The gcodepreview OpenSCAD library*

Author: William F. Adams
willadams at aol dot com

2024/11/29

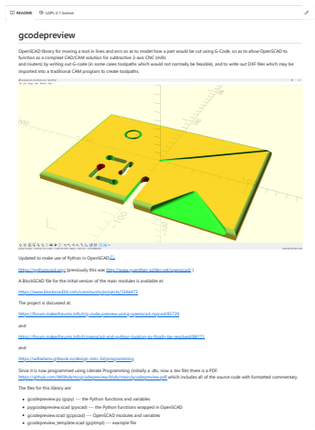
Abstract

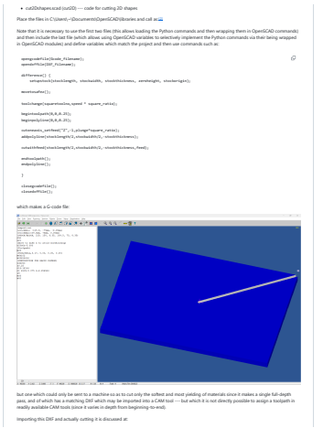
The gcodepreview library allows using OpenPythonSCAD to move a tool in lines and arcs and output dxf and G-code files so as to work as a CAD/CAM program for CNC.

Contents

*This file (gcodepreview) has version number v0.71, last revised 2024/11/29.

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 may
      become 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)
 95 rdme
 96 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.
 97 rdme
 98 rdme Added features since initial upload:
 99 rdme
100 rdme  - endpolyline(); --- this command allows ending one polyline so as
             to allow multiple lines in a DXF
101 rdme  - separate dxf files are written out for each tool where tool is
             ball/square/V and small/large (10/31/23)
102 rdme  - re-writing as a Literate Program using the LaTeX package docmfp
             (begun 4/12/24)
103 rdme  - support for additional tooling shapes such as dovetail and
             keyhole tools
104 rdme
105 rdme Version 0.2 adds support for arcs
106 rdme
107 rdme  - DXF: support for arcs (which may be used to make circles)
             (6/1/24)
108 rdme  - Specialty toolpaths such as Keyhole which may be used for
             dovetail as well as keyhole cutters
109 rdme
110 rdme Version 0.3
111 rdme
112 rdme  - Support for curves along the 3rd dimension
113 rdme  - support for roundover tooling
114 rdme
115 rdme Version 0.4
116 rdme
117 rdme  - Rewrite using literati documentclass, suppression of SVG code
118 rdme  - dxfrectangle (without G-code support)
119 rdme
120 rdme Version 0.5
121 rdme
122 rdme  - more shapes
123 rdme  - consolidate rectangles, arcs, and circles in gcodepreview.scad
124 rdme
125 rdme Version 0.6
126 rdme
127 rdme  - notes on modules
128 rdme  - change file for setupstock
129 rdme
130 rdme Version 0.61
131 rdme
132 rdme  - validate all code so that it runs without errors from sample
133 rdme  - NEW: Note that this version is archived as gcodepreview-
             openscad_0_6.tex and the matching PDF is available as well
134 rdme
135 rdme  Version 0.7
136 rdme
137 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
138 rdme
139 rdme Possible future improvements:
140 rdme
141 rdme  - rewrite OpenSCAD wrapper
142 rdme  - support for additional tooling shapes (bowl bits with flat
             bottom, tapered ball nose, lollipop cutters)
143 rdme  - create a single line font for use where text is wanted
144 rdme
145 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.
146 rdme
147 rdme Deprecated feature:
148 rdme
149 rdme  - exporting SVGs --- coordinate system differences between
             OpenSCAD/DXFs and SVGs would require managing the inversion of
             the coordinate system (using METAPOST, which shares the same
             orientation and which can write out SVGs may be used 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.

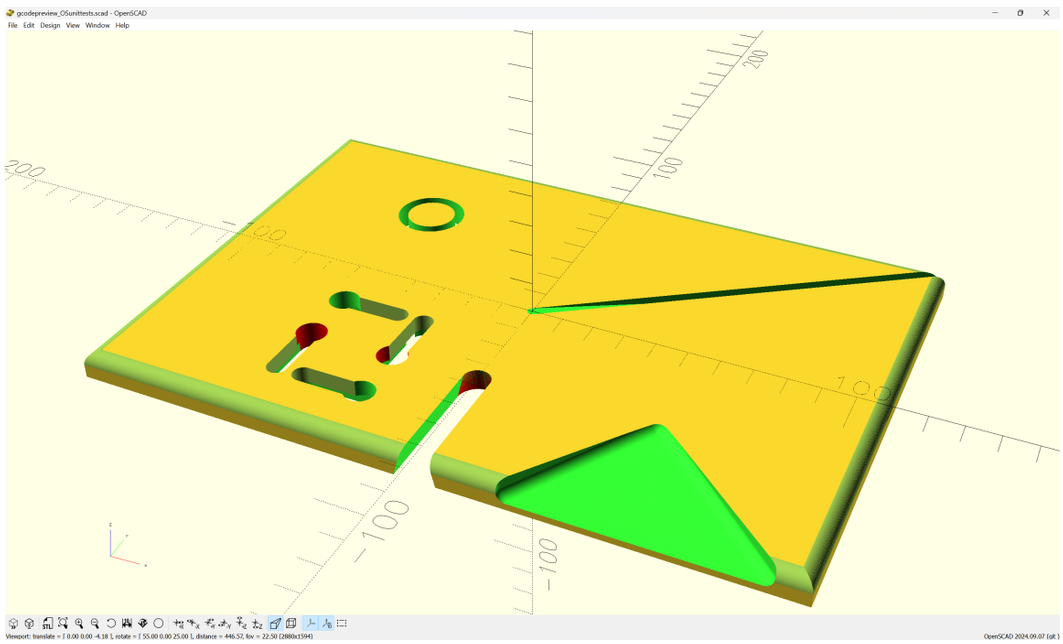
2.1.1 gcodepreviewtemplate.scad

```
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; // [56142:56142, 56125:56125, 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 dxfline(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);
```

```
gcptmpl cutarcSWCCdxf(-stockXwidth/4, stockYheight/4-stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight/16, small_square_tool_num);
gcptmpl cutarcSECCdxf(-(stockXwidth/4-stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight/16, small_square_tool_num);
gcptmpl
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl toolchange(KH_tool_num,speed * KH_ratio);
gcptmpl rapid(-stockXwidth/8,-stockYheight/4,0);
gcptmpl
gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "N", stockYheight/8, KH_tool_num);
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl rapid(-stockXwidth/4,-stockYheight/4,0);
gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "S", stockYheight/8, KH_tool_num);
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl rapid(-stockXwidth/4,-stockYheight/8,0);
gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "E", stockYheight/8, KH_tool_num);
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl rapid(-stockXwidth/8,-stockYheight/8*3,0);
gcptmpl cutkeyhole_toolpath((stockZthickness), (stockZthickness), "W", stockYheight/8, KH_tool_num);
gcptmpl
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl toolchange(DT_tool_num,speed * DT_ratio);
gcptmpl rapid(0,-(stockYheight/2+tool_diameter(DT_tool_num,0)),0);
gcptmpl
gcptmpl cutoneaxis_setfeed("Z",-stockZthickness,plunge*DT_ratio);
gcptmpl cutwithfeed(0,-(stockYheight/4),-stockZthickness,feed*DT_ratio);
gcptmpl rapid(0,-(stockYheight/2+tool_diameter(DT_tool_num,0)),-stockZthickness);
gcptmpl
gcptmpl rapid(getxpos(),getypos(),stockZthickness);
gcptmpl toolchange(Roundover_tool_num, speed * RO_ratio);
gcptmpl rapid(-(stockXwidth/2),-(stockYheight/2),0);
gcptmpl cutoneaxis_setfeed("Z",-4.509,plunge*RO_ratio);
gcptmpl
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);
gcptmpl
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);
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);
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);
gcptmpl
gcptmpl //for (i = [0 : abs(1) : 80]) {
gcptmpl //  cutwithfeed(stockXwidth/4,-stockYheight/4,-stockZthickness/4, feed);
gcptmpl //  cutwithfeed(stockXwidth/8+(stockXwidth/256*i),-stockYheight/2,-stockZthickness*3/4,feed);
gcptmpl //  }
gcptmpl
gcptmpl hull(){
gcptmpl   cutwithfeed(stockXwidth/4,-stockYheight/4,-stockZthickness/4,feed);
gcptmpl   cutwithfeed(stockXwidth/8,-stockYheight/2,-stockZthickness*3/4, feed);
gcptmpl   cutwithfeed(stockXwidth/8+(stockXwidth*0.3125),-stockYheight/2,-stockZthickness*3/4,feed);
gcptmpl   }
gcptmpl }
gcptmpl
gcptmpl closegcodefile();
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 v0.7 re-write, it is possible to directly use the underlying Python code directly.

```
gcptmplpy #!/usr/bin/env python
gcptmplpy
gcptmplpy import sys
gcptmplpy
gcptmplpy try:
gcptmplpy     if 'gcodepreview' in sys.modules:
gcptmplpy         del sys.modules['gcodepreview']
gcptmplpy except AttributeError:
gcptmplpy     pass
gcptmplpy
gcptmplpy from gcodepreview import *
gcptmplpy
gcptmplpy fa = 2
gcptmplpy fs = 0.125
gcptmplpy
gcptmplpy # [Export] */
gcptmplpy Base_filename = "aexport"
gcptmplpy # [Export] */
gcptmplpy generatedxf = True
gcptmplpy # [Export] */
gcptmplpy generategcode = True
gcptmplpy
gcptmplpy # [Stock] */
gcptmplpy stockXwidth = 220
gcptmplpy # [Stock] */
gcptmplpy stockYheight = 150
gcptmplpy # [Stock] */
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 = 201 # [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 = 202 # [0:0,111:111,101:101,202:202]
gcptmplpy # [CAM] */
gcptmplpy small_ball_tool_num = 101 # [0:0,121:121,111:111,101:101]
gcptmplpy # [CAM] */
gcptmplpy large_V_tool_num = 301 # [0:0,301:301,690:690]
gcptmplpy # [CAM] */
gcptmplpy small_V_tool_num = 390 # [0:0,390:390,301:301]
gcptmplpy # [CAM] */
gcptmplpy DT_tool_num = 814 # [0:0,814:814]
gcptmplpy # [CAM] */
gcptmplpy KH_tool_num = 374 # [0:0,374:374,375:375,376:376,378]
gcptmplpy # [CAM] */
gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 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(True, #generatescad
gcptmplpy                    True, #generategcode
gcptmplpy                    True, #generatedxf
gcptmplpy                    )
gcptmplpy
gcptmplpy gcp.opengcodefile(Base_filename)
gcptmplpy gcp.opendxffile(Base_filename)
gcptmplpy gcp.opendxffiles(Base_filename,
gcptmplpy                  large_square_tool_num,
gcptmplpy                  small_square_tool_num,
gcptmplpy                  large_ball_tool_num,
gcptmplpy                  small_ball_tool_num,
gcptmplpy                  large_V_tool_num,
gcptmplpy                  small_V_tool_num,
gcptmplpy                  DT_tool_num,
gcptmplpy                  KH_tool_num,
gcptmplpy                  Roundover_tool_num,
gcptmplpy                  MISC_tool_num)
gcptmplpy
gcptmplpy gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top","
              Center",retractheight)
gcptmplpy
gcptmplpy gcp.movetosafeZ()
gcptmplpy
gcptmplpy gcp.toolchange(102,10000)
gcptmplpy
gcptmplpy #gcp.rapidXY(6,12)
```

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

```
                stockYheight/8, stockYheight/8-stockYheight/16, -stockZthickness
                , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
                )
gcptmplpy toolpaths = toolpaths.union(gcp.cutarcSECCdxfgc(stockXwidth/4+
                stockYheight/8+stockYheight/16, stockYheight/8, -stockZthickness
                , stockXwidth/4+stockYheight/8, stockYheight/8, stockYheight/16)
                )
gcptmplpy
gcptmplpy #a = gcp.currenttool()
gcptmplpy #arcbegin = a.translate([64.37357214209116, -37.33638368965047,-
                stockZthickness])
gcptmplpy #arcend = a.translate([55.16361631034953, -28.12642785790883,-
                stockZthickness])
gcptmplpy #toolpaths = toolpaths.union(arcbegin)
gcptmplpy #toolpaths = toolpaths.union(arcend)
gcptmplpy
gcptmplpy #cu = cube([10,20,30])
gcptmplpy #c = cu.translate([0,0,gcp.zpos()])
gcptmplpy
gcptmplpy #def cutroundovertool(bx, by, bz, ex, ey, ez, tool_radius_tip,
                tool_radius_width):
gcptmplpy #    n = 90 + fn*3
gcptmplpy #    step = 360/n
gcptmplpy #    shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
gcptmplpy #    toolpath = hull(shaft.translate([bx,by,bz]), shaft.translate([
                ex,ey,ez]))
gcptmplpy #    shaft = cylinder(tool_radius_width*2,tool_radius_tip+
                tool_radius_width,tool_radius_tip+tool_radius_width)
gcptmplpy #    toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
                tool_radius_width]), shaft.translate([ex,ey,ez+tool_radius_width
                ])))
gcptmplpy #    for i in range(1, 90, 1):
gcptmplpy #        angle = i
gcptmplpy #        dx = tool_radius_width*math.cos(math.radians(angle))
gcptmplpy #        dxx = tool_radius_width*math.cos(math.radians(angle+1))
gcptmplpy #        dzz = tool_radius_width*math.sin(math.radians(angle))
gcptmplpy #        dz = tool_radius_width*math.sin(math.radians(angle+1))
gcptmplpy #        dh = abs(dzz-dz)+0.0001
gcptmplpy #        slice = cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                tool_radius_tip+tool_radius_width-dxx)
gcptmplpy #        toolpath = toolpath.union(hull(slice.translate([bx,by,bz+
                dz]), slice.translate([ex,ey,ez+dz])))
gcptmplpy #    return toolpath
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.toolchange(814,10000)
gcptmplpy gcp.rapidXY(0, -(stockYheight/2+12.7))
gcptmplpy gcp.cutZgcfeed(-stockZthickness,plunge)
gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgcfeed(0, -(stockYheight
                /16), -stockZthickness, feed))
gcptmplpy
gcptmplpy
gcptmplpy gcp.rapidZ(0)
gcptmplpy
gcptmplpy #print(gcp.currenttoolnumber())
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.toolchange(56142,10000)
gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
gcptmplpy gcp.cutZgcfeed(-1.531,plunge)
gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgcfeed(stockXwidth
                /2+0.508/2, -(stockYheight/2+0.508/2), -1.531, feed))
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy #gcp.toolchange(56125,10000)
gcptmplpy gcp.cutZgcfeed(-1.531,plunge)
gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgcfeed(stockXwidth
                /2+0.508/2, (stockYheight/2+0.508/2), -1.531, feed))
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.toolchange(374,10000)
gcptmplpy gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight/4+
                stockYheight/16))
gcptmplpy gcp.rapidZ(0)
gcptmplpy #toolpaths = toolpaths.union(gcp.cutlinedxfgcfeed(gcp.xpos(), gcp.
                ypos(), -4, feed))
gcptmplpy #toolpaths = toolpaths.union(gcp.cutZgcfeed(-4,plunge))
gcptmplpy #toolpaths = toolpaths.union(gcp.cutlinedxfgcfeed(stockXwidth/4, -(
```

