The gcodepreview OpenSCAD library*

Author: William F. Adams
willadams at aol dot com

2024/11/11

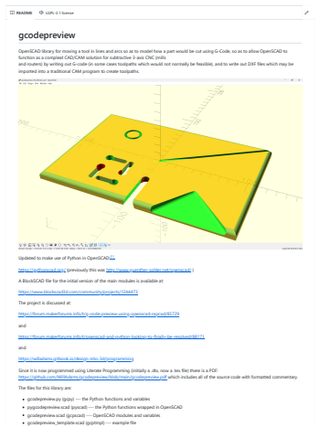
Abstract

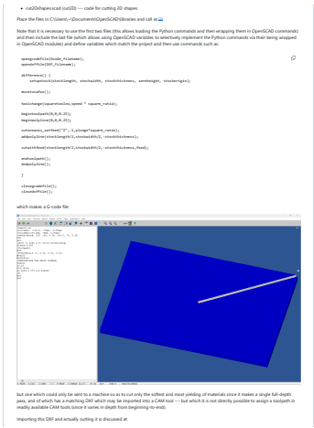
The gcodepreview library allows using OpenPythonSCAD to move a tool in lines and arcs and output dxf and G-code files so as to work as a CAD/CAM program for CNC.

Contents

*This file (gcodepreview) has version number v0.7, last revised 2024/11/11.

1 readme.md





```
1 rdme # gcodepreview
2 rdme
3 rdme OpenPythonSCAD library for moving a tool in lines and arcs so as to
       model how a part would be cut using G-Code, so as to allow
      OpenPythonSCAD to function as a compleat CAD/CAM solution for
      subtractive 3-axis CNC (mills and routers) by writing out G-code
       in addition to 3D modeling (in some cases toolpaths which would
       not normally be feasible), and to write out DXF files which may
       be imported into a traditional CAM program to create toolpaths.
4 rdme
5 rdme ![OpenSCAD Cut Joinery Module](https://raw.githubusercontent.com/
      WillAdams/gcodepreview/main/gcodepreview_unittests.png?raw=true)
6 rdme
7 rdme Updated to make use of Python in OpenSCAD:[^rapcad]
8 rdme
9 rdme [^rapcad]: Previous versions had used RapCAD, so as to take
      advantage of the writeln command, which has since been re-
      written in Python.
10 rdme
11 rdme https://pythonscad.org/ (previously this was http://www.guenther-
      sohler.net/openscad/ )
12 rdme
13 rdme A BlockSCAD file for the initial version of the
14 rdme main modules is available at:
15 rdme
16 rdme https://www.blockscad3d.com/community/projects/1244473
17 rdme
18 rdme The project is discussed at:
19 rdme
20 rdme https://forum.makerforums.info/t/g-code-preview-using-openscad-
      rapcad/85729
21 rdme
22 rdme and
23 rdme
24 rdme https://forum.makerforums.info/t/openscad-and-python-looking-to-
      finally-be-resolved/88171
25 rdme
26 rdme and
27 rdme
28 rdme https://willadams.gitbook.io/design-into-3d/programming
29 rdme
30 rdme Since it is now programmed using Literate Programming (initially a
       .dtx, now a .tex file) there is a PDF: https://github.com/
      WillAdams/gcodepreview/blob/main/gcodepreview.pdf which includes
       all of the source code with formatted commentary.
31 rdme
32 rdme The files for this library are:
33 rdme
34 rdme  - gcodepreview.py (gcpy) --- the Python functions and variables
35 rdme  - pygcodepreview.scad (pyscad) --- the Python functions wrapped in
          OpenSCAD
36 rdme  - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
37 rdme  - gcodepreview_template.scad (gcptmpl) --- example file
38 rdme  - cut2Dshapes.scad (cut2D) --- code for cutting 2D shapes
39 rdme
40 rdme If using from OpenPythonSCAD, place the files in C:\Users\\\~\
      Documents\OpenSCAD\libraries and call as:[^libraries]
41 rdme
```

```
42 rdme [^libraries]: C:\Users\\\~\Documents\RapCAD\libraries is deprecated
       since RapCAD is no longer needed since Python is now used for
      writing out files)
43 rdme
44 rdme     use <gcodepreview.py>;
45 rdme     use <pygcodepreview.scad>;
46 rdme     include <gcodepreview.scad>;
47 rdme
48 rdme Note that it is necessary to use the first two files (this allows
      loading the Python commands and then wrapping them in OpenSCAD
      commands) and then include the last file (which allows using
      OpenSCAD variables to selectively implement the Python commands
      via their being wrapped in OpenSCAD modules) and define
      variables which match the project and then use commands such as:
49 rdme
50 rdme     opengcodefile(Gcode_filename);
51 rdme     opendxffile(DXF_filename);
52 rdme
53 rdme     difference() {
54 rdme         setupstock(stockXwidth, stockYheight, stockZthickness,
                 zeroheight, stockzero);
55 rdme
56 rdme     movetosafez();
57 rdme
58 rdme     toolchange(squaretoolnum,speed * square_ratio);
59 rdme
60 rdme     begintoolpath(0,0,0.25);
61 rdme     beginpolyline(0,0,0.25);
62 rdme
63 rdme     cutoneaxis_setfeed("Z",-1,plunge*square_ratio);
64 rdme     addpolyline(stockXwidth/2,stockYheight/2,-stockZthickness);
65 rdme
66 rdme     cutwithfeed(stockXwidth/2,stockYheight/2,-stockZthickness,feed)
             ;
67 rdme
68 rdme     endtoolpath();
69 rdme     endpolyline();
70 rdme
71 rdme     }
72 rdme
73 rdme     closegcodefile();
74 rdme     closedxffile();
75 rdme
76 rdme which makes a G-code file:
77 rdme
78 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
      WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
79 rdme
80 rdme but one which could only be sent to a machine so as to cut only the
       softest and most yielding of materials since it makes a single
      full-depth pass, and of which has a matching DXF which may be
      imported into a CAM tool --- but which it is not directly
      possible to assign a toolpath in readily available CAM tools (
      since it varies in depth from beginning-to-end).
81 rdme
82 rdme Importing this DXF and actually cutting it is discussed at:
83 rdme
84 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
      /88617/14
85 rdme
86 rdme Alternately, gcodepreview.py may be placed in a Python library
      location and used directly from Python --- note that it is
      possible to use it from a "normal" Python when generating only
      DXFs.
87 rdme
88 rdme Tool numbers match those of tooling sold by Carbide 3D (ob. discl.,
       I work for them).
89 rdme
90 rdme Comments are included in the G-code to match those expected by
      CutViewer.
91 rdme
92 rdme A complete example file is: gcodepreview_template.scad Note that a
      Python template has since been developed as well, allowing usage
       without OpenSCAD code, and another example is
      openscad_gcodepreview_cutjoinery.tres.scad which is made from an
       OpenSCAD Graph Editor file:
93 rdme
94 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
```

```
githubusercontent.com/WillAdams/gcodepreview/main/
OSGE_cutjoinery.png?raw=true)
rdme
rdme Version 0.1 supports setting up stock, origin, rapid positioning,
    making cuts, and writing out matching G-code, and creating a DXF
     with polylines.
rdme
rdme Added features since initial upload:
rdme
rdme  - endpolyline(); --- this command allows ending one polyline so as
         to allow multiple lines in a DXF
rdme  - separate dxf files are written out for each tool where tool is
         ball/square/V and small/large (10/31/23)
rdme  - re-writing as a Literate Program using the LaTeX package docmfp
         (begun 4/12/24)
rdme  - support for additional tooling shapes such as dovetail and
         keyhole tools
rdme
rdme Version 0.2 adds support for arcs
rdme
rdme  - DXF: support for arcs (which may be used to make circles)
         (6/1/24)
rdme  - Specialty toolpaths such as Keyhole which may be used for
         dovetail as well as keyhole cutters
rdme
rdme Version 0.3
rdme
rdme  - Support for curves along the 3rd dimension
rdme  - support for roundover tooling
rdme
rdme Version 0.4
rdme
rdme  - Rewrite using literati documentclass, suppression of SVG code
rdme  - dxfrectangle (without G-code support)
rdme
rdme Version 0.5
rdme
rdme  - more shapes
rdme  - consolidate rectangles, arcs, and circles in gcodepreview.scad
rdme
rdme Version 0.6
rdme
rdme  - notes on modules
rdme  - change file for setupstock
rdme
rdme Version 0.61
rdme
rdme  - validate all code so that it runs without errors from sample
         file
rdme  - NEW: Note that this version is archived as gcodepreview-
         openscad_0_6.tex and the matching PDF is available as well
rdme
rdme  Version 0.7
rdme
rdme   - re-write completely in Python --- note that it is possible to
          use from within OpenPythonSCAD and an OpenSCAD wrapper is not
          functional at this time --- note that the OpenSCAD wrapper
          will need to be rewritten
rdme
rdme Possible future improvements:
rdme
rdme  - rewrite OpenSCAD wrapper
rdme  - restore support for additional tooling shapes (dovetail,
         roundover)
rdme  - support for additional tooling shapes such as tapered ball-nose
         tools or lollipop cutters or thread-cutting tools
rdme
rdme Note for G-code generation that it is up to the user to implement
    Depth per Pass so as to not take a single full-depth pass.
    Working from a DXF of course allows one to off-load such
    considerations to a specialized CAM tool.
rdme
rdme Deprecated feature:
rdme
rdme  - exporting SVGs --- while this was begun, it turns out that these
          would be written out upside down due to coordinate system
         differences between OpenSCAD/DXFs and SVGs requiring managing
         the inversion of the coordinate system (it is possible that
```

```
    METAPOST, which shares the same orientation and which can write
     out SVGs will be used instead for future versions)
```

2 gcodepreview

This library for OpenPythonSCAD works by using Python code as a back-end so as to persistently store and access variables, and to write out files while both modeling the motion of a 3-axis CNC machine and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL or STEP or other model format). There are multiple modes for this, doing so requires up to three files:

- A Python file: gcodepreview.py (`gcpy`) — this has variables in the traditional sense which may be used for tracking machine position and so forth. Note that where it is placed/loaded from will depend on whether it is imported into a Python file:
  `import gcodepreview_standalone as gcp`
  or used in an OpenSCAD file:
  `use <gcodepreview.py>;`
  with additional OpenSCAD modules which allow accessing it
- An OpenSCAD file: pygcodepreview.scad (`pyscad`) — which wraps the Python code in OpenSCAD (note that it too is included by `use <pygcodepreview.scad>`)
- An OpenSCAD file: gcodepreview.scad (`gcpscad`) — which uses the other two files and which is `include`d allowing it to access OpenSCAD variables for branching

Note that this architecture requires that many OpenSCAD modules are essentially "Dispatchers" which pass information from one aspect of the environment to another.

2.1 gcodepreviewtemplate

The various commands are shown all together in templates so as to provide examples of usage, and to ensure that the various files are used/included as necessary, all variables are set up with the correct names, and that files are opened before being written to, and that each is closed at the end.

Note that while the template files seem overly verbose, they specifically incorporate variables for each tool shape, possibly in two different sizes, and a feed rate parameter or ratio for each, which may be used (by setting a tool #) or ignored (by leaving the variable at zero (0).

It should be that this section is all the documentation which some users will need (and arguably is still too much). The balance of the document after this section shows all the code and implementation details.

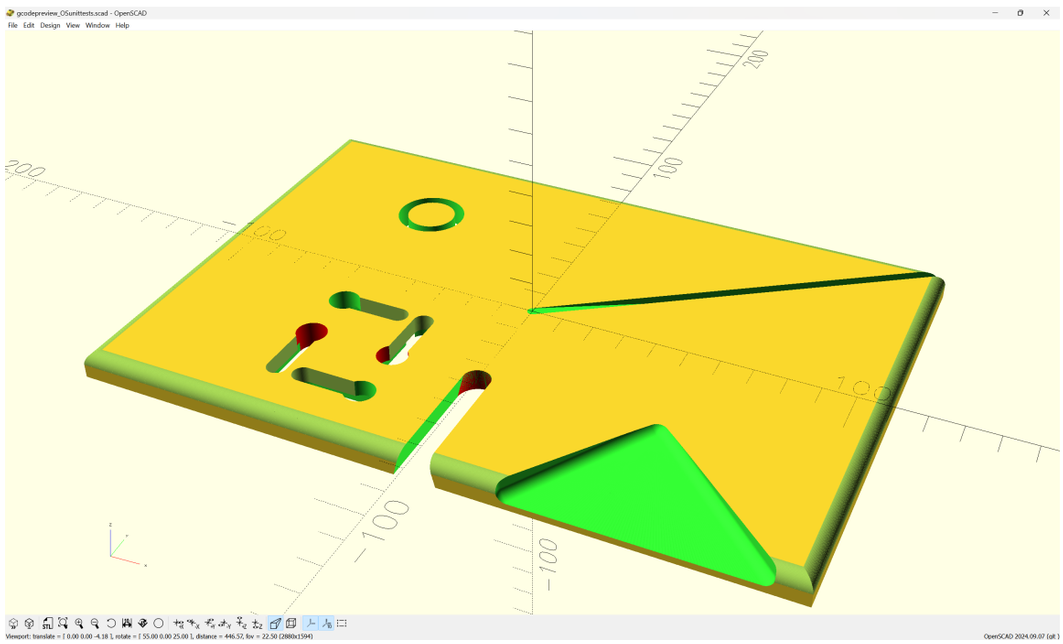
2.1.1 gcodepreviewtemplate.scad

```
1 gcptmpl //!OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>;
4 gcptmpl use <pygcodepreview.scad>;
5 gcptmpl include <gcodepreview.scad>;
6 gcptmpl
7 gcptmpl $fa = 2;
8 gcptmpl $fs = 0.125;
9 gcptmpl
10 gcptmpl /* [Stock] */
11 gcptmpl stockXwidth = 219;
12 gcptmpl /* [Stock] */
13 gcptmpl stockYheight = 150;
14 gcptmpl /* [Stock] */
15 gcptmpl stockZthickness = 8.35;
16 gcptmpl /* [Stock] */
17 gcptmpl zeroheight = "Top"; // [Top, Bottom]
18 gcptmpl /* [Stock] */
19 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
       ]
20 gcptmpl /* [Stock] */
21 gcptmpl retractheight = 9;
22 gcptmpl
23 gcptmpl /* [Export] */
24 gcptmpl Base_filename = "export";
25 gcptmpl /* [Export] */
26 gcptmpl generatedxf = true;
27 gcptmpl /* [Export] */
28 gcptmpl generategcode = true;
29 gcptmpl ///* [Export] */
30 gcptmpl //generatesvg = false;
31 gcptmpl
32 gcptmpl /* [CAM] */
33 gcptmpl toolradius = 1.5875;
34 gcptmpl /* [CAM] */
```

```
35 gcptmpl large_square_tool_num = 0; // [0:0,112:112,102:102,201:201]
36 gcptmpl /* [CAM] */
37 gcptmpl small_square_tool_num = 102; // [0:0,122:122,112:112,102:102]
38 gcptmpl /* [CAM] */
39 gcptmpl large_ball_tool_num = 0; // [0:0,111:111,101:101,202:202]
40 gcptmpl /* [CAM] */
41 gcptmpl small_ball_tool_num = 0; // [0:0,121:121,111:111,101:101]
42 gcptmpl /* [CAM] */
43 gcptmpl large_V_tool_num = 0; // [0:0,301:301,690:690]
44 gcptmpl /* [CAM] */
45 gcptmpl small_V_tool_num = 0; // [0:0,390:390,301:301]
46 gcptmpl /* [CAM] */
47 gcptmpl DT_tool_num = 0; // [0:0,814:814]
48 gcptmpl /* [CAM] */
49 gcptmpl KH_tool_num = 0; // [0:0,374:374,375:375,376:376,378]
50 gcptmpl /* [CAM] */
51 gcptmpl Roundover_tool_num = 0; // [56125:56125, 56142:56142,312:312,
        1570:1570]
52 gcptmpl /* [CAM] */
53 gcptmpl MISC_tool_num = 0; //
54 gcptmpl
55 gcptmpl /* [Feeds and Speeds] */
56 gcptmpl plunge = 100;
57 gcptmpl /* [Feeds and Speeds] */
58 gcptmpl feed = 400;
59 gcptmpl /* [Feeds and Speeds] */
60 gcptmpl speed = 16000;
61 gcptmpl /* [Feeds and Speeds] */
62 gcptmpl small_square_ratio = 0.75; // [0.25:2]
63 gcptmpl /* [Feeds and Speeds] */
64 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
65 gcptmpl /* [Feeds and Speeds] */
66 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
67 gcptmpl /* [Feeds and Speeds] */
68 gcptmpl large_V_ratio = 0.875; // [0.25:2]
69 gcptmpl /* [Feeds and Speeds] */
70 gcptmpl small_V_ratio = 0.625; // [0.25:2]
71 gcptmpl /* [Feeds and Speeds] */
72 gcptmpl DT_ratio = 0.75; // [0.25:2]
73 gcptmpl /* [Feeds and Speeds] */
74 gcptmpl KH_ratio = 0.75; // [0.25:2]
75 gcptmpl /* [Feeds and Speeds] */
76 gcptmpl RO_ratio = 0.5; // [0.25:2]
77 gcptmpl /* [Feeds and Speeds] */
78 gcptmpl MISC_ratio = 0.5; // [0.25:2]
79 gcptmpl
80 gcptmpl filename_gcode = str(Base_filename, ".nc");
81 gcptmpl filename_dxf = str(Base_filename);
82 gcptmpl
83 gcptmpl opengcodefile(filename_gcode);
84 gcptmpl opendxffile(filename_dxf);
85 gcptmpl
86 gcptmpl difference() {
87 gcptmpl setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
        stockzero);
88 gcptmpl
89 gcptmpl movetosafez();
90 gcptmpl
91 gcptmpl toolchange(small_square_tool_num,speed * small_square_ratio);
92 gcptmpl
93 gcptmpl begintoolpath(0,0,0.25);
94 gcptmpl
95 gcptmpl cutoneaxis_setfeed("Z",0,plunge*small_square_ratio);
96 gcptmpl
97 gcptmpl cutwithfeed(stockXwidth/2,stockYheight/2,-stockZthickness,feed);
98 gcptmpl dxfpolyline(getxpos(),getypos(),stockXwidth/2,stockYheight/2,
        small_square_tool_num);
99 gcptmpl
100 gcptmpl endtoolpath();
101 gcptmpl rapid(-(stockXwidth/4-stockYheight/16),stockYheight/4,0);
102 gcptmpl cutoneaxis_setfeed("Z",-stockZthickness,plunge*small_square_ratio);
103 gcptmpl
104 gcptmpl cutarcNECCdxf(-stockXwidth/4, stockYheight/4+stockYheight/16, -
        stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight
        /16, small_square_tool_num);
105 gcptmpl cutarcNWCCdxf(-(stockXwidth/4+stockYheight/16), stockYheight/4, -
        stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight
        /16, small_square_tool_num);
```

```
106 gcptmpl cutarcSWCCdxf(-stockXwidth/4, stockYheight/4-stockYheight/16, -
            stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight
            /16, small_square_tool_num);
107 gcptmpl cutarcSECCdxf(-(stockXwidth/4-stockYheight/16), stockYheight/4, -
            stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight
            /16, small_square_tool_num);
108 gcptmpl
109 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
110 gcptmpl toolchange(KH_tool_num,speed * KH_ratio);
111 gcptmpl rapid(-stockXwidth/8,-stockYheight/4,0);
112 gcptmpl
113 gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "N",
            stockYheight/8, KH_tool_num);
114 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
115 gcptmpl rapid(-stockXwidth/4,-stockYheight/4,0);
116 gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "S",
            stockYheight/8, KH_tool_num);
117 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
118 gcptmpl rapid(-stockXwidth/4,-stockYheight/8,0);
119 gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "E",
            stockYheight/8, KH_tool_num);
120 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
121 gcptmpl rapid(-stockXwidth/8,-stockYheight/8*3,0);
122 gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "W",
            stockYheight/8, KH_tool_num);
123 gcptmpl
124 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
125 gcptmpl toolchange(DT_tool_num,speed * DT_ratio);
126 gcptmpl rapid(0,-(stockYheight/2+tool_diameter(DT_tool_num,0)),0);
127 gcptmpl
128 gcptmpl cutoneaxis_setfeed("Z",-stockZthickness,plunge*DT_ratio);
129 gcptmpl cutwithfeed(0,-(stockYheight/4),-stockZthickness,feed*DT_ratio);
130 gcptmpl rapid(0,-(stockYheight/2+tool_diameter(DT_tool_num,0)),-
            stockZthickness);
131 gcptmpl
132 gcptmpl rapid(getxpos(),getypos(),stockZthickness);
133 gcptmpl toolchange(Roundover_tool_num, speed * RO_ratio);
134 gcptmpl rapid(-(stockXwidth/2),-(stockYheight/2),0);
135 gcptmpl cutoneaxis_setfeed("Z",-4.509,plunge*RO_ratio);
136 gcptmpl
137 gcptmpl cutroundovertool(-(stockXwidth/2++0.507/2), -(stockYheight
            /2+0.507/2), -4.509, stockXwidth/2+0.507/2, -(stockYheight
            /2+0.507/2), -4.509, 0.507/2, 4.509);
138 gcptmpl
139 gcptmpl cutroundover(stockXwidth/2+0.507/2, -(stockYheight/2+0.507/2),
            -4.509, stockXwidth/2+0.507/2, stockYheight/2+0.507/2, -4.509,
            1570);
140 gcptmpl cutroundover(stockXwidth/2+0.507/2, stockYheight/2+0.507/2, -4.509,
             -(stockXwidth/2+0.507/2), stockYheight/2+0.507/2, -4.509, 1570)
            ;
141 gcptmpl cutroundover(-(stockXwidth/2+0.507/2), stockYheight/2+0.507/2,
            -4.509, -(stockXwidth/2+0.507/2), -(stockYheight/2+0.507/2),
            -4.509, 1570);
142 gcptmpl
143 gcptmpl //for (i = [0 : abs(1) : 80]) {
144 gcptmpl //  cutwithfeed(stockXwidth/4,-stockYheight/4,-stockZthickness/4,
            feed);
145 gcptmpl //  cutwithfeed(stockXwidth/8+(stockXwidth/256*i),-stockYheight/2,-
            stockZthickness*3/4,feed);
146 gcptmpl //  }
147 gcptmpl
148 gcptmpl hull(){
149 gcptmpl   cutwithfeed(stockXwidth/4,-stockYheight/4,-stockZthickness/4,feed
              );
150 gcptmpl   cutwithfeed(stockXwidth/8,-stockYheight/2,-stockZthickness*3/4,
              feed);
151 gcptmpl   cutwithfeed(stockXwidth/8+(stockXwidth*0.3125),-stockYheight/2,-
              stockZthickness*3/4,feed);
152 gcptmpl   }
153 gcptmpl }
154 gcptmpl
155 gcptmpl closegcodefile();
156 gcptmpl closedxffile();
```

Which cuts as:



Some comments on the template:

- minimal — it is intended as a framework for a minimal working example (MWE) — it should be possible to comment out unused portions and so arrive at code which tests any aspect of this project
- compleat — a quite wide variety of tools are listed (and probably more will be added in the future), but pre-defining them and having these "hooks" seems the easiest (non-object-oriented) mechanism to handle everything
- shortcuts — as the last example shows, while in real life it is necessary to make many passes with a tool, an expedient shortcut is to forgo the `loop` operation and just use a `hull()` operation

Further features will be added to the template, and the main image updated to reflect the capabilities of the system.

