The gcodepreview PythonSCAD library*

Author: William F. Adams willadams at aol dot com

2025/01/29

Abstract

The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output $\tt DXF$ and $\tt G$ -code files so as to work as a $\tt CAD/CAM$ program for $\tt CNC$.

Contents

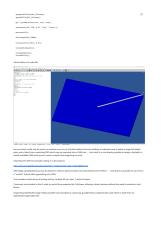
1	read	dme.md	2								
2	Usa	ge and Templates	5								
	2.1	gcpdxf.py	5								
	2.2	gcodepreviewtemplate.py	7								
	2.3	gcodepreviewtemplate.scad	12								
3	gcodepreview										
	3.1	Module Naming Convention	17								
		3.1.1 Parameters and Default Values	19								
	3.2	Implementation files and gcodepreview class	19								
		3.2.1 Position and Variables	22								
		3.2.2 Initial Modules	23								
	3.3	Tools and Changes	25								
		3.3.1 3D Shapes for Tools	26								
		3.3.1.1 Normal Tooling/toolshapes	26								
		3.3.1.2 Tooling for Undercutting Toolpaths	27								
		3.3.1.2.1 Keyhole tools	27								
		3.3.1.2.2 Thread mills	27								
		3.3.1.2.3 Dovetails	28								
		3.3.1.3 Concave toolshapes	28								
		3.3.1.4 Roundover tooling	28								
		3.3.2 toolchange	28								
		3.3.2.1 Selecting Tools	28								
		3.3.2.2 Square and ball nose (including tapered ball nose)	29								
		3.3.2.3 Roundover (corner rounding)	29								
		3.3.2.4 Dovetails	29								
		3.3.2.5 toolchange routine	29								
		3.3.3 tooldiameter	30								
		3.3.4 Feeds and Speeds	32								
	3.4	Movement and Cutting	32								
		3.4.1 Lines	34								
		3.4.2 Arcs for toolpaths and DXFs	35								
		3.4.3 Cutting shapes and expansion	39								
		3.4.3.1 Building blocks	39								
		3.4.3.2 List of shapes	39								
		3.4.3.2.1 circles	41								
			41								
			42								
		3.4.4 Difference of Stock, Rapids, and Toolpaths	49								
	3.5	Output files	49								
		3.5.1 G-code Overview	5C								
		3.5.2 DXF Overview	5C								
		3.5.3 Python and OpenSCAD File Handling	51								
		3.5.3.1 Writing to DXF files	54								
		3.5.3.2 Closings	59								
4	Not	tes	61								
Index											
ın		Hings	65								
		atines	66								
	varı	iables	67								

^{*}This file (gcodepreview) has version number vo.8, last revised 2025/01/29.

1 readme.md 2

1 readme.md







```
1 rdme # gcodepreview
 2 rdme
 3\ \mathrm{rdme}\ \mathrm{PythonSCAD} library for moving a tool in lines and arcs so as to
            model how a part would be cut using G\text{-}Code, so as to allow
            {\tt PythonSCAD} \ \ {\tt to} \ \ {\tt function} \ \ {\tt as} \ \ {\tt a} \ \ {\tt compleat} \ \ {\tt CAD/CAM} \ \ {\tt solution} \ \ {\tt for}
            subtractive 3-axis CNC (mills and routers at this time, 4\text{th-axis}
             support may come in a future version) by writing out G-code in
            addition to 3D modeling (in some cases toolpaths which would not
             normally be feasible), and to write out \widetilde{\text{DXF}} files which may be
            imported into a traditional CAM program to create toolpaths.
 4 rdme
 \texttt{5} \ \texttt{rdme} \ \texttt{![OpenSCAD} \ \texttt{gcodepreview} \ \texttt{Unit} \ \texttt{Tests](https://raw.githubusercontent.}
            com/WillAdams/gcodepreview/main/gcodepreview_unittests.png?raw=
 6 rdme
 7 rdme Updated to make use of Python in OpenSCAD:[^rapcad]
 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://willadams.gitbook.io/design-into-3d/programming
21 rdme
22 \; \mathrm{rdme} \; \mathrm{Since} \; \mathrm{it} \; \mathrm{is} \; \mathrm{now} \; \mathrm{programmed} \; \mathrm{using} \; \mathrm{Literate} \; \mathrm{Programming} \; \mathrm{(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.
23 rdme
24 rdme The files for this library are:
25 rdme
        - gcodepreview.py (gcpy) --- the Python functions and variables - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
26 rdme
27 rdme
28 rdme
        - gcodepreviewtemplate.scad (gcptmpl) --- .scad example file
         - gcodepreviewtemplate.py (gcptmplpy) --- .py example file (which
             requires PythonSCAD)
        - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf file(s) and which will run in "normal" Python
30 rdme
31 rdme
32 rdme If using from PythonSCAD, place the files in C:\Users\\\~\Documents \OpenSCAD\libraries [^libraries]
33 rdme
34 rdme [^libraries]: C:\Users\\\~\Documents\RapCAD\libraries is deprecated
            since RapCAD is no longer needed since Python is now used for
            writing out files.
35 rdme
36 rdme and call as:
37 rdme
38 rdme
            use <gcodepreview.py>
```