```
            stockYheight/4)+25.4, -4, feed))
gcptmplpy #key = gcp.cutlinedxfgcfeed(stockXwidth/2+0.508/2, (stockYheight
            /2+0.508/2), -1.531, feed)
gcptmplpy
gcptmplpy #cutkeyholegcdxf(stockZthickness/2, stockZthickness/2, "N",
            stockYheight/8, KH_tool_num)
gcptmplpy #rapid(getxpos(),getypos(),stockZthickness);
gcptmplpy #rapid(-stockXwidth/4,-stockYheight/4,0);
gcptmplpy #cutkeyhole_toolpath((stockZthickness), (stockZthickness), "S",
            stockYheight/8, KH_tool_num);
gcptmplpy #rapid(getxpos(),getypos(),stockZthickness);
gcptmplpy #rapid(-stockXwidth/4,-stockYheight/8,0);
gcptmplpy key = gcp.cutkeyholegcdxf(0, stockZthickness*0.75, "E",
            stockYheight/9, KH_tool_num)
gcptmplpy toolpaths = toolpaths.union(key)
gcptmplpy #rapid(getxpos(),getypos(),stockZthickness);
gcptmplpy #rapid(-stockXwidth/8,-stockYheight/8*3,0);
gcptmplpy #cutkeyhole_toolpath((stockZthickness), (stockZthickness), "W",
            stockYheight/8, KH_tool_num);
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.rapidXY(stockXwidth/4+stockXwidth/16, -(stockYheight/4+
            stockYheight/16))
gcptmplpy gcp.rapidZ(0)
gcptmplpy toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(0, stockZthickness
            *0.75, "N", stockYheight/9, KH_tool_num))
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.rapidXY(stockXwidth/4+stockXwidth/16, -(stockYheight/4-
            stockYheight/8))
gcptmplpy gcp.rapidZ(0)
gcptmplpy toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(0, stockZthickness
            *0.75, "W", stockYheight/9, KH_tool_num))
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight/4-
            stockYheight/8))
gcptmplpy gcp.rapidZ(0)
gcptmplpy toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(0, stockZthickness
            *0.75, "S", stockYheight/9, KH_tool_num))
gcptmplpy
gcptmplpy gcp.rapidZ(retractheight)
gcptmplpy
gcptmplpy #Last dxf command not being written...
gcptmplpy #empty = gcp.cutlinedxfgcfeed(stockXwidth/2, -(stockYheight
            /2+0.508/2), 1, feed)
gcptmplpy
gcptmplpy part = gcp.stock.difference(toolpaths)
gcptmplpy #part = gcp.stock.union(key)
gcptmplpy
gcptmplpy output(part)
gcptmplpy #output(toolpaths)
gcptmplpy #output(key)
gcptmplpy
gcptmplpy gcp.setzpos(retractheight)
gcptmplpy
gcptmplpy gcp.closegcodefile()
gcptmplpy gcp.closedxffiles()
gcptmplpy gcp.closedxffile()
```

2.1.3 gcpdxf.py

It is also possible to use "plain" Python to create `dxf` files.

```
gcpdxfpy from gcodepreview import *
gcpdxfpy
gcpdxfpy gcp = gcodepreview(False, #generatescad
gcpdxfpy                    False, #generategcode
gcpdxfpy                    True #generatedxf
gcpdxfpy                    )
gcpdxfpy
gcpdxfpy Base_filename = "export"
gcpdxfpy large_square_tool_num = 102
gcpdxfpy small_square_tool_num = 0
gcpdxfpy large_ball_tool_num = 0
gcpdxfpy small_ball_tool_num = 0
gcpdxfpy large_V_tool_num = 0
```

```
gcpdxfpy small_V_tool_num = 0
gcpdxfpy DT_tool_num = 0
gcpdxfpy KH_tool_num = 0
gcpdxfpy Roundover_tool_num = 0
gcpdxfpy MISC_tool_num = 0
gcpdxfpy
gcpdxfpy gcp.opendxffile(Base_filename)
gcpdxfpy gcp.opendxffiles(Base_filename,
gcpdxfpy                  large_square_tool_num,
gcpdxfpy                  small_square_tool_num,
gcpdxfpy                  large_ball_tool_num,
gcpdxfpy                  small_ball_tool_num,
gcpdxfpy                  large_V_tool_num,
gcpdxfpy                  small_V_tool_num,
gcpdxfpy                  DT_tool_num,
gcpdxfpy                  KH_tool_num,
gcpdxfpy                  Roundover_tool_num,
gcpdxfpy                  MISC_tool_num)
gcpdxfpy
gcpdxfpy gcp.dxfarc(large_square_tool_num, 88, 38, 12,   0,  90)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 12, 38, 12,  90, 180)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 12, 12, 12, 180, 270)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 88, 12, 12, 270, 360)
gcpdxfpy
gcpdxfpy gcp.dxfline(large_square_tool_num, 12, 0, 88, 0)
gcpdxfpy gcp.dxfline(large_square_tool_num, 100, 12, 100, 38)
gcpdxfpy gcp.dxfline(large_square_tool_num, 88, 50, 12, 50)
gcpdxfpy gcp.dxfline(large_square_tool_num, 0, 38, 0, 12)
gcpdxfpy
gcpdxfpy gcp.dxfarc(large_square_tool_num, 50, 25, 12,   0,  90)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 50, 25, 12,  90, 180)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 50, 25, 12, 180, 270)
gcpdxfpy gcp.dxfarc(large_square_tool_num, 50, 25, 12, 270, 360)
gcpdxfpy
gcpdxfpy gcp.closedxffiles()
gcpdxfpy gcp.closedxffile()
```

2.2 Implementation files and gcodepreview class

Each file will begin with a comment indicating the file type and further notes/comments on usage where appropriate:

```
gcpy #!/usr/bin/env python
gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe"  --trust-
       python
gcpy #Currently tested with 2024.09.23 and Python 3.11
gcpy #gcodepreview 0.7, for use with OpenPythonSCAD,
gcpy #if using from OpenPythonSCAD see gcodepreview.scad
gcpy
gcpy import sys
gcpy
gcpy # getting openscad functions into namespace
gcpy #https://github.com/gsohler/openscad/issues/39
gcpy try:
gcpy     from openscad import *
gcpy except ModuleNotFoundError as e:
gcpy     print("OpenSCAD module not loaded.")
gcpy
gcpy # add math functions (using radians by default, convert to degrees
       where necessary)
gcpy import math
gcpy
gcpy def gcpversion():
gcpy     return 0.71
```

```
pyscad //!OpenSCAD
pyscad
pyscad //gcodepreview 0.7, see gcodepreview.scad
```

```
gcpscad //!OpenSCAD
gcpscad
gcpscad //gcodepreview 0.7
gcpscad //
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 is 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 utilise needed variables and will contain appropriate functionality. 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 the `init` method which allows passing in and defining the variables which will be used by the other methods in this class.

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

```
43 gcpy #         self.stockzero = stockzero
44 gcpy #         self.retractheight = retractheight
45 gcpy #         self.currenttoolnum = currenttoolnum
46 gcpy #         self.toolradius = toolradius
47 gcpy #         self.plunge = plunge
48 gcpy #         self.feed = feed
49 gcpy #         self.speed = speed
50 gcpy #         global toolpaths
51 gcpy #         self.toolpaths = cylinder(1.5875, 12.7)
52 gcpy #         global generatedxfs
53 gcpy #         if (self.generatescad == True):
54 gcpy          self.generatedxfs = False
```

2.2.1 Output files

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

2.2.1.1 G-code and modules and commands The G-code commands and their matching modules may include (but are not limited to):

Command/Module	G-code
opengcodefile(s)(...); 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(s)(...); 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

```
52 gcpy     def opengcodefile(self, basefilename = "export",
53 gcpy                       currenttoolnum = 102,
54 gcpy                       toolradius = 3.175,
55 gcpy                       plunge = 400,
56 gcpy                       feed = 1600,
57 gcpy                       speed = 10000
58 gcpy                       ):
59 gcpy         self.currenttoolnum = currenttoolnum
60 gcpy         self.toolradius = toolradius
61 gcpy         self.plunge = plunge
62 gcpy         self.feed = feed
63 gcpy         self.speed = speed
64 gcpy         if self.generategcode == True:
65 gcpy             self.gcodefilename = basefilename + ".nc"
66 gcpy             self.gc = open(self.gcodefilename, "w")
67 gcpy
68 gcpy     def opendxffile(self, basefilename = "export"):
69 gcpy         self.basefilename = basefilename
70 gcpy #        global generatedxfs
71 gcpy #        global dxfclosed
72 gcpy         self.dxfclosed = False
73 gcpy         if self.generatedxf == True:
74 gcpy             self.generatedxfs = False
75 gcpy             self.dxffilename = basefilename + ".dxf"
76 gcpy             self.dxf = open(self.dxffilename, "w")
77 gcpy             self.dxfpreamble(-1)
78 gcpy
79 gcpy     def opendxffiles(self, basefilename = "export",
80 gcpy                      large_square_tool_num = 0,
81 gcpy                      small_square_tool_num = 0,
82 gcpy                      large_ball_tool_num = 0,
83 gcpy                      small_ball_tool_num = 0,
84 gcpy                      large_V_tool_num = 0,
85 gcpy                      small_V_tool_num = 0,
86 gcpy                      DT_tool_num = 0,
87 gcpy                      KH_tool_num = 0,
88 gcpy                      Roundover_tool_num = 0,
89 gcpy                      MISC_tool_num = 0):
90 gcpy #        global generatedxfs
91 gcpy         self.basefilename = basefilename
92 gcpy         self.generatedxfs = True
93 gcpy         self.large_square_tool_num = large_square_tool_num
94 gcpy         self.small_square_tool_num = small_square_tool_num
95 gcpy         self.large_ball_tool_num = large_ball_tool_num
96 gcpy         self.small_ball_tool_num = small_ball_tool_num
97 gcpy         self.large_V_tool_num = large_V_tool_num
98 gcpy         self.small_V_tool_num = small_V_tool_num
99 gcpy         self.DT_tool_num = DT_tool_num
100 gcpy        self.KH_tool_num = KH_tool_num
101 gcpy        self.Roundover_tool_num = Roundover_tool_num
102 gcpy        self.MISC_tool_num = MISC_tool_num
103 gcpy        if self.generatedxf == True:
104 gcpy            if (large_square_tool_num > 0):
105 gcpy                self.dxflgsqfilename = basefilename + str(
                        large_square_tool_num) + ".dxf"
106 gcpy #                print("Opening ", str(self.dxflgsqfilename))
107 gcpy                self.dxflgsq = open(self.dxflgsqfilename, "w")
108 gcpy            if (small_square_tool_num > 0):
109 gcpy #                print("Opening small square")
110 gcpy                self.dxfsmsqfilename = basefilename + str(
                        small_square_tool_num) + ".dxf"
111 gcpy                self.dxfsmsq = open(self.dxfsmsqfilename, "w")
112 gcpy            if (large_ball_tool_num > 0):
113 gcpy #                print("Opening large ball")
114 gcpy                self.dxflgblfilename = basefilename + str(
                        large_ball_tool_num) + ".dxf"
115 gcpy                self.dxflgbl = open(self.dxflgblfilename, "w")
116 gcpy            if (small_ball_tool_num > 0):
117 gcpy #                print("Opening small ball")
118 gcpy                self.dxfsmblfilename = basefilename + str(
                        small_ball_tool_num) + ".dxf"
119 gcpy                self.dxfsmbl = open(self.dxfsmblfilename, "w")
120 gcpy            if (large_V_tool_num > 0):
121 gcpy #                print("Opening large V")
122 gcpy                self.dxflgVfilename = basefilename + str(
                        large_V_tool_num) + ".dxf"
```

```
123 gcpy                 self.dxflgV = open(self.dxflgVfilename, "w")
124 gcpy             if (small_V_tool_num > 0):
125 gcpy #                print("Opening small V")
126 gcpy                 self.dxfsmVfilename = basefilename + str(
                            small_V_tool_num) + ".dxf"
127 gcpy                 self.dxfsmV = open(self.dxfsmVfilename, "w")
128 gcpy             if (DT_tool_num > 0):
129 gcpy #                print("Opening DT")
130 gcpy                 self.dxfDTfilename = basefilename + str(DT_tool_num
                            ) + ".dxf"
131 gcpy                 self.dxfDT = open(self.dxfDTfilename, "w")
132 gcpy             if (KH_tool_num > 0):
133 gcpy #                print("Opening KH")
134 gcpy                 self.dxfKHfilename = basefilename + str(KH_tool_num
                            ) + ".dxf"
135 gcpy                 self.dxfKH = open(self.dxfKHfilename, "w")
136 gcpy             if (Roundover_tool_num > 0):
137 gcpy #                print("Opening Rt")
138 gcpy                 self.dxfRtfilename = basefilename + str(
                            Roundover_tool_num) + ".dxf"
139 gcpy                 self.dxfRt = open(self.dxfRtfilename, "w")
140 gcpy             if (MISC_tool_num > 0):
141 gcpy #                print("Opening Mt")
142 gcpy                 self.dxfMtfilename = basefilename + str(
                            MISC_tool_num) + ".dxf"
143 gcpy                 self.dxfMt = open(self.dxfMtfilename, "w")
```

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

```
131 gcpy             if (large_square_tool_num > 0):
132 gcpy                 self.dxfpreamble(large_square_tool_num)
133 gcpy             if (small_square_tool_num > 0):
134 gcpy                 self.dxfpreamble(small_square_tool_num)
135 gcpy             if (large_ball_tool_num > 0):
136 gcpy                 self.dxfpreamble(large_ball_tool_num)
137 gcpy             if (small_ball_tool_num > 0):
138 gcpy                 self.dxfpreamble(small_ball_tool_num)
139 gcpy             if (large_V_tool_num > 0):
140 gcpy                 self.dxfpreamble(large_V_tool_num)
141 gcpy             if (small_V_tool_num > 0):
142 gcpy                 self.dxfpreamble(small_V_tool_num)
143 gcpy             if (DT_tool_num > 0):
144 gcpy                 self.dxfpreamble(DT_tool_num)
145 gcpy             if (KH_tool_num > 0):
146 gcpy                 self.dxfpreamble(KH_tool_num)
147 gcpy             if (Roundover_tool_num > 0):
148 gcpy                 self.dxfpreamble(Roundover_tool_num)
149 gcpy             if (MISC_tool_num > 0):
150 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.

