX  ############
##############################
PROBLEM: tcl/tk version 8.6.8 seems to be broken.
NOTE: Do this right after installing Mingw package a) (the mingw
    toolchain), and before package b)

Download older version, e.g. from
https://sourceforge.net/projects/msys2/files/REPOS/MINGW/x86_64/
or get the files from someone you know have them.
I installed tcl 8.6.7, with filename:
mingw-w64-x86_64-tcl-8.6.7-1-any.pkg.tar.xz
and tk 8.6.7, with filename:
mingw-w64-x86_64-tk-8.6.7-1-any.pkg.tar.xz
(downloaded from sourceforge)

Run "pacboy -R -c tcl:x" to remove current version of tcl and tk
Install both tcl and tk with "pacman -U <pkg_location>/<pkg_name>")
You can (and should?) reinstall gdb afterwards (includes python3).
pacboy -S gdb:x
##############################

Mingw packages:
```

```
a.  pacboy -S toolchain:x (select all as default)
b.  pacboy -S dlfcn:x
c.  pacboy -S gsl:x
d.  pacboy -S openblas:x
e.  pacboy -S libmariadbclient:x
f.  pacboy -S postgresql:x (this should include python2.7)
g.  pacboy -S vtk:x (takes alot of time and space to install!!!

Note to self: Can qt5 be skipped?

After installation change line 34 in vtkLoadPythonTkWidgets.py to
"elif os.name == 'posix' or sys.platform == 'win32':"
For me the file was located inside this folder:
C:\msys64\mingw64\lib\python2.7\site-packages\vtk\tk\

In your home directory in msys (C:msys64/home/<user_name>) you'll have
    a hidden file, .bashrc
Open .bashrc in a text editor and add the following to the end of the
    file:

#Start
alias vtkpython=/mingw64/bin/vtkpython

export PYTHONHOME=/mingw64/lib/python2.7
export PYTHONPATH="$PYTHONHOME:$PYTHONHOME/site-packages:$PYTHONHOME/
    lib-dynload:${PYTHONPATH}:$PYTHONHOME/lib-tk:$PYTHONHOME/site-
    packages/vtk"
export PATH="${PYTHONPATH}:/mingw64/bin:/mingw64/lib:${PATH}"
#End

Note that other python dependent packages, e.g. mercurial, might not
    work with the new pythonpath.
Consider creating a script which runs specifically vtkpython with these
     paths.


####### TESTING ##############

Open a new mingw64 shell and try to run vtkpython.
This should open a python2.7 shell.

Write the following:
import vtk
import Tkinter
from vtk.tk.vtkTkRenderWindowInteractor import *

If no errors appeared your installation should work!

By adding vtk libs to path, a pure python shell should be able to run
    vtk as well


####### EXTRA ##############

pacman also delivers gnuplot and graphviz
pacboy -S gnuplot:x
pacboy -S graphviz:x
```

A.2 Legge visualiseringsskript inn i FlexHX

```
Making a VTK template in python for flexhx

Variables should be stored as pure variables
or dictionaries at the top of the file.

After creating a visualization model:
1. Remove/comment out test variables on top
2. Save template to <somefolder>/flexhx/lib/
3. Add target to Makefile (xxd)
4. Add vtktemplate to objects.lst
5. Make a function that creates a full script (variables and template)
    a. Find correct source file, e.g. /src/fhxvtk.c
    b. Make a function that acquires and writes the variables
    c. Add template at the bottom (loop through all symbols from xxd)
6. Apply function to your model
```

A.3 Fra visualisering til 3D-print

```
Open your vtk output files (.stl and .vtk)
File -> Open -> <your files>

GLYPHING
When glyphing you will have both .stl and .vtk for each actor.
The .stl is a single glyph. The .vtk sets glyph locations.
To combine these data in Paraview, do the following:

0. If necessary: Make .stl-files waterproof in windows 3D-builder
1. Open the .stl and the corresponding .vtk
2. Select the .vtk and select (in toolbar):
   Filters->Alphabetical->Glyph With Custom Source
3. Choose the vtk file as input and stl file as glyph type.
4. Select your new "GlyphWithCustomSource".
   Set "Scale Factor" to 1. Set "Glyph Mode" to "All Points".

Please update this guide if you figure out a way to make the settings
under point 4. the default settings for "Glyph With Cusrom Source".

CLIPPING
Filter -> "Clip closed surface"
Under properties you can choose plane to clip, offset, etc.
Don't use the normal clipping function. It does not close the surface,
    which is needed for 3D-printing.

Note for 3D-printing:
Figures created with polydata might not be waterproof.
Not waterproof -> clipping cannot close object
3D-builder (windows program) can "fix" (waterproof) your .stl polydata
    object.
Replace the old .stl in Paraview.

TIPS:
Download the Paraview guide and read up on how to navigate
Save state often. Paraview crashes frequently.
```