1 readme.md

```
39 rdme
           include <gcodepreview.scad>
40 rdme
41 rdme Note that it is necessary to use the first file (this allows
           loading the Python commands (it used to be necessary to use an
           intermediary .scad file so as to wrap 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 {\tt OpenSCAD} modules) and define variables which
           match the project and then use commands such as:
42 rdme
           opengcodefile(Gcode filename);
43 rdme
           opendxffile(DXF_filename);
44 rdme
45 rdme
           gcp = gcodepreview(true, true, true);
46 rdme
47 rdme
           setupstock(219, 150, 8.35, "Top", "Center");
48 rdme
49 rdme
50 rdme
           movetosafeZ();
51 rdme
           toolchange(102,17000);
52 rdme
53 rdme
           cutline (219/2, 150/2, -8.35);
54 rdme
55 rdme
56 rdme
           stockandtoolpaths();
57 rdme
58 rdme
           closegcodefile();
59 rdme
           closedxffile():
60 rdme
61 rdme which makes a G-code file:
62 rdme
63 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
           WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
65 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 % \left( \frac{1}{2}\right) =\frac{1}{2}\left( \frac{1}{2}\right) 
           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).
66 rdme
67 rdme Importing this DXF and actually cutting it is discussed at:
68 rdme
69 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
           /88617/14
71 rdme Alternately, gcodepreview.py may be placed in a Python library
          location and used directly from Python --- note that it is possible to use it from a "normal" Python when generating only
72 rdme
73 rdme Tool numbers match those of tooling sold by Carbide 3D (ob. discl.,
           I work for them).
75 \ \mathrm{rdme} Comments are included in the G-code to match those expected by
          {\tt CutViewer}, \ {\tt allowing} \ {\tt a} \ {\tt direct} \ {\tt preview} \ {\tt without} \ {\tt the} \ {\tt need} \ {\tt to}
           maintain a tool library.
76 rdme
77 rdme Supporting OpenSCAD usage makes possible such examples as:
          openscad_gcodepreview_cutjoinery.tres.scad which is made from an
           OpenSCAD Graph Editor file:
78 rdme
79 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
           githubusercontent.com/WillAdams/gcodepreview/main/
           OSGE_cutjoinery.png?raw=true)
80 rdme
                        | Notes
81 rdme | Version
82 rdme | ----- | ----- |
                       | Version supports setting up stock, origin, rapid
83 rdme | 0.1
           positioning, making cuts, and writing out matching G-code, and
           creating a DXF with polylines.
                           - separate dxf files are written out for each
84 rdme
           tool where tool is ball/square/V and small/large (10/31/23)
                       | - re-writing as a Literate Program using the
85 rdme
           LaTeX package docmfp (begun 4/12/24)
```

