Name: **Christian Faßbender**
Course: **Computational Mechanics Tools**

Assignment: **Gid Homework**
Date: January 10, 2022

Contents

1 Introduction **1**
2 Interface **2**
3 TCL-Scripting **5**
4 Problems found and solved **7**
5 Cases **9**
5.1 Example A . . . 9
5.2 Example B . . . 12
6 Conclusions **15**

1 Introduction

Frame3DD is a program well suited for the simulation of frame structures. It receives a plain text input file, from which the solver extracts the data of the problem, solves it and outputs the specified quantities of interest like deformation or internal forces. Working with a plain text input and output might be fine for small problems, but becomes very cumbersome for larger problems very quickly. That is why in this assignment the open-source program *Gid* is used to add a Graphical User Interface (GUI) for pre- and postprocessing to the solver. In this way, the user can generate a geometry and mesh in *Gid* and afterwards add directly required data for the solver data like materials, boundary conditions and loads in *Gid*. The solver can then be called from within the GUI and its results can also be postprocessed there. In section 2 it is described how the GUI is implemented, in section 3 it is explained how the data in the GUI is extracted and sent to the solver via the functions in the *.tcl* file. Afterwards, in section 5 it is demonstrated how two example problems can be simulated using the developed GUI and the *Frame3DD* solver. Finally, section 6 gives a summary and conclusions about the work in this assignment.

2 Interface

The first part of the assignment was to generate a graphical user interface for the problem type. This is achieved by the *.spd* file, which generates the tree structure shown below:

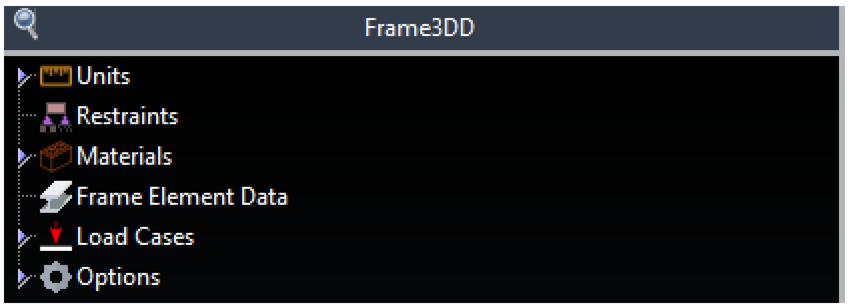


Figure 1: Overview of the graphical user interface of the problem type

Each of the items in the tree structure has further selections to make or values to input.

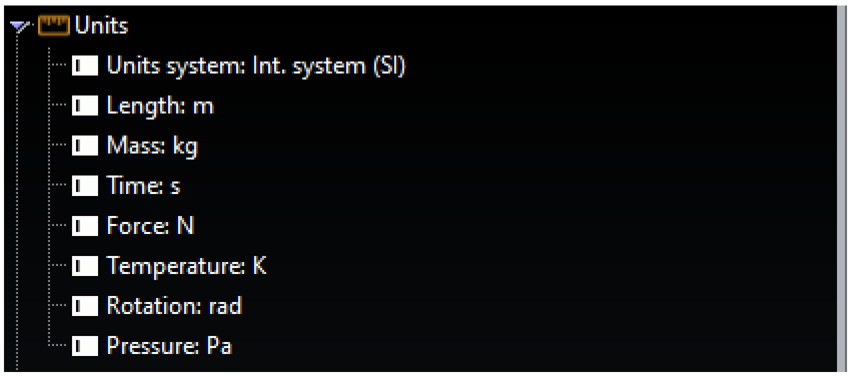

Units
Units system: Int. system (SI)
Length: m
Mass: kg
Time: s
Force: N
Temperature: K
Rotation: rad
Pressure: Pa

Figure 2: The Units menu

Figure 2 expands the first item, where the unit system (Imperial/SI) and the base units for different physical quantities can be selected. Basic expandable folders like *Units* are implemented with the command *container*.

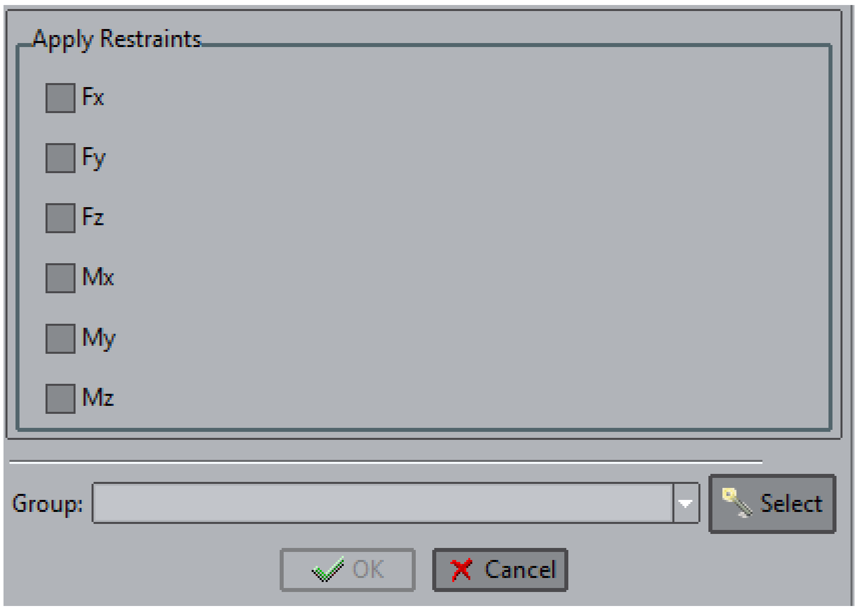


Figure 3: The Restraints menu

Figure 3 shows how restraints can be applied. This menu is generated by the command *condition*, which allows multiple restraints to be added manually. Furthermore, each condition is connected to the geometry and the mesh. Here, the selection only allows nodes to be picked, lines, surfaces or volumes are not valid. Inside the condition there are multiple tickboxes, where each one indicates if a node or group of nodes can react with the corresponding directional force or moment. If it is ticked, it can, otherwise it cannot. Simultaneously this means that a ticked box indicates that the corresponding degree of freedom of that node or group of nodes is locked.

