Technical University of Munich

3D Concrete Reinforcements Using Stress Trajectories

Reference Manual

Software Lab 2021

Chair of Computational Modelling and Simulation

Department of Civil, Geo and Environmental Engineering

Supervisors Dr.-Ing. habil. Stefan Kollmannsberger

Chair of Computational Modelling and Simulation

Dr. Johannes Kreutz

K-hoch-3

Team Members Juan Carlos Alvarado Morones

Emin Can Özen

Rafeek Serhal

Date Munich, 24.01.2022

Table of Contents

3DCRUST: SOFiSTiK Post-Processor

1 Foreword

The following document serves as a reference manual for the use of the 3D CRUST library. Main steps to get the source code running are found in the 3D CRUST GitLab repository and should suffice. Nevertheless, this reference manual includes background and more detailed information on the elements involved and the use of the software.

This library was developed within the framework of the Software Lab 2021 "*3D Concrete Reinforcement Using Stress Trajectories*" at Technical University of Munich.

2 Motivation

This application is meant to provide the user an extended kit of post-processing tools and capabilities for the assessment of Civil Engineering FE-models created in SOFiSTiK FEA. Although the whole pre-processing, calculation and post-processing cycle can be done within the SOFiSTiK FEA software/toolkit, 3D post-processing capabilities within the software are limited. User interaction, manipulation and visualization is not very extensive. This software application aims to overcome this limitation by the integration of a C++ programming interface, different libraries, python scripts, and ParaView.

3 Concepts and Definitions

The following concepts will be treated in the remainder of the manual, a general understanding of them is expected and will help to understand their role in the project and the general integration.

3.1 CDB Files

CDB files, CDBASE or SOFiSTiK database files contain all the data from finite element model. Mesh, load cases boundary conditions, and calculation results are stored there. Basically, the database contains the data organized in logical records which can be accessed sequentially. The data is organized by so-called "keys", which identify the type of data stored in the SOFiSTiK database. Data stored under these keys can be then accessed through the CDB Interface functions.

Further details can be found in reference [2]. Definitions, syntax, and structures are described in detail in reference [5] the "*CDBase Help*" located as "*cdbase.chm*" under the SOFiSTiK installation files.

3.2 CDB Interfaces

The CDB programming interfaces are provided by SOFiSTiK to allow the user to create their own applications to address their specific needs SOFiSTiK doesn't support. Data can be read, manipulated, do further calculations, post-processing, etc. In this software, the CDB interface for C++ is used to extract the stress tensor data from the model to then calculate principal stresses which are then further processed. Documentation for different programming languages can be found in reference [3].

A list of all functions, descriptions, parameters, and syntax are described in reference [5].

3.3 VTK File Formats

The Visualization Toolkit (VTK) is open-source software for manipulating and displaying scientific data. Due to their open-source nature, in the framework of 3DCRUST, the VTK file formats are used to translate the SOFiSTiK produced databases to FE-files that can be handled by other software. In the context of 3DCRUST, two types of files are used.

- **UnstructuredGrid (.vtu) –** Topologically irregular set of points and cells. Used to recreate the nodes, elements and store the corresponding Point/Cell Data as the stress tensor (Point Data).
- **PolyData (.vtp) –** All data within a single file, stored serially. Used for stress trajectories.

For more details see reference [6].

4 Prerequisites and Prior Installation

Before building the source code from the repository, the following software must be installed. These are prerequisites either for creating the build, libraries the repository requires to make use of different functions, or software which is used to pre- or post-process the model.

4.1 SOFiSTiK FEA

SOFiSTiK FEM packages are highly used in the field of Civil Engineering for a range of application fields. It is the pre-processor and solver used for creating the models considered in this software application. ***Note:*** This software was developed using SOFiSTiK FEA 2020 for students. However, functionality should still apply for other versions. Care must be taken to use the corresponding licenses and DLL files.

4.2 CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. It is required to build the application from the source code found in the 3D CRUST repository [1].

4.3 VTK Development Libraries

VTK Development libraries are used mainly by gotraceit library to create the tensor fields from VTU files. These need to be priorly installed in the system.