1 readme.md 4

```
| - support for additional tooling shapes such as
86 rdme
          dovetail and keyhole tools
                      | Adds support for arcs, specialty toolpaths such
87 rdme | 0.2
          as Keyhole which may be used for dovetail as well as keyhole
                      | Support for curves along the 3rd dimension,
88 rdme | 0.3
          roundover tooling
89 rdme | 0.4
                       | Rewrite using literati documentclass, suppression
           of SVG code, dxfrectangle
                      | More shapes, consolidate rectangles, arcs, and
90 rdme | 0.5
          circles in gcodepreview.scad
                      | Notes on modules, change file for setupstock
91 rdme | 0.6
                       \mid Validate all code so that it runs without errors
92 rdme | 0.61
          from sample (NEW: Note that this version is archived as
          gcodepreview-openscad_0_6.tex and the matching PDF is available
          as well|
93 rdme | 0.7
                       | Re-write completely in Python
                      | Re-re-write completely in Python and OpenSCAD,
94 rdme | 0.8
          iteratively testing
                       | Add support for bowl bits with flat bottom
95 rdme | 0.801
96 rdme
97 rdme Possible future improvements:
98 rdme
       - support for additional tooling shapes (tapered ball nose,
          lollipop cutters)
100 rdme
       - create a single line font for use where text is wanted
101 rdme
       - Support Bézier curves (required for fonts if not to be limited
           to lines and arcs) and surfaces
102 rdme
103 \operatorname{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 as
          noted above. Working from a DXF of course allows one to off-load
           such considerations to a specialized CAM tool.
104 rdme
105 rdme Deprecated feature:
106 rdme
        - exporting SVGs --- coordinate system differences between
107 rdme
           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)
108 rdme
109 rdme To-do:
110 rdme
111 \operatorname{rdme} - fix OpenSCAD wrapper and add any missing commands for Python
112 rdme - reposition cutroundover command into cutshape
113 rdme - work on rotary axis option
```

2 Usage and Templates

The gcodepreview library allows the modeling of 2D geometry and 3D shapes using Python or by calling Python from within (Open)PythonSCAD, enabling the creation of 2D DXFS, G-code, or 3D models as a preview of how the file will cut. These abilities may be accessed in "plain" Python (to make DXFS), or Python or OpenSCAD in PythonSCAD (to make G-code and/or for 3D modeling). Providing them in a programmatic context allows making parts or design elements of parts (e.g., joinery) which would be tedious to draw by hand in a traditional CAD or vector drawing application. A further consideration is that this is "Design for Manufacture" taken to its ultimate extreme, and that a part so designed is inherently manufacturable.

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 (note that the sparse template in readme.md eschews variables), and that files are opened before being written to, and that each is closed at the end in the correct order. 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 for a given tool at zero (o).

It should be that the readme at the project page which serves as an overview, and this section (which serves as a tutorial) is all the documentation which most users will need (and arguably is still too much). The balance of the document after this section shows all the code and implementation details, and will where appropriate show examples of usage excerpted from the template files (serving as a how-to guide as well as documenting the code) as well as Indices (which serve as a front-end for reference).



Some comments on the templates:

- minimal each is intended as a framework for a minimal working example (MWE) it should be possible to comment out unused/unneeded 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 mechanism to handle everything.
- shortcuts as the various examples show, 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 and implementing Depth per Pass (but note that this will lose the previewing of scalloped tool marks in places where they might appear otherwise)

One fundamental aspect of this tool is the question of *Layers of Abstraction* (as put forward by Dr. Donald Knuth as the crux of computer science) and *Problem Decomposition* (Prof. John Ousterhout's answer to that topic). To a great degree, the basic implementation of this tool will use G-code as a reference implementation, simultaneously using the abstraction from the mechanical task of machining which it affords as a decomposed version of that task, and creating what is in essence, both a front-end, and a tool, and an API for working with G-code programmatically. This then requires an architecture which allows 3D modeling (OpenSCAD), and writing out files (Python).

Further features will be added to the templates as they are created, and the main image updated to reflect the capabilities of the system.