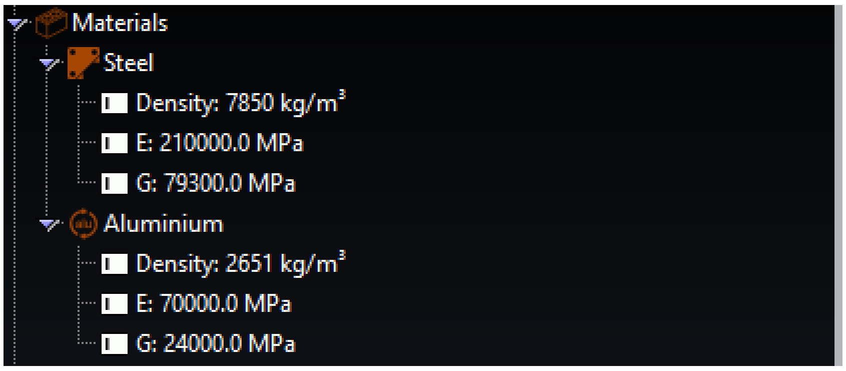
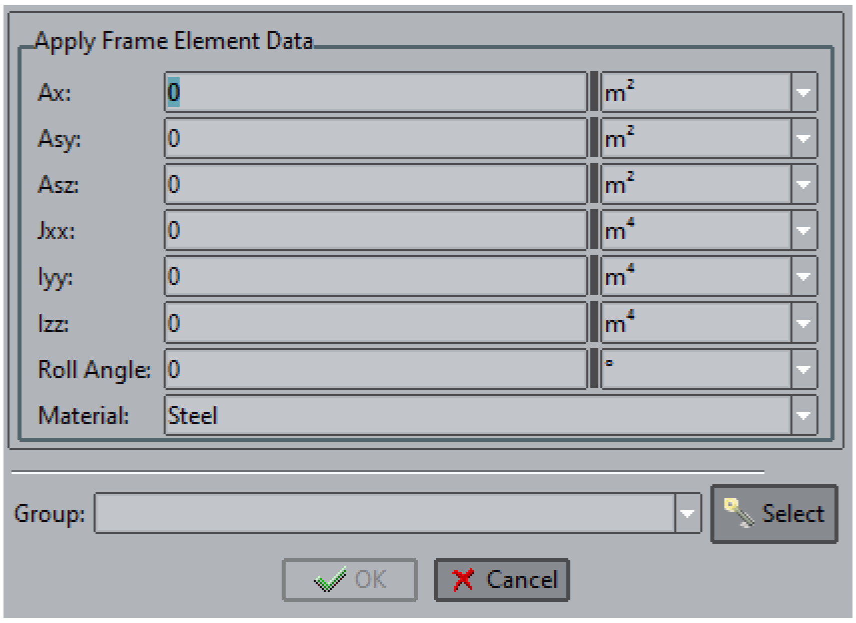

Materials
Steel
Density: 7850 kg/m^3
E: 210000.0 MPa
G: 79300.0 MPa
Aluminium
Density: 2651 kg/m^3
E: 70000.0 MPa
G: 24000.0 MPa

Figure 4: The Materials Menu

Figure 4 points out the material menu. This menu is created from a separate *.xml* file, which is referenced in the *.spd* file. Each material is generated by the *blockdata* command, which functions differently to the *condition* in that its content is not connected to the geometry or mesh. There are two default materials, steel and aluminum, each with its properties the density, the Young's modulus E and the shear modulus G. Further materials can be manually added by right-clicking on the Materials folder and selecting *Create new material*.



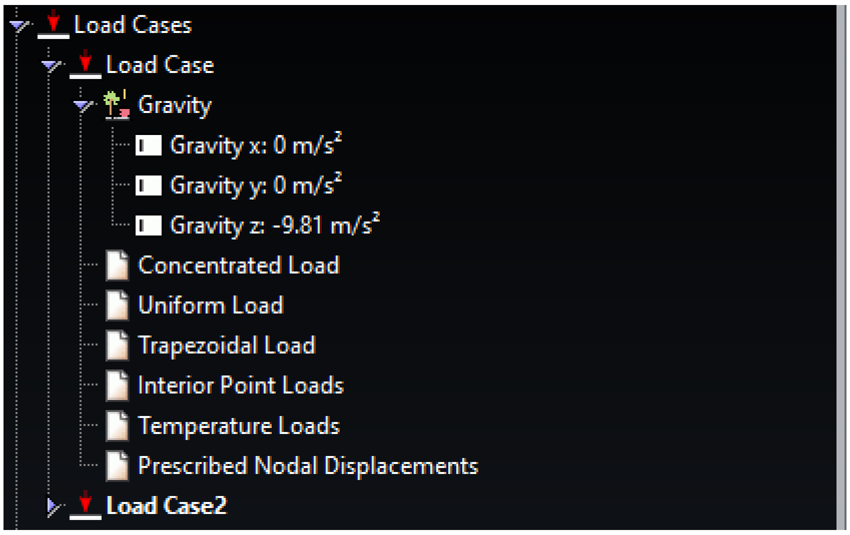
Apply Frame Element Data

Ax:	0	m^2
Asy:	0	m^2
Asz:	0	m^2
Jxx:	0	m^4
Iyy:	0	m^4
Izz:	0	m^4
Roll Angle:	0	°
Material:	Steel	

Group:

Figure 5: The Frame Element Data menu

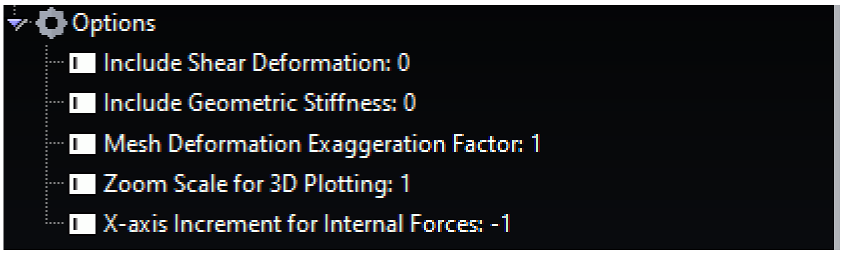
Figure 5 illustrates the menu, in which the frame data can be entered. It is a *condition*, but its data can only be assigned to lines, groups of lines and in the mesh to an element or group of elements respectively. For each are then the cross-sectional area Ax, the shear areas Asy and Asz, the second moments of area Ixx, Iyy and Izz, the roll angle and the material specified.



```
Load Cases
  Load Case
    Gravity
      Gravity x: 0 m/s²
      Gravity y: 0 m/s²
      Gravity z: -9.81 m/s²
    Concentrated Load
    Uniform Load
    Trapezoidal Load
    Interior Point Loads
    Temperature Loads
    Prescribed Nodal Displacements
  Load Case2
```