2.1.2 gcodepreviewtemplate.py

Note that with the vo.7 re-write, it is possible to directly use the underlying Python code directly.

```
1 gcptmplpy #!/usr/bin/env python
2 gcptmplpy
3 gcptmplpy # getting openscad functions into namespace
4 gcptmplpy #https://github.com/gsohler/openscad/issues/39
5 gcptmplpy from openscad import *
6 gcptmplpy
7 gcptmplpy import sys
8 gcptmplpy try:
9 gcptmplpy     if 'gcodepreview' in sys.modules:
10 gcptmplpy         del sys.modules['gcodepreview']
11 gcptmplpy except AttributeError:
12 gcptmplpy     pass
13 gcptmplpy
14 gcptmplpy #Below command only needed if using withing OpenPythonSCAD
15 gcptmplpy from gcodepreview import *
16 gcptmplpy
17 gcptmplpy #fa = 2
18 gcptmplpy #fs = 0.125
19 gcptmplpy
20 gcptmplpy # [Export] */
21 gcptmplpy Base_filename = "export"
22 gcptmplpy # [Export] */
23 gcptmplpy generatedxf = True
24 gcptmplpy # [Export] */
25 gcptmplpy generategcode = True
26 gcptmplpy
27 gcptmplpy # [Stock] */
28 gcptmplpy stockXwidth = 219
29 gcptmplpy # [Stock] */
30 gcptmplpy stockYheight = 150
31 gcptmplpy # [Stock] */
32 gcptmplpy stockZthickness = 8.35
```

```
gcptmplpy # [Stock] */
gcptmplpy zeroheight = "Top" # [Top, Bottom]
gcptmplpy # [Stock] */
gcptmplpy stockzero = "Center" # [Lower-Left, Center-Left, Top-Left, Center]
gcptmplpy # [Stock] */
gcptmplpy retractheight = 9
gcptmplpy
gcptmplpy # [CAM] */
gcptmplpy toolradius = 1.5875
gcptmplpy # [CAM] */
gcptmplpy large_square_tool_num = 0 # [0:0,112:112,102:102,201:201]
gcptmplpy # [CAM] */
gcptmplpy small_square_tool_num = 102 # [0:0,122:122,112:112,102:102]
gcptmplpy # [CAM] */
gcptmplpy large_ball_tool_num = 0 # [0:0,111:111,101:101,202:202]
gcptmplpy # [CAM] */
gcptmplpy small_ball_tool_num = 0 # [0:0,121:121,111:111,101:101]
gcptmplpy # [CAM] */
gcptmplpy large_V_tool_num = 0 # [0:0,301:301,690:690]
gcptmplpy # [CAM] */
gcptmplpy small_V_tool_num = 0 # [0:0,390:390,301:301]
gcptmplpy # [CAM] */
gcptmplpy DT_tool_num = 0 # [0:0,814:814]
gcptmplpy # [CAM] */
gcptmplpy KH_tool_num = 0 # [0:0,374:374,375:375,376:376,378]
gcptmplpy # [CAM] */
gcptmplpy Roundover_tool_num = 0 # [56125:56125, 56142:56142,312:312,
              1570:1570]
gcptmplpy # [CAM] */
gcptmplpy MISC_tool_num = 0 #
gcptmplpy
gcptmplpy # [Feeds and Speeds] */
gcptmplpy plunge = 100
gcptmplpy # [Feeds and Speeds] */
gcptmplpy feed = 400
gcptmplpy # [Feeds and Speeds] */
gcptmplpy speed = 16000
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_square_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy large_V_ratio = 0.875 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_V_ratio = 0.625 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy DT_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy KH_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy RO_ratio = 0.5 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy MISC_ratio = 0.5 # [0.25:2]
gcptmplpy
gcptmplpy gcp = gcodepreview(Base_filename, #"export", basefilename
gcptmplpy                    True, #generategcode
gcptmplpy                    True, #generatedxf
gcptmplpy                    stockXwidth,
gcptmplpy                    stockYheight,
gcptmplpy                    stockZthickness,
gcptmplpy                    zeroheight,
gcptmplpy                    stockzero,
gcptmplpy                    retractheight,
gcptmplpy                    large_square_tool_num,
gcptmplpy                    toolradius,
gcptmplpy                    plunge,
gcptmplpy                    feed,
gcptmplpy                    speed)
gcptmplpy
gcptmplpy gcp.opengcodefile(Base_filename)
gcptmplpy gcp.opendxffiles(Base_filename)
gcptmplpy
gcptmplpy gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top","
              Center",retractheight)
gcptmplpy
gcptmplpy gcp.movetosafeZ()
```

```
109 gcptmplpy
110 gcptmplpy gcp.toolchange(102,10000)
111 gcptmplpy
112 gcptmplpy #gcp.rapidXY(6,12)
113 gcptmplpy gcp.rapidZ(0)
114 gcptmplpy
115 gcptmplpy #print (gcp.xpos())
116 gcptmplpy #print (gcp.ypos())
117 gcptmplpy #psetzpos(7)
118 gcptmplpy #gcp.setzpos(-12)
119 gcptmplpy #print (gcp.zpos())
120 gcptmplpy
121 gcptmplpy #print ("X", str(gcp.xpos()))
122 gcptmplpy #print ("Y", str(gcp.ypos()))
123 gcptmplpy #print ("Z", str(gcp.zpos()))
124 gcptmplpy
125 gcptmplpy toolpaths = gcp.currenttool()
126 gcptmplpy
127 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
                stockYheight/2, -stockZthickness))
128 gcptmplpy
129 gcptmplpy gcp.rapidZ(retractheight)
130 gcptmplpy gcp.toolchange(201,10000)
131 gcptmplpy gcp.rapidXY(0, stockYheight/16)
132 gcptmplpy gcp.rapidZ(0)
133 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
                stockYheight/2, -stockZthickness))
134 gcptmplpy
135 gcptmplpy gcp.setzpos(retractheight)
136 gcptmplpy gcp.toolchange(202,10000)
137 gcptmplpy gcp.rapidXY(0, stockYheight/8)
138 gcptmplpy gcp.rapidZ(0)
139 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
                stockYheight/2, -stockZthickness))
140 gcptmplpy
141 gcptmplpy gcp.setzpos(retractheight)
142 gcptmplpy gcp.toolchange(101,10000)
143 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
144 gcptmplpy gcp.rapidZ(0)
145 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
                stockYheight/2, -stockZthickness))
146 gcptmplpy
147 gcptmplpy gcp.setzpos(retractheight)
148 gcptmplpy gcp.toolchange(390,10000)
149 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
150 gcptmplpy gcp.rapidZ(0)
151 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
                stockYheight/2, -stockZthickness))
152 gcptmplpy gcp.setzpos(retractheight)
153 gcptmplpy
154 gcptmplpy gcp.toolchange(301,10000)
155 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
156 gcptmplpy gcp.rapidZ(0)
157 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*2,
                stockYheight/2, -stockZthickness))
158 gcptmplpy
159 gcptmplpy #gcp.setzpos(retractheight)
160 gcptmplpy #gcp.toolchange(102,10000)
161 gcptmplpy #gcp.rapidXY(stockXwidth/4+stockYheight/16, -(stockYheight/4))
162 gcptmplpy #gcp.rapidZ(0)
163 gcptmplpy ##arcloop(barc, earc, xcenter, ycenter, radius)
164 gcptmplpy #gcp.settzpos(stockZthickness/90)
165 gcptmplpy #toolpaths = toolpaths.union(gcp.arcloop(0, 90, stockXwidth/4, -
                stockYheight/4, stockYheight/16))
166 gcptmplpy
167 gcptmplpy gcp.setzpos(retractheight)
168 gcptmplpy gcp.toolchange(102,10000)
169 gcptmplpy gcp.rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
                stockYheight/8)
170 gcptmplpy gcp.rapidZ(0)
171 gcptmplpy #gcp.settzpos(stockZthickness/90)
172 gcptmplpy #toolpaths = toolpaths.union(gcp.arcloop(0, 90, stockXwidth/4+
                stockYheight/8, stockYheight/8, stockYheight/16))
173 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcNECCdxf(stockXwidth/4+
                stockYheight/8, stockYheight/8+stockYheight/16, -stockZthickness
                , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
                )
174 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcNWCCdxf(stockXwidth/4+
```

```
            stockYheight/8-stockYheight/16, stockYheight/8, -stockZthickness
            , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
            )
175 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcSWCCdxf(stockXwidth/4+
            stockYheight/8, stockYheight/8-stockYheight/16, -stockZthickness
            , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
            )
176 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcSECCdxf(stockXwidth/4+
            stockYheight/8+stockYheight/16, stockYheight/8, -stockZthickness
            , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
            )
177 gcptmplpy
178 gcptmplpy #a = gcp.currenttool()
179 gcptmplpy #arcbegin = a.translate([64.37357214209116, -37.33638368965047,-
            stockZthickness])
180 gcptmplpy #arcend = a.translate([55.16361631034953, -28.12642785790883,-
            stockZthickness])
181 gcptmplpy #toolpaths = toolpaths.union(arcbegin)
182 gcptmplpy #toolpaths = toolpaths.union(arcend)
183 gcptmplpy
184 gcptmplpy #cu = cube([10,20,30])
185 gcptmplpy #c = cu.translate([0,0,gcp.zpos()])
186 gcptmplpy
187 gcptmplpy part = gcp.stock.difference(toolpaths)
188 gcptmplpy
189 gcptmplpy #output(gcp.stock)
190 gcptmplpy #output(gcp.currenttool())
191 gcptmplpy #output(test)
192 gcptmplpy output(part)
193 gcptmplpy #output(toolpaths)
194 gcptmplpy #output (arc)
195 gcptmplpy
196 gcptmplpy gcp.setzpos(retractheight)
197 gcptmplpy
198 gcptmplpy gcp.closegcodefile()
199 gcptmplpy gcp.closedxffile()
200 gcptmplpy #gcp.closedxffiles()
```

2.2 Implementation files and gcodepreview class

Each file will begin with a suitable comment indicating the file type and suitable notes/comments:

```
  1 gcpy #!/usr/bin/env python
  2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe"  --trust-
         python
  3 gcpy #Currently tested with 2024.09.23 and Python 3.11
  4 gcpy #gcodepreview 0.7, for use with OpenPythonSCAD,
  5 gcpy #if using from OpenPythonSCAD see gcodepreview.scad
  6 gcpy
  7 gcpy # getting openscad functions into namespace
  8 gcpy #https://github.com/gsohler/openscad/issues/39
  9 gcpy from openscad import *
 10 gcpy
 11 gcpy # add math functions (using radians by default, convert to degrees
         where necessary)
 12 gcpy import math
```

```
1 pyscad //!OpenSCAD
2 pyscad
3 pyscad //gcodepreview 0.7, see gcodepreview.scad
```

```
1 gcpscad //!OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview 0.7
4 gcpscad //
5 gcpscad //used via use <gcodepreview.py>;
6 gcpscad //         use <pygcodepreview.scad>;
7 gcpscad //         include <gcodepreview.scad>;
8 gcpscad //
```

If all functions are to be handled within Python, then they will need to be gathered into a class which contains them and which may be initialized so as to define shared variables, and then there will need to be objects/commands for each aspect of the program, each of which will initialize

needed variables and will contain appropriate commands. Note that they will be divided between mandatory and optional functions/variables/objects:

- Mandatory
  - stocksetup:
    - stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero, retractheight
  - gcpfiles:
    - basefilename, generatedxf, generategcode
  - largesquaretool:
    - large_square_tool_num, toolradius, plunge, feed, speed
- Optional
  - smallsquaretool:
    - small_square_tool_num, small_square_ratio
  - largeballtool:
    - large_ball_tool_num, large_ball_ratio
  - largeVtool:
    - large_V_tool_num, large_V_ratio
  - smallballtool:
    - small_ball_tool_num, small_ball_ratio
  - smallVtool:
    - small_V_tool_num, small_V_ratio
  - DTtool:
    - DT_tool_num, DT_ratio
  - KHtool:
    - KH_tool_num, KH_ratio
  - Roundovertool:
    - Roundover_tool_num, RO_ratio
  - misctool:
    - MISC_tool_num, MISC_ratio

gcodepreview The first class which is defined is `gcodepreview` which includes an `init` method which allows passing in the variables which will be used by the other methods in this class.

```
14 gcpy class gcodepreview:
15 gcpy
16 gcpy     def __init__(self, basefilename = "export",
17 gcpy                  generategcode = False,
18 gcpy                  generatedxf = False,
19 gcpy                  stockXwidth = 25,
20 gcpy                  stockYheight = 25,
21 gcpy                  stockZthickness = 1,
22 gcpy                  zeroheight = "Top",
23 gcpy                  stockzero = "Lower-left" ,
24 gcpy                  retractheight = 6,
25 gcpy                  currenttoolnum = 102,
26 gcpy                  toolradius = 3.175,
27 gcpy                  plunge = 100,
28 gcpy                  feed = 400,
29 gcpy                  speed = 10000):
30 gcpy         self.basefilename = basefilename
31 gcpy         self.generategcode = generategcode
32 gcpy         self.generatedxf = generatedxf
33 gcpy         self.stockXwidth = stockXwidth
34 gcpy         self.stockYheight = stockYheight
35 gcpy         self.stockZthickness = stockZthickness
36 gcpy         self.zeroheight = zeroheight
37 gcpy         self.stockzero = stockzero
38 gcpy         self.retractheight = retractheight
39 gcpy         self.currenttoolnum = currenttoolnum
40 gcpy         self.toolradius = toolradius
41 gcpy         self.plunge = plunge
42 gcpy         self.feed = feed
43 gcpy         self.speed = speed
44 gcpy #         global toolpaths
45 gcpy #         self.toolpaths = cylinder(1.5875, 12.7)
46 gcpy         global generatedxfs
47 gcpy         self.generatedxfs = False
```

2.2.1 Output files

The gcodepreview class will write out DXF and G-code files.

2.2.1.1 G-code and modules and commands Each module/command will write out certain G-code commands:

Command/Module	G-code
opengcodefile(...); setupstock(...)	(export.nc) (stockMin: -109.5, -75mm, -8.35mm) (stockMax:109.5mm, 75mm, 0.00mm) (STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35) G90 G21
movetosafez()	(Move to safe Z to avoid workholding) G53G0Z-5.000
toolchange(...);	(TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S16000

cutoneaxis_setfeed(...);
 	
cutwithfeed(...);
	(PREPOSITION FOR RAPID PLUNGE)
	G0X0Y0
	Z0.25
	G1Z0F100
	G1 X109.5 Y75 Z-8.35F400
	Z9

closegcodefile();	M05 M02

Conversely, the G-code commands which are supported are generated by the following modules:

G-code	Command/Module
(Design File:) (stockMin:0.00mm, -152.40mm, -34.92mm) (stockMax:109.50mm, -77.40mm, 0.00mm) (STOCK/BLOCK,109.50, 75.00, 34.92,0.00, 152.40, 34.92) G90 G21	opengcodefile(...); setupstock(...)
(Move to safe Z to avoid workholding) G53G0Z-5.000	movetosafez()
(Toolpath: Contour Toolpath 1) M05 (TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S10000	toolchange(...);
(PREPOSITION FOR RAPID PLUNGE)	writecomment(...)
G0X0.000Y-152.400 Z0.250	rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 X57.918Y16.302Z-0.726 Y22.023Z-1.023 X61.190Z-0.681 Y21.643 X57.681 Z12.700	cutwithfeed(...); cutwithfeed(...);
M05 M02	closegcodefile();

The implication here is that it should be possible to read in a G-code file, and for each line/command instantiate a matching command so as to create a 3D model/preview of the file. One possible option would be to make specialized commands for movement which correspond to the various axis combinations (XYZ, XY, XZ, YZ, X, Y, Z).