Detailed procedure to configure and build the VTK Development Libraries can be found in reference [9].

4.4 ParaView

ParaView is an open-source, post-processing application. Serves to do data analysis and visualization. Exploration can be done interactively in 3D or using its batch processing capabilities. In the context of 3DCRUST, once the .vtu and .vtp files have been created, these will be imported to ParaView where visualize and interact with the FE model, stress field and stress trajectories for post processing e.g. filtering values in the stress distribution or principal stresses.

5 Software Design and Concept

Figure 1 shows a simple schematic diagram of the 3D CRUST's structure. This shows the interacting libraries and modular structure.

Core functionality of the software comes from the integration of the C++ CDB interface of SOFiSTiK, vtu11 writer and gotraceit libraries.

The "sof_cdb_w_..." .dll and .lib files contain the functionalities of the Programming interface which are used to access and read the database. The vtu11 library provides the functionality to store the retrieved information from the CDB file and write it into an unstructured grid (.vtu) with the associated Point/Cell Data.

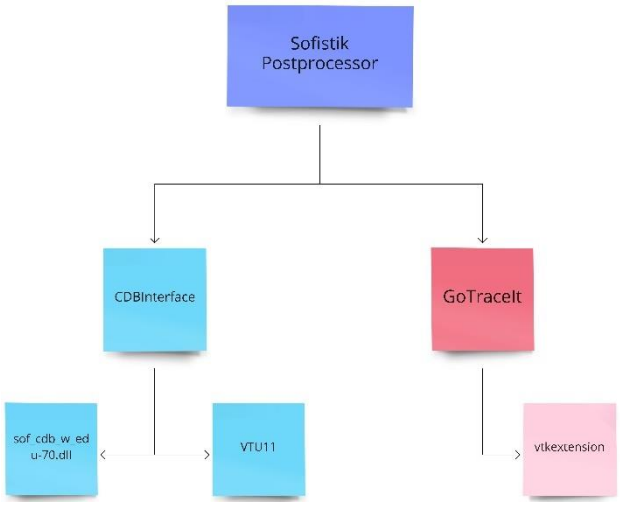

Figure 1 3D CRUST Software Structure

On the other side, gotraceit computes the principal stress trajectories and using its VTK extension, writing the vtp file with the trajectories and principal stress values. 3D CRUST integrates the functionality of both CDBReader and gotraceit into a single interface which can be handled by a Python script.

6 Installation and Configuration

After downloading/cloning the source code from the repository the first step is to create the build. There are a couple of settings to adjust to fit with your own computer, installations, and directories.

6.1 Update SOFiSTiK Directory

As the CDBReader library works with the interface and functions provided by SOFiSTiK, it is necessary to indicate where these files are located. Mainly the header/source files of the interface, as well as the corresponding DLL. For the case of C++ these are usually set directly in the IDE as Visual Studio (reference [4]), since we are using an out-of-source approach, this needs to be indicated in the main CMake file. The **SOFiSTiK directory** ("*SOFISTIK_DIR*") is the main installation folder of the SOFiSTiK version being used. Enter the location of this directory under the "SOFISTIK_DIR" string. As shown in the figure below.