Figure 6: The Load Cases menu

Figure 6 shows the load case menu. Each load case is implemented by *blockdata*, containing a *container* for the directional definition of gravity and *conditions* for different types of loads, which need to be assigned to geometric and mesh objects. These include concentrated, uniform, trapezoidal, interior point and temperature loads as well as prescribed nodal displacements. As each loadcase is defined as a *blockdata*, multiple other load cases with multiple types of loads can be added manually and calculated seperately.



Options
Include Shear Deformation: 0
Include Geometric Stiffness: 0
Mesh Deformation Exaggeration Factor: 1
Zoom Scale for 3D Plotting: 1
X-axis Increment for Internal Forces: -1

Figure 7: The Options menu

The last item in the tree shown in figure 7 is a *container* object with different options, which are to be submitted to the solver. The first two are booleans if shear deformations and or the geometric stiffness should be included in the calculation. Then, a mesh deformation exaggeration factor as well as a zoom scale for 3D plotting can be set. Last, an X-axis increment for the internal force calculation can be specified, where the default value of -1 indicates that the internal force calculations are skipped.

3 TCL-Scripting

After the tree structure has been filled with sufficient data for the problem at hand, the user can click on *Calculate*. This triggers the *.tcl* script to generate the input file from the data in the tree. In detail, the function responsible for generating the input file is called:

```
Frame3DD::WriteCalculationFile {filename}
```

First, basic file information and the base units for the different relevant physical quantities are set by writing:

```
customlib::WriteString "Input Data file for Frame3DD - 3D structural frame analysis\
([gid_groups_conds::give_active_unit F], [gid_groups_conds::give_active_unit L],\
[gid_groups_conds::give_active_unit M], [gid_groups_conds::give_active_unit T],\
[gid_groups_conds::give_active_unit Temp], [gid_groups_conds::give_active_unit Angle],\
[gid_groups_conds::give_active_unit P])"
```

Hereby is *customlib::WriteString* the function that prints a specified string data to the input file, which is used often and thus ommited in the following code snippets. The node coordinates are written by:

```
set format "%5d %14.5e %14.5e %14.5e\t0.0\n"
           # id x y z r
GiD_WriteCalculationFile coordinates  $format
```

Then, the restraints are written by:

```
set restraints_list [list "restraints"]
set restraints_formats [list {"%5d\t" "node" "id"} {"%1d " "property" "fx"
}{"%1d " "property" "fy"} {"%1d " "property" "fz"} {"%1d  " "property" "mx"
}{"%1d  " "property" "my"} {"%1d " "property" "mz"}]
customlib::WriteNodes $restraints_list $restraints_formats
```

These commands write a line with the node id and the values of the tickboxes for each restrained node to the input file. Next, the frame data is written to the input file by:

```
GiD_WriteCalculationFile connectivities -elemtype Linear  $formats_dict
```

Hereby, the argument *connectivities* results in the elements and corresponding nodes to be printed. *formats_dict* is a merged dictionary for each group format dictionary. Each group format dictionary contains the values of the frame for that group of elements. For each group the values of the directional cross sections, the directional moments of inertia, the roll angle and the name of the applied material are obtained. Then the name of the material is used to query its properties from the tree structure and to add them to the *formats_dict*.

Furthermore, the values in the tree could be specified in a different unit than the base unit. This is dealt with (here on the example of A_x) by calling

```
[gid_groups_conds::convert_value_to_active $A__node]
```

which automatically converts to the base unit. This conversion is done for all values in the tree that have units.

The general options are translated to the input file in the following way, subsequently demonstrated for the shear deformation boolean:

```
set document [$::gid_groups_conds::doc documentElement]
set xpath "/Frame3DD_default/container\[@n = 'Options' \]/value\\
    [@n = 'incl_shear_def' \]"
set xml_node [$document selectNodes $xpath]
set shear_deformation [get_domnode_attribute $xml_node v]
```

The other option values are obtained similarly and then all of them are printed to the input file.

As there might be multiple load cases requested by the user, it is looped through them in the *.tcl* script:

```
set xpath "/Frame3DD_default/container\[@n = 'load_cases' \]/blockdata"
set xml_nodes [$document selectNodes $xpath]
foreach load_case $xml_nodes {...    }
```

For each load case the different loading types are written. The gravitational loading is obtained similarly to the option values as it is also a *container* with values inside. The difference is that the xml node is only searched for in the current load case and not in the whole document with the following lines of code:

```
set xpath "./container\[@n = 'gravity' \]/value\[@n = 'g_z' \]"
    set xml_node_z [$load_case selectNodes $xpath]
```

Concentrated loads are written by the function call

```
GiD_WriteCalculationFile nodes  $formats_dict
```

where *formats_dict* is a merged dictionary for each group format dictionary. Each group format dictionary contains the values of the concentrated load condition for that group of nodes. Similarly, uniform loads are written by the same function but with a different argument:

```
GiD_WriteCalculationFile elements -elemtype Linear  $formats_dict
```

Uniforms loads were assigned to elements and elements groups. Furthermore, here the elements written are restricted to be linear. Trapezoidal, internal point and temperature loads are handled very similarly to uniform loads, just with different value arguments. Prescribed nodal displacements are written similarly to concentrated loads as those were assigned to nodes and not to elements. The input file also requires the number of nodes or elements assigned to each loading condition. This is achieved by:

```
set number_of_elements [GiD_WriteCalculationFile elements \
    -count -elemtype Linear $formats_dict]
set number_of_nodes [GiD_WriteCalculationFile nodes -count  $formats_dict]
```

Last, it is written that the number of dynamic modes is 0 as modal analysis is beyond the scope of this project and the writing operation of the input file is terminated. Afterwards, a batch file performs some file operations in the working directory and calls the solver. The results are then converted back to a format readable by *Gid* (*.res)* such that they can be postprocessed.