2.2.1.2 DXF Elements in DXFs are represented as lines or arcs. A minimal file showing both:

```
0
SECTION
2
ENTITIES
0
LWPOLYLINE
90
2
70
0
43
0
10
-31.375
20
-34.9152
10
-31.375
20
-18.75
0
ARC
10
-54.75
20
-37.5
40
4
50
0
51
90
0
ENDSEC
0
EOF
```

The class `gcodepreview` will need additional commands for opening files:

```
49 gcpy     def opengcodefile(self, basefilename = "export"):
50 gcpy         if self.generategcode == True:
51 gcpy             self.gcodefilename = basefilename + ".nc"
52 gcpy             self.gc = open(self.gcodefilename, "w")
53 gcpy
54 gcpy     def opendxffile(self, basefilename = "export"):
55 gcpy         global generatedxfs
56 gcpy         if self.generatedxf == True:
57 gcpy             self.generatedxfs = False
58 gcpy             self.dxffilename = basefilename + ".dxf"
59 gcpy             self.dxf = open(self.dxffilename, "w")
60 gcpy             self.dxfpreamble(-1)
61 gcpy
62 gcpy     def opendxffiles(self, basefilename = "export",
63 gcpy                      large_square_tool_num = 0,
64 gcpy                      small_square_tool_num = 0,
65 gcpy                      large_ball_tool_num = 0,
66 gcpy                      small_ball_tool_num = 0,
67 gcpy                      large_V_tool_num = 0,
68 gcpy                      small_V_tool_num = 0,
69 gcpy                      DT_tool_num = 0,
70 gcpy                      KH_tool_num = 0,
71 gcpy                      Roundover_tool_num = 0,
72 gcpy                      MISC_tool_num = 0):
73 gcpy         global generatedxfs
74 gcpy         self.generatedxfs = True
75 gcpy         self.large_square_tool_num = large_square_tool_num
76 gcpy         self.small_square_tool_num = small_square_tool_num
77 gcpy         self.large_ball_tool_num = large_ball_tool_num
78 gcpy         self.small_ball_tool_num = small_ball_tool_num
79 gcpy         self.large_V_tool_num = large_V_tool_num
80 gcpy         self.small_V_tool_num = small_V_tool_num
81 gcpy         self.DT_tool_num = DT_tool_num
82 gcpy         self.KH_tool_num = KH_tool_num
83 gcpy         self.Roundover_tool_num = Roundover_tool_num
84 gcpy         self.MISC_tool_num = MISC_tool_num
85 gcpy         if self.generatedxf == True:
86 gcpy             if (large_square_tool_num > 0):
```

```
gcpy                self.dxflgsqfilename = basefilename + str(large_square_tool_num) + ".dxf"
gcpy                print("Opening ", str(self.dxflgsqfilename))
gcpy                self.dxflgsq = open(self.dxflgsqfilename, "w")
gcpy                self.dxfpreamble(large_square_tool_num)
gcpy            if (small_square_tool_num > 0):
gcpy                print("Opening small square")
gcpy                self.dxfsmsqfilename = basefilename + str(small_square_tool_num) + ".dxf"
gcpy                self.dxfsmsq = open(self.dxfsmsqfilename, "w")
gcpy                self.dxfpreamble(small_square_tool_num)
gcpy            if (large_ball_tool_num > 0):
gcpy                print("Opening large ball")
gcpy                self.dxflgblfilename = basefilename + str(large_ball_tool_num) + ".dxf"
gcpy                self.dxflgbl = open(self.dxflgblfilename, "w")
gcpy                self.dxfpreamble(large_ball_tool_num)
gcpy            if (small_ball_tool_num > 0):
gcpy                print("Opening small ball")
gcpy                self.dxfsmblfilename = basefilename + str(small_ball_tool_num) + ".dxf"
gcpy                self.dxfsmbl = open(self.dxfsmblfilename, "w")
gcpy                self.dxfpreamble(small_ball_tool_num)
gcpy            if (large_V_tool_num > 0):
gcpy                print("Opening large V")
gcpy                self.dxflgVfilename = basefilename + str(large_V_tool_num) + ".dxf"
gcpy                self.dxflgV = open(self.dxflgVfilename, "w")
gcpy                self.dxfpreamble(large_V_tool_num)
gcpy            if (small_V_tool_num > 0):
gcpy                print("Opening small V")
gcpy                self.dxfsmVfilename = basefilename + str(small_V_tool_num) + ".dxf"
gcpy                self.dxfsmV = open(self.dxfsmVfilename, "w")
gcpy                self.dxfpreamble(small_V_tool_num)
gcpy            if (DT_tool_num > 0):
gcpy                print("Opening DT")
gcpy                self.dxfDTfilename = basefilename + str(DT_tool_num) + ".dxf"
gcpy                self.dxfDT = open(self.dxfDTfilename, "w")
gcpy                self.dxfpreamble(DT_tool_num)
gcpy            if (KH_tool_num > 0):
gcpy                print("Opening KH")
gcpy                self.dxfKHfilename = basefilename + str(KH_tool_num) + ".dxf"
gcpy                self.dxfKH = open(self.dxfKHfilename, "w")
gcpy                self.dxfpreamble(KH_tool_num)
gcpy            if (Roundover_tool_num > 0):
gcpy                print("Opening Rt")
gcpy                self.dxfRtfilename = basefilename + str(Roundover_tool_num) + ".dxf"
gcpy                self.dxfRt = open(self.dxfRtfilename, "w")
gcpy                self.dxfpreamble(Roundover_tool_num)
gcpy            if (MISC_tool_num > 0):
gcpy                print("Opening Mt")
gcpy                self.dxfMtfilename = basefilename + str(MISC_tool_num) + ".dxf"
gcpy                self.dxfMt = open(self.dxfMtfilename, "w")
gcpy                self.dxfpreamble(MISC_tool_num)
```

Note that the commands which interact with files include checks to see if said files are being generated.

writeln The original implementation in RapSCAD used a command `writeln` — fortunately, this command is easily re-created in Python. Note that the dxf commands will be wrapped up with `if/elif` blocks which will write to additional file(s) based on tool number as set up above.

```
gcpy    def writegc(self, *arguments):
gcpy        line_to_write = ""
gcpy        for element in arguments:
gcpy            line_to_write += element
gcpy        self.gc.write(line_to_write)
gcpy        self.gc.write("\n")
gcpy
gcpy    def writedxf(self, toolnumber, *arguments):
gcpy        line_to_write = ""
gcpy        for element in arguments:
gcpy            line_to_write += element
```

```
148 gcpy          if self.generatedxf == True:
149 gcpy              self.dxf.write(line_to_write)
150 gcpy              self.dxf.write("\n")
151 gcpy          if self.generatedxfs == True:
152 gcpy              self.writedxfs(toolnumber, arguments)
153 gcpy
154 gcpy      def writedxfs(self, toolnumber, *arguments):
155 gcpy          print("Processing␣writing␣toolnumber", toolnumber)
156 gcpy          line_to_write = ""
157 gcpy          for element in arguments:
158 gcpy              line_to_write += element
159 gcpy          if (toolnumber == 0):
160 gcpy              return
161 gcpy          elif self.generatedxfs == True:
162 gcpy              if (self.large_square_tool_num > 0):
163 gcpy                  self.dxflgsq.write(line_to_write)
164 gcpy                  self.dxflgsq.write("\n")
165 gcpy              if (self.small_square_tool_num > 0):
166 gcpy                  self.dxfsmsq.write(line_to_write)
167 gcpy                  self.dxfsmsq.write("\n")
168 gcpy              if (self.large_ball_tool_num > 0):
169 gcpy                  self.dxflgbl.write(line_to_write)
170 gcpy                  self.dxflgbl.write("\n")
171 gcpy              if (self.small_ball_tool_num > 0):
172 gcpy                  self.dxfsmbl.write(line_to_write)
173 gcpy                  self.dxfsmbl.write("\n")
174 gcpy              if (self.large_V_tool_num > 0):
175 gcpy                  self.dxflgV.write(line_to_write)
176 gcpy                  self.dxflgV.write("\n")
177 gcpy              if (self.small_V_tool_num > 0):
178 gcpy                  self.dxfsmV.write(line_to_write)
179 gcpy                  self.dxfsmV.write("\n")
180 gcpy              if (self.DT_tool_num > 0):
181 gcpy                  self.dxfDT.write(line_to_write)
182 gcpy                  self.dxfDT.write("\n")
183 gcpy              if (self.KH_tool_num > 0):
184 gcpy                  self.dxfKH.write(line_to_write)
185 gcpy                  self.dxfKH.write("\n")
186 gcpy              if (self.Roundover_tool_num > 0):
187 gcpy                  self.dxfRt.write(line_to_write)
188 gcpy                  self.dxfRt.write("\n")
189 gcpy              if (self.MISC_tool_num > 0):
190 gcpy                  self.dxfMt.write(line_to_write)
191 gcpy                  self.dxfMt.write("\n")
```

which commands will accept a series of arguments and then write them out to a file object for the appropriate file.

2.3 Module Naming Convention

Note that as a convention, where it is necessary for a module to coordinate between Python and OpenSCAD, in certain cases it will be necessary for there to be three separate versions: a Python definition for the manipulation of Python variables and any file routines, originally these were identified as `p<foo>`, but with the use of an object-oriented programming style and dot notation, since v0.7 they will be identified as `gcp.foo` (where `gcp` is the identifier used to import the class); while an `o<foo>` OpenSCAD module which will wrap up the Python function call, and lastly a <foo> OpenSCAD module which will be `<include>`d so as to be able to make use of OpenSCAD variables.

Number will be abbreviated as `num` rather than `no`.

In natural languages such as English, there is an order to various parts of speech such as adjectives — since various prefixes and suffixes will be used for module names, having a consistent ordering/usage will help in consistency and make expression clearer. The ordering should be: sequence, action, function, parameter, filetype, and where possible a hierarchy of large/general to small/specific should be maintained.

- Both prefix and suffix
  - `dxf` (action (write out dxf file), filetype)
- Prefixes
  - `generate` (action)
  - `cut` (action — create 3D object)
  - `move` (action)
  - `rapid` (action — create 3D object so as to show a collision)
  - `open` (action)

  - close (action)
  - set (action/function)

- Nouns
  - arc
  - line

- Suffixes
  - feed (parameter)
  - gcode/gc (filetype)

In particular, this means that the basic cut... command exists (or potentially exists) in the following forms and has matching versions which :

	line			arc		
	cut	dxf	gcode	cut	dxf	gcode
cut	cutline			cutarc		
dxf	cutlinedxf	dxfline		cutarcdxf	dxfarc	
gcode			cutlinedxfgc			cutarcdxfgc
OpenSCAD			linedxfgc			arcdxfgc

while OpenPythonSCAD requires that the current toolpath be returned, stored in a variable, which can then be subtracted from the stock, using OpenSCAD will instead have the toolpaths output in a structure which is differenced from the stock.

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)
- identify which aspect of the project structure is being worked with (cut(ting), dxf, gcode, tool, etc.) and esp. note the use of o(penscad) and p(ython) as prefixes, though the latter is not necessary for definitions within the gcodepreview class which will normally be imported as gcp so that module <foo> will be called as gcp.<foo>

Structurally, this will typically look like:

```
The user-facing module is \DescribeRoutine{FOOBAR}

\lstset{firstnumber=\thegcpscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module FOOBAR(...) {
    oFOOBAR(...);
}

\end{writecode}
\addtocounter{gcpscad}{4}

which calls the internal OpenSCAD Module \DescribeSubroutine{FOOBAR}{oFOOBAR}

\begin{writecode}{a}{pygcodepreview.scad}{scad}
module oFOOBAR(...) {
    pFOOBAR(...);
 }

\end{writecode}
\addtocounter{pyscad}{4}

which in turn calls the internal Python definitioon \DescribeSubroutine{FOOBAR}{pFOOBAR}

\lstset{firstnumber=\thegcpy}
\begin{writecode}{a}{gcodepreview.py}{python}
def pFOOBAR (...)
    ...

\end{writecode}
\addtocounter{gcpy}{3}
```

Further note that this definition will not be necessary for some later modules since they are in turn calling internal modules which already use this structure.

Another consideration is that all commands which write files will check to see if a given filetype is enabled or no.

2.3.1 Initial Modules

setupstock
gcodepreview

The first such routine, (actually a subroutine, see setupstock) gcodepreview will be appropriately enough, to set up the stock, and perform other initializations — initially, the only thing done in Python was to set the value of the persistent (Python) variables, but the rewritten standalone Python version does everything.

gcp.setupstock

The Python code, gcp.setupstock requires that the user set parameters for stock dimensions and so forth, and will create comments in the G-code which incorporate the stock dimensions and its position relative to the zero as set relative to the stock.

```
193 gcpy     def setupstock(self, stockXwidth,
194 gcpy                    stockYheight,
195 gcpy                    stockZthickness,
196 gcpy                    zeroheight,
197 gcpy                    stockzero,
198 gcpy                    retractheight):
199 gcpy         global mpx
200 gcpy         mpx = float(0)
201 gcpy         global mpy
202 gcpy         mpy = float(0)
203 gcpy         global mpz
204 gcpy         mpz = float(0)
205 gcpy         global tpz
206 gcpy         tpz = float(0)
207 gcpy         global currenttoolnum
208 gcpy         currenttoolnum = 102
209 gcpy         global currenttoolshape
210 gcpy         currenttoolshape = cylinder(1.5875,12.7)
211 gcpy         global stock
212 gcpy         self.stock = cube([stockXwidth, stockYheight,
                     stockZthickness])
213 gcpy         if self.generategcode == True:
214 gcpy             self.writegc("(Design␣File:␣" + self.basefilename + ")"
                         )
```

Note that since Python in OpenPythonSCAD defers output of the 3D model, it is possible to define it once, then set up all the specifics for each possible positioning of the stock in terms of origin:

The internal variable stockzero is used in an <if then else> structure to position the 3D model of the stock.

```
215 gcpy         if self.zeroheight == "Top":
216 gcpy             if self.stockzero == "Lower-Left":
217 gcpy                 self.stock = stock.translate([0,0,-self.
                             stockZthickness])
218 gcpy                 if self.generategcode == True:
219 gcpy                     self.writegc("(stockMin:0.00mm,␣0.00mm,␣-",str(
                                 self.stockZthickness),"mm)")
220 gcpy                     self.writegc("(stockMax:",str(self.stockXwidth)
                                 ,"mm,␣",str(stockYheight),"mm,␣0.00mm)")
221 gcpy                     self.writegc("(STOCK/BLOCK,␣",str(self.
                                 stockXwidth),",␣",str(self.stockYheight),",␣
                                 ",str(self.stockZthickness),",␣0.00,␣0.00,␣"
                                 ,str(self.stockZthickness),")")
222 gcpy             if self.stockzero == "Center-Left":
223 gcpy                 self.stock = self.stock.translate([0,-stockYheight
                             / 2,-stockZthickness])
224 gcpy                 if self.generategcode == True:
225 gcpy                     self.writegc("(stockMin:0.00mm,␣-",str(self.
                                 stockYheight/2),"mm,␣-",str(self.
                                 stockZthickness),"mm)")
226 gcpy                     self.writegc("(stockMax:",str(self.stockXwidth)
                                 ,"mm,␣",str(self.stockYheight/2),"mm,␣0.00mm
                                 )")
227 gcpy                     self.writegc("(STOCK/BLOCK,␣",str(self.
                                 stockXwidth),",␣",str(self.stockYheight),",␣
                                 ",str(self.stockZthickness),",␣0.00,␣",str(
                                 self.stockYheight/2),",␣",str(self.
                                 stockZthickness),")");
228 gcpy             if self.stockzero == "Top-Left":
229 gcpy                 self.stock = self.stock.translate([0,-self.
                             stockYheight,-self.stockZthickness])
230 gcpy                 if self.generategcode == True:
231 gcpy                     self.writegc("(stockMin:0.00mm,␣-",str(self.
                                 stockYheight),"mm,␣-",str(self.
                                 stockZthickness),"mm)")
232 gcpy                     self.writegc("(stockMax:",str(self.stockXwidth)
```

```
                                ,"mm,␣0.00mm,␣0.00mm)")
233 gcpy                        self.writegc("(STOCK/BLOCK,␣",str(self.
                                    stockXwidth),",␣",str(self.stockYheight),",␣
                                    ",str(self.stockZthickness),",␣0.00,␣",str(
                                    self.stockYheight),",␣",str(self.
                                    stockZthickness),")")
234 gcpy                    if self.stockzero == "Center":
235 gcpy                        self.stock = self.stock.translate([-self.
                                    stockXwidth / 2,-self.stockYheight / 2,-self.
                                    stockZthickness])
236 gcpy                        if self.generategcode == True:
237 gcpy                            self.writegc("(stockMin:␣-",str(self.
                                        stockXwidth/2),",␣-",str(self.stockYheight
                                        /2),"mm,␣-",str(self.stockZthickness),"mm)")
238 gcpy                            self.writegc("(stockMax:",str(self.stockXwidth
                                        /2),"mm,␣",str(self.stockYheight/2),"mm,␣
                                        0.00mm)")
239 gcpy                            self.writegc("(STOCK/BLOCK,␣",str(self.
                                        stockXwidth),",␣",str(self.stockYheight),",␣
                                        ",str(self.stockZthickness),",␣",str(self.
                                        stockXwidth/2),",␣", str(self.stockYheight
                                        /2),",␣",str(self.stockZthickness),")")
240 gcpy            if self.zeroheight == "Bottom":
241 gcpy                if self.stockzero == "Lower-Left":
242 gcpy                     self.stock = self.stock.translate([0,0,0])
243 gcpy                     if self.generategcode == True:
244 gcpy                         self.writegc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
                                        )")
245 gcpy                         self.writegc("(stockMax:",str(self.stockXwidth
                                        ),"mm,␣",str(self.stockYheight),"mm,␣␣",str
                                        (self.stockZthickness),"mm)")
246 gcpy                         self.writegc("(STOCK/BLOCK,␣",str(self.
                                        stockXwidth),",␣",str(self.stockYheight),",
                                        ␣",str(self.stockZthickness),",␣0.00,␣0.00,
                                        ␣0.00)")
247 gcpy                if self.stockzero == "Center-Left":
248 gcpy                    self.stock = self.stock.translate([0,-self.
                                stockYheight / 2,0])
249 gcpy                    if self.generategcode == True:
250 gcpy                        self.writegc("(stockMin:0.00mm,␣-",str(self.
                                    stockYheight/2),"mm,␣0.00mm)")
251 gcpy                        self.writegc("(stockMax:",str(self.stockXwidth)
                                    ,"mm,␣",str(self.stockYheight/2),"mm,␣-",str
                                    (self.stockZthickness),"mm)")
252 gcpy                        self.writegc("(STOCK/BLOCK,␣",str(self.
                                    stockXwidth),",␣",str(self.stockYheight),",␣
                                    ",str(self.stockZthickness),",␣0.00,␣",str(
                                    self.stockYheight/2),",␣0.00mm)");
253 gcpy                if self.stockzero == "Top-Left":
254 gcpy                    self.stock = self.stock.translate([0,-self.
                                stockYheight,0])
255 gcpy                    if self.generategcode == True:
256 gcpy                        self.writegc("(stockMin:0.00mm,␣-",str(self.
                                    stockYheight),"mm,␣0.00mm)")
257 gcpy                        self.writegc("(stockMax:",str(self.stockXwidth)
                                    ,"mm,␣0.00mm,␣",str(self.stockZthickness),"
                                    mm)")
258 gcpy                        self.writegc("(STOCK/BLOCK,␣",str(self.
                                    stockXwidth),",␣",str(self.stockYheight),",␣
                                    ",str(self.stockZthickness),",␣0.00,␣",str(
                                    self.stockYheight),",␣0.00)")
259 gcpy                if self.stockzero == "Center":
260 gcpy                    self.stock = self.stock.translate([-self.
                                stockXwidth / 2,-self.stockYheight / 2,0])
261 gcpy                    if self.generategcode == True:
262 gcpy                        self.writegc("(stockMin:␣-",str(self.
                                    stockXwidth/2),",␣-",str(self.stockYheight
                                    /2),"mm,␣0.00mm)")
263 gcpy                        self.writegc("(stockMax:",str(self.stockXwidth
                                    /2),"mm,␣",str(self.stockYheight/2),"mm,␣",
                                    str(self.stockZthickness),"mm)")
264 gcpy                        self.writegc("(STOCK/BLOCK,␣",str(self.
                                    stockXwidth),",␣",str(self.stockYheight),",␣
                                    ",str(self.stockZthickness),",␣",str(self.
                                    stockXwidth/2),",␣", str(self.stockYheight
                                    /2),",␣0.00)")
265 gcpy            if self.generategcode == True:
266 gcpy                self.writegc("G90");
```

```
267 gcpy             self.writegc("G21");
```

Note that while the #102 is declared as a default tool, while it was originally necessary to call a tool change after invoking `setupstock` in the 2024.09.03 version of PythonSCAD this requirement went away when an update which interfered with persistently setting a variable directly was fixed.

osetupstock

The intermediary OpenSCAD code, `osetupstock` simply calls the Python version. Note that the parameters are passed all the way down, which was initially for consistency (they were not used) in 0.8 and later, everything happens in the Python file, and the OpenSCAD code is simply a series of descriptors which simply call the Python file.

```
4 pyscad  module osetupstock(stockXwidth, stockYheight, stockZthickness,
              zeroheight, stockzero) {
5 pyscad      psetupstock(stockXwidth, stockYheight, stockZthickness,
                  zeroheight, stockzero);
6 pyscad  }
```

```
 9 gcpscad module setupstock(stockXwidth, stockYheight, stockZthickness,
               zeroheight, stockzero) {
10 gcpscad   osetupstock(stockXwidth, stockYheight, stockZthickness,
                 zeroheight, stockzero);
11 gcpscad }
```

An example usage in OpenSCAD would be:

```
difference() {
setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero);
... // Cutting commands go here
}
```

For Python, the initial 3D model is stored in the variable `stock`:

```
setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero)

cy = cube([1,2,stockZthickness*2])

diff = stock.difference(cy)
#output(diff)
diff.show()
```

2.3.2 Position and Variables

In modeling the machine motion and G-code it will be necessary to have the machine track several variables for machine position, current tool, and depth in toolpath. This will be done using paired functions (which will set and return the matching variable) and a matching (global) variable, as well as additional functions for setting the matching variable(s).