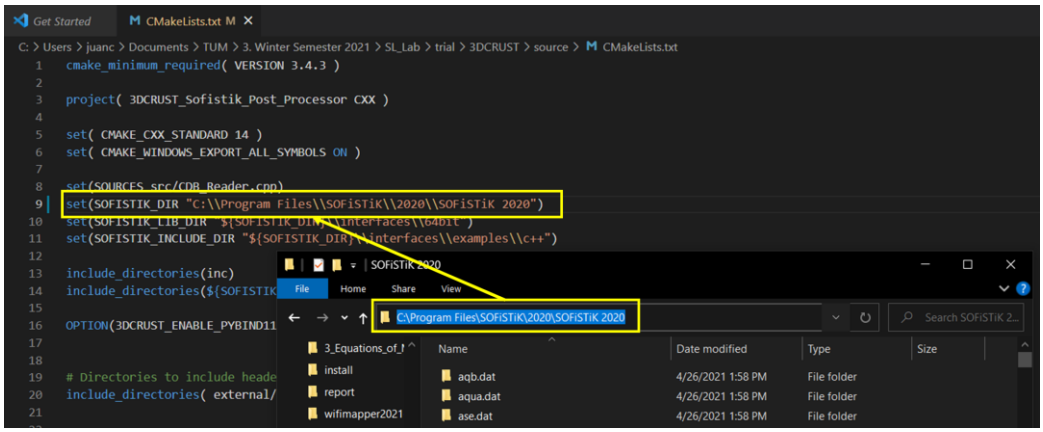

Figure 2 SOFiSTiK Directory in CMake File

6.2 Enable pybind11 and VTK extensions

After setting selecting the source and build directories in CMake, make sure to enable the pybind11 and VTK extension options in CMake.

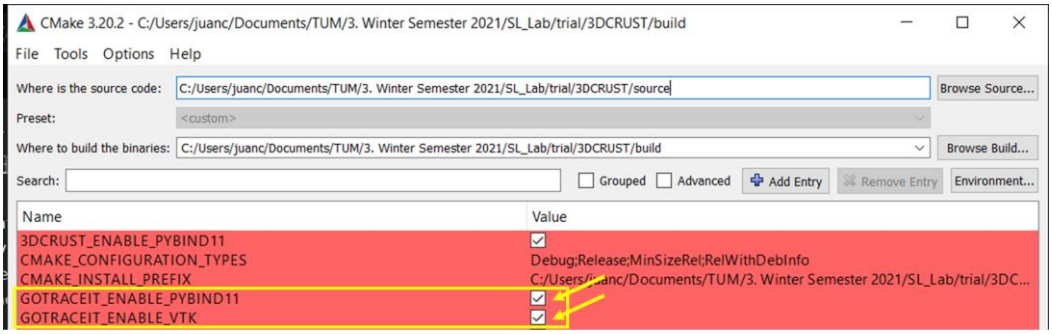

Figure 3 Enable pybind11 and VTK extensions

6.3 Path of CDB Interface Functions DLL

Under the "3DCRUST.cpp" source file main function, update the argument corresponding to the SOFiSTiK .dll file.

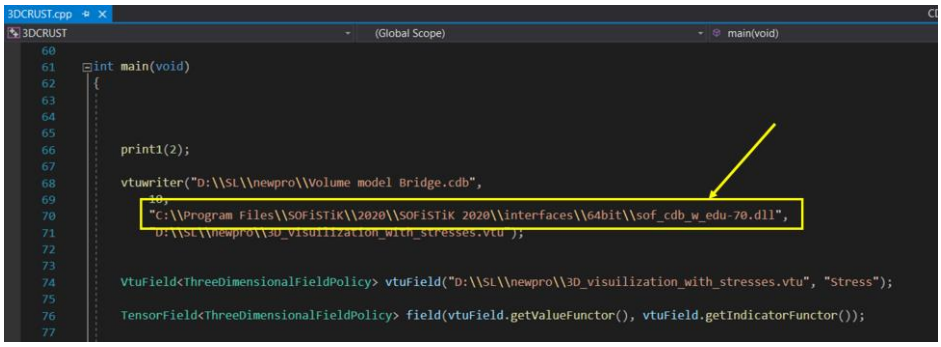

Figure 4 Update .dll location in 3DCRUST.cpp C++ source file

Note that here as shown here, since the education version was used, the corresponding .dll file was given as an argument. Enter the corresponding file to the SOFiSTiK version being used.

6.4 Build INSTALL

After creating the build, open the project make sure to set the configuration to release mode (1), select the **INSTALL** project and select Build as shown in Figure 5 below.