```
152 gcpy     def writegc(self, *arguments):
153 gcpy         line_to_write = ""
154 gcpy         for element in arguments:
155 gcpy             line_to_write += element
156 gcpy         self.gc.write(line_to_write)
157 gcpy         self.gc.write("\n")
158 gcpy
159 gcpy     def writedxf(self, toolnumber, *arguments):
160 gcpy #         global dxfclosed
161 gcpy         line_to_write = ""
162 gcpy         for element in arguments:
163 gcpy             line_to_write += element
164 gcpy         if self.generatedxf == True:
165 gcpy             if self.dxfclosed == False:
166 gcpy                 self.dxf.write(line_to_write)
167 gcpy                 self.dxf.write("\n")
168 gcpy         if self.generatedxfs == True:
169 gcpy             self.writedxfs(toolnumber, line_to_write)
170 gcpy
171 gcpy     def writedxfs(self, toolnumber, line_to_write):
```

```
172 gcpy #         print("Processing writing toolnumber", toolnumber)
173 gcpy #         line_to_write = ""
174 gcpy #         for element in arguments:
175 gcpy #             line_to_write += element
176 gcpy          if (toolnumber == 0):
177 gcpy              return
178 gcpy          elif self.generatedxfs == True:
179 gcpy              if (self.large_square_tool_num == toolnumber):
180 gcpy                  self.dxflgsq.write(line_to_write)
181 gcpy                  self.dxflgsq.write("\n")
182 gcpy              if (self.small_square_tool_num == toolnumber):
183 gcpy                  self.dxfsmsq.write(line_to_write)
184 gcpy                  self.dxfsmsq.write("\n")
185 gcpy              if (self.large_ball_tool_num == toolnumber):
186 gcpy                  self.dxflgbl.write(line_to_write)
187 gcpy                  self.dxflgbl.write("\n")
188 gcpy              if (self.small_ball_tool_num == toolnumber):
189 gcpy                  self.dxfsmbl.write(line_to_write)
190 gcpy                  self.dxfsmbl.write("\n")
191 gcpy              if (self.large_V_tool_num == toolnumber):
192 gcpy                  self.dxflgV.write(line_to_write)
193 gcpy                  self.dxflgV.write("\n")
194 gcpy              if (self.small_V_tool_num == toolnumber):
195 gcpy                  self.dxfsmV.write(line_to_write)
196 gcpy                  self.dxfsmV.write("\n")
197 gcpy              if (self.DT_tool_num == toolnumber):
198 gcpy                  self.dxfDT.write(line_to_write)
199 gcpy                  self.dxfDT.write("\n")
200 gcpy              if (self.KH_tool_num == toolnumber):
201 gcpy                  self.dxfKH.write(line_to_write)
202 gcpy                  self.dxfKH.write("\n")
203 gcpy              if (self.Roundover_tool_num == toolnumber):
204 gcpy                  self.dxfRt.write(line_to_write)
205 gcpy                  self.dxfRt.write("\n")
206 gcpy              if (self.MISC_tool_num == toolnumber):
207 gcpy                  self.dxfMt.write(line_to_write)
208 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. Note that the DXF files for specific tools will expect that the tool numbers be set in the matching variables from the template. Further note that while it is possible to use tools which are not so defined, the toolpaths will not be written into DXF files for any tool numbers which do not match the variables from the template (but will appear in the main `.dxf`).

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`, and the short form will be used internally for variable names, while the compleat word will be used in commands.

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 (if necessary), 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
  - `write` (action) — used to write to files
  - `begin` (sequence) — note that sequencing may not be necessary, not having been used in the 0.7 re-write
  - `continue` (sequence)
  - `end` (sequence)
  - `cut` (action — create 3D object)
  - `rapid` (action — create 3D object so as to show a collision)

  - open (action)
  - close (action)
  - set (action/function) — note that the matching get is implicit in functions which return variables, e.g., xpos()
  - current

- Nouns
  - arc
  - line
  - Bézier — a possible future addition, will likely be rendered bezier

- Suffixes
  - feed (parameter)
  - gcode/gc (filetype)
  - pos — position
  - tool
  - number/num — note that num is used internally for variable names, making it straightforward to ensure that functions and variables have different names for purposes of scope

Further note that commands which are implicitly for the generation of G-code, such as toolchange() will omit gc for the sake of conciseness.

In particular, this means that the basic cut... and associated commands exist (or potentially exist) in the following forms and have matching versions which may be used when programming in Python or OpenSCAD:

	line			arc		
	cut	dxf	gcode	cut	dxf	gcode
cut	cutline		cutlinegc	cutarc		cutarcgc
dxf	cutlinedxf	dxfline		cutarcdxf	dxfarc	
gcode		dxflinegc	linegc		dxfarcgc	arcgc
	cutlinedxfgc			cutarcdxfgc		

Note that certain commands (dxflinegc, dxfarcgc, linegc, arcgc) are unlikely to be needed, and may not be implemented. Note that there may be additional versions as required for the convenience of notation or cutting, in particular, a set of cutarc<quadrant><direction>dxf commands was warranted during the initial development of arc-related commands.

OpenPythonSCAD requires that the current toolpath be returned and 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 declared 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, when developing OpenSCAD commands which make use of Python 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.

```
210 gcpy     def setupstock(self, stockXwidth,
211 gcpy                    stockYheight,
212 gcpy                    stockZthickness,
213 gcpy                    zeroheight,
214 gcpy                    stockzero,
215 gcpy                    retractheight):
216 gcpy         self.stockXwidth = stockXwidth
217 gcpy         self.stockYheight = stockYheight
218 gcpy         self.stockZthickness = stockZthickness
219 gcpy         self.zeroheight = zeroheight
220 gcpy         self.stockzero = stockzero
221 gcpy         self.retractheight = retractheight
222 gcpy #        global mpx
223 gcpy         self.mpx = float(0)
224 gcpy #        global mpy
225 gcpy         self.mpy = float(0)
226 gcpy #        global mpz
227 gcpy         self.mpz = float(0)
228 gcpy #        global tpz
229 gcpy         self.tpz = float(0)
230 gcpy #        global currenttoolnum
231 gcpy         self.currenttoolnum = 102
232 gcpy #        global currenttoolshape
233 gcpy         self.currenttoolshape = cylinder(12.7, 1.5875)
234 gcpy #        global stock
235 gcpy         self.stock = cube([stockXwidth, stockYheight,
                     stockZthickness])
236 gcpy         if self.generategcode == True:
237 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 and write out the G-code comment which defines it.

```
232 gcpy         if self.zeroheight == "Top":
233 gcpy             if self.stockzero == "Lower-Left":
234 gcpy                 self.stock = stock.translate([0,0,-self.
                            stockZthickness])
235 gcpy                 if self.generategcode == True:
236 gcpy                     self.writegc("(stockMin:0.00mm,␣0.00mm,␣-",str(
                                self.stockZthickness),"mm)")
237 gcpy                     self.writegc("(stockMax:",str(self.stockXwidth)
                                ,"mm,␣",str(stockYheight),"mm,␣0.00mm)")
```

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

```
272 gcpy                    if self.generategcode == True:
273 gcpy                        self.writegc("(stockMin:0.00mm, -",str(self.
                                stockYheight),"mm, 0.00mm)")
274 gcpy                        self.writegc("(stockMax:",str(self.stockXwidth)
                                ,"mm, 0.00mm, ",str(self.stockZthickness),"
                                mm)")
275 gcpy                        self.writegc("(STOCK/BLOCK, ",str(self.
                                stockXwidth),", ",str(self.stockYheight),", 
                                ",str(self.stockZthickness),", 0.00, ",str(
                                self.stockYheight),", 0.00)")
276 gcpy                if self.stockzero == "Center":
277 gcpy                    self.stock = self.stock.translate([-self.
                            stockXwidth / 2,-self.stockYheight / 2,0])
278 gcpy                    if self.generategcode == True:
279 gcpy                        self.writegc("(stockMin: -",str(self.
                                stockXwidth/2),", -",str(self.stockYheight
                                /2),"mm, 0.00mm)")
280 gcpy                        self.writegc("(stockMax:",str(self.stockXwidth
                                /2),"mm, ",str(self.stockYheight/2),"mm, ",
                                str(self.stockZthickness),"mm)")
281 gcpy                        self.writegc("(STOCK/BLOCK, ",str(self.
                                stockXwidth),", ",str(self.stockYheight),", 
                                ",str(self.stockZthickness),", ",str(self.
                                stockXwidth/2),", ", str(self.stockYheight
                                /2),", 0.00)")
282 gcpy            if self.generategcode == True:
283 gcpy                self.writegc("G90");
284 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 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:

```
286 gcpy      def xpos(self):
287 gcpy #         global mpx
288 gcpy          return self.mpx
289 gcpy
290 gcpy      def ypos(self):
291 gcpy #         global mpy
292 gcpy          return self.mpy
293 gcpy
294 gcpy      def zpos(self):
295 gcpy #         global mpz
296 gcpy          return self.mpz
297 gcpy
298 gcpy      def tzpos(self):
299 gcpy #         global tpz
300 gcpy          return self.tpz
```

psetxpos
psetypos
psetzpos
psettzpos

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

```
302 gcpy      def setxpos(self, newxpos):
303 gcpy #         global mpx
304 gcpy          self.mpx = newxpos
305 gcpy
306 gcpy      def setypos(self, newypos):
307 gcpy #         global mpy
308 gcpy          self.mpy = newypos
309 gcpy
310 gcpy      def setzpos(self, newzpos):
311 gcpy #         global mpz
312 gcpy          self.mpz = newzpos
313 gcpy
314 gcpy      def settzpos(self, newtzpos):
315 gcpy #         global tpz
316 gcpy          self.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:

```
318 gcpy     def settool(self,tn):
319 gcpy #         global currenttoolnum
320 gcpy         self.currenttoolnum = tn
321 gcpy
322 gcpy     def currenttoolnumber(self):
323 gcpy #         global currenttoolnum
324 gcpy         return self.currenttoolnum
325 gcpy
326 gcpy     def currentroundovertoolnumber(self):
327 gcpy #         global Roundover_tool_num
328 gcpy         return self.Roundover_tool_num
```

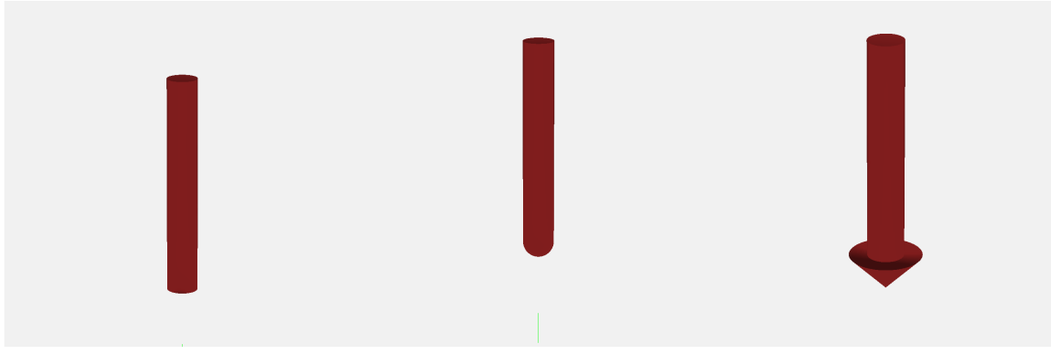
`osettool` `current tool` and matching OpenSCAD modules: `osettool` and `current tool` set and return the current tool:

```
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:

```
330 gcpy     def endmill_square(self, es_diameter, es_flute_length):
331 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:

```
333 gcpy     def gcp_endmill_ball(self, es_diameter, es_flute_length):
334 gcpy         b = sphere(r=(es_diameter / 2))
335 gcpy         s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                     es_flute_length, center=False)
336 gcpy         p = union(b,s)
337 gcpy         return p.translate([0, 0, (es_diameter / 2)])
```

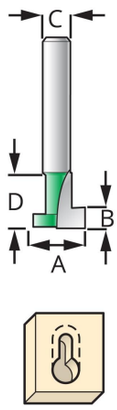
`gcp endmill v` The `gcp endmill v` is modeled as a cylinder with a zero width base and a second cylinder for the shaft (note that Python's `math` defaults to radians, hence the need to convert from degrees):

```
339 gcpy     def gcp_endmill_v(self, es_v_angle, es_diameter):
340 gcpy         es_v_angle = math.radians(es_v_angle)
341 gcpy         v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                     2) / math.tan((es_v_angle / 2))), center=False)
342 gcpy         s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h
                     =((es_diameter * 8) ), center=False)
343 gcpy         sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                     es_v_angle / 2)))])
344 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
- Threadmill — used for cutting threads, normally a single form geometry is used on a CNC.

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 in two parts, first the cutting base:

```
346 gcpy     def gcp_keyhole(self, es_diameter, es_flute_length):
347 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:

```
349 gcpy     def gcp_keyhole_shaft(self, es_diameter, es_flute_length):
350 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)

```
352 gcpy     def gcp_dovetail(self, dt_bottomdiameter, dt_topdiameter,
                 dt_height, dt_angle):
353 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 (or four in the instance of keyhole tools), 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-43723.

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`.

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` (if the toolshape will work with the `hull` command) which may be done in this module, or it will be necessary to check for the specific toolnumber in the `cutline` module and handle the tooling in a separate module as is currently done for roundover tooling.

```
355 gcpy     def currenttool(self):
356 gcpy #         global currenttoolshape
357 gcpy         return self.currenttoolshape
```

Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available:

2.4.2.2 Square and ball nose (including tapered ball nose)

`TOOL/MILL, Diameter, Corner radius, Height, Taper Angle`

2.4.2.3 Roundover (corner rounding)

`TOOL/CRMILL, Diameter1, Diameter2,Radius, Height, Length`

Unfortunately, tools which support undercuts such as dovetails are not supported (CAMotics will work for such tooling).