The first such variables are for XYZ position:

mpx

- `mpx`

mpy

- `mpy`

mpz

- `mpz`

Similarly, for some toolpaths it will be necessary to track the depth along the Z-axis as the toolpath is cut out:

tpz

- `tpz`

It will further be necessary to have a variable for the current tool:

currenttoolnum

- `currenttoolnum`

Note that the `currenttoolnum` variable should always be used for any specification of a tool, being read in whenever a tool is to be made use of, or a parameter or aspect of the tool needs to be used in a calculation.

For each intended command it will be necessary to implement an appropriate aspect in each file. The Python file will manage the Python variables and handle things which can only be done in Python, while there will be two OpenSCAD files as noted above, one which calls the Python code (this will be `used`), while the other will be able to access and use OpenSCAD variables, as well as implement Customizer options (this will be `included`).

xpos
ypos
zpos

It will be necessary to have Python functions (`xpos`, `ypos`, and `zpos`) which return the current values of the machine position in Cartesian coordinates:

```
269 gcpy     def xpos(self):
270 gcpy         global mpx
271 gcpy         return mpx
272 gcpy
273 gcpy     def ypos(self):
274 gcpy         global mpy
275 gcpy         return mpy
276 gcpy
277 gcpy     def zpos(self):
278 gcpy         global mpz
279 gcpy         return mpz
280 gcpy
281 gcpy     def tzpos(self):
282 gcpy         global tpz
283 gcpy         return tpz
```

psetxpos
psetypos
psetzpos
psettzpos

and in turn, functions which set the positions: psetxpos, psetypos, psetzpos, and psettzpos

```
285 gcpy     def setxpos(self, newxpos):
286 gcpy         global mpx
287 gcpy         mpx = newxpos
288 gcpy
289 gcpy     def setypos(self, newypos):
290 gcpy         global mpy
291 gcpy         mpy = newypos
292 gcpy
293 gcpy     def setzpos(self, newzpos):
294 gcpy         global mpz
295 gcpy         mpz = newzpos
296 gcpy
297 gcpy     def settzpos(self, newtzpos):
298 gcpy         global tpz
299 gcpy         tpz = newtzpos
```

setxpos
setypos
setzpos
settzpos
getxpos
getypos
getzpos
gettzpos

and as noted above, there will need to be matching OpenSCAD versions which will set: setxpos, setypos, setzpos, and settzpos; as well as return the value: getxpos, getypos, getzpos, and gettzpos Note that for routines where the variable is directly passed from OpenSCAD to Python it is possible to have OpenSCAD directly call the matching Python module with no needto use an intermediary OpenSCAD module.

```
 8 pyscad //function getxpos() = xpos();
 9 pyscad //function getypos() = ypos();
10 pyscad //function getzpos() = zpos();
11 pyscad //function gettzpos() = tzpos();
12 pyscad //
13 pyscad //module setxpos(newxpos) {
14 pyscad //    psetxpos(newxpos);
15 pyscad //}
16 pyscad //
17 pyscad //module setypos(newypos) {
18 pyscad //    psetypos(newypos);
19 pyscad //}
20 pyscad //
21 pyscad //module setzpos(newzpos) {
22 pyscad //    psetzpos(newzpos);
23 pyscad //}
24 pyscad //
25 pyscad //module settzpos(newtzpos) {
26 pyscad //    psettzpos(newtzpos);
27 pyscad //}
28 pyscad //
```

oset

oset while for setting all three of the variables, there is an internal OpenSCAD module:

```
102 gcpscad //module oset(ex, ey, ez) {
103 gcpscad //    setxpos(ex);
104 gcpscad //    setypos(ey);
105 gcpscad //    setzpos(ez);
106 gcpscad //}
107 gcpscad //
```

osettz

and some toolpaths will require the storing and usage of an intermediate value via osettz for the Z-axis position during calculation:

```
108 gcpscad //module osettz(tz) {
```

```
109 gcpscad //     settzpos(tz);
110 gcpscad //}
111 gcpscad //
```

2.4 Tools and Changes

currenttoolnumber settool

Similarly Python functions and variables will be used in: `currenttoolnumber` (note that it is important to use a different name than the variable `currenttoolnum` and `settool` (it may be that the latter will be removed) to track and set and return the current tool:

```
301 gcpy      def settool(self,tn):
302 gcpy          global currenttoolnum
303 gcpy          currenttoolnum = tn
304 gcpy
305 gcpy      def currenttoolnumber(self):
306 gcpy          global currenttoolnum
307 gcpy          return currenttoolnum
```

osettool current tool

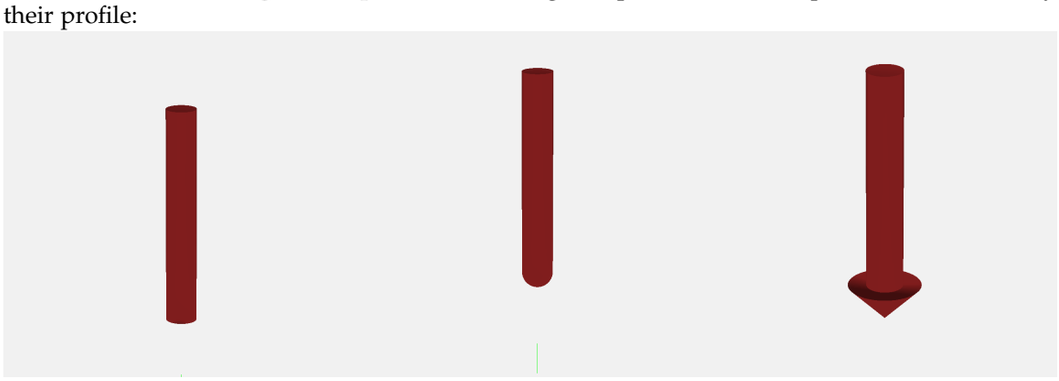
and matching OpenSCAD modules: `osettool` and `current tool` set and return the current tool:

```
29 pyscad module osettool(tn){
30 pyscad     psettool(tn);
31 pyscad }
32 pyscad
33 pyscad function current_tool() = pcurrent_tool();
```

2.4.1 3D Shapes for Tools

Each tool must be modeled in 3D using an OpenSCAD module.

2.4.1.1 Normal Tooling/toolshapes Most tooling has quite standard shapes and are defined by their profile:



- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle their simple and easily understood geometry makes them a standard choice (a radiused form with a flat bottom, often described as a "bowl bit" is not implemented as-of-yet)
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302 and 390) — pointed at the tip, they are available in a variety of angles and diameters and may be used for decorative V carving, or for chamfering or cutting specific angles (note that the commonly available radiused form is not implemented at this time, *e.g.*, #501 and 502)

Most tools are easily implemented with concise 3D descriptions which may be connected with a simple `hull` operation:

endmill square

The `endmill square` is a simple cylinder:

```
309 gcpy      def endmill_square(self, es_diameter, es_flute_length):
310 gcpy          return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                      h=es_flute_length, center = False)
```

gcp endmill ball

The `gcp endmill ball` is modeled as a hemisphere joined with a cylinder:

```
312 gcpy      def gcp_endmill_ball(self, es_diameter, es_flute_length):
313 gcpy          b = sphere(r=(es_diameter / 2))
314 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                      es_flute_length, center=False)
315 gcpy          p = union(b,s)
```

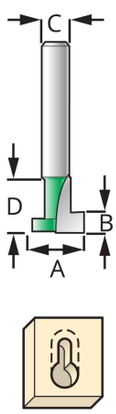
```
316 gcpy         return p.translate([0, 0, (es_diameter / 2)])
```

gcp endmill v The gcp endmill v is modeled as a cylinder with a zero width base and a second cylinder for the shaft (note that Python's math defaults to radians, hence the need to convert from degrees):

```
318 gcpy     def gcp_endmill_v(self, es_v_angle, es_diameter):
319 gcpy         es_v_angle = math.radians(es_v_angle)
320 gcpy         v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                     2) / math.tan((es_v_angle / 2))), center=False)
321 gcpy         s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h
                     =((es_diameter * 8) ), center=False)
322 gcpy         sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                     es_v_angle / 2)))])
323 gcpy         return union(v,sh)
```

2.4.1.2 Tooling for Keyhole Toolpaths Keyhole toolpaths (see: subsection 3.2 are intended for use with tooling which projects beyond the the narrower shaft and so will cut usefully underneath the visible surface. Also described as "undercut" tooling, but see below.

There are several notable candidates for such tooling:



Keyhole Router Bits

#	A	B	C	D
374	3/8″	1/8″	1/4″	3/8″
375	9.525mm	3.175mm	8mm	9.525mm
376	1/2″	3/16″	1/4″	1/2″
378	12.7mm	4.7625mm	8mm	12.7mm

- Keyhole tools — intended to cut slots for retaining hardware used for picture hanging, they may be used to create slots for other purposes Note that it will be necessary to model these twice, once for the shaft, the second time for the actual keyhole cutting https://assetssc.leevalley.com/en-gb/shop/tools/power-tool-accessories/router-bits/30113-keyhole-router-bits
- Dovetail cutters — used for the joinery of the same name, they cut a large area at the bottom which slants up to a narrower region at a defined angle
- Lollipop cutters — normally used for 3D work, as their name suggests they are essentially a (cutting) ball on a narrow stick (the tool shaft), they are mentioned here only for compleatness' sake and are not (at this time) implemented

2.4.1.3 Thread mills The implementation of arcs cutting along the Z-axis raises the possibility of cutting threads using "thread mills". See: https://community.carbide3d.com/t/thread-milling-in-metal-on-the-shapeoko-3/5332

gcp keyhole **2.4.1.4 Keyhole** The gcp keyhole is modeled by the cutting base:

```
325 gcpy     def gcp_keyhole(self, es_diameter, es_flute_length):
326 gcpy         return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                     h=es_flute_length, center=false)
```

and a second call for an additional cylinder for the shaft will be necessary:

```
328 gcpy     def gcp_keyhole_shaft(self, es_diameter, es_flute_length):
329 gcpy         return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                     h=es_flute_length, center=false)
```

gcp dovetail The gcp dovetail is modeled as a cylinder with the differing bottom and top diameters determining the angle (though dt_angle is still required as a parameter)

```
331 gcpy     def gcp_dovetail(self, dt_bottomdiameter, dt_topdiameter,
                 dt_height, dt_angle):
```

```
332 gcpy          return cylinder(r1=(dt_bottomdiameter / 2), r2=(
                      dt_topdiameter / 2), h= dt_height, center=false)
```

2.4.1.5 Concave toolshapes While normal tooling may be represented with a single `hull` operation betwixt two 3D toolshapes, concave tooling such as roundover/radius tooling require multiple slices of the tool shape which are then `hull`ed together. Something of this can be seen in the manual work-around for previewing them: https://community.carbide3d.com/t/using-unsupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723.

Ideally, it would be possible to simply identify such tooling using the tool # in the code used for normal toolshapes as above, but the most expedient option is to simply use a specific command for this. Since such tooling is quite limited in its use and normally only used at the surface of the part along an edge, this separation is easily justified.

Because it is necessary to divide the tooling into vertical slices and call the hull operation for each slice the tool definitions are tightly coupled with the module. Note that there are two
`cutroundover` different modules, the public-facing version which includes the tool number:`cutroundover`

2.4.1.6 Roundover tooling It is not possible to represent all tools using tool changes as coded above which require using a `hull` operation between 3D representations of the tools at the beginning and end points. Tooling which cannot be so represented will be implemented separately below, see paragraph 2.4.1.5.

```
112 gcpscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
113 gcpscad     if (radiustn == 56125) {
114 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
115 gcpscad     } else if (radiustn == 56142) {
116 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
117 gcpscad     } else if (radiustn == 312) {
118 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
119 gcpscad     } else if (radiustn == 1570) {
120 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
121 gcpscad     }
122 gcpscad }
```

which then calls the actual `cutroundovertool` module passing in the tip radius and the radius of the rounding. Note that this module sets its quality relative to the value of `$fn`.

```
124 gcpscad module cutroundovertool(bx, by, bz, ex, ey, ez, tool_radius_tip,
                tool_radius_width) {
125 gcpscad n = 90 + $fn*3;
126 gcpscad step = 360/n;
127 gcpscad
128 gcpscad hull(){
129 gcpscad     translate([bx,by,bz])
130 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
131 gcpscad     translate([ex,ey,ez])
132 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
133 gcpscad }
134 gcpscad
135 gcpscad hull(){
136 gcpscad translate([bx,by,bz+tool_radius_width])
137 gcpscad cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
                tool_radius_tip+tool_radius_width);
138 gcpscad
139 gcpscad translate([ex,ey,ez+tool_radius_width])
140 gcpscad   cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
                  tool_radius_tip+tool_radius_width);
141 gcpscad }
142 gcpscad
143 gcpscad for (i=[0:step:90]) {
144 gcpscad     angle = i;
145 gcpscad     dx = tool_radius_width*cos(angle);
146 gcpscad     dxx = tool_radius_width*cos(angle+step);
147 gcpscad     dzz = tool_radius_width*sin(angle);
148 gcpscad     dz = tool_radius_width*sin(angle+step);
149 gcpscad     dh = dz-dzz;
150 gcpscad     hull(){
151 gcpscad         translate([bx,by,bz+dz])
152 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
153 gcpscad         translate([ex,ey,ez+dz])
154 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
155 gcpscad         }
156 gcpscad     }
```

```
157 gcpscad }
```

2.4.2 toolchange

toolchange and apply the appropriate commands for a `toolchange`. Note that it is expected that this code will be updated as needed when new tooling is introduced as additional modules which require specific tooling are added below.

Note that the comments written out in G-code correspond to that used by the G-code previewing tool CutViewer (which is unfortunately, no longer readily available).

A further concern is that early versions often passed the tool into a module using a parameter. That ceased to be necessary in the 2024.09.03 version of PythonSCAD, and all modules should read the tool # from `currenttoolnumber()`. Note that this variable has changed names from the original `currenttool` which is now used to store the current tool *shape* (or 3D model).

It is possible that rather than hard-coding the tool definitions, a future update will instead read them in from an external file — the `.csv` format used for tool libraries in Carbide Create seems a likely candidate and worth exploring.

Note that there are many varieties of tooling and not all will be implemented, especially in the early versions of this project

2.4.2.1 Selecting Tools The original implementation created the model for the tool at the current position, wrapping the twain for each end of a given movement in a `hull()` command. This approach will not work within Python, so it will be necessary to instead assign and select the tool currenttoolshape as part of the cutting command indirectly by first storing it in the variable `currenttoolshape`.

```
334 gcpy     def currenttool(self):
335 gcpy         global currenttoolshape
336 gcpy         return self.currenttoolshape
```

```
338 gcpy     def toolchange(self,tool_number,speed):
339 gcpy         global currenttoolshape
340 gcpy
341 gcpy         self.settool(tool_number)
342 gcpy         if (self.generategcode == True):
343 gcpy             self.writegc("(Toolpath)")
344 gcpy             self.writegc("M05")
345 gcpy         if (tool_number == 201):
346 gcpy             self.writegc("(TOOL/MILL,6.35,␣0.00,␣0.00,␣0.00)")
347 gcpy             self.currenttoolshape = self.endmill_square(6.35,
                         19.05)
348 gcpy         elif (tool_number == 202):
349 gcpy             self.writegc("(TOOL/MILL,6.35,␣3.17,␣0.00,␣0.00)")
350 gcpy             self.currenttoolshape = self.gcp_endmill_ball(6.35,
                         19.05)
351 gcpy         elif (tool_number == 102):
352 gcpy             self.writegc("(TOOL/MILL,3.17,␣0.00,␣0.00,␣0.00)")
353 gcpy             self.currenttoolshape = self.endmill_square(3.175,
                         12.7)
354 gcpy         elif (tool_number == 101):
355 gcpy             self.writegc("(TOOL/MILL,3.17,␣1.58,␣0.00,␣0.00)")
356 gcpy             self.currenttoolshape = self.gcp_endmill_ball(3.175,
                         12.7)
357 gcpy         elif (tool_number == 301):
358 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣6.35,␣45.00)")
359 gcpy             self.currenttoolshape = self.gcp_endmill_v(90, 12.7)
360 gcpy         elif (tool_number == 302):
361 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣10.998,␣30.00)")
362 gcpy             self.currenttoolshape = self.gcp_endmill_v(60, 12.7)
363 gcpy         elif (tool_number == 390):
364 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣1.5875,␣45.00)")
365 gcpy             self.currenttoolshape = self.gcp_endmill_v(90, 3.175)
366 gcpy         elif (tool_number == 374):
367 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
368 gcpy         elif (tool_number == 375):
369 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
370 gcpy         elif (tool_number == 376):
371 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
372 gcpy         elif (tool_number == 378):
373 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
374 gcpy         elif (tool_number == 814):
375 gcpy             writegc("(TOOL/MILL,12.7,␣6.367,␣12.7,␣0.00)")
```

With the tools delineated, the module is closed out and the toolchange information written into the G-code as well as the command to start the spindle at the specified speed.

```
376 gcpy          self.writegc("M6T",str(tool_number))
377 gcpy          self.writegc("M03S",str(speed))
```

For example:

```
toolchange(small_square_tool_num,speed);
```

(the assumption is that all speed rates in a file will be the same, so as to account for the most frequent use case of a trim router with speed controlled by a dial setting)

2.4.3 tooldiameter

It will also be necessary to be able to provide the diameter of the current tool. Arguably, this would be much easier using an object-oriented programming style/dot notation.

One aspect of tool parameters which will need to be supported is shapes which create different profiles based on how deeply the tool is cutting into the surface of the material at a given point. To accommodate this, it will be necessary to either track the thickness of uncut material at any given point, or, to specify the depth of cut as a parameter which is what the initial version will implement.

`tool diameter` The public-facing OpenSCAD code, `tool diameter` simply calls the matching OpenSCAD module which wraps the Python code:

```
159 gcpscad function tool_diameter(td_tool, td_depth) = otool_diameter(td_tool,
        td_depth);
```

`otool diameter` the matching OpenSCAD function, `otool diameter` calls the Python function:

```
 35 pyscad function otool_diameter(td_tool, td_depth) = ptool_diameter(td_tool
        , td_depth);
```

`ptool diameter` the Python code, `ptool diameter` returns appropriate values based on the specified tool number and depth:

```
379 gcpy #def ptool_diameter(ptd_tool, ptd_depth):
380 gcpy # Square 122,112,102,201
381 gcpy #    if ptd_tool == 122:
382 gcpy #        return 0.79375
383 gcpy #    if ptd_tool == 112:
384 gcpy #        return 1.5875
385 gcpy #    if ptd_tool == 102:
386 gcpy #        return 3.175
387 gcpy #    if ptd_tool == 201:
388 gcpy #        return 6.35
389 gcpy ## Ball 121,111,101,202
390 gcpy #    if ptd_tool == 122:
391 gcpy #        return
392 gcpy #        if ptd_depth > 0.396875:
393 gcpy #            return 0.79375
394 gcpy #        else:
395 gcpy #            return 0
396 gcpy #    if ptd_tool == 112:
397 gcpy #        if ptd_depth > 0.79375:
398 gcpy #            return 1.5875
399 gcpy #        else:
400 gcpy #            return 0
401 gcpy #    if ptd_tool == 101:
402 gcpy #        if ptd_depth > 1.5875:
403 gcpy #            return 3.175
404 gcpy #        else:
405 gcpy #            return 0
406 gcpy #    if ptd_tool == 202:
407 gcpy #        if ptd_depth > 3.175:
408 gcpy #            return 6.35
409 gcpy #        else:
410 gcpy #            return 0
411 gcpy ## V 301, 302, 390
412 gcpy #    if ptd_tool == 301:
413 gcpy #        return 0
414 gcpy #    if ptd_tool == 302:
415 gcpy #        return 0
416 gcpy #    if ptd_tool == 390:
417 gcpy #        return 0
418 gcpy ## Keyhole
419 gcpy #    if ptd_tool == 374:
420 gcpy #        if ptd_depth < 3.175:
```

```
421 gcpy #             return 9.525
422 gcpy #         else:
423 gcpy #             return 6.35
424 gcpy #     if ptd_tool == 375:
425 gcpy #         if ptd_depth < 3.175:
426 gcpy #             return 9.525
427 gcpy #         else:
428 gcpy #             return 8
429 gcpy #     if ptd_tool == 376:
430 gcpy #         if ptd_depth < 4.7625:
431 gcpy #             return 12.7
432 gcpy #         else:
433 gcpy #             return 6.35
434 gcpy #     if ptd_tool == 378:
435 gcpy #         if ptd_depth < 4.7625:
436 gcpy #             return 12.7
437 gcpy #         else:
438 gcpy #             return 8
439 gcpy ## Dovetail
440 gcpy #     if ptd_tool == 814:
441 gcpy #         if ptd_depth > 12.7:
442 gcpy #             return 6.35
443 gcpy #         else:
444 gcpy #             return 12.7
445 gcpy #
```

tool radius

Since it is often necessary to utilise the radius of the tool, an additional command, `tool radius` to return this value is worthwhile:

```
161 gcpscad function tool_radius(td_tool, td_depth) = otool_diameter(td_tool,
        td_depth)/2;
```

(Note that zero (0) and other not fully calculated values will need to be replaced with code which calculates the appropriate values.)