4 Problems found and solved

The problems encountered while working on this assignment were mainly related to the generation of the input file by the *.tcl* scripting:

- The first one was ensuring that the units were printed to the input file as intended by the user. The units of each value that has units can be changed in the tree interface as for example figure 4 illustrates. Therefore, just printing the base unit of the corresponding physical quantity to the input file by

  ```
  [gid_groups_conds::give_active_unit P]
  ```

  is not enough. This is because, if the user manually changed the unit for the value, the change of the magnitude of the value is reflected in the input file, the change of the unit however is not. This issue was resolved by using

  ```
  [gid_groups_conds::convert_value_to_active $E_node]
  ```

  which automatically converts the magnitude of the value to be represented to the base unit for the input file. Thus, all quantities in the input file are specified in the base units even if the user has given a quantity in a different unit.

- If multiple load cases are present in the tree structure, it was hard to find the correct entries for different load cases. In the beginning, the *.tcl* script recognized that there were multiple load cases but still printed only the loads of the first load case if multiple load cases had the same type of loading. This was caused by searching for the entries with

  ```
  set xml_nodes [$document selectNodes $xpath]
  ```

  which searches for the name specified in *xpath* in the whole document. The solution to this issue was achieved by only searching inside the load case loop and only for xml nodes inside that specific load case by using

  ```
  set xml_nodes [$load_case selectNodes $xpath]
  ```

Hereby was *load_case* the iterating variable of the loop.

- Antoher problem occured when writing the node coordinates to the input file. Initially the function

  ```
  customlib::WriteCoordinates "%5d %14.5e %14.5e %14.5e\t0.0\n"
  ```

  was used. However, this seemed to scale the nodal coordinates in an unwanted manner. For example if the geometry was created before opening the problem type *Frame3DD* and the length units were then changed to *mm*, the nodal coordinates were scaled by 1/1000. This was not the intended behavior as the geometry was created without a unit first and the unit should be added later by the user in the problem type without scaling the geometry. This issue was resolved by using the following code snippet instead:

  ```
  set format "%5d %14.5e %14.5e %14.5e\t0.0\n"
  GiD_WriteCalculationFile coordinates $format
  ```

- The last problem was that the meshing does not preserve the node numbering from the geometry even if only element is generated per line. In general, this is not a concern for the common user as it can be checked visually to which nodes and elements certain data should be applied. However, when trying to reproduce the examples of the *Frame3DD* documentation, this makes it hard to know where exactly to apply loads and restraints as orientation on the node number of the documentation input file is not valid. An easy solution for this would have been to apply the conditions while being in the geometry view as there the node numbers are as intended, but unfortunately then the condition cannot be assigned to nodes or elements of the mesh. I could not find a way to match the node numbering in geometry and mesh without unreasonable effort. So, the workaround is to check where the node/element lies by the numbers in the geometry view, remember its position and then apply the condition to the node/element with the same position in the mesh view. The different numbering results in the input files from the documentation and the ones built in this assignment not being the same, however the physical information that is represented by them is the same and valid.

Many smaller problems were resolved by looking in detail at the *Gid Customization Help.*

5 Cases

The examples presented in this report are the first two from the *Frame3DD* documentation.

5.1 Example A

The first one is a frame structure with the following geometry,

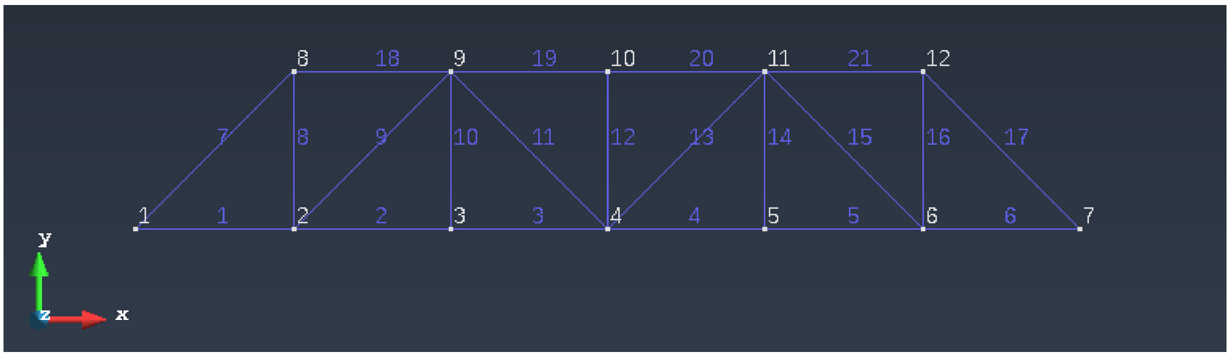
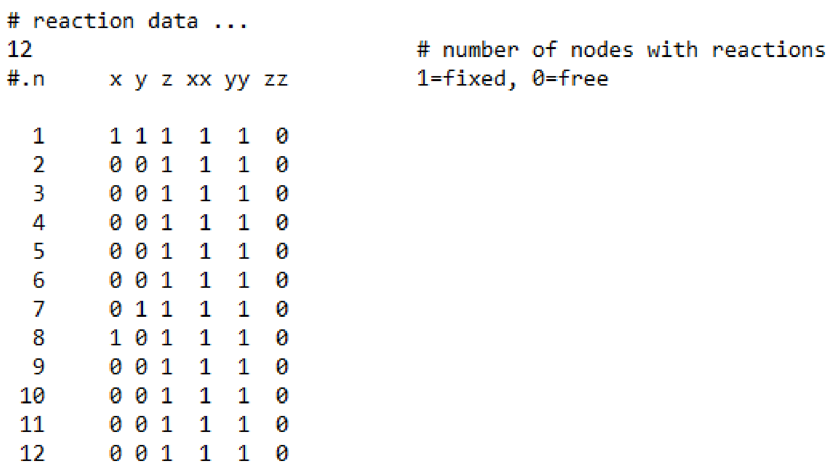