```
359 gcpy     def toolchange(self,tool_number,speed):
360 gcpy #         global currenttoolshape
361 gcpy         self.currenttoolshape = self.endmill_square(0.001, 0.001)
362 gcpy
363 gcpy         self.settool(tool_number)
364 gcpy         if (self.generategcode == True):
365 gcpy             self.writegc("(Toolpath)")
366 gcpy             self.writegc("M05")
367 gcpy         if (tool_number == 201):
368 gcpy             self.writegc("(TOOL/MILL,6.35,␣0.00,␣0.00,␣0.00)")
369 gcpy             self.currenttoolshape = self.endmill_square(6.35,
                         19.05)
370 gcpy         elif (tool_number == 102):
371 gcpy             self.writegc("(TOOL/MILL,3.175,␣0.00,␣0.00,␣0.00)")
```

```
372 gcpy             self.currenttoolshape = self.endmill_square(3.175,
                         12.7)
373 gcpy         elif (tool_number == 112):
374 gcpy             self.writegc("(TOOL/MILL,1.5875,␣0.00,␣0.00,␣0.00)")
375 gcpy             self.currenttoolshape = self.endmill_square(1.5875,
                         6.35)
376 gcpy         elif (tool_number == 122):
377 gcpy             self.writegc("(TOOL/MILL,0.79375,␣0.00,␣0.00,␣0.00)")
378 gcpy             self.currenttoolshape = self.endmill_square(0.79375,
                         1.5875)
379 gcpy         elif (tool_number == 202):
380 gcpy             self.writegc("(TOOL/MILL,6.35,␣3.175,␣0.00,␣0.00)")
381 gcpy             self.currenttoolshape = self.gcp_endmill_ball(6.35,
                         19.05)
382 gcpy         elif (tool_number == 101):
383 gcpy             self.writegc("(TOOL/MILL,3.175,␣1.5875,␣0.00,␣0.00)")
384 gcpy             self.currenttoolshape = self.gcp_endmill_ball(3.175,
                         12.7)
385 gcpy         elif (tool_number == 111):
386 gcpy             self.writegc("(TOOL/MILL,1.5875,␣0.79375,␣0.00,␣0.00)")
387 gcpy             self.currenttoolshape = self.gcp_endmill_ball(1.5875,
                         6.35)
388 gcpy         elif (tool_number == 121):
389 gcpy             self.writegc("(TOOL/MILL,3.175,␣0.79375,␣0.00,␣0.00)")
390 gcpy             self.currenttoolshape = self.gcp_endmill_ball(0.79375,
                         1.5875)
391 gcpy         elif (tool_number == 327):
392 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣13.4874,␣30.00)")
393 gcpy             self.currenttoolshape = self.gcp_endmill_v(60, 26.9748)
394 gcpy         elif (tool_number == 301):
395 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣6.35,␣45.00)")
396 gcpy             self.currenttoolshape = self.gcp_endmill_v(90, 12.7)
397 gcpy         elif (tool_number == 302):
398 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣10.998,␣30.00)")
399 gcpy             self.currenttoolshape = self.gcp_endmill_v(60, 12.7)
400 gcpy         elif (tool_number == 390):
401 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣1.5875,␣45.00)")
402 gcpy             self.currenttoolshape = self.gcp_endmill_v(90, 3.175)
403 gcpy         elif (tool_number == 374):
404 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
405 gcpy         elif (tool_number == 375):
406 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
407 gcpy         elif (tool_number == 376):
408 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
409 gcpy         elif (tool_number == 378):
410 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
411 gcpy         elif (tool_number == 814):
412 gcpy             self.writegc("(TOOL/MILL,12.7,␣6.367,␣12.7,␣0.00)")
413 gcpy             #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
                         )
414 gcpy             #https://www.leevalley.com/en-us/shop/tools/power-tool-
                         accessories/router-bits/30172-dovetail-bits?item=18
                         J1607
415 gcpy             self.currenttoolshape = self.gcp_dovetail(12.7, 6.367,
                         12.7, 14)
416 gcpy         elif (tool_number == 56125):#0.508/2, 1.531
417 gcpy             self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
                         ␣3.175)")
418 gcpy         elif (tool_number == 56142):#0.508/2, 2.921
419 gcpy             self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
                         5.55625,␣1.5875)")
420 gcpy #         elif (tool_number == 312):#1.524/2, 3.175
421 gcpy #             self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
        Radius, Height, Length)")
422 gcpy         elif (tool_number == 1570):#0.507/2, 4.509
423 gcpy             self.writegc("(TOOL/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
                         12.7,␣4.7625)")
```

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.

```
424 gcpy         self.writegc("M6T",str(tool_number))
425 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:

```
124 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:

```
427 gcpy     def tool_diameter(self, ptd_tool, ptd_depth):
428 gcpy # Square 122,112,102,201
429 gcpy         if ptd_tool == 122:
430 gcpy             return 0.79375
431 gcpy         if ptd_tool == 112:
432 gcpy             return 1.5875
433 gcpy         if ptd_tool == 102:
434 gcpy             return 3.175
435 gcpy         if ptd_tool == 201:
436 gcpy             return 6.35
437 gcpy # Ball 121,111,101,202
438 gcpy         if ptd_tool == 122:
439 gcpy             if ptd_depth > 0.396875:
440 gcpy                 return 0.79375
441 gcpy             else:
442 gcpy                 return ptd_tool
443 gcpy         if ptd_tool == 112:
444 gcpy             if ptd_depth > 0.79375:
445 gcpy                 return 1.5875
446 gcpy             else:
447 gcpy                 return ptd_tool
448 gcpy         if ptd_tool == 101:
449 gcpy             if ptd_depth > 1.5875:
450 gcpy                 return 3.175
451 gcpy             else:
452 gcpy                 return ptd_tool
453 gcpy         if ptd_tool == 202:
454 gcpy             if ptd_depth > 3.175:
455 gcpy                 return 6.35
456 gcpy             else:
457 gcpy                 return ptd_tool
458 gcpy # V 301, 302, 390
459 gcpy         if ptd_tool == 301:
460 gcpy             return ptd_tool
461 gcpy         if ptd_tool == 302:
462 gcpy             return ptd_tool
463 gcpy         if ptd_tool == 390:
464 gcpy             return ptd_tool
465 gcpy # Keyhole
466 gcpy         if ptd_tool == 374:
467 gcpy             if ptd_depth < 3.175:
468 gcpy                 return 9.525
469 gcpy             else:
470 gcpy                 return 6.35
471 gcpy         if ptd_tool == 375:
472 gcpy             if ptd_depth < 3.175:
473 gcpy                 return 9.525
474 gcpy             else:
475 gcpy                 return 8
476 gcpy         if ptd_tool == 376:
```

```
477 gcpy             if ptd_depth < 4.7625:
478 gcpy                 return 12.7
479 gcpy             else:
480 gcpy                 return 6.35
481 gcpy         if ptd_tool == 378:
482 gcpy             if ptd_depth < 4.7625:
483 gcpy                 return 12.7
484 gcpy             else:
485 gcpy                 return 8
486 gcpy # Dovetail
487 gcpy         if ptd_tool == 814:
488 gcpy             if ptd_depth > 12.7:
489 gcpy                 return 6.35
490 gcpy             else:
491 gcpy                 return 12.7
```

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:

```
493 gcpy     def tool_radius(self, ptd_tool, ptd_depth):
494 gcpy         tr = self.tool_diameter(ptd_tool, ptd_depth)/2
495 gcpy         return tr
```

(Note that where values are not fully calculated values currently the passed in tool number is returned which 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.

```
497 gcpy #def popengcodefile(fn):
498 gcpy #     global f
499 gcpy #     f = open(fn, "w")
500 gcpy #
501 gcpy #def popendxffile(fn):
502 gcpy #     global dxf
503 gcpy #     dxf = open(fn, "w")
504 gcpy #
505 gcpy #def popendxflgblfile(fn):
506 gcpy #     global dxflgbl
507 gcpy #     dxflgbl = open(fn, "w")
508 gcpy #
509 gcpy #def popendxflgsqfile(fn):
510 gcpy #     global dxflgsq
511 gcpy #     dxflgsq = open(fn, "w")
512 gcpy #
```

```
513 gcpy #def popendxflgVfile(fn):
514 gcpy #     global dxflgV
515 gcpy #     dxflgV = open(fn, "w")
516 gcpy #
517 gcpy #def popendxfsmblfile(fn):
518 gcpy #     global dxfsmbl
519 gcpy #     dxfsmbl = open(fn, "w")
520 gcpy #
521 gcpy #def popendxfsmsqfile(fn):
522 gcpy #     global dxfsmsq
523 gcpy #     dxfsmsq = open(fn, "w")
524 gcpy #
525 gcpy #def popendxfsmVfile(fn):
526 gcpy #     global dxfsmV
527 gcpy #     dxfsmV = open(fn, "w")
528 gcpy #
529 gcpy #def popendxfKHfile(fn):
530 gcpy #     global dxfKH
531 gcpy #     dxfKH = open(fn, "w")
532 gcpy #
533 gcpy #def popendxfDTfile(fn):
534 gcpy #     global dxfDT
535 gcpy #     dxfDT = open(fn, "w")
536 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`

```
126 gcpscad module opengcodefile(fn) {
127 gcpscad if (generategcode == true) {
128 gcpscad     oopengcodefile(fn);
129 gcpscad //     echo(fn);
130 gcpscad     owritecomment(fn);
```

```
131 gcpscad      }
132 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`

```
537 gcpy #def writedxflgbl(*arguments):
538 gcpy #     line_to_write = ""
539 gcpy #     for element in arguments:
540 gcpy #         line_to_write += element
541 gcpy #     dxflgbl.write(line_to_write)
542 gcpy #     print(line_to_write)
543 gcpy #     dxflgbl.write("\n")
544 gcpy #
545 gcpy #def writedxflgsq(*arguments):
546 gcpy #     line_to_write = ""
547 gcpy #     for element in arguments:
548 gcpy #         line_to_write += element
549 gcpy #     dxflgsq.write(line_to_write)
550 gcpy #     print(line_to_write)
551 gcpy #     dxflgsq.write("\n")
552 gcpy #
553 gcpy #def writedxflgV(*arguments):
554 gcpy #     line_to_write = ""
555 gcpy #     for element in arguments:
556 gcpy #         line_to_write += element
557 gcpy #     dxflgV.write(line_to_write)
558 gcpy #     print(line_to_write)
559 gcpy #     dxflgV.write("\n")
560 gcpy #
561 gcpy #def writedxfsmbl(*arguments):
562 gcpy #     line_to_write = ""
563 gcpy #     for element in arguments:
564 gcpy #         line_to_write += element
565 gcpy #     dxfsmbl.write(line_to_write)
566 gcpy #     print(line_to_write)
567 gcpy #     dxfsmbl.write("\n")
568 gcpy #
569 gcpy #def writedxfsmsq(*arguments):
570 gcpy #     line_to_write = ""
571 gcpy #     for element in arguments:
572 gcpy #         line_to_write += element
573 gcpy #     dxfsmsq.write(line_to_write)
574 gcpy #     print(line_to_write)
575 gcpy #     dxfsmsq.write("\n")
576 gcpy #
577 gcpy #def writedxfsmV(*arguments):
578 gcpy #     line_to_write = ""
579 gcpy #     for element in arguments:
580 gcpy #         line_to_write += element
581 gcpy #     dxfsmV.write(line_to_write)
```

```
582 gcpy #     print(line_to_write)
583 gcpy #     dxfsmV.write("\n")
584 gcpy #
585 gcpy #def writedxfKH(*arguments):
586 gcpy #     line_to_write = ""
587 gcpy #     for element in arguments:
588 gcpy #         line_to_write += element
589 gcpy #     dxfKH.write(line_to_write)
590 gcpy #     print(line_to_write)
591 gcpy #     dxfKH.write("\n")
592 gcpy #
593 gcpy #def writedxfDT(*arguments):
594 gcpy #     line_to_write = ""
595 gcpy #     for element in arguments:
596 gcpy #         line_to_write += element
597 gcpy #     dxfDT.write(line_to_write)
598 gcpy #     print(line_to_write)
599 gcpy #     dxfDT.write("\n")
600 gcpy #
```

owritecomment
dxfwriteone
dxfwritelgbl
dxfwritelgsq
dxfwritelgV
dxfwritesmbl
dxfwritesmsq
dxfwritesmV

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.

```
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 }
```

owrite...

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...

```
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.

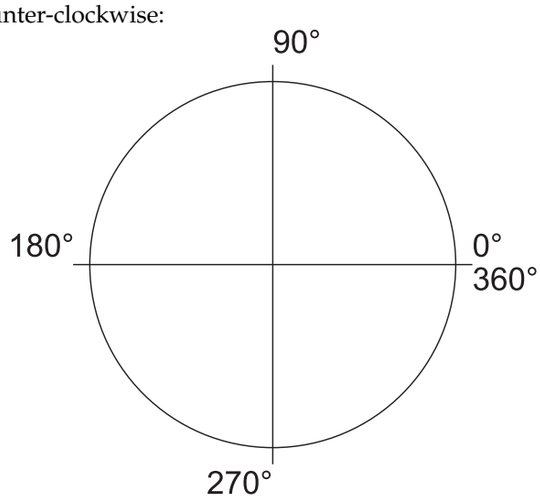
dxfpreamble

```
601 gcpy     def dxfpreamble(self, tn):
602 gcpy #         self.writedxf(tn,str(tn))
603 gcpy         self.writedxf(tn,"0")
604 gcpy         self.writedxf(tn,"SECTION")
605 gcpy         self.writedxf(tn,"2")
606 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.