2.4.4 Feeds and Speeds

feed
plunge
speed

There are several possibilities for handling feeds and speeds. Currently, base values for `feed`, `plunge`, and `speed` are used, which may then be adjusted using various `<tooldescriptor>_ratio` values, as an acknowledgement of the likelihood of a trim router being used as a spindle, the assumption is that the `speed` will remain unchanged.

One notable possibility for the future would be to load it from the .csv files used for User tool libraries in Carbide Create. Ideally, any use of such values in modules would be such that some other scheme could replace that usage with minimal editing and updating.

The tools which need to be calculated thus are those in addition to the `large_square` tool:

- `small_square_ratio`
- `small_ball_ratio`
- `large_ball_ratio`
- `small_V_ratio`
- `large_V_ratio`
- `KH_ratio`
- `DT_ratio`

2.5 OpenSCAD File Handling

popengcodefile
popendxffile
popendxflgsqfile
popendxfsmsqfile
popendxflgblfile
popendxfsmblfile
popendxflgVfile
popendxfsmVfile

For writing to files it will be necessary to have commands: `popengcodefile`, `popendxffile`, `popendxflgsqfile`, `popendxfsmsqfile`, `popendxflgblfile`, `popendxfsmblfile`, `popendxflgVfile`, and `popendxfsmVfile`. There is a separate function for each type of file, and for DXFs, there are multiple file instances, one for each combination of different type and size of tool which it is expected a project will work with. Each such file will be suffixed with the tool number.

Integrating G-code and DXF generation with everything else would be ideal, but will require ensuring that each command which moves the tool creates a matching command for both files.

```
446 gcpy #def popengcodefile(fn):
447 gcpy #     global f
448 gcpy #     f = open(fn, "w")
449 gcpy #
450 gcpy #def popendxffile(fn):
451 gcpy #     global dxf
452 gcpy #     dxf = open(fn, "w")
```

```
453 gcpy #
454 gcpy #def popendxflgblfile(fn):
455 gcpy #    global dxflgbl
456 gcpy #    dxflgbl = open(fn, "w")
457 gcpy #
458 gcpy #def popendxflgsqfile(fn):
459 gcpy #    global dxflgsq
460 gcpy #    dxflgsq = open(fn, "w")
461 gcpy #
462 gcpy #def popendxflgVfile(fn):
463 gcpy #    global dxflgV
464 gcpy #    dxflgV = open(fn, "w")
465 gcpy #
466 gcpy #def popendxfsmblfile(fn):
467 gcpy #    global dxfsmbl
468 gcpy #    dxfsmbl = open(fn, "w")
469 gcpy #
470 gcpy #def popendxfsmsqfile(fn):
471 gcpy #    global dxfsmsq
472 gcpy #    dxfsmsq = open(fn, "w")
473 gcpy #
474 gcpy #def popendxfsmVfile(fn):
475 gcpy #    global dxfsmV
476 gcpy #    dxfsmV = open(fn, "w")
477 gcpy #
478 gcpy #def popendxfKHfile(fn):
479 gcpy #    global dxfKH
480 gcpy #    dxfKH = open(fn, "w")
481 gcpy #
482 gcpy #def popendxfDTfile(fn):
483 gcpy #    global dxfDT
484 gcpy #    dxfDT = open(fn, "w")
485 gcpy #
```

`oopengcodefile` `oopendxffile` There will need to be matching OpenSCAD modules `oopengcodefile`, and `oopendxffile`, for the Python functions.

```
37 pyscad module oopengcodefile(fn) {
38 pyscad     popengcodefile(fn);
39 pyscad }
40 pyscad
41 pyscad module oopendxffile(fn) {
42 pyscad //     echo(fn);
43 pyscad     popendxffile(fn);
44 pyscad }
45 pyscad
46 pyscad module oopendxflgblfile(fn) {
47 pyscad     popendxflgblfile(fn);
48 pyscad }
49 pyscad
50 pyscad module oopendxflgsqfile(fn) {
51 pyscad     popendxflgsqfile(fn);
52 pyscad }
53 pyscad
54 pyscad module oopendxflgVfile(fn) {
55 pyscad     popendxflgVfile(fn);
56 pyscad }
57 pyscad
58 pyscad module oopendxfsmblfile(fn) {
59 pyscad     popendxfsmblfile(fn);
60 pyscad }
61 pyscad
62 pyscad module oopendxfsmsqfile(fn) {
63 pyscad //     echo(fn);
64 pyscad     popendxfsmsqfile(fn);
65 pyscad }
66 pyscad
67 pyscad module oopendxfsmVfile(fn) {
68 pyscad     popendxfsmVfile(fn);
69 pyscad }
70 pyscad
71 pyscad module oopendxfKHfile(fn) {
72 pyscad     popendxfKHfile(fn);
73 pyscad }
74 pyscad
75 pyscad module oopendxfDTfile(fn) {
76 pyscad     popendxfDTfile(fn);
77 pyscad }
```

opengcodefile With matching OpenSCAD commands: `opengcodefile`

```
163 gcpscad module opengcodefile(fn) {
164 gcpscad if (generategcode == true) {
165 gcpscad     oopengcodefile(fn);
166 gcpscad //     echo(fn);
167 gcpscad     owritecomment(fn);
168 gcpscad     }
169 gcpscad }
```

opendxffile For each DXF file, there will need to be a Preamble created by `opendxffile` in addition to opening the file in the file system:

```
171 gcpscad module opendxffile(fn) {
172 gcpscad   if (generatedxf == true) {
173 gcpscad       oopendxffile(str(fn,".dxf"));
174 gcpscad //     echo(fn);
175 gcpscad       dxfwriteone("0");
176 gcpscad       dxfwriteone("SECTION");
177 gcpscad       dxfwriteone("2");
178 gcpscad       dxfwriteone("ENTITIES");
179 gcpscad     if (large_ball_tool_num >  0) {    oopendxflgblfile(str(fn,".",
                    large_ball_tool_num,".dxf"));
180 gcpscad       dxfpreamble(large_ball_tool_num);
181 gcpscad     }
182 gcpscad     if (large_square_tool_num >  0) {    oopendxflgsqfile(str(fn
                    ,".",large_square_tool_num,".dxf"));
183 gcpscad       dxfpreamble(large_square_tool_num);
184 gcpscad     }
185 gcpscad     if (large_V_tool_num >  0) {    oopendxflgVfile(str(fn,".",
                    large_V_tool_num,".dxf"));
186 gcpscad       dxfpreamble(large_V_tool_num);
187 gcpscad     }
188 gcpscad     if (small_ball_tool_num >  0) { oopendxfsmblfile(str(fn,".",
                    small_ball_tool_num,".dxf"));
189 gcpscad       dxfpreamble(small_ball_tool_num);
190 gcpscad     }
191 gcpscad     if (small_square_tool_num >  0) {    oopendxfsmsqfile(str(fn
                    ,".",small_square_tool_num,".dxf"));
192 gcpscad //     echo(str("tool number ",small_square_tool_num));
193 gcpscad       dxfpreamble(small_square_tool_num);
194 gcpscad     }
195 gcpscad     if (small_V_tool_num >  0) {    oopendxfsmVfile(str(fn,".",
                    small_V_tool_num,".dxf"));
196 gcpscad       dxfpreamble(small_V_tool_num);
197 gcpscad     }
198 gcpscad     if (KH_tool_num >  0) {    oopendxfKHfile(str(fn,".",
                    KH_tool_num,".dxf"));
199 gcpscad       dxfpreamble(KH_tool_num);
200 gcpscad     }
201 gcpscad     if (DT_tool_num >  0) {    oopendxfDTfile(str(fn,".",
                    DT_tool_num,".dxf"));
202 gcpscad       dxfpreamble(DT_tool_num);
203 gcpscad     }
204 gcpscad   }
205 gcpscad }
```

2.5.1 Writing to files

When the command to open dxf files is called it is passed all of the variables for the various tool types/sizes, and based on a value being greater than zero, the matching file is opened, and in addition, the main DXF which is always written to is opened as well. On the gripping hand, each element which may be written to a DXF file will have a user module as well as an internal module which will be called by it so as to write to the file for the current tool. It will be necessary for the dxfwrite `dxfwrite` command to evaluate the tool number which is passed in, and to use an appropriate command or set of commands to then write out to the appropriate file for a given tool (if positive) or not do anything (if zero), and to write to the master file if a negative value is passed in (this allows the various DXF template commands to be written only once and then called at need).
has a matching command each tool/size combination:

writedxflgbl
- Ball nose, large (lgbl) `writedxflgbl`

writedxfsmbl
- Ball nose, small (smbl) `writedxfsmbl`

writedxflgsq
- Square, large (lgsq) `writedxflgsq`

writedxfsmsq
- Square, small (smsq) `writedxfsmsq`

writedxflgV
- V, large (lgV) `writedxflgV`

writedxfsmV
- V, small (smV) `writedxfsmV`

writedxfKH
- Keyhole (KH) `writedxfKH`

writedxfDT
- Dovetail (DT) `writedxfDT`

```
486 gcpy #def writedxflgbl(*arguments):
487 gcpy #     line_to_write = ""
488 gcpy #     for element in arguments:
489 gcpy #         line_to_write += element
490 gcpy #     dxflgbl.write(line_to_write)
491 gcpy #     print(line_to_write)
492 gcpy #     dxflgbl.write("\n")
493 gcpy #
494 gcpy #def writedxflgsq(*arguments):
495 gcpy #     line_to_write = ""
496 gcpy #     for element in arguments:
497 gcpy #         line_to_write += element
498 gcpy #     dxflgsq.write(line_to_write)
499 gcpy #     print(line_to_write)
500 gcpy #     dxflgsq.write("\n")
501 gcpy #
502 gcpy #def writedxflgV(*arguments):
503 gcpy #     line_to_write = ""
504 gcpy #     for element in arguments:
505 gcpy #         line_to_write += element
506 gcpy #     dxflgV.write(line_to_write)
507 gcpy #     print(line_to_write)
508 gcpy #     dxflgV.write("\n")
509 gcpy #
510 gcpy #def writedxfsmbl(*arguments):
511 gcpy #     line_to_write = ""
512 gcpy #     for element in arguments:
513 gcpy #         line_to_write += element
514 gcpy #     dxfsmbl.write(line_to_write)
515 gcpy #     print(line_to_write)
516 gcpy #     dxfsmbl.write("\n")
517 gcpy #
518 gcpy #def writedxfsmsq(*arguments):
519 gcpy #     line_to_write = ""
520 gcpy #     for element in arguments:
521 gcpy #         line_to_write += element
522 gcpy #     dxfsmsq.write(line_to_write)
523 gcpy #     print(line_to_write)
524 gcpy #     dxfsmsq.write("\n")
525 gcpy #
526 gcpy #def writedxfsmV(*arguments):
527 gcpy #     line_to_write = ""
528 gcpy #     for element in arguments:
529 gcpy #         line_to_write += element
530 gcpy #     dxfsmV.write(line_to_write)
531 gcpy #     print(line_to_write)
532 gcpy #     dxfsmV.write("\n")
533 gcpy #
534 gcpy #def writedxfKH(*arguments):
535 gcpy #     line_to_write = ""
536 gcpy #     for element in arguments:
537 gcpy #         line_to_write += element
538 gcpy #     dxfKH.write(line_to_write)
539 gcpy #     print(line_to_write)
540 gcpy #     dxfKH.write("\n")
541 gcpy #
542 gcpy #def writedxfDT(*arguments):
543 gcpy #     line_to_write = ""
544 gcpy #     for element in arguments:
545 gcpy #         line_to_write += element
546 gcpy #     dxfDT.write(line_to_write)
547 gcpy #     print(line_to_write)
548 gcpy #     dxfDT.write("\n")
549 gcpy #
```

owritecomment
dxfwriteone
dxfwritelgbl
dxfwritelgsq

Separate OpenSCAD modules, `owritecomment`, `dxfwriteone`, `dxfwritelgbl`, `dxfwritelgsq`,

`dxfwritelgV`, `dxfwritesmbl`, `dxfwritesmsq`, and `dxfwritesmV` will be used for either writing out comments in G-code (.nc) files or adding to a DXF file — for each different tool in a file there will be a matching module to write to it.

dxfwritelgV
dxfwritesmbl
dxfwritesmsq
dxfwritesmV

```
 79 pyscad module owritecomment(comment) {
 80 pyscad     writeln("(",comment,")");
 81 pyscad }
 82 pyscad
 83 pyscad module dxfwriteone(first) {
 84 pyscad     writedxf(first);
 85 pyscad //     writeln(first);
 86 pyscad //     echo(first);
 87 pyscad }
 88 pyscad
 89 pyscad module dxfwritelgbl(first) {
 90 pyscad     writedxflgbl(first);
 91 pyscad }
 92 pyscad
 93 pyscad module dxfwritelgsq(first) {
 94 pyscad     writedxflgsq(first);
 95 pyscad }
 96 pyscad
 97 pyscad module dxfwritelgV(first) {
 98 pyscad     writedxflgV(first);
 99 pyscad }
100 pyscad
101 pyscad module dxfwritesmbl(first) {
102 pyscad     writedxfsmbl(first);
103 pyscad }
104 pyscad
105 pyscad module dxfwritesmsq(first) {
106 pyscad     writedxfsmsq(first);
107 pyscad }
108 pyscad
109 pyscad module dxfwritesmV(first) {
110 pyscad     writedxfsmV(first);
111 pyscad }
112 pyscad
113 pyscad module dxfwriteKH(first) {
114 pyscad     writedxfKH(first);
115 pyscad }
116 pyscad
117 pyscad module dxfwriteDT(first) {
118 pyscad     writedxfDT(first);
119 pyscad }
```

Since it is not convenient to stitch together and then write out multiple elements, the most expedient thing to do is to have discrete commands for each possible number of arguments, one through thirteen, `owrite...`

owrite...

```
121 pyscad module owriteone(first) {
122 pyscad     writeln(first);
123 pyscad }
124 pyscad
125 pyscad module owritetwo(first, second) {
126 pyscad     writeln(first, second);
127 pyscad }
128 pyscad
129 pyscad module owritethree(first, second, third) {
130 pyscad     writeln(first, second, third);
131 pyscad }
132 pyscad
133 pyscad module owritefour(first, second, third, fourth) {
134 pyscad     writeln(first, second, third, fourth);
135 pyscad }
136 pyscad
137 pyscad module owritefive(first, second, third, fourth, fifth) {
138 pyscad     writeln(first, second, third, fourth, fifth);
139 pyscad }
140 pyscad
141 pyscad module owritesix(first, second, third, fourth, fifth, sixth) {
142 pyscad     writeln(first, second, third, fourth, fifth, sixth);
143 pyscad }
144 pyscad
145 pyscad module owriteseven(first, second, third, fourth, fifth, sixth,
            seventh) {
146 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh);
```

```
147 pyscad }
148 pyscad
149 pyscad module owriteeight(first, second, third, fourth, fifth, sixth,
        seventh,eighth) {
150 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth);
151 pyscad }
152 pyscad
153 pyscad module owritenine(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth) {
154 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth);
155 pyscad }
156 pyscad
157 pyscad module owriteten(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth) {
158 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth);
159 pyscad }
160 pyscad
161 pyscad module owriteeleven(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh) {
162 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh);
163 pyscad }
164 pyscad
165 pyscad module owritetwelve(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth) {
166 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh, twelfth);
167 pyscad }
168 pyscad
169 pyscad module owritethirteen(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth) {
170 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh, twelfth, thirteenth);
171 pyscad }
```

2.5.1.1 Writing to DXFs This module requires that the tool number be passed in, and after writing out `dxfpreamble`, that value will be used to write out to the appropriate file with a series of `if` statements.

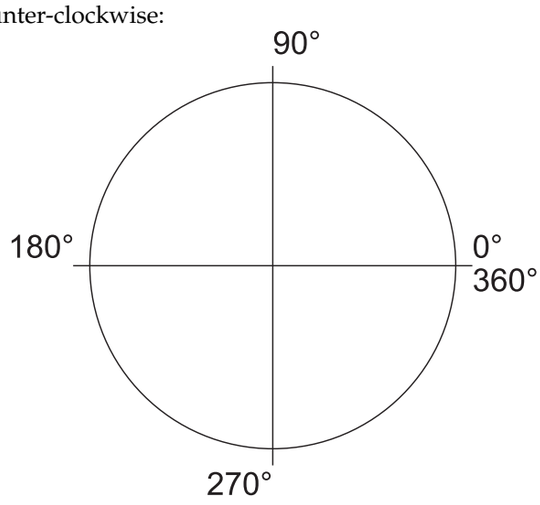
dxfpreamble

```
550 gcpy     def dxfpreamble(self, tn):
551 gcpy #         self.writedxf(tn,str(tn))
552 gcpy         self.writedxf(tn,"0")
553 gcpy         self.writedxf(tn,"SECTION")
554 gcpy         self.writedxf(tn,"2")
555 gcpy         self.writedxf(tn,"ENTITIES")
```

2.5.1.2 DXF Lines and Arcs There are two notable elements which may be written to a DXF:

- a line: LWPOLYLINE is one possible implementation: `dxfbpl` (dxfbpl)
- ARC — a notable option would be for the arc to close on itself, creating a circle: `dxfarc` (dxfarc)

DXF orders arcs counter-clockwise:



Note that arcs of greater than 90 degrees are not rendered accurately, so, for the sake of precision, they should be limited to a swing of 90 degrees or less. Further note that 4 arcs may be stitched together to make a circle:

```
dxfarc(10, 10, 5,   0,  90, small_square_tool_num);
dxfarc(10, 10, 5,  90, 180, small_square_tool_num);
dxfarc(10, 10, 5, 180, 270, small_square_tool_num);
dxfarc(10, 10, 5, 270, 360, small_square_tool_num);
```

A further refinement would be to connect multiple line segments/arcs into a larger polyline, but since most CAM tools implicitly join elements on import, that is not necessary.

There are three possible interactions for DXF elements and toolpaths:

- describe the motion of the tool
- define a perimeter of an area which will be cut by a tool
- define a centerpoint for a specialty toolpath such as Drill or Keyhhole

and it is possible that multiple such elements could be instantiated for a given toolpath.