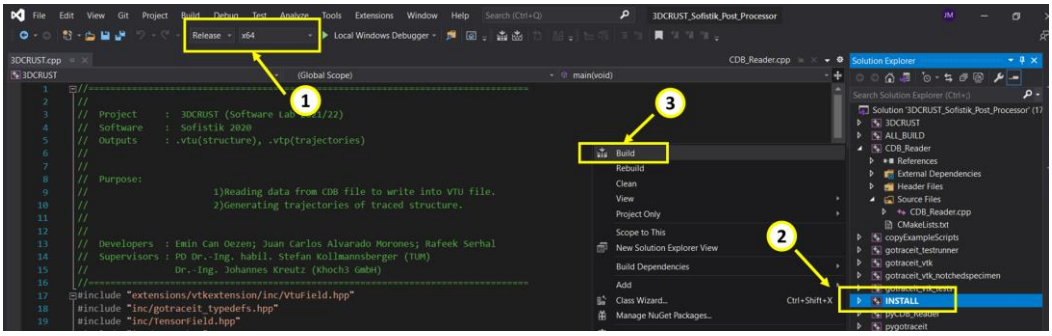

Figure 5 INSTALL Build

7 Workflow of the library

Figure 6 diagram shows the flow of data e.g., input/output files, and interactions between modules.

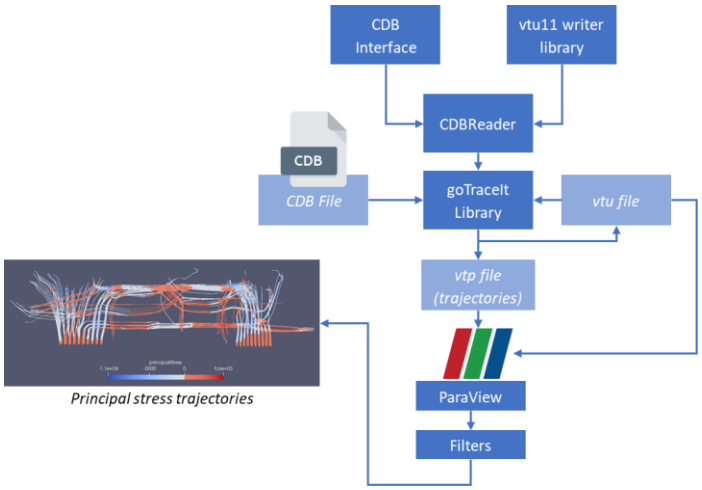

Figure 6 Workflow of the 3D CRUST SOFiSTiK Post-Processor

As seen, after integration of the different modules, interaction concentrates in gotraceit, where the newly created unstructured grid file (.vtu) is used by the library as an input to create the stress trajectories and outputs the vtp file. These two will go into ParaView for post-processing.

7.1 File Inputs

Besides the built project in C++, Python bindings can be used to avoid interaction with the source code. User interaction is done through a Python script created inside the *build/install* folder. Inputs are entered. We can distinguish between a) file inputs, and b) trajectories parameters. In this section the file inputs are described, in section 7.2 information on the trajectory parameters is given. The script that will deliver the needed vtu and vtp files for post-processing is the "*VTUwriter_trajectory_generator_filtered*" script.

- **CDB_path:** Enter the location of the desired CDB to postprocess including name file.
- **Load_Case:** In most of the cases, the databases will contain results for more than one load case. This can be checked within the "*System Visualization*" application of SOFiSTiK. Every vtu will correspond to a load case. Enter here the corresponding number.
- **Sofistik_dll:** As mentioned previously, the CDBReader library works with the CDB functions provided by SOFiSTiK. For this to work it is needed to provide the location of such .dll file.
- **VTU_file:** Enter the desired location and name for the vtu file. Take note of the location where the file will be created.
- **VTP_output:** Similarly, location and name for the vtp file containing the trajectories. It is important to note that the gotraceit output still needs to be filtered. This will be used for an intermediate temporary file.
- **VTP_filtered:** Location and name for the vtp file containing the trajectories. Gotraceit may provide principal stress results as isolated points. Hence, a line (stress trajectory) cannot be created as there are no points to connect but a single point (this can be observed in the vtp PointData array. This will cause ParaView to crash or will be unable to apply the filters needed for postprocessing. To overcome this, the algorithm of 3D CRUST duplicates single points and removes empty lines.

```
CDB_path="D:\\SL\\newpro\\Volume model Bridge.cdb"
Load_Case=10
Sofistik_dll="D:\\Program Files\\2020\\SOFiSTiK 2020\\interfaces\\64bit\\sof_cdb_w_edu-70.dll"
VTU_file= "D:\\SL\\newpro\\3D_visuilization_with_stresses.vtu"
VTP_output="D:\\SL\\newpro\\trajectories.vtp"
VTP_filtered="D:\\SL\\newpro\\trajectories_filtered.vtp"