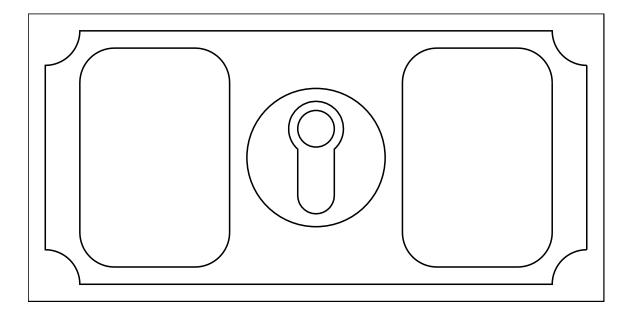
2.1 gcpdxf.py

The most basic usage, with the fewest dependencies is to use "plain" Python to create dxf files. Note that this example includes an optional command (openscad.)nimport(<URL>) which if enabled/uncommented (and the following line commented out), will import the library from Github, sidestepping the need to download and install the library.

```
9 gcpdxfpy
10 gcpdxfpy # [Stock] */
11 gcpdxfpy stockXwidth = 100
12 gcpdxfpy # [Stock] */
13 gcpdxfpy stockYheight = 50
14 gcpdxfpy
15 gcpdxfpy # [Export] */
16 gcpdxfpy Base_filename = "dxfexport"
17 gcpdxfpy
18 gcpdxfpy
19 gcpdxfpy # [CAM] */
20 gcpdxfpy large_square_tool_num = 102
21 gcpdxfpy # [CAM] */
22 gcpdxfpy small_square_tool_num = 0
23 gcpdxfpy # [CAM] */
24 gcpdxfpy large_ball_tool_num = 0
25 gcpdxfpy # [CAM] */
26 gcpdxfpy small_ball_tool_num = 0
27 gcpdxfpy # [CAM] */
28 gcpdxfpy large_V_tool_num = 0
29 gcpdxfpy # [CAM] */
30 gcpdxfpy small_V_tool_num = 0
31 gcpdxfpy # [CAM] */
32 \text{ gcpdxfpy } DT_{tool_num} = 374
33 gcpdxfpy # [CAM] */
34 gcpdxfpy KH_tool_num = 0
35 gcpdxfpy # [CAM] */
36 gcpdxfpy Roundover_tool_num = 0
37 gcpdxfpy # [CAM] */
38 gcpdxfpy MISC_tool_num = 0
39 gcpdxfpy
40 gcpdxfpy # [Design] */
41 gcpdxfpy inset = 3
42 gcpdxfpy # [Design] */
43 \text{ gcpdxfpy radius} = 6
44 gcpdxfpy # [Design] */
45 gcpdxfpy cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
46 gcpdxfpy
47 gcpdxfpy gcp.opendxffile(Base_filename)
48 gcpdxfpy #gcp.opendxffiles(Base_filename,
49 gcpdxfpy #
                              large_square_tool_num,
50 gcpdxfpy #
                              small_square_tool_num,
51 gcpdxfpy #
                              large\_ball\_tool\_num,
                              small_ball_tool_num,
52 gcpdxfpy #
53 gcpdxfpy #
                             large_V_tool_num,
                              small_V_tool_num,
54 gcpdxfpy #
                             DT\_tool\_num,
55 gcpdxfpy #
56 gcpdxfpy #
                              KH tool num,
57 gcpdxfpy #
                              Roundover_tool_num,
                             MISC_tool_num)
58 gcpdxfpy #
59 gcpdxfpy
60 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, 0, 0, stockXwidth,
             stockYheight)
61 gcpdxfpv
62 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, inset, radius,
                                                                         0, 90)
63 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, inset,
             radius, 90, 180)
64 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
               - inset, radius, 180, 270)
65 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
             radius, 270, 360)
66 gcpdxfpy
67 gcpdxfpy gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
             stockYheight - (inset + radius))
68 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, inset,
              stockXwidth - (inset + radius), inset)
69 gcpdxfpy gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
             radius,stockXwidth - inset,stockYheight - (inset + radius))
70 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, stockYheight-
             inset,stockXwidth - (inset + radius),stockYheight - inset)
72 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, radius +inset, radius,
              stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
              cornerstyle, radius)
73 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
              2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
              (radius * 2), cornerstyle, radius)
74 gcpdxfpy #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius
```

```
)
75 gcpdxfpy #gcp.dxfrectanglechamfer(large_square_tool_num, 64, 7, 24, 36, radius)
76 gcpdxfpy #gcp.dxfrectangleflippedfillet(large_square_tool_num, 64, 7, 24, 36, radius)
77 gcpdxfpy
78 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2, radius * 2)
79 gcpdxfpy
80 gcpdxfpy gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270, 11.5875)
81 gcpdxfpy #gcp.closedxffiles()
83 gcpdxfpy gcp.closedxffile()
```

which creates:



and which may be imported into pretty much any CAD or CAM application. Note that the lines referencing multiple files (open/closedxffiles) may be uncommented if the project wants separate dxf files for different tools.