When writing out to a DXF file there is a pair of commands, a public facing command which takes in a tool number in addition to the coordinates which then writes out to the main DXF file and then calls an internal command to which repeats the call with the tool number so as to write it out to the matching file.

```
557 gcpy      def dxfpolyline(self, tn, xbegin,ybegin,xend,yend):
558 gcpy          self.writedxf(tn,"0")
559 gcpy          self.writedxf(tn,"LWPOLYLINE")
560 gcpy          self.writedxf(tn,"90")
561 gcpy          self.writedxf(tn,"2")
562 gcpy          self.writedxf(tn,"70")
563 gcpy          self.writedxf(tn,"0")
564 gcpy          self.writedxf(tn,"43")
565 gcpy          self.writedxf(tn,"0")
566 gcpy          self.writedxf(tn,"10")
567 gcpy          self.writedxf(tn,str(xbegin))
568 gcpy          self.writedxf(tn,"20")
569 gcpy          self.writedxf(tn,str(ybegin))
570 gcpy          self.writedxf(tn,"10")
571 gcpy          self.writedxf(tn,str(xend))
572 gcpy          self.writedxf(tn,"20")
573 gcpy          self.writedxf(tn,str(yend))
```

The original implementation of polylines worked, but may be removed.

```
207 gcpscad module dxfbpl(tn,bx,by) {
208 gcpscad     dxfwrite(tn,"0");
209 gcpscad     dxfwrite(tn,"POLYLINE");
210 gcpscad     dxfwrite(tn,"8");
211 gcpscad     dxfwrite(tn,"default");
212 gcpscad     dxfwrite(tn,"66");
213 gcpscad     dxfwrite(tn,"1");
214 gcpscad     dxfwrite(tn,"70");
215 gcpscad     dxfwrite(tn,"0");
216 gcpscad     dxfwrite(tn,"0");
217 gcpscad     dxfwrite(tn,"VERTEX");
218 gcpscad     dxfwrite(tn,"8");
219 gcpscad     dxfwrite(tn,"default");
220 gcpscad     dxfwrite(tn,"70");
221 gcpscad     dxfwrite(tn,"32");
222 gcpscad     dxfwrite(tn,"10");
223 gcpscad     dxfwrite(tn,str(bx));
224 gcpscad     dxfwrite(tn,"20");
225 gcpscad     dxfwrite(tn,str(by));
226 gcpscad }
227 gcpscad
228 gcpscad module beginpolyline(bx,by,bz) {
229 gcpscad if (generatedxf == true) {
230 gcpscad     dxfwriteone("0");
231 gcpscad     dxfwriteone("POLYLINE");
232 gcpscad     dxfwriteone("8");
233 gcpscad     dxfwriteone("default");
234 gcpscad     dxfwriteone("66");
235 gcpscad     dxfwriteone("1");
236 gcpscad     dxfwriteone("70");
237 gcpscad     dxfwriteone("0");
238 gcpscad     dxfwriteone("0");
239 gcpscad     dxfwriteone("VERTEX");
```

```
240 gcpscad     dxfwriteone("8");
241 gcpscad     dxfwriteone("default");
242 gcpscad     dxfwriteone("70");
243 gcpscad     dxfwriteone("32");
244 gcpscad     dxfwriteone("10");
245 gcpscad     dxfwriteone(str(bx));
246 gcpscad     dxfwriteone("20");
247 gcpscad     dxfwriteone(str(by));
248 gcpscad     dxfbpl(current_tool(),bx,by);}
249 gcpscad }
250 gcpscad
251 gcpscad module dxfapl(tn,bx,by) {
252 gcpscad     dxfwrite(tn,"0");
253 gcpscad     dxfwrite(tn,"VERTEX");
254 gcpscad     dxfwrite(tn,"8");
255 gcpscad     dxfwrite(tn,"default");
256 gcpscad     dxfwrite(tn,"70");
257 gcpscad     dxfwrite(tn,"32");
258 gcpscad     dxfwrite(tn,"10");
259 gcpscad     dxfwrite(tn,str(bx));
260 gcpscad     dxfwrite(tn,"20");
261 gcpscad     dxfwrite(tn,str(by));
262 gcpscad }
263 gcpscad
264 gcpscad module addpolyline(bx,by,bz) {
265 gcpscad if (generatedxf == true) {
266 gcpscad     dxfwriteone("0");
267 gcpscad     dxfwriteone("VERTEX");
268 gcpscad     dxfwriteone("8");
269 gcpscad     dxfwriteone("default");
270 gcpscad     dxfwriteone("70");
271 gcpscad     dxfwriteone("32");
272 gcpscad     dxfwriteone("10");
273 gcpscad     dxfwriteone(str(bx));
274 gcpscad     dxfwriteone("20");
275 gcpscad     dxfwriteone(str(by));
276 gcpscad     dxfapl(current_tool(),bx,by);
277 gcpscad     }
278 gcpscad }
279 gcpscad
280 gcpscad module dxfcpl(tn) {
281 gcpscad     dxfwrite(tn,"0");
282 gcpscad     dxfwrite(tn,"SEQEND");
283 gcpscad }
284 gcpscad
285 gcpscad module closepolyline() {
286 gcpscad   if (generatedxf == true) {
287 gcpscad     dxfwriteone("0");
288 gcpscad     dxfwriteone("SEQEND");
289 gcpscad     dxfcpl(current_tool());
290 gcpscad   }
291 gcpscad }
292 gcpscad
293 gcpscad module writecomment(comment) {
294 gcpscad   if (generategcode == true) {
295 gcpscad     owritecomment(comment);
296 gcpscad   }
297 gcpscad }
```

At the end of the project it will be necessary to close each file using the commands: `pclosegcodefile` `pclosegcodefile`, and `closedxffile`. In some instances it may be necessary to write additional `closedxffile` information, depending on the file format. Note that these commands will need to be within the `gcodepreview` class.

```
575 gcpy     def dxfpostamble(self,tn):
576 gcpy #         self.writedxf(tn,str(tn))
577 gcpy          self.writedxf(tn,"0")
578 gcpy          self.writedxf(tn,"ENDSEC")
579 gcpy          self.writedxf(tn,"0")
580 gcpy          self.writedxf(tn,"EOF")
```

```
582 gcpy     def gcodepostamble(self):
583 gcpy          self.writegc("Z12.700")
584 gcpy          self.writegc("M05")
585 gcpy          self.writegc("M02")
```

It will be necessary to call the dxfpostamble (with appropriate checks and trappings so as to ensure that each dxf file is ended and closed so as to be valid.

```
587 gcpy     def closegcodefile(self):
588 gcpy         self.gcodepostamble()
589 gcpy         self.gc.close()
590 gcpy
591 gcpy     def closedxffile(self):
592 gcpy         if self.generatedxf == True:
593 gcpy             self.dxfpostamble(-1)
594 gcpy             self.dxf.close()
595 gcpy
596 gcpy     def closedxffiles(self):
597 gcpy         if self.generatedxfs == True:
598 gcpy             if (self.large_square_tool_num > 0):
599 gcpy                 self.dxfpostamble(self.large_square_tool_num)
600 gcpy             if (self.small_square_tool_num > 0):
601 gcpy                 self.dxfpostamble(self.small_square_tool_num)
602 gcpy             if (self.large_ball_tool_num > 0):
603 gcpy                 self.dxfpostamble(self.large_ball_tool_num)
604 gcpy             if (self.small_ball_tool_num > 0):
605 gcpy                 self.dxfpostamble(self.small_ball_tool_num)
606 gcpy             if (self.large_V_tool_num > 0):
607 gcpy                 self.dxfpostamble(self.large_V_tool_num)
608 gcpy             if (self.small_V_tool_num > 0):
609 gcpy                 self.dxfpostamble(self.small_V_tool_num)
610 gcpy             if (self.DT_tool_num > 0):
611 gcpy                 self.dxfpostamble(self.DT_tool_num)
612 gcpy             if (self.KH_tool_num > 0):
613 gcpy                 self.dxfpostamble(self.KH_tool_num)
614 gcpy             if (self.Roundover_tool_num > 0):
615 gcpy                 self.dxfpostamble(self.Roundover_tool_num)
616 gcpy             if (self.MISC_tool_num > 0):
617 gcpy                 self.dxfpostamble(self.MISC_tool_num)
618 gcpy
619 gcpy             if (self.large_square_tool_num > 0):
620 gcpy                 self.dxflgsq.close()
621 gcpy             if (self.small_square_tool_num > 0):
622 gcpy                 self.dxfsmsq.close()
623 gcpy             if (self.large_ball_tool_num > 0):
624 gcpy                 self.dxflgbl.close()
625 gcpy             if (self.small_ball_tool_num > 0):
626 gcpy                 self.dxfsmbl.close()
627 gcpy             if (self.large_V_tool_num > 0):
628 gcpy                 self.dxflgV.close()
629 gcpy             if (self.small_V_tool_num > 0):
630 gcpy                 self.dxfsmV.close()
631 gcpy             if (self.DT_tool_num > 0):
632 gcpy                 self.dxfDT.close()
633 gcpy             if (self.KH_tool_num > 0):
634 gcpy                 self.dxfKH.close()
635 gcpy             if (self.Roundover_tool_num > 0):
636 gcpy                 self.dxfRt.close()
637 gcpy             if (self.MISC_tool_num > 0):
638 gcpy                 self.dxfMt.close()
```

oclosegcodefile
oclosedxffile

In addition to the Python forms, there will need to be matching OpenSCAD commands to call them: `oclosegcodefile`, and `oclosedxffile`.

```
173 pyscad module oclosegcodefile() {
174 pyscad     pclosegcodefile();
175 pyscad }
176 pyscad
177 pyscad module oclosedxffile() {
178 pyscad     pclosedxffile();
179 pyscad }
180 pyscad
181 pyscad module oclosedxflgblfile() {
182 pyscad     pclosedxflgblfile();
183 pyscad }
184 pyscad
185 pyscad module oclosedxflgsqfile() {
186 pyscad     pclosedxflgsqfile();
187 pyscad }
188 pyscad
189 pyscad module oclosedxflgVfile() {
190 pyscad     pclosedxflgVfile();
```

```
191 pyscad }
192 pyscad
193 pyscad module oclosedxfsmblfile() {
194 pyscad     pclosedxfsmblfile();
195 pyscad }
196 pyscad
197 pyscad module oclosedxfsmsqfile() {
198 pyscad     pclosedxfsmsqfile();
199 pyscad }
200 pyscad
201 pyscad module oclosedxfsmVfile() {
202 pyscad     pclosedxfsmVfile();
203 pyscad }
204 pyscad
205 pyscad module oclosedxfDTfile() {
206 pyscad     pclosedxfDTfile();
207 pyscad }
208 pyscad
209 pyscad module oclosedxfKHfile() {
210 pyscad     pclosedxfKHfile();
211 pyscad }
```

closegcodefile
closedxffile
dxfpostamble

The commands: `closegcodefile`, and `closedxffile` are used to close the files at the end of a program. For efficiency, each references the command: `dxfpostamble` which when called provides the boilerplate needed at the end of their respective files.

```
299 gcpscad module closegcodefile() {
300 gcpscad   if (generategcode == true) {
301 gcpscad     owriteone("M05");
302 gcpscad     owriteone("M02");
303 gcpscad     oclosegcodefile();
304 gcpscad   }
305 gcpscad }
306 gcpscad
307 gcpscad module dxfpostamble(arg) {
308 gcpscad     dxfwrite(arg,"0");
309 gcpscad     dxfwrite(arg,"ENDSEC");
310 gcpscad     dxfwrite(arg,"0");
311 gcpscad     dxfwrite(arg,"EOF");
312 gcpscad }
313 gcpscad
314 gcpscad module closedxffile() {
315 gcpscad   if (generatedxf == true) {
316 gcpscad     dxfwriteone("0");
317 gcpscad     dxfwriteone("ENDSEC");
318 gcpscad     dxfwriteone("0");
319 gcpscad     dxfwriteone("EOF");
320 gcpscad     oclosedxffile();
321 gcpscad //    echo("CLOSING");
322 gcpscad     if (large_ball_tool_num >  0) {    dxfpostamble(
                    large_ball_tool_num);
323 gcpscad       oclosedxflgblfile();
324 gcpscad     }
325 gcpscad     if (large_square_tool_num >  0) {    dxfpostamble(
                    large_square_tool_num);
326 gcpscad       oclosedxflgsqfile();
327 gcpscad     }
328 gcpscad     if (large_V_tool_num >  0) {    dxfpostamble(large_V_tool_num);
329 gcpscad       oclosedxflgVfile();
330 gcpscad     }
331 gcpscad     if (small_ball_tool_num >  0) {    dxfpostamble(
                    small_ball_tool_num);
332 gcpscad       oclosedxfsmblfile();
333 gcpscad     }
334 gcpscad     if (small_square_tool_num >  0) {    dxfpostamble(
                    small_square_tool_num);
335 gcpscad       oclosedxfsmsqfile();
336 gcpscad     }
337 gcpscad     if (small_V_tool_num >  0) {    dxfpostamble(small_V_tool_num);
338 gcpscad       oclosedxfsmVfile();
339 gcpscad     }
340 gcpscad     if (DT_tool_num >  0) {    dxfpostamble(DT_tool_num);
341 gcpscad       oclosedxfDTfile();
342 gcpscad     }
343 gcpscad     if (KH_tool_num >  0) {    dxfpostamble(KH_tool_num);
344 gcpscad       oclosedxfKHfile();
345 gcpscad     }
346 gcpscad   }
```

```
347 gcpscad }
```

2.6 Movement and Cutting

otm
ocut
orapid
rapid
rapidbx

With all the scaffolding in place, it is possible to model the tool: `otm`, (colors the tool model so as to differentiate cut areas) and cutting: `ocut`, as well as Rapid movements to position the tool to begin a cut: `orapid`, `rapid`, and `rapidbx` which will also need to write out files which represent the desired machine motions.

The first command needs to be a move to/from the safe Z height. In G-code this would be:

```
(Move to safe Z to avoid workholding)
G53G0Z-5.000
```

but in the 3D model, since we do not know how tall the Z-axis is, we simply move to safe height and use that as a starting point:

```
640 gcpy     def movetosafeZ(self):
641 gcpy #         global toolpaths
642 gcpy          self.writegc("(Move␣to␣safe␣Z␣to␣avoid␣workholding)")
643 gcpy          self.writegc("G53G0Z-5.000")
644 gcpy          self.setzpos(self.retractheight)
645 gcpy          toolpath = cylinder(1.5875,12.7)
646 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
647 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
648 gcpy          return toolpath
```

toolpath

Note that a hard-coded cylinder is used since the command will be used prior to a toolchange. This also allows initializing the `toolpath` so that later commands may add to it.

There are three different movements in G-code which will need to be handled. Rapid commands will be used for G0 movements and will not appear in DXFs but will appear in G-code files, while cut (G1) and arc (G2/G3) commands will appear in both G-code and DXF files.

```
650 gcpy     def rapid(self, ex, ey, ez):
651 gcpy #         global toolpath
652 gcpy #         global toolpaths
653 gcpy          self.writegc("G00␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                      )
654 gcpy          start = self.currenttool()
655 gcpy          start = start.translate([self.xpos(), self.ypos(), self.
                      zpos()])
656 gcpy          toolpath = hull(start, start.translate([ex,ey,ez]))
657 gcpy          self.setxpos(ex)
658 gcpy          self.setypos(ey)
659 gcpy          self.setzpos(ez)
660 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
661 gcpy          return toolpath
```

```
663 gcpy     def rapidXY(self, ex, ey):
664 gcpy #         global toolpath
665 gcpy #         global toolpaths
666 gcpy          self.writegc("G00␣X", str(ex), "␣Y", str(ey))
667 gcpy          start = self.currenttool()
668 gcpy          start = start.translate([self.xpos(), self.ypos(), self.
                      zpos()])
669 gcpy          toolpath = hull(start, start.translate([ex,ey,self.zpos()])
                      )
670 gcpy          self.setxpos(ex)
671 gcpy          self.setypos(ey)
672 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
673 gcpy          return toolpath
```

```
675 gcpy     def rapidZ(self, ez):
676 gcpy #         global toolpath
677 gcpy #         global toolpaths
678 gcpy          self.writegc("G00␣Z", str(ez))
679 gcpy          start = self.currenttool()
680 gcpy          start = start.translate([self.xpos(), self.ypos(), self.
                      zpos()])
681 gcpy          toolpath = hull(start, start.translate([self.xpos(),self.
                      ypos(),ez]))
682 gcpy          self.setzpos(ez)
683 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
```

```
684 gcpy         return toolpath
```

cut... The Python commands `cut...` add the `currenttool` to the `toolpath` `hull`ed together at the current position and the end position of the move.

```
686 gcpy     def cutlinedxfgc(self,ex, ey, ez):
687 gcpy #         global toolpath
688 gcpy #         global toolpaths
689 gcpy         self.dxfpolyline(self.currenttool(), self.xpos(), self.ypos
                    (), ex, ey)
690 gcpy         self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                    )
691 gcpy         start = self.currenttool()
692 gcpy         start = start.translate([self.xpos(), self.ypos(), self.
                    zpos()])
693 gcpy         end = self.currenttool()
694 gcpy         toolpath = hull(start, end.translate([ex,ey,ez]))
695 gcpy         self.setxpos(ex)
696 gcpy         self.setypos(ey)
697 gcpy         self.setzpos(ez)
698 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
699 gcpy         return toolpath
700 gcpy
701 gcpy     def cutline(self,ex, ey, ez):
702 gcpy #         global toolpath
703 gcpy #         global toolpaths
704 gcpy         start = self.currenttool()
705 gcpy         start = start.translate([self.xpos(), self.ypos(), self.
                    zpos()])
706 gcpy         end = self.currenttool()
707 gcpy         toolpath = hull(start, end.translate([ex,ey,ez]))
708 gcpy         self.setxpos(ex)
709 gcpy         self.setypos(ey)
710 gcpy         self.setzpos(ez)
711 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
712 gcpy         return toolpath
```

```
349 gcpscad module otm(ex, ey, ez, r,g,b) {
350 gcpscad color([r,g,b]) hull(){
351 gcpscad     translate([xpos(), ypos(), zpos()]){
352 gcpscad       select_tool(current_tool());
353 gcpscad     }
354 gcpscad     translate([ex, ey, ez]){
355 gcpscad       select_tool(current_tool());
356 gcpscad     }
357 gcpscad   }
358 gcpscad oset(ex, ey, ez);
359 gcpscad }
360 gcpscad
361 gcpscad module ocut(ex, ey, ez) {
362 gcpscad   //color([0.2,1,0.2]) hull(){
363 gcpscad   otm(ex, ey, ez, 0.2,1,0.2);
364 gcpscad }
365 gcpscad
366 gcpscad module orapid(ex, ey, ez) {
367 gcpscad   //color([0.93,0,0]) hull(){
368 gcpscad   otm(ex, ey, ez, 0.93,0,0);
369 gcpscad }
370 gcpscad
371 gcpscad module rapidbx(bx, by, bz, ex, ey, ez) {
372 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
373 gcpscad   if (generategcode == true) {
374 gcpscad     writecomment("rapid");
375 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
376 gcpscad   }
377 gcpscad     orapid(ex, ey, ez);
378 gcpscad }
379 gcpscad
380 gcpscad module rapid(ex, ey, ez) {
381 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
382 gcpscad   if (generategcode == true) {
383 gcpscad       writecomment("rapid");
384 gcpscad       owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
385 gcpscad   }
386 gcpscad   orapid(ex, ey, ez);
387 gcpscad }
```

```
388 gcpscad
389 gcpscad module movetosafez() {
390 gcpscad   //this should be move to retract height
391 gcpscad   if (generategcode == true) {
392 gcpscad       writecomment("Move to safe Z to avoid workholding");
393 gcpscad       owriteone("G53G0Z-5.000");
394 gcpscad   }
395 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
396 gcpscad }
397 gcpscad
398 gcpscad module begintoolpath(bx,by,bz) {
399 gcpscad   if (generategcode == true) {
400 gcpscad     writecomment("PREPOSITION FOR RAPID PLUNGE");
401 gcpscad     owritefour("G0X", str(bx), "Y",str(by));
402 gcpscad     owritetwo("Z", str(bz));
403 gcpscad   }
404 gcpscad   orapid(bx,by,bz);
405 gcpscad }
406 gcpscad
407 gcpscad module movetosafeheight() {
408 gcpscad   //this should be move to machine position
409 gcpscad   if (generategcode == true) {
410 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
411 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge)
412 gcpscad     if (zeroheight == "Top") {
413 gcpscad       owritetwo("Z",str(retractheight));
414 gcpscad     }
415 gcpscad   }
416 gcpscad     orapid(getxpos(), getypos(), retractheight+55);
417 gcpscad }
418 gcpscad
419 gcpscad module cutoneaxis_setfeed(axis,depth,feed) {
420 gcpscad   if (generategcode == true) {
421 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
422 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge) G1Z7.612F381.0
423 gcpscad     if (zeroheight == "Top") {
424 gcpscad       owritefive("G1",axis,str(depth),"F",str(feed));
425 gcpscad     }
426 gcpscad   }
427 gcpscad   if (axis == "X") {setxpos(depth);
428 gcpscad     ocut(depth, getypos(), getzpos());}
429 gcpscad     if (axis == "Y") {setypos(depth);
430 gcpscad       ocut(getxpos(), depth, getzpos());
431 gcpscad     }
432 gcpscad       if (axis == "Z") {setzpos(depth);
433 gcpscad         ocut(getxpos(), getypos(), depth);
434 gcpscad       }
435 gcpscad }
436 gcpscad
437 gcpscad module cut(ex, ey, ez) {
438 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
439 gcpscad   if (generategcode == true) {
440 gcpscad      owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
441 gcpscad   }
442 gcpscad   //if (generatesvg == true) {
443 gcpscad   //    owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
444 gcpscad   //    orapid(getxpos(), getypos(), retractheight+5);
445 gcpscad   //    writesvgline(getxpos(),getypos(),ex,ey);
446 gcpscad   //}
447 gcpscad   ocut(ex, ey, ez);
448 gcpscad }
449 gcpscad
450 gcpscad module cutwithfeed(ex, ey, ez, feed) {
451 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
452 gcpscad   if (generategcode == true) {
453 gcpscad   //    writecomment("rapid");
454 gcpscad     owriteeight("G1 X",str(ex)," Y", str(ey), " Z", str(ez),"F",str
                (feed));
455 gcpscad   }
456 gcpscad   ocut(ex, ey, ez);
457 gcpscad }
458 gcpscad
459 gcpscad module endtoolpath() {
460 gcpscad   if (generategcode == true) {
461 gcpscad   //Z31.750
462 gcpscad   //    owriteone("G53G0Z-5.000");
463 gcpscad     owritetwo("Z",str(retractheight));
464 gcpscad   }
```

```
465 gcpscad    orapid(getxpos(),getypos(),retractheight);
466 gcpscad }
```

3 Cutting shapes, cut2Dshapes, and expansion

Certain basic shapes (arcs, circles, rectangles), will be incorporated in the main code. Other shapes will be added to the additional/optional file, `cut2Dshapes.scad` as they are developed, and of course the user is free to develop their own systems.