```
608 gcpy      def dxfline(self, tn, xbegin,ybegin,xend,yend):
609 gcpy          self.writedxf(tn,"0")
610 gcpy          self.writedxf(tn,"LWPOLYLINE")
611 gcpy          self.writedxf(tn,"90")
612 gcpy          self.writedxf(tn,"2")
613 gcpy          self.writedxf(tn,"70")
614 gcpy          self.writedxf(tn,"0")
615 gcpy          self.writedxf(tn,"43")
616 gcpy          self.writedxf(tn,"0")
617 gcpy          self.writedxf(tn,"10")
618 gcpy          self.writedxf(tn,str(xbegin))
619 gcpy          self.writedxf(tn,"20")
620 gcpy          self.writedxf(tn,str(ybegin))
621 gcpy          self.writedxf(tn,"10")
622 gcpy          self.writedxf(tn,str(xend))
623 gcpy          self.writedxf(tn,"20")
624 gcpy          self.writedxf(tn,str(yend))
```

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

```
134 gcpscad module dxfbpl(tn,bx,by) {
135 gcpscad     dxfwrite(tn,"0");
136 gcpscad     dxfwrite(tn,"POLYLINE");
137 gcpscad     dxfwrite(tn,"8");
138 gcpscad     dxfwrite(tn,"default");
139 gcpscad     dxfwrite(tn,"66");
140 gcpscad     dxfwrite(tn,"1");
141 gcpscad     dxfwrite(tn,"70");
142 gcpscad     dxfwrite(tn,"0");
143 gcpscad     dxfwrite(tn,"0");
144 gcpscad     dxfwrite(tn,"VERTEX");
145 gcpscad     dxfwrite(tn,"8");
146 gcpscad     dxfwrite(tn,"default");
147 gcpscad     dxfwrite(tn,"70");
```

```
148 gcpscad     dxfwrite(tn,"32");
149 gcpscad     dxfwrite(tn,"10");
150 gcpscad     dxfwrite(tn,str(bx));
151 gcpscad     dxfwrite(tn,"20");
152 gcpscad     dxfwrite(tn,str(by));
153 gcpscad }
154 gcpscad
155 gcpscad module beginpolyline(bx,by,bz) {
156 gcpscad if (generatedxf == true) {
157 gcpscad     dxfwriteone("0");
158 gcpscad     dxfwriteone("POLYLINE");
159 gcpscad     dxfwriteone("8");
160 gcpscad     dxfwriteone("default");
161 gcpscad     dxfwriteone("66");
162 gcpscad     dxfwriteone("1");
163 gcpscad     dxfwriteone("70");
164 gcpscad     dxfwriteone("0");
165 gcpscad     dxfwriteone("0");
166 gcpscad     dxfwriteone("VERTEX");
167 gcpscad     dxfwriteone("8");
168 gcpscad     dxfwriteone("default");
169 gcpscad     dxfwriteone("70");
170 gcpscad     dxfwriteone("32");
171 gcpscad     dxfwriteone("10");
172 gcpscad     dxfwriteone(str(bx));
173 gcpscad     dxfwriteone("20");
174 gcpscad     dxfwriteone(str(by));
175 gcpscad     dxfbpl(current_tool(),bx,by);}
176 gcpscad }
177 gcpscad
178 gcpscad module dxfapl(tn,bx,by) {
179 gcpscad     dxfwrite(tn,"0");
180 gcpscad     dxfwrite(tn,"VERTEX");
181 gcpscad     dxfwrite(tn,"8");
182 gcpscad     dxfwrite(tn,"default");
183 gcpscad     dxfwrite(tn,"70");
184 gcpscad     dxfwrite(tn,"32");
185 gcpscad     dxfwrite(tn,"10");
186 gcpscad     dxfwrite(tn,str(bx));
187 gcpscad     dxfwrite(tn,"20");
188 gcpscad     dxfwrite(tn,str(by));
189 gcpscad }
190 gcpscad
191 gcpscad module addpolyline(bx,by,bz) {
192 gcpscad if (generatedxf == true) {
193 gcpscad     dxfwriteone("0");
194 gcpscad     dxfwriteone("VERTEX");
195 gcpscad     dxfwriteone("8");
196 gcpscad     dxfwriteone("default");
197 gcpscad     dxfwriteone("70");
198 gcpscad     dxfwriteone("32");
199 gcpscad     dxfwriteone("10");
200 gcpscad     dxfwriteone(str(bx));
201 gcpscad     dxfwriteone("20");
202 gcpscad     dxfwriteone(str(by));
203 gcpscad     dxfapl(current_tool(),bx,by);
204 gcpscad     }
205 gcpscad }
206 gcpscad
207 gcpscad module dxfcpl(tn) {
208 gcpscad     dxfwrite(tn,"0");
209 gcpscad     dxfwrite(tn,"SEQEND");
210 gcpscad }
211 gcpscad
212 gcpscad module closepolyline() {
213 gcpscad   if (generatedxf == true) {
214 gcpscad     dxfwriteone("0");
215 gcpscad     dxfwriteone("SEQEND");
216 gcpscad     dxfcpl(current_tool());
217 gcpscad   }
218 gcpscad }
219 gcpscad
220 gcpscad module writecomment(comment) {
221 gcpscad   if (generategcode == true) {
222 gcpscad     owritecomment(comment);
223 gcpscad   }
224 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.

```
626 gcpy      def dxfpostamble(self,tn):
627 gcpy #         self.writedxf(tn,str(tn))
628 gcpy          self.writedxf(tn,"0")
629 gcpy          self.writedxf(tn,"ENDSEC")
630 gcpy          self.writedxf(tn,"0")
631 gcpy          self.writedxf(tn,"EOF")
```

```
633 gcpy      def gcodepostamble(self):
634 gcpy          self.writegc("Z12.700")
635 gcpy          self.writegc("M05")
636 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.

```
638 gcpy      def closegcodefile(self):
639 gcpy          self.gcodepostamble()
640 gcpy          self.gc.close()
641 gcpy
642 gcpy      def closedxffile(self):
643 gcpy          if self.generatedxf == True:
644 gcpy #             global dxfclosed
645 gcpy              self.dxfclosed = True
646 gcpy              self.dxfpostamble(-1)
647 gcpy              self.dxf.close()
648 gcpy
649 gcpy      def closedxffiles(self):
650 gcpy          if self.generatedxfs == True:
651 gcpy              if (self.large_square_tool_num > 0):
652 gcpy                  self.dxfpostamble(self.large_square_tool_num)
653 gcpy              if (self.small_square_tool_num > 0):
654 gcpy                  self.dxfpostamble(self.small_square_tool_num)
655 gcpy              if (self.large_ball_tool_num > 0):
656 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
657 gcpy              if (self.small_ball_tool_num > 0):
658 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
659 gcpy              if (self.large_V_tool_num > 0):
660 gcpy                  self.dxfpostamble(self.large_V_tool_num)
661 gcpy              if (self.small_V_tool_num > 0):
662 gcpy                  self.dxfpostamble(self.small_V_tool_num)
663 gcpy              if (self.DT_tool_num > 0):
664 gcpy                  self.dxfpostamble(self.DT_tool_num)
665 gcpy              if (self.KH_tool_num > 0):
666 gcpy                  self.dxfpostamble(self.KH_tool_num)
667 gcpy              if (self.Roundover_tool_num > 0):
668 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
669 gcpy              if (self.MISC_tool_num > 0):
670 gcpy                  self.dxfpostamble(self.MISC_tool_num)
671 gcpy
672 gcpy              if (self.large_square_tool_num > 0):
673 gcpy                  self.dxflgsq.close()
674 gcpy              if (self.small_square_tool_num > 0):
675 gcpy                  self.dxfsmsq.close()
676 gcpy              if (self.large_ball_tool_num > 0):
677 gcpy                  self.dxflgbl.close()
678 gcpy              if (self.small_ball_tool_num > 0):
679 gcpy                  self.dxfsmbl.close()
680 gcpy              if (self.large_V_tool_num > 0):
681 gcpy                  self.dxflgV.close()
682 gcpy              if (self.small_V_tool_num > 0):
683 gcpy                  self.dxfsmV.close()
684 gcpy              if (self.DT_tool_num > 0):
685 gcpy                  self.dxfDT.close()
686 gcpy              if (self.KH_tool_num > 0):
687 gcpy                  self.dxfKH.close()
688 gcpy              if (self.Roundover_tool_num > 0):
689 gcpy                  self.dxfRt.close()
690 gcpy              if (self.MISC_tool_num > 0):
691 gcpy                  self.dxfMt.close()
```

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

oclosegcodefile
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.

```
226 gcpscad module closegcodefile() {
227 gcpscad   if (generategcode == true) {
228 gcpscad     owriteone("M05");
229 gcpscad     owriteone("M02");
230 gcpscad     oclosegcodefile();
231 gcpscad   }
232 gcpscad }
233 gcpscad
234 gcpscad module dxfpostamble(arg) {
235 gcpscad     dxfwrite(arg,"0");
236 gcpscad     dxfwrite(arg,"ENDSEC");
237 gcpscad     dxfwrite(arg,"0");
238 gcpscad     dxfwrite(arg,"EOF");
239 gcpscad }
240 gcpscad
241 gcpscad module closedxffile() {
242 gcpscad   if (generatedxf == true) {
243 gcpscad     dxfwriteone("0");
244 gcpscad     dxfwriteone("ENDSEC");
245 gcpscad     dxfwriteone("0");
246 gcpscad     dxfwriteone("EOF");
247 gcpscad     oclosedxffile();
248 gcpscad //    echo("CLOSING");
249 gcpscad     if (large_ball_tool_num >  0) {    dxfpostamble(
                    large_ball_tool_num);
250 gcpscad       oclosedxflgblfile();
251 gcpscad     }
252 gcpscad     if (large_square_tool_num >  0) {    dxfpostamble(
                    large_square_tool_num);
253 gcpscad       oclosedxflgsqfile();
```

```
254 gcpscad     }
255 gcpscad     if (large_V_tool_num >  0) {    dxfpostamble(large_V_tool_num);
256 gcpscad       oclosedxflgVfile();
257 gcpscad     }
258 gcpscad     if (small_ball_tool_num >  0) {    dxfpostamble(
                    small_ball_tool_num);
259 gcpscad       oclosedxfsmblfile();
260 gcpscad     }
261 gcpscad     if (small_square_tool_num >  0) {    dxfpostamble(
                    small_square_tool_num);
262 gcpscad       oclosedxfsmsqfile();
263 gcpscad     }
264 gcpscad     if (small_V_tool_num >  0) {    dxfpostamble(small_V_tool_num);
265 gcpscad       oclosedxfsmVfile();
266 gcpscad     }
267 gcpscad     if (DT_tool_num >  0) {    dxfpostamble(DT_tool_num);
268 gcpscad       oclosedxfDTfile();
269 gcpscad     }
270 gcpscad     if (KH_tool_num >  0) {    dxfpostamble(KH_tool_num);
271 gcpscad       oclosedxfKHfile();
272 gcpscad     }
273 gcpscad   }
274 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:

```
693 gcpy     def movetosafeZ(self):
694 gcpy #         global toolpaths
695 gcpy         self.writegc("(Move␣to␣safe␣Z␣to␣avoid␣workholding)")
696 gcpy         self.writegc("G53G0Z-5.000")
697 gcpy         self.setzpos(self.retractheight)
698 gcpy         toolpath = cylinder(1.5875,12.7)
699 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
700 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
701 gcpy         return toolpath
```

toolpaths

Note that a hard-coded cylinder is used since the command will be used prior to a toolchange. In the future there may be a command for initializing the `toolpaths` so that later cut 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 straight line cut (G1) and arc (G2/G3) commands will appear in both G-code and DXF files.

```
703 gcpy     def rapid(self, ex, ey, ez):
704 gcpy #         global toolpath
705 gcpy #         global toolpaths
706 gcpy         self.writegc("G00␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                    )
707 gcpy         start = self.currenttool()
708 gcpy         start = start.translate([self.xpos(), self.ypos(), self.
                    zpos()])
709 gcpy         toolpath = hull(start, start.translate([ex,ey,ez]))
710 gcpy         self.setxpos(ex)
711 gcpy         self.setypos(ey)
712 gcpy         self.setzpos(ez)
713 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
714 gcpy         return toolpath
```

```
716 gcpy     def rapidXY(self, ex, ey):
717 gcpy #         global toolpath
```

```
718 gcpy #         global toolpaths
719 gcpy          self.writegc("G00␣X", str(ex), "␣Y", str(ey))
720 gcpy          start = self.currenttool()
721 gcpy          start = start.translate([self.xpos(), self.ypos(), self.
                     zpos()])
722 gcpy          toolpath = hull(start, start.translate([ex,ey,self.zpos()])
                     )
723 gcpy          self.setxpos(ex)
724 gcpy          self.setypos(ey)
725 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
726 gcpy          return toolpath
```

```
728 gcpy      def rapidZ(self, ez):
729 gcpy #         global toolpath
730 gcpy #         global toolpaths
731 gcpy          self.writegc("G00␣Z", str(ez))
732 gcpy          start = self.currenttool()
733 gcpy          start = start.translate([self.xpos(), self.ypos(), self.
                     zpos()])
734 gcpy          toolpath = hull(start, start.translate([self.xpos(),self.
                     ypos(),ez]))
735 gcpy          self.setzpos(ez)
736 gcpy #         self.toolpaths = union([self.toolpaths, toolpath])
737 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.