As shown/implied by the above code, the following commands/shapes are implemented:

- dxfrectangle (specify lower-left and upper-right corners)

 dxfrectangleround (specified as "Fillet" and radius for the round option)

 dxfrectanglechamfer (specified as "Chamfer" and radius for the round option)

 dxfrectangleflippedfillet (specified as "Flipped Fillet" and radius for the option)
- dxfcircle (specifying their center and radius)
- dxfline (specifying begin/end points)
- dxfarc (specifying arc center, radius, and beginning/ending angles)
- dxfKH (specifying origin, depth, angle, distance)

2.2 gcodepreviewtemplate.py

Note that since the vo.7 re-write, it is possible to directly use the underlying Python code. Using Python to generate 3D previews of how DXFS or G-code will cut requires the use of PythonSCAD.

```
1 gcptmplpy #!/usr/bin/env python
2 gcptmplpy
3 gcptmplpy import sys
4 gcptmplpy
5 gcptmplpy \mathtt{try}:
              if 'gcodepreview' in sys.modules:
6 gcptmplpy
                     del sys.modules['gcodepreview']
7 gcptmplpy
8 gcptmplpy except AttributeError:
9 gcptmplpy
                pass
10 gcptmplpy
11 gcptmplpy from gcodepreview import *
12 gcptmplpy
13 gcptmplpy fa = 2
14 gcptmplpy fs = 0.125
```

```
15 gcptmplpy
16 gcptmplpy # [Export] */
17 gcptmplpy Base_filename = "aexport"
18 gcptmplpy # [Export] */
19 gcptmplpy generatepaths = False
20 gcptmplpy # [Export] */
21 gcptmplpy generatedxf = True
22 gcptmplpy # [Export] */
23 gcptmplpy generategcode = True
24 gcptmplpy
25 gcptmplpy # [Stock] */
26 gcptmplpy stockXwidth = 220
27 gcptmplpy # [Stock] */
28 gcptmplpy stockYheight = 150
29 gcptmplpy # [Stock] */
30 gcptmplpy stockZthickness = 8.35
31 gcptmplpy # [Stock] */
32 gcptmplpy zeroheight = "Top" # [Top, Bottom]
33 gcptmplpy # [Stock] */
34 gcptmplpy stockzero = "Center" # [Lower-Left, Center-Left, Top-Left, Center]
35 gcptmplpy # [Stock] */
36 gcptmplpy retractheight = 9
37 gcptmplpy
38 gcptmplpy # [CAM] */
39 gcptmplpy toolradius = 1.5875
40 gcptmplpy # [CAM] */
41 gcptmplpy large_square_tool_num = 201 # [0:0,112:112,102:102,201:201]
42 gcptmplpy # [CAM] */
43 gcptmplpy small_square_tool_num = 102 # [0:0,122:122,112:112,102:102]
44 gcptmplpy # [CAM] */
45 gcptmplpy large_ball_tool_num = 202 # [0:0,111:111,101:101,202:202]
46 gcptmplpy # [CAM] */
47 gcptmplpy small_ball_tool_num = 101 # [0:0,121:121,111:111,101:101]
48 gcptmplpy # [CAM] */
49 gcptmplpy large_V_tool_num = 301 # [0:0,301:301,690:690]
50 gcptmplpy # [CAM] */
51 gcptmplpy small_V_tool_num = 390 # [0:0,390:390,301:301]
52 gcptmplpy # [CAM] */
53 gcptmplpy DT_tool_num = 814 # [0:0,814:814]
54 gcptmplpy # [CAM] */
55 gcptmplpy KH_tool_num = 374 # [0:0,374:374,375:375,376:376,378:378]
56 gcptmplpy # [CAM] */
57 gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 1570:1570]
58 gcptmplpy # [CAM] */
59 gcptmplpy MISC_tool_num = 0 # [648:648, 45982:45982]
60 gcptmplpy #648 threadmill_shaft(2.4, 0.75, 18)
61 gcptmplpy #45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
               Inch Shank
62 gcptmplpy
63 gcptmplpy # [Feeds and Speeds] */
64 gcptmplpy plunge = 100
65 gcptmplpy # [Feeds and Speeds] */
66 gcptmplpy feed = 400
67 gcptmplpy # [Feeds and Speeds] */
68 gcptmplpy speed = 16000
69 gcptmplpy # [Feeds and Speeds] */
70 gcptmplpy small_square_ratio = 0.75 # [0.25:2] 71 gcptmplpy # [Feeds and Speeds] */
72 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
73 gcptmplpy # [Feeds and Speeds] */
74 gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
75 gcptmplpy # [Feeds and Speeds] */
76 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
77 gcptmplpy # [Feeds and Speeds] */
78 gcptmplpy small_V_ratio = 0.625 \# [0.25:2]
79 gcptmplpy # [Feeds and Speeds] */
80 gcptmplpy DT_ratio = 0.75 \# [0.25:2]
81 gcptmplpy # [Feeds and Speeds] */
82 gcptmplpy KH_ratio = 0.75 \# [0.25:2]
83 gcptmplpy # [Feeds and Speeds] */
84 gcptmplpy RO_ratio = 0.5 \# [0.25:2]
85 gcptmplpy # [Feeds and Speeds] */
86 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
87 gcptmplpy
88 gcptmplpy gcp = gcodepreview(generatepaths,
89 gcptmplpy
                                generategcode,
                                generatedxf,
90 gcptmplpy
91 gcptmplpy
```