Figure 8: Frame Geometry of Example A

where the vertical as well as the horizontal distance between each junction is 120 inches. The white numbers indicate node numbers, whereas the blue ones show element numbers. The mesh seed (120) is chosen such that each line is represented by one element. The geometry is constrained by the following nodal reactions:

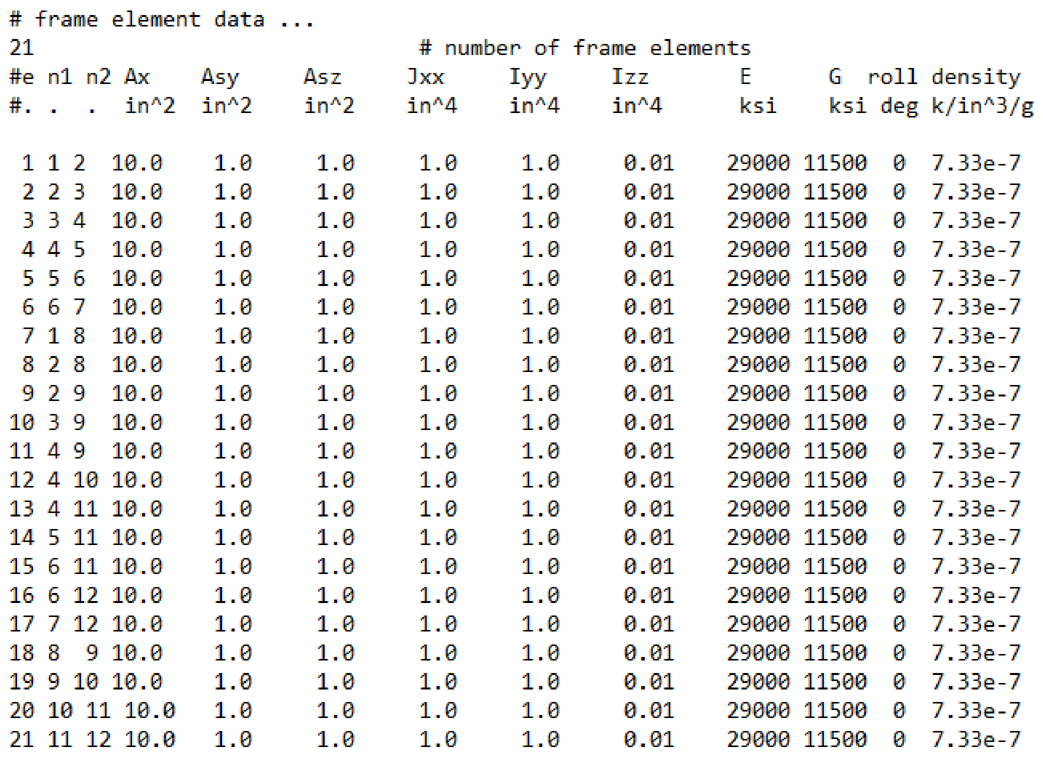


```
# reaction data ...
12                              # number of nodes with reactions
#.n     x y z xx yy zz          1=fixed, 0=free

  1     1 1 1  1  1  0
  2     0 0 1  1  1  0
  3     0 0 1  1  1  0
  4     0 0 1  1  1  0
  5     0 0 1  1  1  0
  6     0 0 1  1  1  0
  7     0 1 1  1  1  0
  8     1 0 1  1  1  0
  9     0 0 1  1  1  0
 10     0 0 1  1  1  0
 11     0 0 1  1  1  0
 12     0 0 1  1  1  0
```

Figure 9: Restraints of Example A

The frame element data is chosen as follows, where the material properties are included:



```
# frame element data ...
21                              # number of frame elements
#e n1 n2 Ax     Asy     Asz     Jxx     Iyy     Izz       E      G  roll density
#.  .  .  in^2  in^2    in^2    in^4    in^4    in^4      ksi    ksi deg k/in^3/g

 1 1 2  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 2 2 3  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 3 3 4  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 4 4 5  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 5 5 6  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 6 6 7  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 7 1 8  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 8 2 8  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
 9 2 9  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
10 3 9  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
11 4 9  10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
12 4 10 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
13 4 11 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
14 5 11 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
15 6 11 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
16 6 12 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
17 7 12 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
18 8  9 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
19 9 10 10.0    1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
20 10 11 10.0   1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
21 11 12 10.0   1.0     1.0     1.0     1.0     0.01    29000 11500  0  7.33e-7
```

Figure 10: Frame Element Data of Example A

Example A has two load cases, but in order not to flood this section with postprocessing figures, only the first one is presented. The interested reader may take a look at the figures for the other load cases in the folder *images_report* in the delivery or generate further ones by his-/herself using the files in the folder *CaseA.gid*. There are concentrated forces and a prescribed nodal displacement present in the first load case:

```
                                # Begin Static Load Case 1 of 2

# gravitational acceleration for self-weight loading (global)
#.gX            gY              gZ
#.in./s^2       in./s^2         in./s^2
  0             0               0

5                               # number of loaded nodes
#.n     Fx       Fy      Fz     Mxx     Myy     Mzz
#.      kip      kip     kip    in.k    in.k    in.k
 2      0.0     -10.0    0.0    0.0     0.0     0.0
 3      0.0     -20.0    0.0    0.0     0.0     0.0
 4      0.0     -20.0    0.0    0.0     0.0     0.0
 5      0.0     -10.0    0.0    0.0     0.0     0.0
 6      0.0     -20.0    0.0    0.0     0.0     0.0

0                               # number of uniform loads
0                               # number of trapezoidal loads
0                               # number of internal concentrated loads
0                               # number of temperature loads

1                               # number of nodes with prescribed displacements
#.n    Dx      Dy      Dz      Dxx     Dyy     Dzz
#.     in      in      in      rad.    rad.    rad.
  8     0.1     0.0     0.0     0.0     0.0     0.0
                                # End   Static Load Case 1 of 2
```

Figure 11: Load Case 1 of Example A

The input file of the *Frame3DD* documentation contained the unit *kip*, which converts to $1000 lbf$. The results generated by this input file are shown in the following:

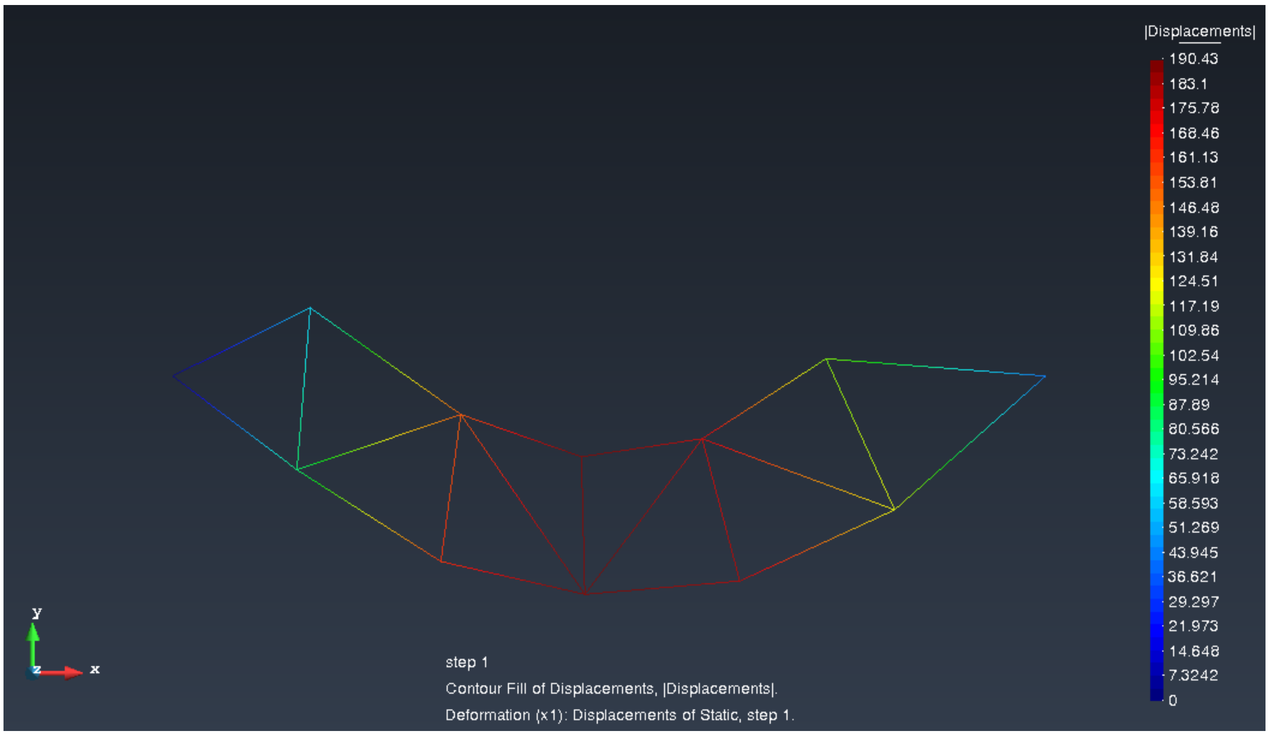


Figure 12: Deformation in Load Case 1 of Example A

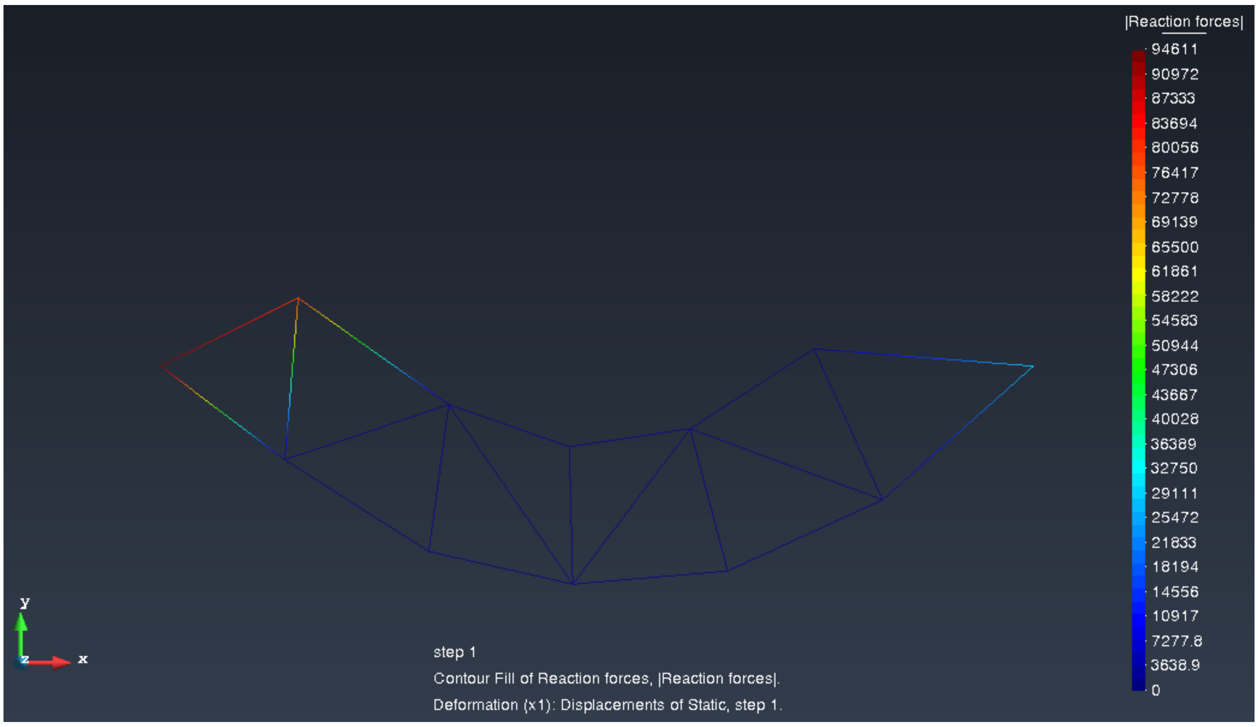


Figure 13: Reaction Forces in Load Case 1 of Example A

The resulting displacements and reaction forces shown in figure 12 and 13 are rather large due to the high loads. The deformations reproduce a frame structure bending under transverse loads with a maximum displacement of about 190 inches. In total the results seem plausible.