```
739 gcpy      def cutline(self,ex, ey, ez):
740 gcpy #         global toolpath
741 gcpy #         global toolpaths
742 gcpy #         print("cutline tool #", self.currenttoolnumber())
743 gcpy          if (self.currenttoolnumber() == 56142):
744 gcpy #             print("cutline tool internal #", self.
        currenttoolnumber())
745 gcpy              toolpath = self.cutroundovertool(self.xpos(), self.ypos
                         (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
746 gcpy          elif (self.currenttoolnumber() == 56125):
747 gcpy              toolpath = self.cutroundovertool(self.xpos(), self.ypos
                         (), self.zpos(), ex, ey, ez, 0.508/2, 2.921)
748 gcpy #         elif (self.currenttoolnumber() == 312):
749 gcpy #             toolpath = self.cutroundovertool(self.xpos(), self.
        ypos(), self.zpos(), ex, ey, ez, 1.524/2, 3.175)
750 gcpy          elif (self.currenttoolnumber() == 1570):
751 gcpy              toolpath = self.cutroundovertool(self.xpos(), self.ypos
                         (), self.zpos(), ex, ey, ez, 0.507/2, 4.509)
752 gcpy          elif (self.currenttoolnumber() == 374):
753 gcpy #             self.writegc("(TOOL/MILL,9.53, 0.00, 3.17, 0.00)")
754 gcpy              shaft = cylinder(9.525, 6.35/2, 6.35/2)
755 gcpy              shaftend = shaft
756 gcpy              shaftbegin = shaft.translate([self.xpos(), self.ypos(),
                          self.zpos()])
757 gcpy              shaftpath = hull(shaftbegin, shaftend.translate([ex,ey,
                         ez]))
758 gcpy              start = cylinder(3.175, 9.525/2, 9.525/2)
759 gcpy              end = start
760 gcpy              start = start.translate([self.xpos(), self.ypos(), self
                         .zpos()])
761 gcpy              cutpath = hull(start, end.translate([ex,ey,ez]))
762 gcpy              toolpath = union(shaftpath, cutpath)
763 gcpy          elif (self.currenttoolnumber() == 375):
764 gcpy #             self.writegc("(TOOL/MILL,9.53, 0.00, 3.17, 0.00)")
765 gcpy              shaft = cylinder(9.525, 8/2, 8/2)
766 gcpy              shaftend = shaft
767 gcpy              shaftbegin = shaft.translate([self.xpos(), self.ypos(),
                          self.zpos()])
768 gcpy              shaftpath = hull(shaftbegin, shaftend.translate([ex,ey,
                         ez]))
769 gcpy              start = cylinder(3.175, 9.525/2, 9.525/2)
770 gcpy              end = start
771 gcpy              start = start.translate([self.xpos(), self.ypos(), self
                         .zpos()])
772 gcpy              cutpath = hull(start, end.translate([ex,ey,ez]))
773 gcpy              toolpath = union(shaftpath, cutpath)
774 gcpy          elif (self.currenttoolnumber() == 376):
775 gcpy #             self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
```

```
776 gcpy             shaft = cylinder(9.525, 6.35/2, 6.35/2)
777 gcpy             shaftend = shaft
778 gcpy             shaftbegin = shaft.translate([self.xpos(), self.ypos(),
                         self.zpos()])
779 gcpy             shaftpath = hull(shaftbegin, shaftend.translate([ex,ey,
                        ez]))
780 gcpy             start = cylinder(3.175, 12.7/2, 12.7/2)
781 gcpy             end = start
782 gcpy             start = start.translate([self.xpos(), self.ypos(), self
                        .zpos()])
783 gcpy             cutpath = hull(start, end.translate([ex,ey,ez]))
784 gcpy             toolpath = union(shaftpath, cutpath)
785 gcpy         elif (self.currenttoolnumber() == 378):
786 gcpy #            self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
787 gcpy             shaft = cylinder(9.525, 8/2, 8/2)
788 gcpy             shaftend = shaft
789 gcpy             shaftbegin = shaft.translate([self.xpos(), self.ypos(),
                         self.zpos()])
790 gcpy             shaftpath = hull(shaftbegin, shaftend.translate([ex,ey,
                        ez]))
791 gcpy             start = cylinder(3.175, 12.7/2, 12.7/2)
792 gcpy             end = start
793 gcpy             start = start.translate([self.xpos(), self.ypos(), self
                        .zpos()])
794 gcpy             cutpath = hull(start, end.translate([ex,ey,ez]))
795 gcpy             toolpath = union(shaftpath, cutpath)
796 gcpy         else:
797 gcpy             start = self.currenttool()
798 gcpy             start = start.translate([self.xpos(), self.ypos(), self
                        .zpos()])
799 gcpy             end = self.currenttool()
800 gcpy             toolpath = hull(start, end.translate([ex,ey,ez]))
801 gcpy         self.setxpos(ex)
802 gcpy         self.setypos(ey)
803 gcpy         self.setzpos(ez)
804 gcpy #        self.toolpaths = union([self.toolpaths, toolpath])
805 gcpy         return toolpath
806 gcpy
807 gcpy     def cutZgcfeed(self, ez, feed):
808 gcpy         self.writegc("G01␣Z", str(ez), "F",str(feed))
809 gcpy         return self.cutline(self.xpos(),self.ypos(),ez)
810 gcpy
811 gcpy     def cutlinedxfgc(self,ex, ey, ez):
812 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
813 gcpy         self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                    )
814 gcpy         return self.cutline(ex, ey, ez)
815 gcpy
816 gcpy     def cutlinedxfgcfeed(self,ex, ey, ez, feed):
817 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
818 gcpy         self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                    , "␣F", str(feed))
819 gcpy         return self.cutline(ex, ey, ez)
820 gcpy
821 gcpy     def cutroundovertool(self, bx, by, bz, ex, ey, ez,
                tool_radius_tip, tool_radius_width):
822 gcpy #        n = 90 + fn*3
823 gcpy #        print("Tool dimensions", tool_radius_tip,
            tool_radius_width, "begin ",bx, by, bz,"end ", ex, ey, ez)
824 gcpy         step = 4 #360/n
825 gcpy         shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
826 gcpy         toolpath = hull(shaft.translate([bx,by,bz]), shaft.
                    translate([ex,ey,ez]))
827 gcpy         shaft = cylinder(tool_radius_width*2,tool_radius_tip+
                    tool_radius_width,tool_radius_tip+tool_radius_width)
828 gcpy         toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
                    tool_radius_width]), shaft.translate([ex,ey,ez+
                    tool_radius_width])))
829 gcpy         for i in range(1, 90, 1):
830 gcpy             angle = i
831 gcpy             dx = tool_radius_width*math.cos(math.radians(angle))
832 gcpy             dxx = tool_radius_width*math.cos(math.radians(angle+1))
833 gcpy             dzz = tool_radius_width*math.sin(math.radians(angle))
834 gcpy             dz = tool_radius_width*math.sin(math.radians(angle+1))
835 gcpy             dh = abs(dzz-dz)+0.0001
836 gcpy             slice = cylinder(dh,tool_radius_tip+tool_radius_width-
```

```
                    dx,tool_radius_tip+tool_radius_width-dxx)
837 gcpy          toolpath = toolpath.union(hull(slice.translate([bx,by,
                    bz+dz]), slice.translate([ex,ey,ez+dz])))
838 gcpy      return toolpath
```

```
276 gcpscad module otm(ex, ey, ez, r,g,b) {
277 gcpscad color([r,g,b]) hull(){
278 gcpscad     translate([xpos(), ypos(), zpos()]){
279 gcpscad       select_tool(current_tool());
280 gcpscad     }
281 gcpscad     translate([ex, ey, ez]){
282 gcpscad       select_tool(current_tool());
283 gcpscad     }
284 gcpscad   }
285 gcpscad oset(ex, ey, ez);
286 gcpscad }
287 gcpscad
288 gcpscad module ocut(ex, ey, ez) {
289 gcpscad   //color([0.2,1,0.2]) hull(){
290 gcpscad   otm(ex, ey, ez, 0.2,1,0.2);
291 gcpscad }
292 gcpscad
293 gcpscad module orapid(ex, ey, ez) {
294 gcpscad   //color([0.93,0,0]) hull(){
295 gcpscad   otm(ex, ey, ez, 0.93,0,0);
296 gcpscad }
297 gcpscad
298 gcpscad module rapidbx(bx, by, bz, ex, ey, ez) {
299 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
300 gcpscad   if (generategcode == true) {
301 gcpscad     writecomment("rapid");
302 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
303 gcpscad   }
304 gcpscad     orapid(ex, ey, ez);
305 gcpscad }
306 gcpscad
307 gcpscad module rapid(ex, ey, ez) {
308 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
309 gcpscad   if (generategcode == true) {
310 gcpscad       writecomment("rapid");
311 gcpscad       owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
312 gcpscad   }
313 gcpscad   orapid(ex, ey, ez);
314 gcpscad }
315 gcpscad
316 gcpscad module movetosafez() {
317 gcpscad   //this should be move to retract height
318 gcpscad   if (generategcode == true) {
319 gcpscad       writecomment("Move to safe Z to avoid workholding");
320 gcpscad       owriteone("G53G0Z-5.000");
321 gcpscad   }
322 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
323 gcpscad }
324 gcpscad
325 gcpscad module begintoolpath(bx,by,bz) {
326 gcpscad   if (generategcode == true) {
327 gcpscad     writecomment("PREPOSITION FOR RAPID PLUNGE");
328 gcpscad     owritefour("G0X", str(bx), "Y",str(by));
329 gcpscad     owritetwo("Z", str(bz));
330 gcpscad   }
331 gcpscad   orapid(bx,by,bz);
332 gcpscad }
333 gcpscad
334 gcpscad module movetosafeheight() {
335 gcpscad   //this should be move to machine position
336 gcpscad   if (generategcode == true) {
337 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
338 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge)
339 gcpscad     if (zeroheight == "Top") {
340 gcpscad       owritetwo("Z",str(retractheight));
341 gcpscad     }
342 gcpscad   }
343 gcpscad     orapid(getxpos(), getypos(), retractheight+55);
344 gcpscad }
345 gcpscad
346 gcpscad module cutoneaxis_setfeed(axis,depth,feed) {
347 gcpscad   if (generategcode == true) {
```

```
348 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
349 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge) G1Z7.612F381.0
350 gcpscad     if (zeroheight == "Top") {
351 gcpscad       owritefive("G1",axis,str(depth),"F",str(feed));
352 gcpscad     }
353 gcpscad   }
354 gcpscad   if (axis == "X") {setxpos(depth);
355 gcpscad     ocut(depth, getypos(), getzpos());}
356 gcpscad     if (axis == "Y") {setypos(depth);
357 gcpscad       ocut(getxpos(), depth, getzpos());
358 gcpscad     }
359 gcpscad       if (axis == "Z") {setzpos(depth);
360 gcpscad         ocut(getxpos(), getypos(), depth);
361 gcpscad       }
362 gcpscad }
363 gcpscad
364 gcpscad module cut(ex, ey, ez) {
365 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
366 gcpscad   if (generategcode == true) {
367 gcpscad      owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
368 gcpscad   }
369 gcpscad   //if (generatesvg == true) {
370 gcpscad   //    owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
371 gcpscad   //    orapid(getxpos(), getypos(), retractheight+5);
372 gcpscad   //    writesvgline(getxpos(),getypos(),ex,ey);
373 gcpscad   //}
374 gcpscad   ocut(ex, ey, ez);
375 gcpscad }
376 gcpscad
377 gcpscad module cutwithfeed(ex, ey, ez, feed) {
378 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
379 gcpscad   if (generategcode == true) {
380 gcpscad   //    writecomment("rapid");
381 gcpscad     owriteeight("G1 X",str(ex)," Y", str(ey), " Z", str(ez),"F",str
                (feed));
382 gcpscad   }
383 gcpscad   ocut(ex, ey, ez);
384 gcpscad }
385 gcpscad
386 gcpscad module endtoolpath() {
387 gcpscad   if (generategcode == true) {
388 gcpscad   //Z31.750
389 gcpscad   //    owriteone("G53G0Z-5.000");
390 gcpscad     owritetwo("Z",str(retractheight));
391 gcpscad   }
392 gcpscad   orapid(getxpos(),getypos(),retractheight);
393 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

- 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?

- 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 dove-tail 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
- 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

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.

arcloop
narcloop

```
840 gcpy     def arcloop(self, barc, earc, xcenter, ycenter, radius):
841 gcpy #         global toolpath
842 gcpy         toolpath = self.currenttool()
843 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
844 gcpy         i = barc
845 gcpy         while i < earc:
846 gcpy             toolpath = toolpath.union(self.cutline(xcenter + radius
                         * math.cos(math.radians(i)), ycenter + radius *
                        math.sin(math.radians(i)), self.zpos()-(self.tzpos()
                        )))
847 gcpy             self.setxpos(xcenter + radius * math.cos(math.radians(i
                        )))
848 gcpy             self.setypos(ycenter + radius * math.sin(math.radians(i
                        )))
849 gcpy             i += 1
850 gcpy #         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
            radius, barc, earc)
851 gcpy         return toolpath
852 gcpy
853 gcpy     def narcloop(barc,earc, xcenter, ycenter, radius):
854 gcpy #         global toolpath
855 gcpy         toolpath = self.currenttool()
856 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                    .zpos()])
857 gcpy         i = barc
858 gcpy         while i > earc:
859 gcpy             toolpath = toolpath.union(self.cutline(xcenter + radius
                         * math.cos(math.radians(i)), ycenter + radius *
                        math.sin(math.radians(i)), self.zpos()-(self.tzpos()
                        )))
860 gcpy             self.setxpos(xcenter + radius * math.cos(math.radians(i
                        )))
861 gcpy             self.setypos(ycenter + radius * math.sin(math.radians(i
                        )))
862 gcpy #             print(str(self.xpos()), str(self.ypos()))
863 gcpy             i += -1
864 gcpy #         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
            radius, barc, earc)
865 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.

```
867 gcpy     def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
                endangle):
868 gcpy         if (self.generatedxf == True):
869 gcpy             self.writedxf(tn, "0")
870 gcpy             self.writedxf(tn, "ARC")
871 gcpy             self.writedxf(tn, "10")
872 gcpy             self.writedxf(tn, str(xcenter))
873 gcpy             self.writedxf(tn, "20")
874 gcpy             self.writedxf(tn, str(ycenter))
```

