# Structural 3D File Conversion Tool

CSCE A470 Final Report

Group Members: James Stewman and Orry Snyder

Client: Dr. Scott Hamel

12/12/2024

**Abstract**

Our project is a tool that can convert files created in RISA-3D and ModelSmart 3D, each with its own proprietary filetype (.r3d and .3dd, respectively), into a more widely supported filetype (.obj) that can be used with various 3D modeling and printing applications.

## 1. Intro

Our project stemmed from an issue faced by the UAA students who participate in the ASCE Student Steel Bridge Competition and elementary-age students who participate in the UAA Summer Engineering Academies. This issue is that they create bridges in software that use proprietary file types, which cannot be easily imported into other 3D modeling software such as AutoDesk Fusion 360. With our tool, these two groups can quickly convert these proprietary files into 3D mesh files that are easily importable into AutoDesk Fusion 360 or other 3D modeling software.

## 2. Planning

Initially, we aimed to get something that worked decently with files generated by RISA-3D written as fast as possible. Once we got a rough prototype working, we realized that the math we used for the key part of our conversion tool would not work for our project. We then worked on figuring out what mathematics/libraries would work best for our project and settled on some of NumPy's linear algebra functions. Once we got that working, we split our focus on building a GUI and restructuring how we read and store information from the source files. Once we had completed these two tasks, we sent off that version to Professor Hamel for user testing and focused on the ModelSmart 3D file type. Since we had restructured how our program worked, we could then read and parse these files in a manner that allowed us to use the same mathematical and file generation functions for both file types. Once both file type conversions were working and we were happy with our tool, we created the executable file and fully released it on GitHub.

**3. Requirements**

Our tool's primary requirement is that the user use Windows 10/11 as their operating system. There are three methods for running our program, each with its own set of requirements.

- 3.1 Method one
    - The first method is to run one of the two exe files that are published in our GitHub repository.
    - There are no additional requirements other than the aforementioned operating system requirement.
- 3.2 Method two
    - The second method is to run the .bat file published in our GitHub repository.
    - This batch file will automatically install any missing dependencies and execute the convert.py source file.
    - The only additional requirement for this method is that Python is installed.
- 3.3 Method three
    - The third method is to run the source files published in our GitHub repository.
    - This method requires the user to install Python, NumPy, and tkinter-tooltip.

**4. Design**

For this project, we decided to use Python as our programming language. We chose Python because it is a language that we are both familiar with and that our client is also familiar with. This will allow him to maintain the program better after we are done. For the output file type, we chose the wavefront OBJ file format [6] because while there is no standard 3D file format, it is a standardized file type that many applications can use.

*4.1 Reading in the source files.*

To read the data from the files, we went through the file line by line and parsed the data. This allows for better scalability when reading larger files and is more time-efficient. RISA-3D and Modelsmart 3D have very detailed file formats found in [1] and [2], respectively. We were able to use the guides to read the data that we needed for the main functionality.

*4.2 Conversion Algorithm*

After the data is read from the file, we can begin the conversion process. The first step is to go through each member and determine which planar views the member is in by finding the minimum and maximum of the x, y, and z coordinates and comparing its coordinates to that. Then, we determine if the member is circular or rectangular and compute the faces. For the rectangular face, we used NumPy [3] to help us with the linear algebra. We first created a direction vector from the two endpoints of a member and normalized it. Since each member can be in any direction, we can't just add the width and height to the centerpoint and go from there. We must find two more vectors orthogonal to the directional vector that we can expand upon to create the face. An example of this can be viewed in Figure 1.