5.2 Example B

The second case resembles a pyramid shape consisting of lines:

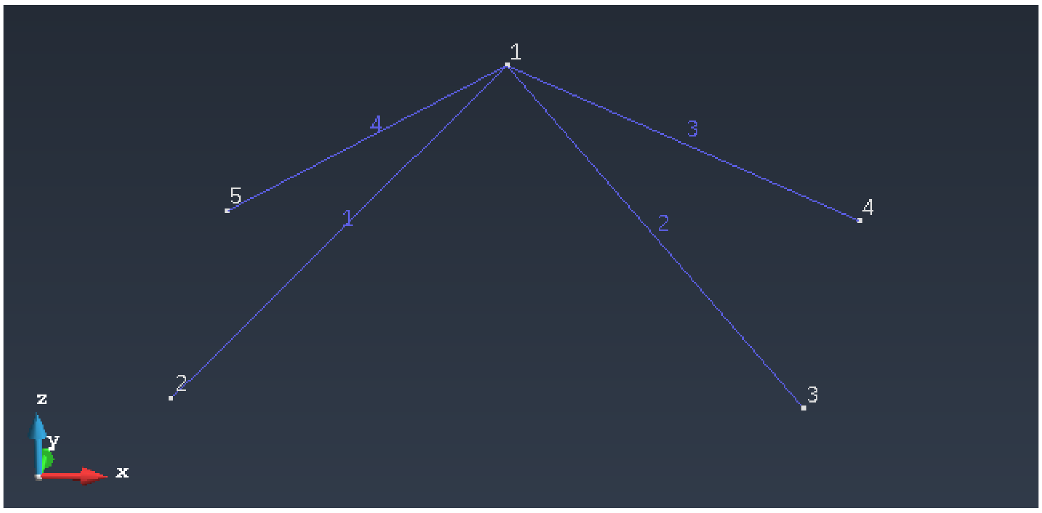


Figure 14: Frame Geometry of Example B

Each line connects a x-distance of 1200 mm, a y-distance of 900 mm and a z-distance of 1000 mm. Again, the white numbers indicate node numbers, whereas the blue ones show element numbers. The mesh size $h = \sqrt{1200^2 + 900^2 + 1000^2} \approx 1803$ is chosen such that each line is represented by one element. The geometry is constrained by the following nodal reactions:

```
4                                # number of nodes with reactions
#.n     x  y  z xx yy zz            1=fixed, 0=free

  2     1  1  1  1  1  1
  3     1  1  1  1  1  1
  4     1  1  1  1  1  1
  5     1  1  1  1  1  1
```

Figure 15: Restraints of Example B

The frame element data is chosen as follows, where the material properties are included:

```
4                               # number of frame elements
#.e n1 n2 Ax    Asy     Asz     Jxx     Iyy     Izz     E       G    roll density
#   .  .  mm^2  mm^2    mm^2    mm^4    mm^4    mm^4    MPa     MPa   deg T/mm^3

1 2 1   36.0    20.0    20.0    1000    492     492     200000  79300  0 7.85e-9
2 1 3   36.0    20.0    20.0    1000    492     492     200000  79300  0 7.85e-9
3 1 4   36.0    20.0    20.0    1000    492     492     200000  79300  0 7.85e-9
4 5 1   36.0    20.0    20.0    1000    492     492     200000  79300  0 7.85e-9
```

Figure 16: Frame Element Data of Example B

Example B has three load cases, however, only the first one is presented here. Again, further images can be found in the folder *images_report* of the delivery or generated by his-/herself using the files in the folder *CaseB*. The first load case consists of a gravitational and a concentrated loading:

```
                                # Begin Static Load Case 1 of 3

# gravitational acceleration for self-weight loading (global)
#.gX           gY              gZ
#.mm/s^2       mm/s^2          mm/s^2
  0            0               -9806.33

1                              # number of loaded nodes
#.e      Fx       Fy     Fz      Mxx      Myy      Mzz
#        N        N      N       N.mm     N.mm     N.mm
 1      100     -200    -100    0.0      0.0      0.0
0                              # number of uniform loads
0                              # number of trapezoidal loads
0                              # number of internal concentrated loads
0                              # number of temperature loads
0                              # number of nodes with prescribed displacements
                               # End   Static Load Case 1 of 3
```

Figure 17: Load Case 1 of Example B

These loadings yielded the following deformation and axial force results:

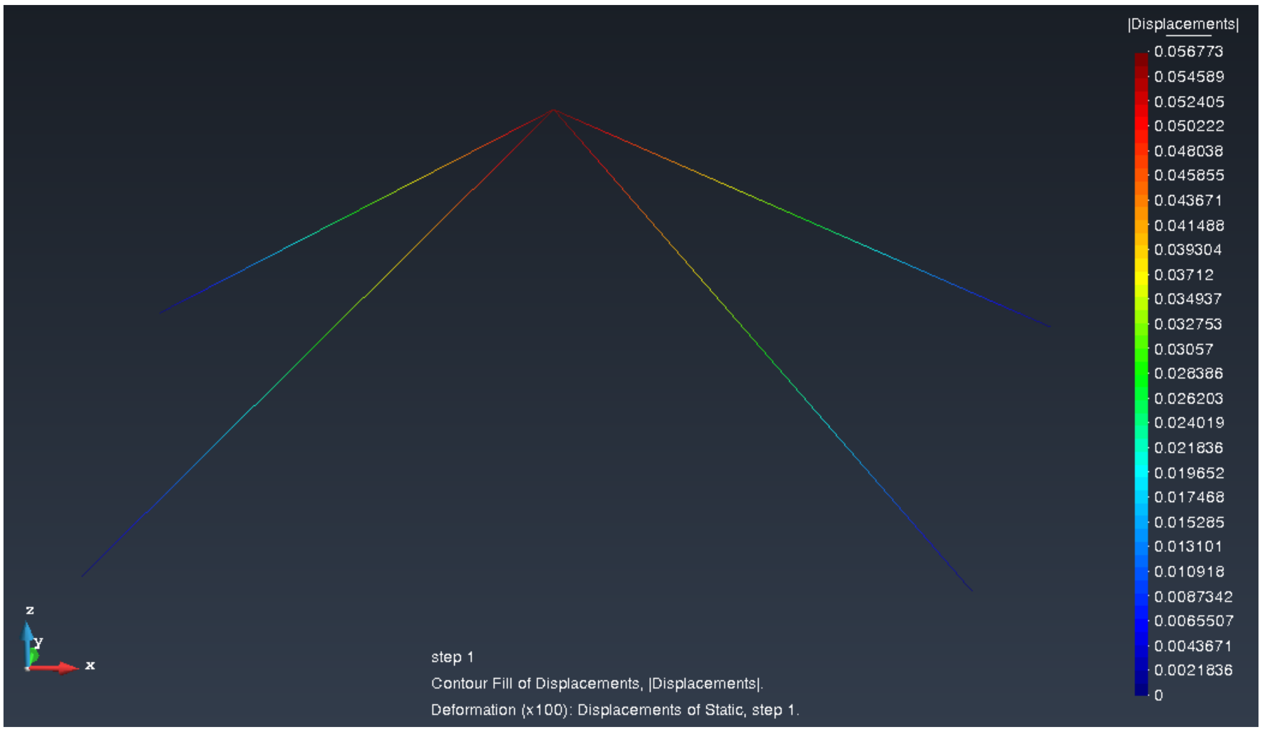


Figure 18: Deformation in Load Case 1 of Example B

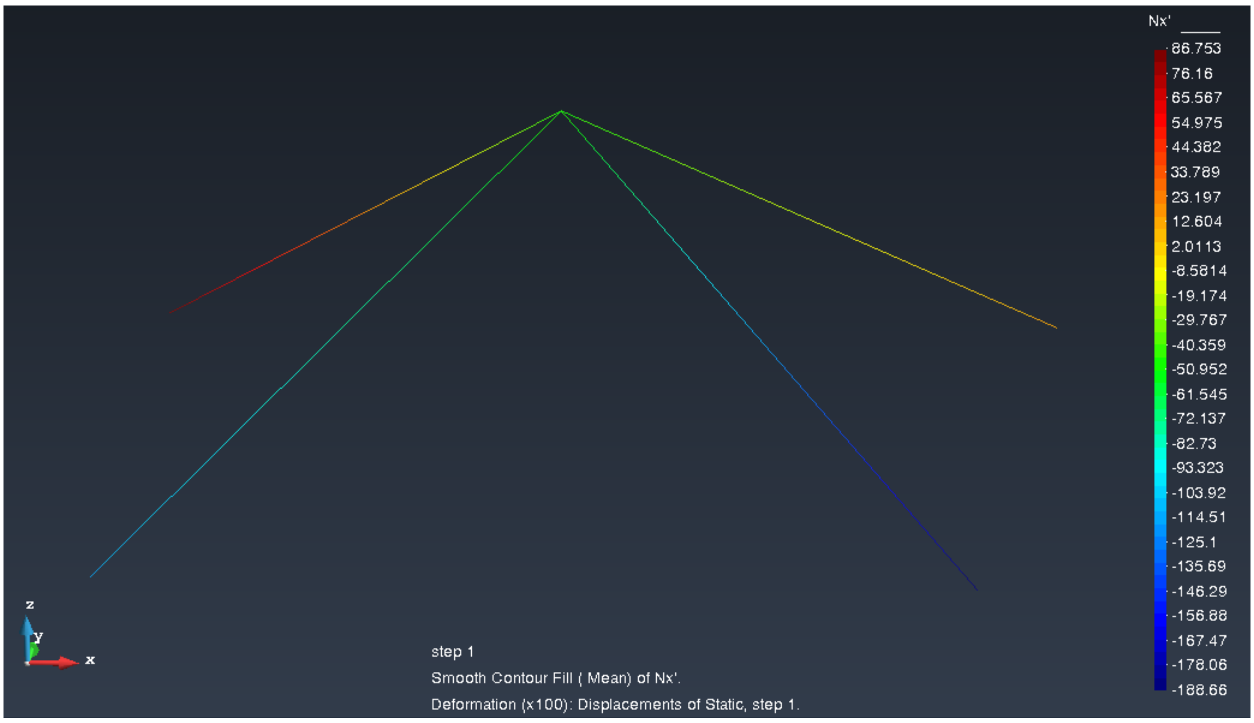


Figure 19: Axial Force in Load Case 1 of Example B

The absolute displacements shown in figure 18 are largest at the top of the pyramid, which is plausible as all but the top node were fixed. The magnitude and direction of the displacements is also reasonable as the length unit is mm and the top of the pyramid is deformed downwards. This is because both the gravitational as well as the point load are applied in negative z-direction. The axial forces vary linearly within the element due to the constant distributed self weight loading. Figure 19 illustrates the axial forces. Furthermore, it shows that the majority of the frame system is under compression mainly due to the loadings in negative z-direction. However, there are also parts of the frame more to the back on the left side in tension. In this way, the x- and y components of the concentrated load, which act in positive x- and negative y-direction, are supported.

6 Conclusions

In the scope of this assignment a GUI for the solver *Frame3DD* was developed. For this a *.spd* file was implemented, which contains the information necessary to build the graphical tree stucture in *Gid*. Furthermore, an extensive *.tcl* script was written, which converts the data given to the tree structure by the user to an input file for the *Frame3DD* solver. With its and the help of a batch file, the solver can be called directly from within *Gid*. After the calculations, the results can also directly postprocessed in *Gid*. This procedure was demonstrated using two examples from the *Frame3DD* documentation. The developed GUI and the fact that *Frame3DD* was integrated into *Gid* made calculating these two examples a very streamlined, fast and easy process. Further generalized, the work of this assignment can be used to simplify the user experience for *Frame3DD* significantly. The initial effort put into the *.spd* and *tcl* files is therefore more than worth it as it accelerates the process of simulating further cases in the future compared to using the solver in a standalone way. Many parts of this work can also serve as help to extend similar functionalities to different solvers to enjoy the benefits of having a solver of choice integrated in *Gid*.