```
875 gcpy             self.writedxf(tn, "40")
876 gcpy             self.writedxf(tn, str(radius))
877 gcpy             self.writedxf(tn, "50")
878 gcpy             self.writedxf(tn, str(anglebegin))
879 gcpy             self.writedxf(tn, "51")
880 gcpy             self.writedxf(tn, str(endangle))
881 gcpy
882 gcpy     def gcodearc(self, xcenter, ycenter, radius, anglebegin,
                 endangle, tn):
883 gcpy         if (self.generategcode == True):
884 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.

```
886 gcpy     def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
887 gcpy #         global toolpath
888 gcpy         toolpath = self.currenttool()
889 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                     .zpos()])
890 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                     radius,0,90)
891 gcpy         if (self.zpos == ez):
892 gcpy             self.settzpos(0)
893 gcpy         else:
894 gcpy             self.settzpos((self.zpos()-ez)/90)
895 gcpy         toolpath = self.arcloop(1,90, xcenter, ycenter, radius)
896 gcpy         self.setxpos(ex)
897 gcpy         self.setypos(ey)
898 gcpy         self.setzpos(ez)
899 gcpy         return toolpath
900 gcpy
901 gcpy     def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
902 gcpy #         global toolpath
903 gcpy         toolpath = self.currenttool()
904 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                     .zpos()])
905 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                     radius,90,180)
906 gcpy         if (self.zpos == ez):
907 gcpy             self.settzpos(0)
908 gcpy         else:
909 gcpy             self.settzpos((self.zpos()-ez)/90)
910 gcpy         toolpath = self.arcloop(91,180, xcenter, ycenter, radius)
911 gcpy         self.setxpos(ex)
912 gcpy         self.setypos(ey)
913 gcpy         self.setzpos(ez)
914 gcpy         return toolpath
915 gcpy
916 gcpy     def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
917 gcpy #         global toolpath
918 gcpy         toolpath = self.currenttool()
919 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                     .zpos()])
920 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                     radius,180,270)
921 gcpy         if (self.zpos == ez):
922 gcpy             self.settzpos(0)
923 gcpy         else:
924 gcpy             self.settzpos((self.zpos()-ez)/90)
925 gcpy         toolpath = self.arcloop(181,270, xcenter, ycenter, radius)
926 gcpy         self.setxpos(ex)
927 gcpy         self.setypos(ey)
928 gcpy         self.setzpos(ez)
929 gcpy         return toolpath
930 gcpy
931 gcpy     def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
932 gcpy #         global toolpath
933 gcpy         toolpath = self.currenttool()
934 gcpy         toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                     .zpos()])
935 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                     radius,270,360)
936 gcpy         if (self.zpos == ez):
937 gcpy             self.settzpos(0)
938 gcpy         else:
939 gcpy             self.settzpos((self.zpos()-ez)/90)
940 gcpy         toolpath = self.arcloop(271,360, xcenter, ycenter, radius)
```

```
941 gcpy          self.setxpos(ex)
942 gcpy          self.setypos(ey)
943 gcpy          self.setzpos(ez)
944 gcpy          return toolpath
945 gcpy
946 gcpy      def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
947 gcpy #          global toolpath
948 gcpy          toolpath = self.currenttool()
949 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
950 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                      radius,0,90)
951 gcpy          if (self.zpos == ez):
952 gcpy              self.settzpos(0)
953 gcpy          else:
954 gcpy              self.settzpos((self.zpos()-ez)/90)
955 gcpy          toolpath = self.narcloop(89,0, xcenter, ycenter, radius)
956 gcpy          self.setxpos(ex)
957 gcpy          self.setypos(ey)
958 gcpy          self.setzpos(ez)
959 gcpy          return toolpath
960 gcpy
961 gcpy      def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
962 gcpy #          global toolpath
963 gcpy          toolpath = self.currenttool()
964 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
965 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                      radius,270,360)
966 gcpy          if (self.zpos == ez):
967 gcpy              self.settzpos(0)
968 gcpy          else:
969 gcpy              self.settzpos((self.zpos()-ez)/90)
970 gcpy          toolpath = self.narcloop(359,270, xcenter, ycenter, radius)
971 gcpy          self.setxpos(ex)
972 gcpy          self.setypos(ey)
973 gcpy          self.setzpos(ez)
974 gcpy          return toolpath
975 gcpy
976 gcpy      def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
977 gcpy #          global toolpath
978 gcpy          toolpath = self.currenttool()
979 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
980 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                      radius,180,270)
981 gcpy          if (self.zpos == ez):
982 gcpy              self.settzpos(0)
983 gcpy          else:
984 gcpy              self.settzpos((self.zpos()-ez)/90)
985 gcpy          toolpath = self.narcloop(269,180, xcenter, ycenter, radius)
986 gcpy          self.setxpos(ex)
987 gcpy          self.setypos(ey)
988 gcpy          self.setzpos(ez)
989 gcpy          return toolpath
990 gcpy
991 gcpy      def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
992 gcpy #          global toolpath
993 gcpy          toolpath = self.currenttool()
994 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
995 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                      radius,90,180)
996 gcpy          if (self.zpos == ez):
997 gcpy              self.settzpos(0)
998 gcpy          else:
999 gcpy              self.settzpos((self.zpos()-ez)/90)
1000 gcpy          toolpath = self.narcloop(179,90, xcenter, ycenter, radius)
1001 gcpy          self.setxpos(ex)
1002 gcpy          self.setypos(ey)
1003 gcpy          self.setzpos(ez)
1004 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
```

```
1006 gcpy     def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1007 gcpy         self.writegc("G03␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                      , "␣R", str(radius))
1008 gcpy
1009 gcpy     def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1010 gcpy         self.writegc("G02␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                      , "␣R", str(radius))
```

The above commands may be called if G-code is also wanted with writing out G-code added:

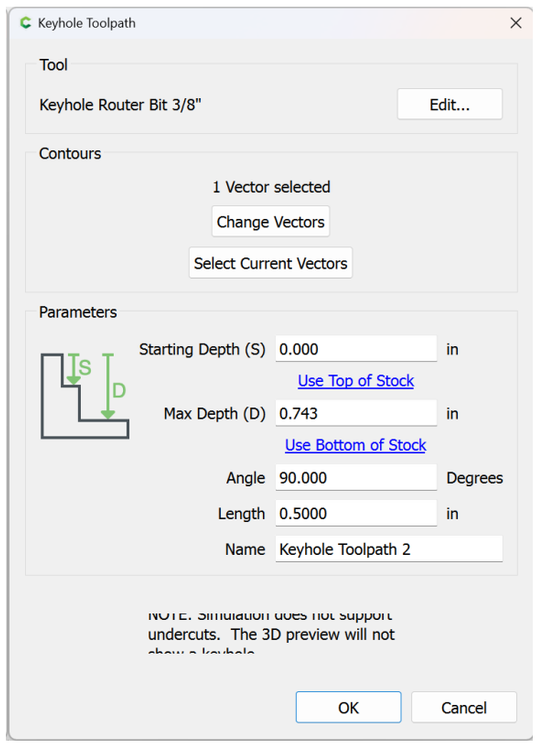
```
1012 gcpy     def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1013 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1014 gcpy         return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1015 gcpy
1016 gcpy     def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1017 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1018 gcpy         return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1019 gcpy
1020 gcpy     def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1021 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1022 gcpy         return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1023 gcpy
1024 gcpy     def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1025 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1026 gcpy         return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1027 gcpy
1028 gcpy     def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1029 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1030 gcpy         return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1031 gcpy
1032 gcpy     def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1033 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1034 gcpy         return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1035 gcpy
1036 gcpy     def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1037 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1038 gcpy         return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
1039 gcpy
1040 gcpy     def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1041 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1042 gcpy         return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
                      radius)
```

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

The interface which is being modeled is that of Carbide Create:



Hence the parameters:

- Starting Depth == `kh_start_depth`
- Max Depth == `kh_max_depth`
- Angle == `kht_direction`
- Length == `kh_distance`
- Tool == `kh_tool_num`

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:

```
1044 gcpy     def cutkeyholegcdxf(self, kh_start_depth, kh_max_depth,
                  kht_direction, kh_distance, kh_tool_num):
1045 gcpy         if (kht_direction == "N"):
1046 gcpy             toolpath = self.cutKHgcdxf(kh_start_depth, kh_max_depth
                        , 90, kh_distance, kh_tool_num)
1047 gcpy             return toolpath
1048 gcpy         elif (kht_direction == "S"):
1049 gcpy             toolpath = self.cutKHgcdxf(kh_start_depth, kh_max_depth
                        , 270, kh_distance, kh_tool_num)
1050 gcpy             return toolpath
1051 gcpy         elif (kht_direction == "E"):
1052 gcpy             toolpath = self.cutKHgcdxf(kh_start_depth, kh_max_depth
                        , 0, kh_distance, kh_tool_num)
1053 gcpy             return toolpath
1054 gcpy         elif (kht_direction == "W"):
1055 gcpy             toolpath = self.cutKHgcdxf(kh_start_depth, kh_max_depth
                        , 180, kh_distance, kh_tool_num)
1056 gcpy             return toolpath
```

`cutKHgcdxf` The original version of the command, `cutKHgcdxf` 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:

```
1058 gcpy      def cutKHgcdxf(self, kh_start_depth, kh_max_depth, kh_angle,
                   kh_distance, kh_tool_num):
1059 gcpy          oXpos = self.xpos()
1060 gcpy          oYpos = self.ypos()
1061 gcpy #Circle at entry hole
1062 gcpy #    def dxfarc(self, xcenter, ycenter, radius, anglebegin,
               endangle, tn):
1063 gcpy #        print(self.tool_radius(kh_tool_num, 7))
1064 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                   tool_radius(kh_tool_num, 7),  0, 90)
1065 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                   tool_radius(kh_tool_num, 7), 90,180)
1066 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                   tool_radius(kh_tool_num, 7),180,270)
1067 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                   tool_radius(kh_tool_num, 7),270,360)
1068 gcpy          toolpath = self.cutline(self.xpos(), self.ypos(), -
                   kh_max_depth)
```