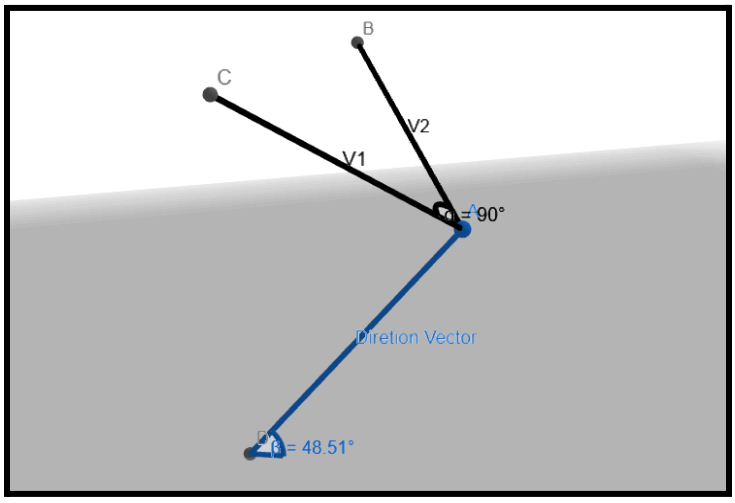



Figure 1. Generated with Geogrbra 3D calculator [5]. This figure shows an example of our generated direction vector with the two other orthogonal vectors so that we can make our faces.

There is also the possibility that a member is rotated in the source software. To account for this, we used Rodrigues' Rotation Formula [4] to get a rotation matrix to ensure our orthogonal vectors account for this rotation. The vertices and faces are then added to master lists to be used later for the output. For cylindrical faces, we again start by finding the direction vector so we know where to center the pie slices. Then, using input from the GUI, we determine how fine the user wants their cylindrical shapes to be since you cannot represent a true cylinder in the obj format, as seen in Figure 2 and Figure 3.

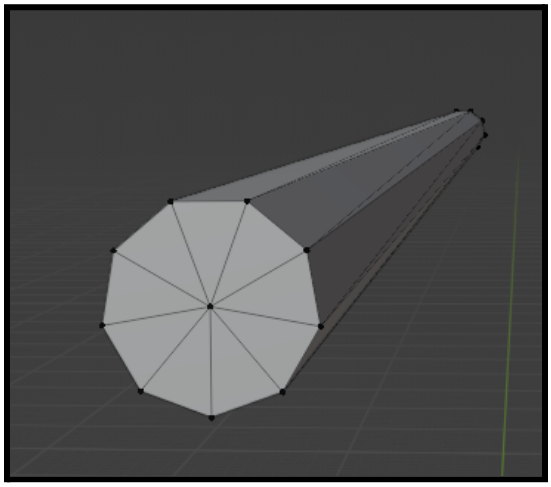


Figure 2. Cylinder with nine vertices.

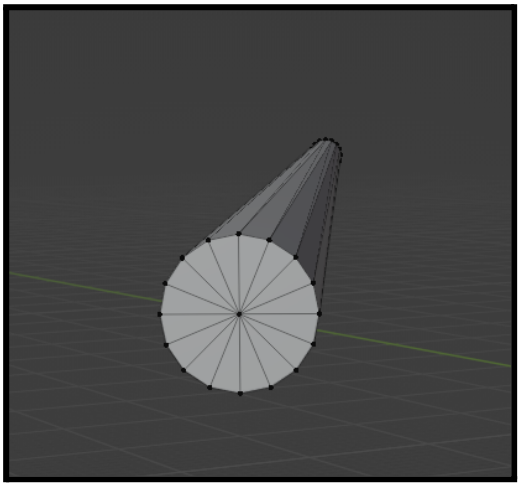


Figure 3. Cylinder with sixteen vertices

We use trigonometry to create the pie-shaped faces and vertices of the cylinders, then add them to the same master lists at the end. Finally, we use the master lists of faces and vertices to write the lists to the wavefront obj file.

*4.3 Program Flow*

The general flow of the program can be seen in Figure 4. The first step involves the user selecting the source file(s), the output destination folder for the obj files, and any additional options. Then, we check if the selected files are of the Modelsmart 3D or RISA-3D file type. We use the process described in 4.1 to parse the data, then complete the main conversion as described in 4.2, which is finally written to our output file.