```

Figure 7 File Inputs of the Python Script

7.2 Trajectories Parameters

Tracing of 3D principal stress trajectories is based depends mainly on these user inputs.

- **boxOrigin:** Decides the point of origin for the box. From this point the "box" will be extruded.
- **boxDimensions:** Determines height, width, and length of the box where seeds are placed.
- **SeedDirections:** Specifies the direction for the 3D principal stresses trajectories.

This box is where the gotraceit library will start tracing from. Selection of origin, dimensions and direction seeds depends on user understanding of the structure behavior. A hint for seeding locations is to create the box either at support boundary conditions and/or load application area. Moreover, the direction of seeds depends on the user's direction visualization assumption.

```
def main():
    vtuField = gtvtk.VtuField3D(VTU_file,"Stress")
    field = gt.TensorField3D(vtuField.getValueFunctor(),vtuField.getIndicatorFunctor())
    stepSize = 0.001
    maximumNumberOfSteps = 1000

    boxDimensions = np.array([0.5, 0.5, 3.0])
    boxOrigin  = np.array([-0.5, -3.5, 0])

    seedPoints = []
    seedDirections = []

    for x in np.linspace(boxOrigin[0],boxOrigin[0]+boxDimensions[0],20):
        for y in np.linspace(boxOrigin[1],boxOrigin[1]+boxDimensions[1],20):
            seedPoints.append([x,y,boxOrigin[2]])

    seedDirections = [[0.0,0.0,1.0]] * len(seedPoints)
```

Figure 8 Trajectories Parameters set-up in the Python script

8 Post-processing in ParaView

Finally, once the files are created, import both vtu and vtp into ParaView for post-processing. To visualize the structure and stress field, enable visibility of the vtu file. Stress trajectories are then visualized by activating the vtp file. For a clearer visualization of the stress values, and tension/compression areas, apply consecutively over the vtp a *Threshold* filter and color palette. This way, certain value ranges can be disregarded for visualization purposes.

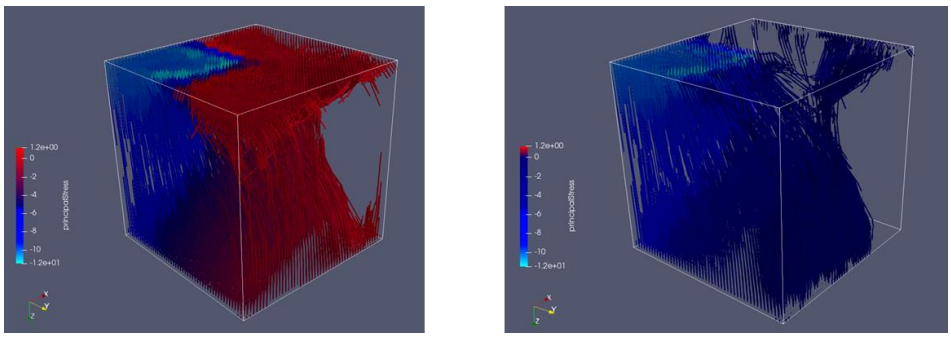

Figure 9 Threshold filter application

8.1 Post-Processing Tools

For a better visualization of the trajectories three tools inside ParaView are mainly used.

- **Color Map preset:** Determines the coloring scale used for the principal stress fields, colors can be selected, number of divisions and values can be set manually.
- **Tube Filter:** Turns lines from the vtp into tubes of adjustable thickness for a clear visualization and distinction of one trajectory to another. Take care of not selecting a value too big.
- **Threshold Filter:** Allows to filter out certain values from the stress field in the Display Window. Useful to see which areas exceed certain stress range.