It is most expedient to test out new features in a new/separate file insofar as the file structures will allow (tool definitions for example will need to consolidated in 2.4.2) which will need to be included in the projects which will make use of said features until such time as they are added into the main `gcodepreview.scad` file.

A basic requirement for two-dimensional regions will be to define them so as to cut them out. Two different geometric treatments will be necessary: modeling the geometry which defines the region to be cut out (output as a DXF); and modeling the movement of the tool, the toolpath which will be used in creating the 3D model and outputting the G-code.

In the TUG presentation/paper: http://tug.org/TUGboat/tb40-2/tb125adams-3d.pdf a list of 2D shapes was put forward — which of these will need to be created, or if some more general solution will be put forward is uncertain. For the time being, shapes will be implemented on an as-needed basis, as modified by the interaction with the requirements of toolpaths.

The program Carbide Create has toolpath types and options which are as follows:

- Contour — No Offset — the default, this is already supported in the existing code
- Contour — Outside Offset
- Contour — Inside Offset
- Pocket — such toolpaths/geometry should include the rounding of the tool at the corners, c.f., cutrectangledxf
- Drill — note that this is implemented as the plunging of a tool centered on a circle and normally that circle is the same diameter as the tool which is used.
- Keyhole — also beginning from a circle, a nice feature for this would be to include/model the areas which should be cleared for the sake of reducing wear on the tool and ensuring chip clearance

Some further considerations:

- relationship of geometry to toolpath — arguably there should be an option for each toolpath (we will use Carbide Create as a reference implementation) which is to be supported. Note that there are several possibilities: modeling the tool movement, describing the outline which the tool will cut, modeling a reference shape for the toolpath
- tool geometry — it should be possible to include support for specialty tooling such as dovetail cutters and to get an accurate 3D model, esp. for tooling which undercuts since they cannot be modeled in Carbide Create.
- feeds and speeds — if outputting G-code it would be nice to be able to import feeds and speeds from external files such as the .csv files used for user tool libraries in Carbide Create
- Starting and Max Depth — are there CAD programs which will make use of Z-axis information in a DXF? — would it be possible/necessary to further differentiate the DXF geometry? (currently written out separately for each toolpath in addition to one combined file)

3.1 Arcs for toolpaths and DXFs

A further consideration here is that G-code supports arcs in addition to the lines and polylines already implemented.

Implementing arcs wants at least the following options for quadrant and direction:

- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC — upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW

- 0
  - circle
  - ellipse (oval) (requires some sort of non-arc curve)
    - egg-shaped
  - annulus (one circle within another, forming a ring)
  - superellipse (see astroid below)
- 1
  - cone with rounded end (arc)see also "sector" under 3 below
- 2
  - semicircle/circular/half-circle segment (arc and a straight line); see also sector below
  - arch—curve possibly smoothly joining a pair of straight lines with a flat bottom
  - lens/vesica piscis (two convex curves)
  - lune/crescent (one convex, one concave curve)
  - heart (two curves)
  - tomoe (comma shape)—non-arc curves
- 3
  - triangle
    - equilateral
    - isosceles
    - right triangle
    - scalene
  - (circular) sector (two straight edges, one convex arc)
    - quadrant (90°)
    - sextants (60°)
    - octants (45°)
  - deltoid curve (three concave arcs)
  - Reuleaux triangle (three convex arcs)
  - arbelos (one convex, two concave arcs)
  - two straight edges, one concave arc—an example is the hyperbolic sector[1]
  - two convex, one concave arc
- 4
  - rectangle (including square) — cutrectangledxf, cutoutrectangledxf, rectangleoutlinedxf
  - parallelogram
  - rhombus
  - trapezoid/trapezium
  - kite
  - ring/annulus segment (straight line, concave arc, straight line, convex arc)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

Is the list of shapes for which there are not widely known names interesting for its lack of notoriety?

- two straight edges, one concave arcoddly, an asymmetric form (hyperbolic sector) has a name, but not the symmetrical—while the colloquial/prosaic arrowhead was considered, it was rejected as being better applied to the shape below. (Its also the shape used for the spaceship in the game Asteroids (or Hyperspace), but that is potentially confusing with astroid.) At the conference, Dr. Knuth suggested dart as a suitable term.
- two convex, one concave arcwith the above named, the term arrowhead is freed up to use as the name for this shape.
- three straight lines and one concave arc.

The first in particular is sorely needed for this project (its the result of inscribing a circle in a square or other regular geometric shape). Do these shapes have names in any other languages which might be used instead?

- cutarcNECC
- cutcircleCW — while it wont matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCCdxf

It will be necessary to have two separate representations of arcs — the DXF may be easily and directly supported with a single command, but representing the matching tool movement in OpenSCAD will require a series of short line movements which approximate the arc. At this time, the current version of Carbide Create only imports circles in DXF as curves, any other example is converted into polylines — unfortunately, the implementation of this is not such as would allow directly matching that representation. A work-around to import a DXF as curves is to convert the arc into a reasonable number of line segments so as to approximate the arc.

Note that there are the following representations/interfaces for representing an arc:

- G-code — G2 (clockwise) and G3 (counter-clockwise) arcs may be specified, and since the endpoint is the positional requirement, it is most likely best to use the offset to the center (I and J), rather than the radius parameter (K) `G2/3 ...`
- DXF — `dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)`
- approximation of arc using lines (OpenSCAD) — note that this may also be used in DXF so as to sidestep the question of how many line segments there would be for a given arc representation

Cutting the quadrant arcs will greatly simplify the calculation and interface for the modules. A full set of 8 will be necessary, then circles may either be stitched together manually or a pair of modules made for them.

At this time, despite what the module names imply (`cutarcNWCWdxf`, &c.), only cutting and DXF generation is supported. Adding support for G-code will be done at a later time. Since these modules will ultimately support G-code, the interface will assume the stored `xpos` and `ypos` as the origin. Parameters which will need to be passed in are:

- `tn`
- `ex`
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical toolpaths
- `xcenter` — the center position will be specified as an absolute position which will require calculating the offset when it is used for G-code's `IJ`, for which `xctr/yctr` are suggested
- `ycenter`
- `radius` — while this could be calculated, passing it in as a parameter is both convenient and acts as a check on the other parameters

`arcloop` `narcloop`

Since OpenSCAD does not have an arc movement command it is necessary to iterate through a loop: `arcloop` (clockwise), `narcloop` (counterclockwise) to handle the drawing and processing of the `cut()` toolpaths as short line segments which additionally affords a single point of control for adding additional features such as allowing the depth to vary as one cuts along an arc. Note that the definition matches the `DXF` definition of defining the center position with a matching radius, but it will be necessary to move the tool to the actual origin, and to calculate the end position when writing out a `G2/G3` arc.

```
714 gcpy     def arcloop(self, barc, earc, xcenter, ycenter, radius):
715 gcpy #         global toolpath
716 gcpy         toolpath = self.currenttool()
717 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                     .zpos()])
718 gcpy         i = barc
719 gcpy         while i < earc:
720 gcpy             toolpath = toolpath.union(self.cutline(xcenter + radius
                          * math.cos(math.radians(i)), ycenter + radius *
                         math.sin(math.radians(i)), self.zpos()-(self.tzpos()
                         )))
721 gcpy             self.setxpos(xcenter + radius * math.cos(math.radians(i
                         )))
722 gcpy             self.setypos(ycenter + radius * math.sin(math.radians(i
                         )))
723 gcpy             i += 1
724 gcpy #         self.dxfarc(xcenter, ycenter, radius, barc, earc, self.
            currenttoolnumber())
725 gcpy         return toolpath
```

```
726 gcpy
727 gcpy     def narcloop(barc,earc, xcenter, ycenter, radius):
728 gcpy #         global toolpath
729 gcpy         toolpath = self.currenttool()
730 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
731 gcpy         i = barc
732 gcpy         while i > earc:
733 gcpy             toolpath = toolpath.union(self.cutline(xcenter + radius
                         * math.cos(math.radians(i)), ycenter + radius *
                        math.sin(math.radians(i)), self.zpos()-(self.tzpos()
                        )))
734 gcpy             self.setxpos(xcenter + radius * math.cos(math.radians(i
                        )))
735 gcpy             self.setypos(ycenter + radius * math.sin(math.radians(i
                        )))
736 gcpy             print(str(self.xpos()), str(self.ypos()))
737 gcpy             i += -1
738 gcpy #         self.dxfarc(xcenter, ycenter, radius, barc, earc, self.
        currenttoolnumber())
739 gcpy         return toolpath
```

There are specific commands for writing out the DXF and G-code files. Note that for the G-code version it will be necessary to calculate the end-position.

```
741 gcpy     def dxfarc(self, xcenter, ycenter, radius, anglebegin, endangle
                , tn):
742 gcpy         if (self.generatedxf == True):
743 gcpy             self.writedxf(tn, "0")
744 gcpy             self.writedxf(tn, "ARC")
745 gcpy             self.writedxf(tn, "10")
746 gcpy             self.writedxf(tn, str(xcenter))
747 gcpy             self.writedxf(tn, "20")
748 gcpy             self.writedxf(tn, str(ycenter))
749 gcpy             self.writedxf(tn, "40")
750 gcpy             self.writedxf(tn, str(radius))
751 gcpy             self.writedxf(tn, "50")
752 gcpy             self.writedxf(tn, str(anglebegin))
753 gcpy             self.writedxf(tn, "51")
754 gcpy             self.writedxf(tn, str(endangle))
755 gcpy
756 gcpy     def gcodearc(self, xcenter, ycenter, radius, anglebegin,
                endangle, tn):
757 gcpy         if (self.generategcode == True):
758 gcpy             self.writegc(tn, "(0)")
```

The various textual versions are quite obvious, and due to the requirements of G-code, it is easiest to include the G-code in them if it is wanted.

```
760 gcpy     def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
761 gcpy #         global toolpath
762 gcpy         toolpath = self.currenttool()
763 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
764 gcpy         self.dxfarc(xcenter,ycenter,radius,0,90, self.
                    currenttoolnumber())
765 gcpy         if (self.zpos == ez):
766 gcpy             self.settzpos(0)
767 gcpy         else:
768 gcpy             self.settzpos((self.zpos()-ez)/90)
769 gcpy         toolpath = self.arcloop(1,90, xcenter, ycenter, radius)
770 gcpy         self.setxpos(ex)
771 gcpy         self.setypos(ey)
772 gcpy         self.setzpos(ez)
773 gcpy         return toolpath
774 gcpy
775 gcpy     def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
776 gcpy #         global toolpath
777 gcpy         toolpath = self.currenttool()
778 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
779 gcpy         self.dxfarc(xcenter,ycenter,radius,90,180, self.
                    currenttoolnumber())
780 gcpy         if (self.zpos == ez):
781 gcpy             self.settzpos(0)
782 gcpy         else:
783 gcpy             self.settzpos((self.zpos()-ez)/90)
```

```
784 gcpy          toolpath = self.arcloop(91,180, xcenter, ycenter, radius)
785 gcpy          self.setxpos(ex)
786 gcpy          self.setypos(ey)
787 gcpy          self.setzpos(ez)
788 gcpy          return toolpath
789 gcpy
790 gcpy      def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
791 gcpy #          global toolpath
792 gcpy          toolpath = self.currenttool()
793 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
794 gcpy          self.dxfarc(xcenter,ycenter,radius,180,270, self.
                      currenttoolnumber())
795 gcpy          if (self.zpos == ez):
796 gcpy              self.settzpos(0)
797 gcpy          else:
798 gcpy              self.settzpos((self.zpos()-ez)/90)
799 gcpy          toolpath = self.arcloop(181,270, xcenter, ycenter, radius)
800 gcpy          self.setxpos(ex)
801 gcpy          self.setypos(ey)
802 gcpy          self.setzpos(ez)
803 gcpy          return toolpath
804 gcpy
805 gcpy      def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
806 gcpy #          global toolpath
807 gcpy          toolpath = self.currenttool()
808 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
809 gcpy          self.dxfarc(xcenter,ycenter,radius,270,360, self.
                      currenttoolnumber())
810 gcpy          if (self.zpos == ez):
811 gcpy              self.settzpos(0)
812 gcpy          else:
813 gcpy              self.settzpos((self.zpos()-ez)/90)
814 gcpy          toolpath = self.arcloop(271,360, xcenter, ycenter, radius)
815 gcpy          self.setxpos(ex)
816 gcpy          self.setypos(ey)
817 gcpy          self.setzpos(ez)
818 gcpy          return toolpath
819 gcpy
820 gcpy      def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
821 gcpy #          global toolpath
822 gcpy          toolpath = self.currenttool()
823 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
824 gcpy          self.dxfarc(xcenter,ycenter,radius,0,90, self.
                      currenttoolnumber())
825 gcpy          if (self.zpos == ez):
826 gcpy              self.settzpos(0)
827 gcpy          else:
828 gcpy              self.settzpos((self.zpos()-ez)/90)
829 gcpy          toolpath = self.narcloop(89,0, xcenter, ycenter, radius)
830 gcpy          self.setxpos(ex)
831 gcpy          self.setypos(ey)
832 gcpy          self.setzpos(ez)
833 gcpy          return toolpath
834 gcpy
835 gcpy      def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
836 gcpy #          global toolpath
837 gcpy          toolpath = self.currenttool()
838 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
839 gcpy          self.dxfarc(xcenter,ycenter,radius,270,360, self.
                      currenttoolnumber())
840 gcpy          if (self.zpos == ez):
841 gcpy              self.settzpos(0)
842 gcpy          else:
843 gcpy              self.settzpos((self.zpos()-ez)/90)
844 gcpy          toolpath = self.narcloop(359,270, xcenter, ycenter, radius)
845 gcpy          self.setxpos(ex)
846 gcpy          self.setypos(ey)
847 gcpy          self.setzpos(ez)
848 gcpy          return toolpath
849 gcpy
850 gcpy      def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
851 gcpy #          global toolpath
852 gcpy          toolpath = self.currenttool()
853 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
```

```
                    .zpos()])
854 gcpy          self.dxfarc(xcenter,ycenter,radius,180,270, self.
                    currenttoolnumber())
855 gcpy          if (self.zpos == ez):
856 gcpy              self.settzpos(0)
857 gcpy          else:
858 gcpy              self.settzpos((self.zpos()-ez)/90)
859 gcpy          toolpath = self.narcloop(269,180, xcenter, ycenter, radius)
860 gcpy          self.setxpos(ex)
861 gcpy          self.setypos(ey)
862 gcpy          self.setzpos(ez)
863 gcpy          return toolpath
864 gcpy
865 gcpy      def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
866 gcpy #          global toolpath
867 gcpy          toolpath = self.currenttool()
868 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
869 gcpy          self.dxfarc(xcenter,ycenter,radius,90,180, self.
                    currenttoolnumber())
870 gcpy          if (self.zpos == ez):
871 gcpy              self.settzpos(0)
872 gcpy          else:
873 gcpy              self.settzpos((self.zpos()-ez)/90)
874 gcpy          toolpath = self.narcloop(179,90, xcenter, ycenter, radius)
875 gcpy          self.setxpos(ex)
876 gcpy          self.setypos(ey)
877 gcpy          self.setzpos(ez)
878 gcpy          return toolpath
```

Using such commands to create a circle is quite straight-forward:

```
cutarcNECCdxf(-stockXwidth/4, stockYheight/4+stockYheight/16, -stockZthickness, -stockXwidth/4, stockYh
cutarcNWCCdxf(-(stockXwidth/4+stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stock
cutarcSWCCdxf(-stockXwidth/4, stockYheight/4-stockYheight/16, -stockZthickness, -stockXwidth/4, stockYh
cutarcSECCdxf(-(stockXwidth/4-stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stock
```

3.2 Keyhole toolpath and undercut tooling

cutkeyhole toolpath

The first topologically unusual toolpath is `cutkeyhole toolpath` — where other toolpaths have a direct correspondence between the associated geometry and the area cut, that Keyhole toolpaths may be used with tooling which undercuts will result in the creation of two different physical physical regions: the visible surface matching the union of the tool perimeter at the entry point and the linear movement of the shaft and the larger region of the tool perimeter at the depth which the tool is plunged to and moved along.

Tooling for such toolpaths is defined at paragraph 2.4.1.2

Due to the possibility of rotation, for the in-between positions there are more cases than one would think for each quadrant there are the following possibilities:

- one node on the clockwise side is outside of the quadrant
- two nodes on the clockwise side are outside of the quadrant
- all nodes are w/in the quadrant
- one node on the counter-clockwise side is outside of the quadrant
- two nodes on the counter-clockwise side are outside of the quadrant

Supporting all of these would require trigonometric comparisons in the If else blocks, so only the 4 quadrants, N, S, E, and W will be supported in the initial version. This will be done by wrapping the command with a version which only accepts those options:

```
468 gcpscad module cutkeyhole_toolpath(kh_start_depth, kh_max_depth,
              kht_direction, kh_distance, kh_tool_num) {
469 gcpscad if (kht_direction == "N") {
470 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 90,
                kh_distance, kh_tool_num);
471 gcpscad   } else if (kht_direction == "S") {
472 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 270,
                kh_distance, kh_tool_num);
473 gcpscad   } else if (kht_direction == "E") {
474 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 0,
                kh_distance, kh_tool_num);
475 gcpscad   } else if (kht_direction == "W") {
476 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 180,
                kh_distance, kh_tool_num);
477 gcpscad   }
478 gcpscad }
```

cutKH toolpath degrees

The original version of the command, `cutKH toolpath degrees` retains an interface which allows calling it for arbitrary beginning and ending points of an arc. Note that code is still present for the partial calculation of one quadrant (for the case of all nodes within the quadrant).