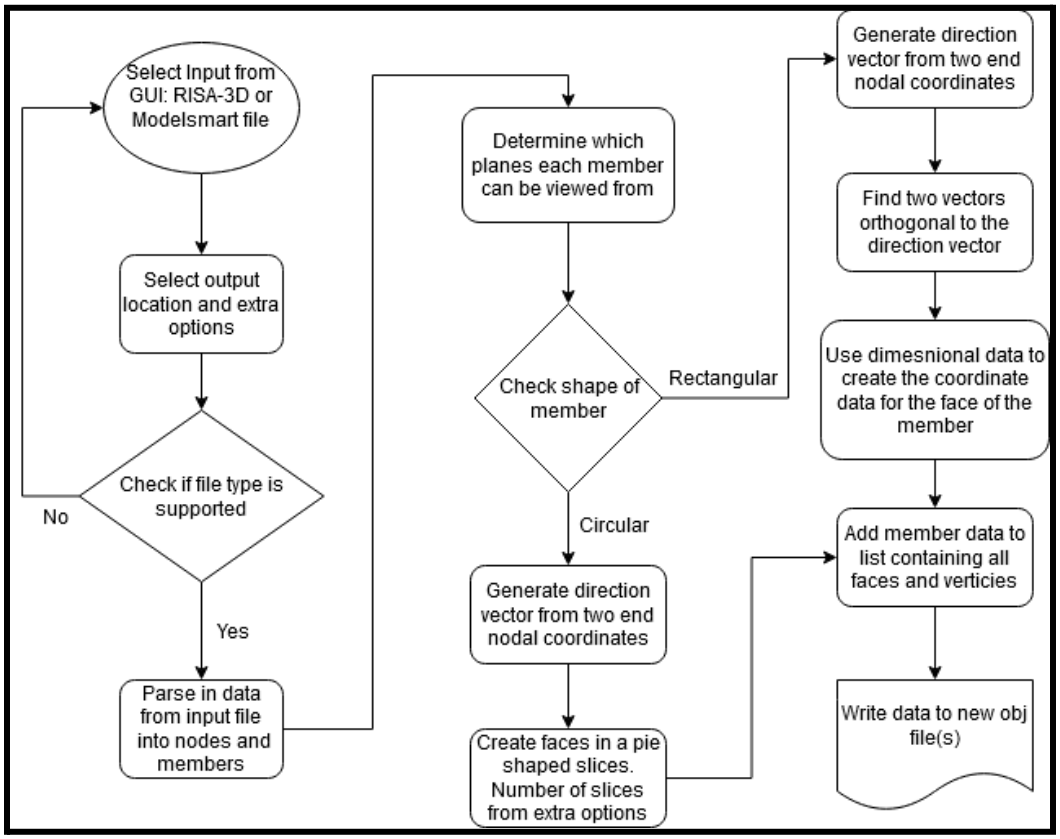



Figure 4. A flowchart that shows the general flow of the program without all of our error checking.

*4.5 User Interface*

The two interface windows can be seen in Figure 5 and Figure 6

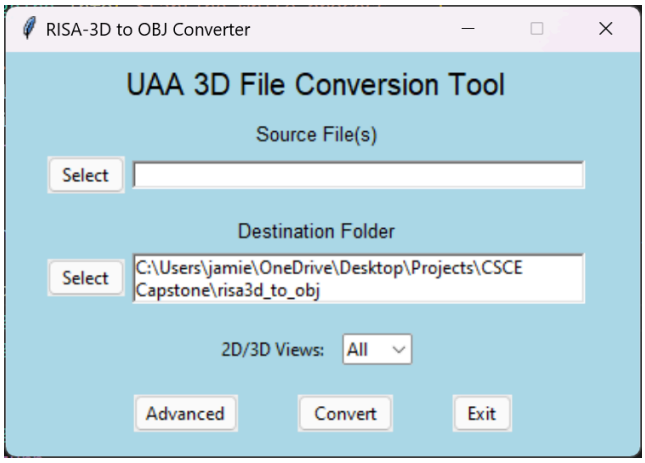



Figure 5. Main window of user interface.