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

```
1070 gcpy #pre-calculate needed values
1071 gcpy          r = self.tool_radius(kh_tool_num, 7)
1072 gcpy #        print(r)
1073 gcpy          rt = self.tool_radius(kh_tool_num, 1)
1074 gcpy #        print(rt)
1075 gcpy          ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                   tool_radius(kh_tool_num, 7))**2)
1076 gcpy #        print(ro)
1077 gcpy          angle = math.degrees(math.acos(ro/rt))
1078 gcpy #Outlines of entry hole and slot
1079 gcpy          if (kh_angle == 0):
1080 gcpy #Lower left of entry hole
1081 gcpy              self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                       tool_radius(kh_tool_num, 1),180,270)
1082 gcpy #Upper left of entry hole
1083 gcpy              self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                       tool_radius(kh_tool_num, 1),90,180)
1084 gcpy #Upper right of entry hole
1085 gcpy #             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                41.810, 90)
1086 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                       angle, 90)
1087 gcpy #Lower right of entry hole
1088 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                       270, 360-angle)
1089 gcpy #             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
               tool_radius(kh_tool_num, 1),270, 270+math.acos(math.radians(self
               .tool_diameter(kh_tool_num, 5)/self.tool_diameter(kh_tool_num,
               1))))
1090 gcpy #Actual line of cut
1091 gcpy #             self.dxfline(kh_tool_num, self.xpos(),self.ypos(),self
               .xpos()+kh_distance,self.ypos())
1092 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
1093 gcpy              self.dxfarc(kh_tool_num, self.xpos()+kh_distance,self.
                       ypos(),self.tool_diameter(kh_tool_num, (kh_max_depth
                       +4.36))/2,0,90)
1094 gcpy #lower right of end of slot
1095 gcpy              self.dxfarc(kh_tool_num, self.xpos()+kh_distance,self.
                       ypos(),self.tool_diameter(kh_tool_num, (kh_max_depth
                       +4.36))/2,270,360)
1096 gcpy #upper right slot
1097 gcpy              self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()-(
                       self.tool_diameter(kh_tool_num,7)/2), self.xpos()+
                       kh_distance, self.ypos()-(self.tool_diameter(
                       kh_tool_num,7)/2))
1098 gcpy #             self.dxfline(kh_tool_num, self.xpos()+(sqrt((self.
               tool_diameter(kh_tool_num,1)^2)-(self.tool_diameter(kh_tool_num
               ,5)^2))/2), self.ypos()+self.tool_diameter(kh_tool_num, (
               kh_max_depth))/2, ( (kh_max_depth-6.34))/2)^2-(self.
               tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.xpos
               ()+kh_distance, self.ypos()+self.tool_diameter(kh_tool_num, (
               kh_max_depth))/2, kh_tool_num)
1099 gcpy #end position at top of slot
1100 gcpy #lower right slot
1101 gcpy              self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()+(
                       self.tool_diameter(kh_tool_num,7)/2), self.xpos()+
```

```
                        kh_distance, self.ypos()+(self.tool_diameter(
                        kh_tool_num,7)/2))
1102 gcpy #         dxfline(kh_tool_num, self.xpos()+(sqrt((self.tool_diameter
          (kh_tool_num,1)^2)-(self.tool_diameter(kh_tool_num,5)^2))/2),
          self.ypos()-self.tool_diameter(kh_tool_num, (kh_max_depth))/2, (
           (kh_max_depth-6.34))/2)^2-(self.tool_diameter(kh_tool_num, (
          kh_max_depth-6.34))/2)^2, self.xpos()+kh_distance, self.ypos()-
          self.tool_diameter(kh_tool_num, (kh_max_depth))/2, KH_tool_num)
1103 gcpy #end position at top of slot
1104 gcpy #    hull(){
1105 gcpy #      translate([xpos(), ypos(), zpos()]){
1106 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1107 gcpy #      }
1108 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1109 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1110 gcpy #      }
1111 gcpy #    }
1112 gcpy #    hull(){
1113 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1114 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1115 gcpy #      }
1116 gcpy #      translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
          {
1117 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1118 gcpy #      }
1119 gcpy #    }
1120 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
1121 gcpy #    cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed
          );
1122 gcpy #    setxpos(getxpos()-kh_distance);
1123 gcpy #  } else if (kh_angle > 0 && kh_angle < 90) {
1124 gcpy #//echo(kh_angle);
1125 gcpy #  dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
          kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_num);
1126 gcpy #  dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
          kh_max_depth))/2,180+kh_angle,270+kh_angle, KH_tool_num);
1127 gcpy #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);
1128 gcpy #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);
1129 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
1130 gcpy #  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);
1131 gcpy #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);
1132 gcpy #dxfline( 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))),
1133 gcpy # 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))),
1134 gcpy # getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
          , (kh_max_depth+4.36))/2)*sin(kh_angle)),
1135 gcpy # getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
          , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
1136 gcpy #//echo("a",tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
1137 gcpy #//echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
1138 gcpy #echo("Aangle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36)
          )/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
1139 gcpy #//echo(kh_angle);
1140 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
          kh_distance*sin(kh_angle)),-kh_max_depth,feed);
1141 gcpy             toolpath = toolpath.union(self.cutline(self.xpos()+
                        kh_distance, self.ypos(), -kh_max_depth))
1142 gcpy         elif (kh_angle == 90):
1143 gcpy #Lower left of entry hole
1144 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),180,270)
1145 gcpy #Lower right of entry hole
1146 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),270,360)
1147 gcpy #left slot
1148 gcpy             self.dxfline(kh_tool_num, self.xpos()-r, self.ypos()+ro
```

```
                        , self.xpos()-r, self.ypos()+kh_distance)
1149 gcpy #right slot
1150 gcpy             self.dxfline(kh_tool_num, self.xpos()+r, self.ypos()+ro
                        , self.xpos()+r, self.ypos()+kh_distance)
1151 gcpy #upper left of end of slot
1152 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos()+
                        kh_distance,r,90,180)
1153 gcpy #upper right of end of slot
1154 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos()+
                        kh_distance,r,0,90)
1155 gcpy #Upper right of entry hole
1156 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        0, 90-angle)
1157 gcpy #Upper left of entry hole
1158 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        90+angle, 180)
1159 gcpy             toolpath = toolpath.union(self.cutline(self.xpos(),
                        self.ypos()+kh_distance, -kh_max_depth))
1160 gcpy         elif (kh_angle == 180):
1161 gcpy #Lower right of entry hole
1162 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),270,360)
1163 gcpy #Upper right of entry hole
1164 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),0,90)
1165 gcpy #Upper left of entry hole
1166 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        90, 180-angle)
1167 gcpy #Lower left of entry hole
1168 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        180+angle, 270)
1169 gcpy #upper slot
1170 gcpy             self.dxfline(kh_tool_num, self.xpos()-ro, self.ypos()-r
                        , self.xpos()-kh_distance, self.ypos()-r)
1171 gcpy #lower slot
1172 gcpy             self.dxfline(kh_tool_num, self.xpos()-ro, self.ypos()+r
                        , self.xpos()-kh_distance, self.ypos()+r)
1173 gcpy #upper left of end of slot
1174 gcpy             self.dxfarc(kh_tool_num, self.xpos()-kh_distance,self.
                        ypos(),r,90,180)
1175 gcpy #lower left of end of slot
1176 gcpy             self.dxfarc(kh_tool_num, self.xpos()-kh_distance,self.
                        ypos(),r,180,270)
1177 gcpy             toolpath = toolpath.union(self.cutline(self.xpos()-
                        kh_distance, self.ypos(), -kh_max_depth))
1178 gcpy         elif (kh_angle == 270):
1179 gcpy #Upper left of entry hole
1180 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),90,180)
1181 gcpy #Upper right of entry hole
1182 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),0,90)
1183 gcpy #left slot
1184 gcpy             self.dxfline(kh_tool_num, self.xpos()-r, self.ypos()-ro
                        , self.xpos()-r, self.ypos()-kh_distance)
1185 gcpy #right slot
1186 gcpy             self.dxfline(kh_tool_num, self.xpos()+r, self.ypos()-ro
                        , self.xpos()+r, self.ypos()-kh_distance)
1187 gcpy #lower left of end of slot
1188 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos()-
                        kh_distance,r,180,270)
1189 gcpy #lower right of end of slot
1190 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos()-
                        kh_distance,r,270,360)
1191 gcpy #lower right of entry hole
1192 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        180, 270-angle)
1193 gcpy #lower left of entry hole
1194 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        270+angle, 360)
1195 gcpy             toolpath = toolpath.union(self.cutline(self.xpos(),
                        self.ypos()-kh_distance, -kh_max_depth))
1196 gcpy #         print(self.zpos())
1197 gcpy         self.setxpos(oXpos)
1198 gcpy         self.setypos(oYpos)
1199 gcpy         return toolpath
1200 gcpy
1201 gcpy #  } else if (kh_angle == 90) {
```

```
1202 gcpy #    //Lower left of entry hole
1203 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
1204 gcpy #    //Lower right of entry hole
1205 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
1206 gcpy #    //Upper right of entry hole
1207 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,acos(tool_diameter(
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
1208 gcpy #    //Upper left of entry hole
1209 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,180-acos(tool_diameter(
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180,KH_tool_num)
           ;
1210 gcpy #    //Actual line of cut
1211 gcpy #    dxfline(getxpos(),getypos(),getxpos(),getypos()+kh_distance);
1212 gcpy #    //upper right of slot
1213 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
           KH_tool_num, (kh_max_depth+4.36))/2,0,90, KH_tool_num);
1214 gcpy #    //upper left of slot
1215 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
           KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
1216 gcpy #    //right of slot
1217 gcpy #    dxfline(
1218 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1219 gcpy #        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,
1220 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1221 gcpy #    //end position at top of slot
1222 gcpy #        getypos()+kh_distance,
1223 gcpy #        KH_tool_num);
1224 gcpy #    dxfline(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);
1225 gcpy #    hull(){
1226 gcpy #      translate([xpos(), ypos(), zpos()]){
1227 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1228 gcpy #      }
1229 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1230 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1231 gcpy #      }
1232 gcpy #    }
1233 gcpy #    hull(){
1234 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1235 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1236 gcpy #      }
1237 gcpy #      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
           {
1238 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1239 gcpy #      }
1240 gcpy #    }
1241 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
1242 gcpy #    cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed
           );
1243 gcpy #    setypos(getypos()-kh_distance);
1244 gcpy #  } else if (kh_angle == 180) {
1245 gcpy #    //Lower right of entry hole
1246 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
1247 gcpy #    //Upper right of entry hole
1248 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
1249 gcpy #    //Upper left of entry hole
1250 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,90, 90+acos(tool_diameter(
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
1251 gcpy #    //Lower left of entry hole
1252 gcpy #    dxfarc(getxpos(),getypos(),9.525/2, 270-acos(tool_diameter(
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num
           );
1253 gcpy #    //upper left of slot
1254 gcpy #    dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
           KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
1255 gcpy #    //lower left of slot
1256 gcpy #    dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
           KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
1257 gcpy #    //Actual line of cut
1258 gcpy #    dxfline(getxpos(),getypos(),getxpos()-kh_distance,getypos());
1259 gcpy #    //upper left slot
1260 gcpy #    dxfline(
1261 gcpy #        getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
```

```
            tool_diameter(KH_tool_num,5)^2))/2),
1262 gcpy #       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,
1263 gcpy #       getxpos()-kh_distance,
1264 gcpy #    //end position at top of slot
1265 gcpy #       getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1266 gcpy #       KH_tool_num);
1267 gcpy #    //lower right slot
1268 gcpy #    dxfline(
1269 gcpy #       getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
            tool_diameter(KH_tool_num,5)^2))/2),
1270 gcpy #       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,
1271 gcpy #       getxpos()-kh_distance,
1272 gcpy #    //end position at top of slot
1273 gcpy #       getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1274 gcpy #       KH_tool_num);
1275 gcpy #    hull(){
1276 gcpy #      translate([xpos(), ypos(), zpos()]){
1277 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1278 gcpy #      }
1279 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1280 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1281 gcpy #      }
1282 gcpy #    }
1283 gcpy #    hull(){
1284 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
1285 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1286 gcpy #      }
1287 gcpy #      translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
            {
1288 gcpy #        gcp_keyhole_shaft(6.35, 9.525);
1289 gcpy #      }
1290 gcpy #    }
1291 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
1292 gcpy #    cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed
            );
1293 gcpy #    setxpos(getxpos()+kh_distance);
1294 gcpy #  } else if (kh_angle == 270) {
1295 gcpy #    //Upper right of entry hole
1296 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
1297 gcpy #    //Upper left of entry hole
1298 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,90,180, KH_tool_num);
1299 gcpy #    //lower right of slot
1300 gcpy #    dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
            KH_tool_num, (kh_max_depth+4.36))/2,270,360, KH_tool_num);
1301 gcpy #    //lower left of slot
1302 gcpy #    dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
            KH_tool_num, (kh_max_depth+4.36))/2,180,270, KH_tool_num);
1303 gcpy #    //Actual line of cut
1304 gcpy #    dxfline(getxpos(),getypos(),getxpos(),getypos()-kh_distance);
1305 gcpy #    //right of slot
1306 gcpy #    dxfline(
1307 gcpy #       getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1308 gcpy #       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,
1309 gcpy #       getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1310 gcpy #    //end position at top of slot
1311 gcpy #       getypos()-kh_distance,
1312 gcpy #       KH_tool_num);
1313 gcpy #    //left of slot
1314 gcpy #    dxfline(
1315 gcpy #       getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1316 gcpy #       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,
1317 gcpy #       getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1318 gcpy #    //end position at top of slot
1319 gcpy #       getypos()-kh_distance,
1320 gcpy #       KH_tool_num);
1321 gcpy #    //Lower right of entry hole
1322 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,360-acos(tool_diameter(
            KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num
            );
1323 gcpy #    //Lower left of entry hole
```

```
1324 gcpy #     dxfarc(getxpos(),getypos(),9.525/2,180, 180+acos(tool_diameter
             (KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
1325 gcpy #     hull(){
1326 gcpy #       translate([xpos(), ypos(), zpos()]){
1327 gcpy #         gcp_keyhole_shaft(6.35, 9.525);
1328 gcpy #       }
1329 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1330 gcpy #         gcp_keyhole_shaft(6.35, 9.525);
1331 gcpy #       }
1332 gcpy #     }
1333 gcpy #     hull(){
1334 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1335 gcpy #         gcp_keyhole_shaft(6.35, 9.525);
1336 gcpy #       }
1337 gcpy #       translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
             {
1338 gcpy #         gcp_keyhole_shaft(6.35, 9.525);
1339 gcpy #       }
1340 gcpy #     }
1341 gcpy #     cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
1342 gcpy #     cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed
             );
1343 gcpy #     setypos(getypos()+kh_distance);
1344 gcpy #   }
1345 gcpy #}
```

Lastly, to use the class it will be necessary to load it:

```
1177 gcpy from gcodepreview import *
```

which may then allow loading the the class as expected. https://github.com/gsohler/openscad/issues/48

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`.

```
395 gcpscad module begincutdxf(rh, ex, ey, ez, fr) {
396 gcpscad   rapid(getxpos(),getypos(),rh);
397 gcpscad   cutwithfeed(ex,ey,ez,fr);
398 gcpscad }
```

```
400 gcpscad module continuecutdxf(ex, ey, ez, fr) {
401 gcpscad   cutwithfeed(ex,ey,ez,fr);
402 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:

```
404 gcpscad module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
        {//passes
405 gcpscad   movetosafez();
406 gcpscad   hull(){
407 gcpscad     //  for (i = [0 : abs(1) : passes]) {
408 gcpscad     //      rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(
                  current_tool()))/passes,bx+tool_radius(rtn),1);
409 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                  (current_tool()))/passes,by+tool_radius(rtn),bz-rdepth,feed)
                  ;
410 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                  (current_tool()))/passes,by+rheight-tool_radius(rtn),bz-
                  rdepth,feed);
411 gcpscad
412 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                  feed);
413 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                  rdepth,feed);
414 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(
                  rtn),bz-rdepth,feed);
415 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                  rdepth,feed);
416 gcpscad   }
417 gcpscad   //dxfarc(xcenter,ycenter,radius,anglebegin,endangle, tn)
418 gcpscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                  ,180,270, rtn);
419 gcpscad   //dxfline(xbegin,ybegin,xend,yend, tn)
420 gcpscad   dxfline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn),
                  rtn);
421 gcpscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),90,180, rtn);
422 gcpscad   dxfline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(rtn)
                  ,by+rheight, rtn);
423 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),0,90, rtn);
424 gcpscad   dxfline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                  tool_radius(rtn), rtn);
425 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                  (rtn),270,360, rtn);
426 gcpscad   dxfline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by, rtn
                  );
427 gcpscad }
```

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

```
429 gcpscad module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth,
              rtn) {//passes
430 gcpscad   movetosafez();
431 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                  feed);
432 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                  rdepth,feed);
433 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn
                  ),bz-rdepth,feed);
434 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                  rdepth,feed);
435 gcpscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                  ,180,270, rtn);
436 gcpscad   dxfline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn),
                  rtn);
437 gcpscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),90,180, rtn);
438 gcpscad   dxfline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(rtn)
                  ,by+rheight, rtn);
439 gcpscad   dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                  tool_radius(rtn),0,90, rtn);
440 gcpscad   dxfline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                  tool_radius(rtn), rtn);
```

```
441 gcpscad    dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                   (rtn),270,360, rtn);
442 gcpscad    dxfline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by, rtn
                   );
443 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):

```
445 gcpscad  module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
446 gcpscad    dxfline(bx,by,bx,by+rheight, rtn);
447 gcpscad    dxfline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
448 gcpscad    dxfline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
449 gcpscad    dxfline(bx+rwidth,by,bx,by, rtn);
450 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:

```
452 gcpscad  module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                 {
453 gcpscad    movetosafez();
454 gcpscad    cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                   feed);
455 gcpscad    cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-
                   rdepth,feed);
456 gcpscad    cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn
                   ),bz-rdepth,feed);
457 gcpscad    cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-
                   rdepth,feed);
458 gcpscad    cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                   feed);
459 gcpscad    dxfline(bx,by,bx,by+rheight, rtn);
460 gcpscad    dxfline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
461 gcpscad    dxfline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
462 gcpscad    dxfline(bx+rwidth,by,bx,by, rtn);
463 gcpscad  }
```

4 Future

Images

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

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

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

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:

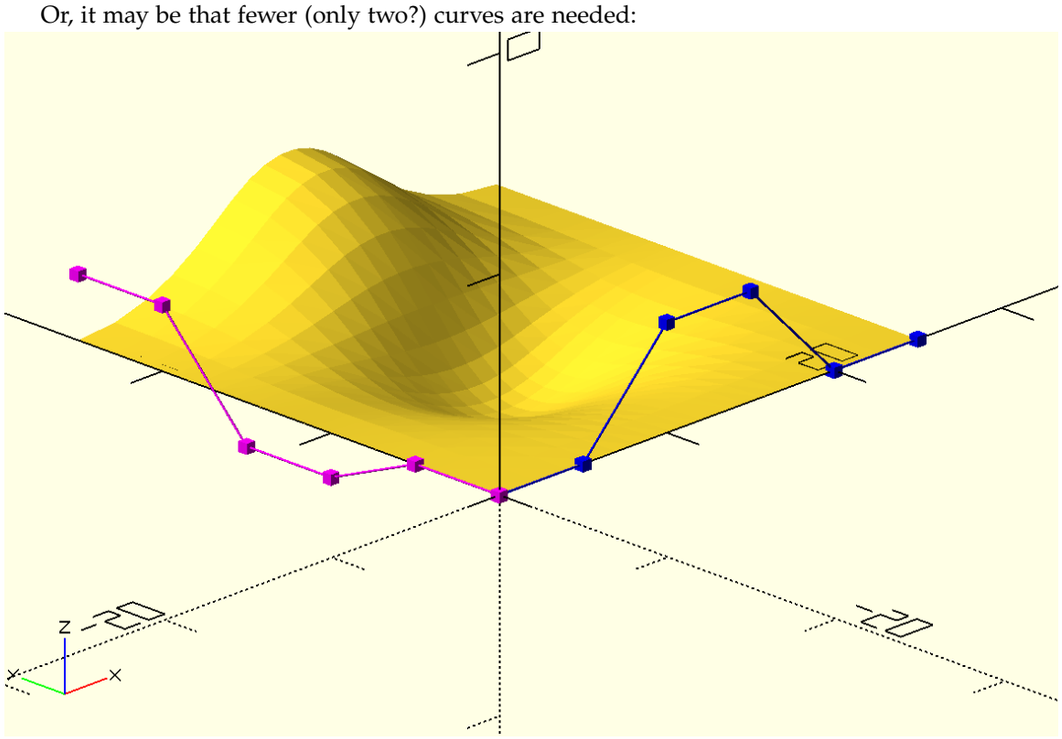
- XY
- XZ
- ZY

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/

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