**To Do: Output in form of input**

**Intro**

This is a guide that covers implementing an option to enable DFNgen to produce DFN output in the form of user defined fracture by coordinate files.

What we need to do is dump all the data for each polygon (Poly structure) to an output file, and then be able to read it back in. After the DFN completes, they polys listed by the list *finalFractures* in DFNmain.cpp are the fractures to keep.

**Input file option**

Use any of the user defined rectangles or ellipses options for examples and read Add\_new\_user\_options\_DFNgen.docx in the dfnworks-main git repo *Documentation* folder.

**Output polygons**

This section of code should be added to output.cpp. There are many files being written here and you can easily follow the example of one of the other functions for opening and writing a file.

This new function needs access to the *acceptedPoly* list, the *finalFractures* list, and the output path.

Open the file and loop for the number of items in the *finalFractures* list. *finalFractures* holds the indexes of the final polygons after the isolated fracture removal. finalFractures is used as the index to the accepted poly arry.

E.g. Poly a\_finalPolygon = acceptedPoly[finalFractures[i]]

Look at the poly structure structures.h and print all variables and vertices, to a file. Printing the vertices will require another loop for the amount of nodes the polygon has.

Be sure to write the number of nodes to the file before the vertices, this will help when reading the polygons back in.

**Reading the output back in as input**

A new function with a separate .cpp file and new .h file should be created for this. The user defined rectangle insertion function can be used as an example (see insertUserRects.cpp). This new function should look very similar to the function in insertUserRects.cpp.

Most input files are read into the program in readInput.cpp. I suggest the parsing of the polygon file to happen in readInput.cpp also, though it will also work if done in the new function. Parse the entire polygon file into something like a temporary *std::vector<Poly> tempPolyLIst* list. Use the *tempPolyList* and use it to assign *newPoly* in the new function (see below)*.*

E.g. Poly newPoly = tempPolyList[i] (See insertUserRects.cpp)

I suggest copying insertUserRects.cpp and using it as a template of sorts to start coding the new function.

The single example line of code above can replace lines 27 to 73.

A call to this new function should happen in DFNmain.cpp, either before or after the user defined rectangles and ellipses insertion takes place (Search for comment “User Defined Shapes Insertion”).

After line 73, the code will be almost exactly the same except that the area of the polygon and bounding boxes will not need to be calculated, as it should be already there in the polygon file.