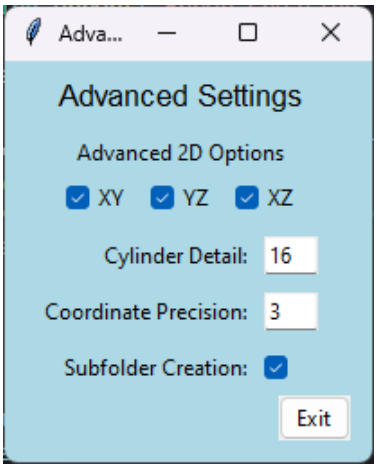



Figure 6. Advanced user interface window

**5. Development Process**

Initially, most of our time was spent on the primary conversion process. We knew the conversion would take the longest, so we devised a crude way to parse in the file to get working on the more complex aspects of the conversion. Our first iteration of the conversion involved attempting to use quaternion angles to rotate vectors in space easily.

However, neither of us had worked with quaternions before, and we were getting close to our deadlines. We switched gears to see if it was possible to do it purely using linear algebra, and we came up with our current algorithm for conversion. During this time, we also started to sprinkle some aspects of the GUI.

Once an iteration worked, we began our testing phase with the client. Immediately, we ran into issues from assumptions we made that were untrue. We did not believe you could change the shape data of members, so we assumed it would always be in a specific format. This was not the case, so we had to dig into the RISA-3D file and see where this data was stored. At this point, we were also giving an executable file to the client that would crash

unexpectedly. We had not devised a proper error-handling solution besides print statements, so we added a log file that would print the errors it encountered.

**6. Analysis & Results**

Our program works very well, as it does nearly everything we initially set out to do. Our only shortcoming specific to the RISA-3D files is that some members, which are rotated about all three axes, have some orientation issues that could be resolved by implementing normal vectors. We also encountered an interesting problem with some ModelSmart 3D files where some members were rotated 90° in the final OBJ file compared to how they appear natively in ModelSmart 3D. There was also a problem specific to Tinkercad. For some reason, Tinkercad puts large gouges and holes in imported models. However, this issue appears to be on Tinkercad's end, as files created in other 3D modeling software and then imported into Tinkercad exhibited the same gouges and holes.

Figures 7 & 8 show a sample RISA-3D file before and after the file conversion.

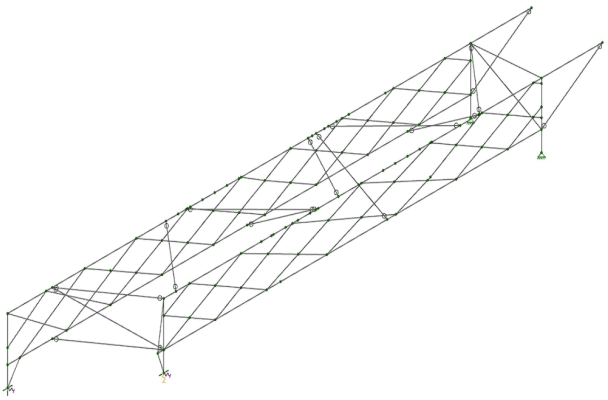


Figure 7. Before conversion

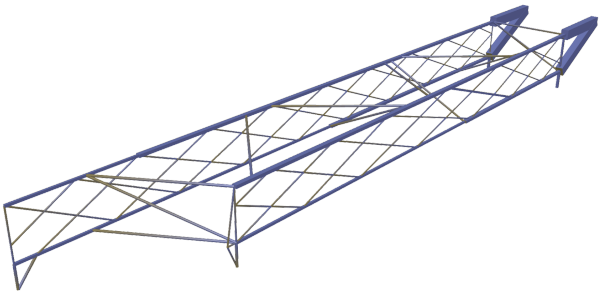


Figure 8. After conversion

Figures 9 & 10 show a sample ModelSmart 3D file before and after the file conversion.