The first task is to place a circle at the origin which is invariant of angle:

```
480 gcpscad module cutKH_toolpath_degrees(kh_start_depth, kh_max_depth,
        kh_angle, kh_distance, kh_tool_num) {
481 gcpscad //Circle at entry hole
482 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (7))/2,0,90,
        KH_tool_num);
483 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (7))
        /2,90,180, KH_tool_num);
484 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (7))
        /2,180,270, KH_tool_num);
485 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (7))
        /2,270,360, KH_tool_num);
```

Then it will be necessary to test for each possible case in a series of If Else blocks:

```
  1 gcpscad //Outlines of entry hole and slot
  2 gcpscad   if (kh_angle == 0) {
  3 gcpscad     //Lower left of entry hole
  4 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
  5 gcpscad     //Upper left of entry hole
  6 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,90,180, KH_tool_num);
  7 gcpscad     //Upper right of entry hole
  8 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,90-acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 90,
                    KH_tool_num);
  9 gcpscad     //Lower right of entry hole
 10 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,270, 270+acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num)
                    ;
 11 gcpscad     //Actual line of cut
 12 gcpscad     dxfpolyline(getxpos(),getypos(),getxpos()+kh_distance,getypos()
                    );
 13 gcpscad     //upper right of slot
 14 gcpscad     dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(
                    KH_tool_num, (kh_max_depth+4.36))/2,0,90, KH_tool_num);
 15 gcpscad     //lower right of slot
 16 gcpscad     dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(
                    KH_tool_num, (kh_max_depth+4.36))/2,270,360, KH_tool_num);
 17 gcpscad     //upper right slot
 18 gcpscad     dxfpolyline(
 19 gcpscad         getxpos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                        tool_diameter(KH_tool_num,5)^2))/2),
 20 gcpscad         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
                        (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
                        kh_max_depth-6.34))/2)^2,
 21 gcpscad         getxpos()+kh_distance,
 22 gcpscad     //end position at top of slot
 23 gcpscad         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
 24 gcpscad         KH_tool_num);
 25 gcpscad     //lower right slot
 26 gcpscad     dxfpolyline(
 27 gcpscad         getxpos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                        tool_diameter(KH_tool_num,5)^2))/2),
 28 gcpscad         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
                        (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
                        kh_max_depth-6.34))/2)^2,
 29 gcpscad         getxpos()+kh_distance,
 30 gcpscad     //end position at top of slot
 31 gcpscad         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
 32 gcpscad         KH_tool_num);
 33 gcpscad     hull(){
 34 gcpscad       translate([xpos(), ypos(), zpos()]){
 35 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
 36 gcpscad       }
 37 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
 38 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
 39 gcpscad       }
 40 gcpscad     }
 41 gcpscad     hull(){
 42 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
 43 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
 44 gcpscad       }
 45 gcpscad       translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth]){
```

```
46 gcpscad          gcp_keyhole_shaft(6.35, 9.525);
47 gcpscad        }
48 gcpscad      }
49 gcpscad      cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
50 gcpscad      cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed)
                    ;
51 gcpscad      setxpos(getxpos()-kh_distance);
52 gcpscad    } else if (kh_angle > 0 && kh_angle < 90) {
53 gcpscad //echo(kh_angle);
54 gcpscad    dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
                   kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_num);
55 gcpscad    dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
                   kh_max_depth))/2,180+kh_angle,270+kh_angle, KH_tool_num);
56 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (kh_max_depth
               ))/2,kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth
               +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)),90+
               kh_angle, KH_tool_num);
57 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (kh_max_depth
               ))/2,270+kh_angle,360+kh_angle-asin((tool_diameter(KH_tool_num,
               (kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num, (
               kh_max_depth))/2)), KH_tool_num);
58 gcpscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
59 gcpscad    getypos()+(kh_distance*sin(kh_angle)),tool_diameter(KH_tool_num,
                   (kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle, KH_tool_num);
60 gcpscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(kh_distance
               *sin(kh_angle)),tool_diameter(KH_tool_num, (kh_max_depth+4.36))
               /2,270+kh_angle,360+kh_angle, KH_tool_num);
61 gcpscad dxfpolyline( getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))
               /2*cos(kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth
               +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
62 gcpscad  getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(
                kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
                /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
63 gcpscad  getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num,
                (kh_max_depth+4.36))/2)*sin(kh_angle)),
64 gcpscad  getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num,
                (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
65 gcpscad //echo("a",tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
66 gcpscad //echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
67 gcpscad echo("Aangle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
               /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
68 gcpscad //echo(kh_angle);
69 gcpscad  cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
                kh_distance*sin(kh_angle)),-kh_max_depth,feed);
70 gcpscad  setxpos(getxpos()-(kh_distance*cos(kh_angle)));
71 gcpscad  setypos(getypos()-(kh_distance*sin(kh_angle)));
72 gcpscad   } else if (kh_angle == 90) {
73 gcpscad     //Lower left of entry hole
74 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
75 gcpscad     //Lower right of entry hole
76 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
77 gcpscad     //Upper right of entry hole
78 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,0,acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num)
                    ;
79 gcpscad     //Upper left of entry hole
80 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,180-acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180,
                    KH_tool_num);
81 gcpscad     //Actual line of cut
82 gcpscad     dxfpolyline(getxpos(),getypos(),getxpos(),getypos()+kh_distance
                    );
83 gcpscad     //upper right of slot
84 gcpscad     dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
                    KH_tool_num, (kh_max_depth+4.36))/2,0,90, KH_tool_num);
85 gcpscad     //upper left of slot
86 gcpscad     dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
                    KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
87 gcpscad     //right of slot
88 gcpscad     dxfpolyline(
89 gcpscad         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
90 gcpscad         getypos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                        tool_diameter(KH_tool_num,5)^2))/2),//( (kh_max_depth
                        -6.34))/2)^2-(tool_diameter(KH_tool_num, (kh_max_depth
                        -6.34))/2)^2,
91 gcpscad         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
92 gcpscad     //end position at top of slot
93 gcpscad         getypos()+kh_distance,
```

```
94 gcpscad          KH_tool_num);
95 gcpscad      dxfpolyline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth)
                    )/2, getypos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                    tool_diameter(KH_tool_num,5)^2))/2), getxpos()-tool_diameter
                    (KH_tool_num, (kh_max_depth+6.35))/2,getypos()+kh_distance,
                    KH_tool_num);
96 gcpscad      hull(){
97 gcpscad        translate([xpos(), ypos(), zpos()]){
98 gcpscad          gcp_keyhole_shaft(6.35, 9.525);
99 gcpscad        }
100 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
101 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
102 gcpscad       }
103 gcpscad     }
104 gcpscad     hull(){
105 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
106 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
107 gcpscad       }
108 gcpscad       translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth]){
109 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
110 gcpscad       }
111 gcpscad     }
112 gcpscad     cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
113 gcpscad     cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed)
                    ;
114 gcpscad     setypos(getypos()-kh_distance);
115 gcpscad   } else if (kh_angle == 180) {
116 gcpscad     //Lower right of entry hole
117 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
118 gcpscad     //Upper right of entry hole
119 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
120 gcpscad     //Upper left of entry hole
121 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,90, 90+acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num)
                    ;
122 gcpscad     //Lower left of entry hole
123 gcpscad     dxfarc(getxpos(),getypos(),9.525/2, 270-acos(tool_diameter(
                    KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270,
                    KH_tool_num);
124 gcpscad     //upper left of slot
125 gcpscad     dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
                    KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
126 gcpscad     //lower left of slot
127 gcpscad     dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
                    KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
128 gcpscad     //Actual line of cut
129 gcpscad     dxfpolyline(getxpos(),getypos(),getxpos()-kh_distance,getypos()
                    );
130 gcpscad     //upper left slot
131 gcpscad     dxfpolyline(
132 gcpscad         getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                        tool_diameter(KH_tool_num,5)^2))/2),
133 gcpscad         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
                        (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
                        kh_max_depth-6.34))/2)^2,
134 gcpscad         getxpos()-kh_distance,
135 gcpscad     //end position at top of slot
136 gcpscad         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
137 gcpscad         KH_tool_num);
138 gcpscad     //lower right slot
139 gcpscad     dxfpolyline(
140 gcpscad         getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                        tool_diameter(KH_tool_num,5)^2))/2),
141 gcpscad         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
                        (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
                        kh_max_depth-6.34))/2)^2,
142 gcpscad         getxpos()-kh_distance,
143 gcpscad     //end position at top of slot
144 gcpscad         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
145 gcpscad         KH_tool_num);
146 gcpscad     hull(){
147 gcpscad       translate([xpos(), ypos(), zpos()]){
148 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
149 gcpscad       }
150 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
151 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
152 gcpscad       }
153 gcpscad     }
```

```
154 gcpscad     hull(){
155 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
156 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
157 gcpscad       }
158 gcpscad       translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth]){
159 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
160 gcpscad       }
161 gcpscad     }
162 gcpscad     cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
163 gcpscad     cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed)
                ;
164 gcpscad     setxpos(getxpos()+kh_distance);
165 gcpscad   } else if (kh_angle == 270) {
166 gcpscad     //Upper right of entry hole
167 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
168 gcpscad     //Upper left of entry hole
169 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,90,180, KH_tool_num);
170 gcpscad     //lower right of slot
171 gcpscad     dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
                KH_tool_num, (kh_max_depth+4.36))/2,270,360, KH_tool_num);
172 gcpscad     //lower left of slot
173 gcpscad     dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
                KH_tool_num, (kh_max_depth+4.36))/2,180,270, KH_tool_num);
174 gcpscad     //Actual line of cut
175 gcpscad     dxfpolyline(getxpos(),getypos(),getxpos(),getypos()-kh_distance
                );
176 gcpscad     //right of slot
177 gcpscad     dxfpolyline(
178 gcpscad         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
179 gcpscad         getypos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                    tool_diameter(KH_tool_num,5)^2))/2),//( (kh_max_depth
                    -6.34))/2)^2-(tool_diameter(KH_tool_num, (kh_max_depth
                    -6.34))/2)^2,
180 gcpscad         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
181 gcpscad     //end position at top of slot
182 gcpscad         getypos()-kh_distance,
183 gcpscad         KH_tool_num);
184 gcpscad     //left of slot
185 gcpscad     dxfpolyline(
186 gcpscad         getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
187 gcpscad         getypos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
                    tool_diameter(KH_tool_num,5)^2))/2),//( (kh_max_depth
                    -6.34))/2)^2-(tool_diameter(KH_tool_num, (kh_max_depth
                    -6.34))/2)^2,
188 gcpscad         getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
189 gcpscad     //end position at top of slot
190 gcpscad         getypos()-kh_distance,
191 gcpscad         KH_tool_num);
192 gcpscad     //Lower right of entry hole
193 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,360-acos(tool_diameter(
                KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360,
                KH_tool_num);
194 gcpscad     //Lower left of entry hole
195 gcpscad     dxfarc(getxpos(),getypos(),9.525/2,180, 180+acos(tool_diameter(
                KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num)
                ;
196 gcpscad     hull(){
197 gcpscad       translate([xpos(), ypos(), zpos()]){
198 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
199 gcpscad       }
200 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
201 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
202 gcpscad       }
203 gcpscad     }
204 gcpscad     hull(){
205 gcpscad       translate([xpos(), ypos(), zpos()-kh_max_depth]){
206 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
207 gcpscad       }
208 gcpscad       translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth]){
209 gcpscad         gcp_keyhole_shaft(6.35, 9.525);
210 gcpscad       }
211 gcpscad     }
212 gcpscad     cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
213 gcpscad     cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed)
                ;
214 gcpscad     setypos(getypos()+kh_distance);
215 gcpscad   }
216 gcpscad }
```

3.3 Shapes and tool movement

The majority of commands will be more general, focusing on tooling which is generally supported by this library, moving in lines and arcs so as to describe shapes which lend themselves to representation with those tool and which match up with both toolpaths and supported geometry in Carbide Create, and the usage requirements of the typical user.

3.3.1 Generalized commands and cuts

The first consideration is a naming convention which will allow a generalized set of associated commands to be defined. The initial version will only create OpenSCAD commands for 3D modeling and write out matching DXF files. At a later time this will be extended with G-code support.

begincutdxf **3.3.1.1 begincutdxf** The first command, `begincutdxf` will need to allow the machine to rapid to the beginning point of the cut and then rapid down to the surface of the stock, and then plunge down to the depth of the cut. The implementation will need to allow for a hook where the Depth per Pass is applied to the plunge operation so that multiple passes are made.

The first module will ensure that the tool is safely up above the stock and will rapid to the position specified at the retract height (moving to that position as an initial step, then will `cutwithfeed` to the specified position at the specified feed rate. Despite `dxf` being included in the filename no change is made to the dxf file at this time, this simply indicates that this file is preparatory to the
continuecutdxf use of `continuecutdxf`.

```
593 gcpscad module begincutdxf(rh, ex, ey, ez, fr) {
594 gcpscad   rapid(getxpos(),getypos(),rh);
595 gcpscad   cutwithfeed(ex,ey,ez,fr);
596 gcpscad }
```

```
598 gcpscad module continuecutdxf(ex, ey, ez, fr) {
599 gcpscad   cutwithfeed(ex,ey,ez,fr);
600 gcpscad }
```

3.3.1.2 Rectangles Cutting rectangles while writing out their perimeter in the DXF files (so that they may be assigned a matching toolpath in a traditional CAM program upon import) will require the origin coordinates, height and width and depth of the pocket, and the tool # so that the corners may have a radius equal to the tool which is used. Whether a given module is an interior pocket or an outline (interior or exterior) will be determined by the specifics of the module and its usage/positioning, with `outline` being added to those modules which cut perimeter.

A further consideration is that cut orientation as an option should be accounted for if writing out G-code, as well as stepover, and the nature of initial entry (whether ramping in would be implemented, and if so, at what angle). Advanced toolpath strategies such as trochoidal milling could also be implemented.

cutrectangledxf Th routine `cutrectangledxf` cuts the outline of a rectangle creating sharp corners. Note that the initial version would work as a beginning point for vertical cutting if the `hull()` operation was removed and the loop was uncommented:

```
602 gcpscad module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                {//passes
603 gcpscad   movetosafez();
604 gcpscad   hull(){
605 gcpscad     //  for (i = [0 : abs(1) : passes]) {
606 gcpscad     //      rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(
                    current_tool()))/passes,bx+tool_radius(rtn),1);
607 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+tool_radius(rtn),bz-rdepth,feed)
                    ;
608 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
609 gcpscad
610 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                    feed);
611 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                    rdepth,feed);
612 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(
                    rtn),bz-rdepth,feed);
613 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
614 gcpscad   }
615 gcpscad   //dxfarc(xcenter,ycenter,radius,anglebegin,endangle, tn)
616 gcpscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                  ,180,270, rtn);
```

```
617 gcpscad   //dxfpolyline(xbegin,ybegin,xend,yend, tn)
618 gcpscad   dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
                  , rtn);
619 gcpscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),90,180, rtn);
620 gcpscad   dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
                  rtn),by+rheight, rtn);
621 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),0,90, rtn);
622 gcpscad   dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                  tool_radius(rtn), rtn);
623 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                  (rtn),270,360, rtn);
624 gcpscad   dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
                   rtn);
625 gcpscad }
```

cutrectangleoutlinedxf A matching command: `cutrectangleoutlinedxf` cuts the outline of a rounded rectangle and is a simplification of the above:

```
627 gcpscad module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth,
                rtn) {//passes
628 gcpscad   movetosafez();
629 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                  feed);
630 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                  rdepth,feed);
631 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn
                  ),bz-rdepth,feed);
632 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                  rdepth,feed);
633 gcpscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                  ,180,270, rtn);
634 gcpscad   dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
                  , rtn);
635 gcpscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),90,180, rtn);
636 gcpscad   dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
                  rtn),by+rheight, rtn);
637 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),0,90, rtn);
638 gcpscad   dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                  tool_radius(rtn), rtn);
639 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                  (rtn),270,360, rtn);
640 gcpscad   dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
                   rtn);
641 gcpscad }
```

rectangleoutlinedxf Which suggests a further command, `rectangleoutlinedxf` for simply adding a rectangle (a potential use of which would be in Job Setup to add the stock outline to DXFs to assist in registration of jobs with multiple tools):

```
643 gcpscad module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
644 gcpscad   dxfpolyline(bx,by,bx,by+rheight, rtn);
645 gcpscad   dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
646 gcpscad   dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
647 gcpscad   dxfpolyline(bx+rwidth,by,bx,by, rtn);
648 gcpscad }
```

the initial section performs the cutting operation for the 3D preview while the latter section writes out the outline to the DXF files.

cutoutrectangledxf A variant of the cutting version of that file, `cutoutrectangledxf` will cut to the outside:

```
650 gcpscad module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                {
651 gcpscad   movetosafez();
652 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                  feed);
653 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-
                  rdepth,feed);
654 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn
                  ),bz-rdepth,feed);
655 gcpscad   cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-
                  rdepth,feed);
656 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
```

```
                feed);
gcpscad    dxfpolyline(bx,by,bx,by+rheight, rtn);
gcpscad    dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
gcpscad    dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
gcpscad    dxfpolyline(bx+rwidth,by,bx,by, rtn);
gcpscad }
```

4 Future

4.1 Images

Would it be helpful to re-create code algorithms/sections using OpenSCAD Graph Editor so as to represent/illustrate the program?

4.2 Import G-code

Use a tool to read in a G-code file, then create a 3D model which would serve as a preview of the cut?

- https://stackoverflow.com/questions/34638372/simple-python-program-to-read-gcode-file
- https://pypi.org/project/gcodeparser/
- https://github.com/fragmuffin/pygcode/wiki

4.3 Bézier curves in 2 dimensions

Take a Bézier curve definition and approximate it as arcs and write them into a DXF?
https://pomax.github.io/bezierinfo/
c.f., https://linuxcnc.org/docs/html/gcode/g-code.html#gcode:g5

4.4 Bézier curves in 3 dimensions

One question is how many Bézier curves would it be necessary to have to define a surface in 3 dimensions. Attributes for this which are desirable/necessary:

- concise — a given Bézier curve should be represented by just the point coordinates, so two on-curve points, two off-curve points, each with a pair of coordinates
- For a given shape/region it will need to be possible to have a matching definition exactly match up with it so that one could piece together a larger more complex shape from smaller/simpler regions
- similarly it will be necessary for it to be possible to sub-divide a defined region — for example it should be possible if one had 4 adjacent regions, then the four quadrants at the intersection of the four regions could be used to construct a new region — is it possible to derive a new Bézier curve from half of two other curves?

For the three planes:

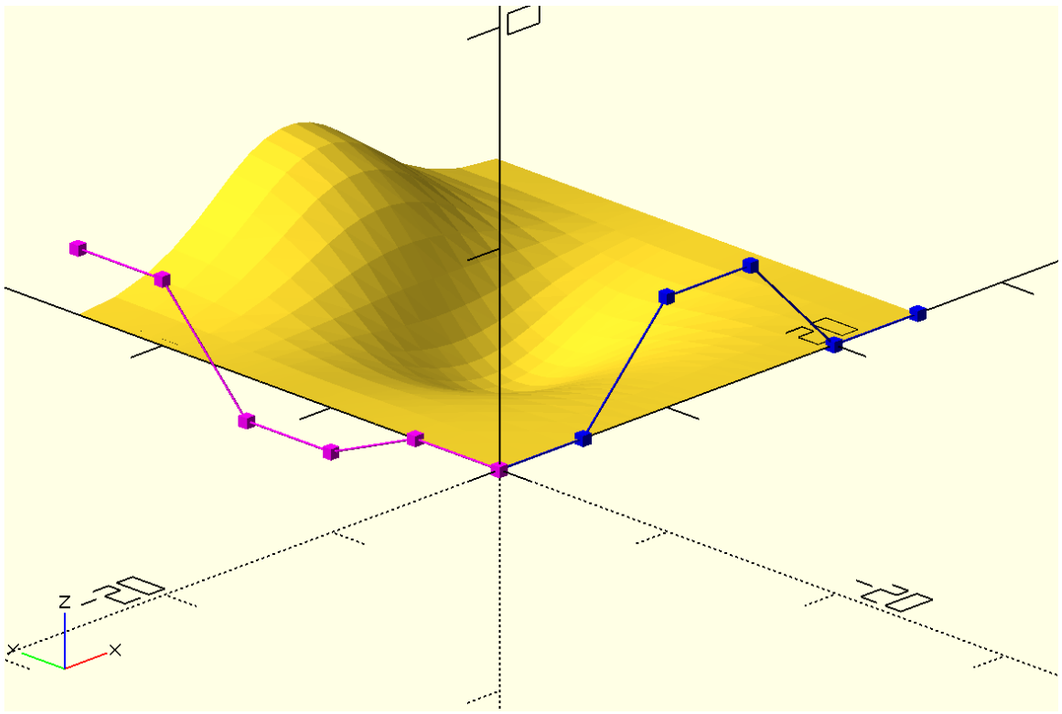
- XY
- XZ
- ZY

it should be possible to have three Bézier curves (left-most/right-most or front-back or top/bottom for two, and a mid-line for the third), so a region which can be so represented would be definable by:

```
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs
```

which is a marked contrast to representations such as:
https://github.com/DavidPhillipOster/Teapot
and regions which could not be so represented could be sub-divided until the representation is workable.

Or, it may be that fewer (only two?) curves are needed:



https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html
c.f., https://github.com/BelfrySCAD/BOSL2/wiki/nurbs.scad and https://old.reddit.com/r/OpenPythonSCAD/comments/1gjcz4z/pythonscad_will_get_a_new_spline_function/

5 Other Resources

Holidays are from https://nationaltoday.com/

5.1 DXFs

http://www.paulbourke.net/dataformats/dxf/
https://paulbourke.net/dataformats/dxf/min3d.html

References

[ConstGeom] Walmsley, Brian. *Construction Geometry*. 2d ed., Centennial College Press, 1981.

[MkCalc] Horvath, Joan, and Rich Cameron. *Make: Calculus: Build models to learn, visualize, and explore*. First edition., Make: Community LLC, 2022.

[MkGeom] Horvath, Joan, and Rich Cameron. *Make: Geometry: Learn by 3D Printing, Coding and Exploring*. First edition., Make: Community LLC, 2021.

[MkTrig] Horvath, Joan, and Rich Cameron. *Make: Trigonometry: Build your way from triangles to analytic geometry*. First edition., Make: Community LLC, 2023.

[PractShopMath] Begnal, Tom. *Practical Shop Math: Simple Solutions to Workshop Fractions, Formulas + Geometric Shapes*. Updated edition, Spring House Press, 2018.

[RS274] Thomas R. Kramer, Frederick M. Proctor, Elena R. Messina.
https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374
https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3

[SoftwareDesign] Ousterhout, John K. *A Philosophy of Software Design*. First Edition., Yaknyam Press, Palo Alto, Ca., 2018

Index

Routines

Variables