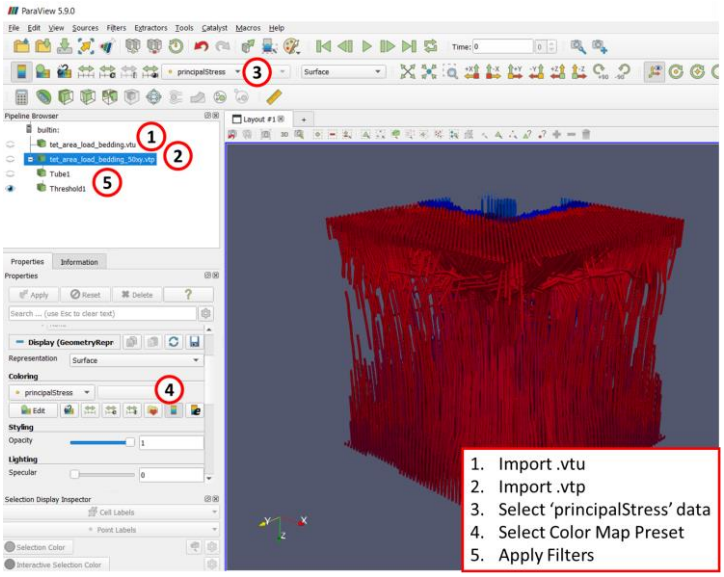

Figure 10 Post-Processing Pipeline in ParaView

For all this tools, some suggested templates have been created. A JSON file containing a blue/red scale for distinction of tension and compression, and a python script which can be run within the ParaView Python Shell (*Tools > Python Shell > Run Script*) or as a Macro (*Macros > Import a new macro…*). However, keep in mind this are a sort of templates, or starting points which can be fined tuned as the user sees fit.

A correct definition of seeding parameters and successful application of ParaView filters can lead to the following output from the SOFiSTiK database.

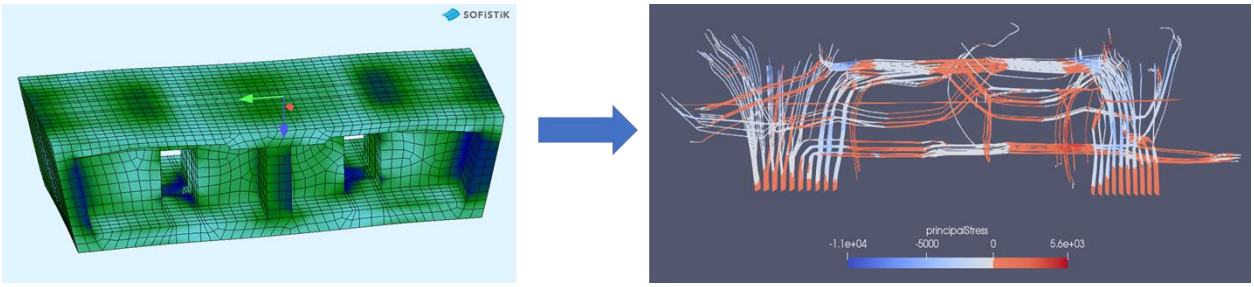

Figure 11 Comparison from a) CDBase to b) ParaView Stress Trajectories

9 References

[1] 3DCRUST / 3DCRUST · GitLab (lrz.de)
[2] CDB description (CDBase Help) — CDB Interfaces 2020 (sofistik.de)
[3] CDB Interfaces Manual — CDB Interfaces 2020 (sofistik.de)
[4] Configure project — CDB Interfaces 2020 (sofistik.de)
[5] SOFiHelp – CDBase Help, SOFiSTiK 2020 Installation files
[6] https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf
[7] Download | CMake
[8] cie_sam / gotraceit · GitLab (lrz.de)
[9] VTK/Configure and Build - KitwarePublic
[10] VTK - KitwarePublic
[11] Download | VTK
[12] Documentation | VTK
[13] Download | ParaView
[14] GitHub - phmkopp/vtu11: Vtu11 is a small C++ header-only library to write unstructured grids using the vtu file format