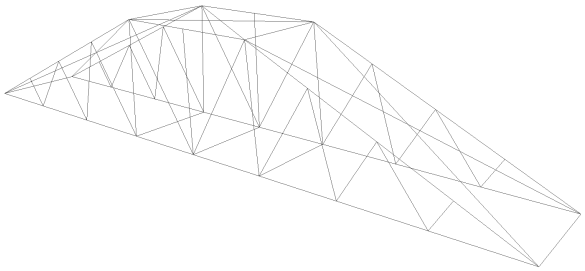


Figure 9. Before conversion

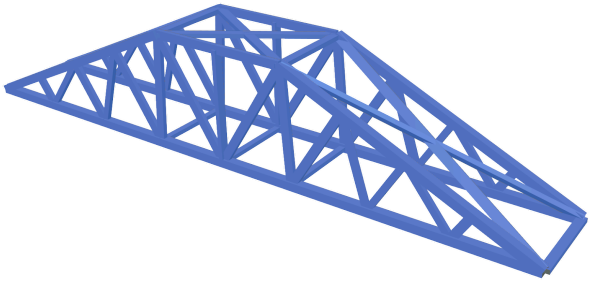


Figure 10. After conversion

### 7. Code Review

| **Severity (H M L Q)** | **Description of Defect** | Response |
|---|---|---|
| L, Q | Instead of appending to the empty corners list, would it be more efficient to define it upon declaration? | Changed the corners list declaration to define each corner upon declaration instead of appending them. |
| H | If the normalized vector is 0, a divide by 0 error occurs. | Implemented a check that verifies the normalized vector is not 0. |
| L | There is a lack of comments that explain the math, especially function-level docstrings. | Added docstrings to all functions. |
| M | Exception handling for arithmetic (e.g., div 0) for debugging purposes. | Added logging functionality for when division by zero errors arise. It was not necessary for the program to crash when this happened. |
| M-H | Duplicate use of the variable name “i”. | Changed one of the variable names. |
| L | Variable “arc_range” is not used. | Removed the unused variable. |
| L-M | magic numbers --> named constants | These numbers were converted to be based on the circle_size variable. |
| L | Typo: “verticies” instead of “vertices.” | Fixed the typo. |
| L-M | “dir” is a built-in Python function. | Renamed the variable to dir_vec |

Table 1. This table contains our fixes/responses to the code review.

**8. Future works**

Some potential future additions can be seen in the list below.

- Fix rotations of objects that vary in all angles.
- Allow the “2D” output to be usable in CAD vs. only being usable for 3D printing (.DXF file type).
- Add normal vectors to faces in the obj file to allow for materials to be added to meshes.
- Potential fix to Tinkercad .obj issues.
- Make the cylindrical member have rectangular faces on the sides and triangles on the caps.

**9. Conclusion**

Our conversion tool was designed with the goal of creating software that the Civil Engineering department could use for some of the programs they host. We were able to meet most of the original requirements on time, except for importing to Tinkercad. The structural bridge team was able to use our software for their design and expressed that it was working well. There were some extra functionalities that came up near the end, but we did not have the time to add these.

We solidified our linear algebra and trigonometry techniques during this project. Working on an actual project with changing features was a new learning experience to see through to the end. We were also able to refine our communication and group work skills.

Overall, our software, Structural 3D File Conversion Tool, was a good learning experience for us when working with a client and performing work for them.

**References**

[1] “Appendix D – File Format,” *Risa.com*, 2024. https://risa.com/risahelp/risa3d/Content/Appendices/Appendix%20D%20(3D).htm

[2] Pre-Engineering Software Corporation, “ModelSmart3D Registered User Resources - TSQ,” *Pre-engineering.com*, 2018. http://www.pre-engineering.com/modelsmart3d/tsq.html (accessed Dec. 13, 2024).

[3] Numpy, “NumPy,” *Numpy.org*, 2024. https://numpy.org/

[4] S. Belongie, “Rodrigues’ Rotation Formula,” *mathworld.wolfram.com*. https://mathworld.wolfram.com/RodriguesRotationFormula.html

[5] GeoGebra, “3D Calculator - GeoGebra,” *www.geogebra.org*. https://www.geogebra.org/3d?lang=en

[6] Wikipedia Contributors, “Wavefront .obj file,” *Wikipedia*, Dec. 10, 2018. https://en.wikipedia.org/wiki/Wavefront_.obj_file

**Appendix A**

# Structural 3D File Conversion Tool
# User Manual

## Overview

This program converts trusses and/or bridges developed in either ModelSmart 3D (.3dd) or RISA-3D (.r3d) into an OBJ file (.obj), which is much more widely supported in various types of 3D modeling software, such as AutoDesk Fusion 360, or even 3D printing slicers, like UltiMaker Cura. You can find the beta release here: https://github.com/CompliedBytes/structural_to_obj/releases

## Features

- Converts RISA-3D files (.r3d) to OBJ format.
- Converts Modelsmart files (.3dd) to OBJ format.
- Generates OBJ files compatible with Fusion 360.
- Supports batch conversion of multiple files and file types
- Can generate a single .obj or multiple .obj view projections for 3D printing.

## Compatibility

### Tested versions of RISA-3D

- 20.0501
- 22.01000

### Tested versions of Modelsmart 3D

- Version 4

### Supported Operating Systems

- Windows 10 & 11

# Installation

## Executable File

1. Run the executable.

## Batch File

1. Install Python.
2. Clone the repository.
3. Run the batch script.

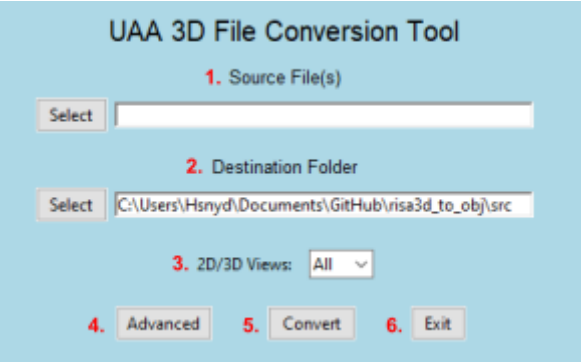
## Source Files

1. Install Python.
2. Clone/download the repository.
3. Install the dependencies

   ```
   pip install -r requirements.txt
   ```
4. Run the convert.py file located in the src folder.

# Instructions

## Primary Options




1. Source File(s)
   Select the .r3d/.3dd source files you'd like to convert into .obj files.

   Select multiple files if you'd like to convert more than one file at a time.

2. Destination Folder
   Select the folder where you'd like to save the converted files.

   The program defaults to the folder where the main program file is running.

3. 2D/3D Views
   The default option is **All**.

   2D
   Only 3D members that fall along each 2D plane (XY, YZ, XZ plane) are generated. This setting is useful for those who wish to 3D print their bridges.

   Note: In the Advanced settings menu, you can choose which 2D plane views are generated. The default is all three planar views.

   3D
   Generate the entire 3D model in a single file.

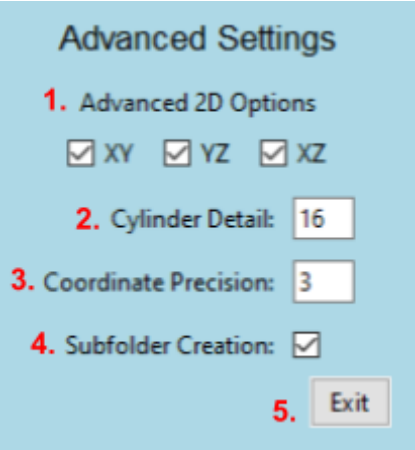
   All
   Generate the 2D planar views and the whole 3D model in separate files.

4. Advanced
   Click to open the advanced settings menu.

5. Convert
   Click to convert the source files.

6. Exit
   Click to exit the program.

## Advanced Settings Menu




1. Advanced 2D Options
   Specify which 2D planar views you'd like to generate.

   The default setting is all three planes.

2. Cylinder Detail
   Specify the number of side faces for cylinder generation. The more side faces, the more round the cylinders will appear.

   The default setting is 16 faces.

3. Coordinate Precision
   Specify the number of decimal places to round nodal coordinates. The more decimal places, the higher the level of precision in the generated .obj file.

   The default setting is 0.001 of a unit.

4. Subfolder Creation
   Create a folder for each source file and store all the generated .obj files in the newly created folder.

   The default setting is enabled.

5. Exit
   Click this button to exit the advanced settings menu.