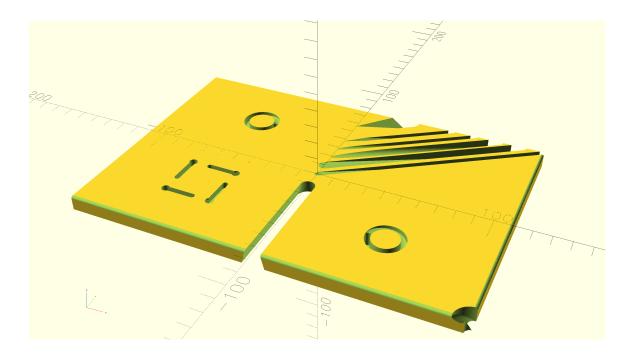
```
92 gcptmplpy
 93 gcptmplpy gcp.opengcodefile(Base_filename)
 94 gcptmplpy gcp.opendxffile(Base_filename)
95 gcptmplpy gcp.opendxffiles(Base_filename,
96 gcptmplpy
                                large_square_tool_num,
 97 gcptmplpy
                                small_square_tool_num,
                                large_ball_tool_num,
98 gcptmplpy
                                small_ball_tool_num,
99 gcptmplpy
100 gcptmplpy
                               large_V_tool_num,
                               small_V_tool_num,
101 gcptmplpy
                                DT_tool_num,
102 gcptmplpy
103 gcptmplpy
                                KH tool num,
                                Roundover_tool_num,
104 gcptmplpy
                                MISC_tool_num)
105 gcptmplpy
106~{\tt gcptmplpy}~{\tt gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top","}
                Center", retractheight)
107 gcptmplpy
108 gcptmplpy #print(pygcpversion())
109 gcptmplpy
110 gcptmplpy #print(gcp.myfunc(4))
111 gcptmplpy
112 gcptmplpy #print(gcp.getvv())
113 gcptmplpy
114 gcptmplpy \#ts = cylinder(12.7, 1.5875, 1.5875)
115 gcptmplpy #toolpaths = gcp.cutshape(stockXwidth/2,stockYheight/2,-
                stockZthickness)
116 gcptmplpy
117 gcptmplpy gcp.movetosafeZ()
118 gcptmplpy
119 gcptmplpy gcp.toolchange(102,10000)
120 gcptmplpy
121 gcptmplpy \#gcp.rapidXY(6,12)
122 gcptmplpy gcp.rapidZ(0)
123 gcptmplpy
124 gcptmplpy \#print (gcp.xpos())
125 gcptmplpy #print (gcp.ypos())
126 gcptmplpy #psetzpos(7)
127 gcptmplpy \#gcp.setzpos(-12)
128 gcptmplpy #print (gcp.zpos())
129 gcptmplpy
130 gcptmplpy #print ("X", str(gcp.xpos()))
131 gcptmplpy #print ("Y", str(gcp.ypos()))
132 gcptmplpy #print ("Z", str(gcp.zpos()))
133 gcptmplpy
134 gcptmplpy toolpaths = gcp.currenttool()
135 gcptmplpy
136 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2,stockYheight/2,-
               stockZthickness)
137 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
                stockYheight/2, -stockZthickness))
138 gcptmplpy
139 gcptmplpy gcp.rapidZ(retractheight)
140 gcptmplpy gcp.toolchange(201,10000)
141 gcptmplpy gcp.rapidXY(0, stockYheight/16)
142 gcptmplpy gcp.rapidZ(0)
143 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
                stockYheight/2, -stockZthickness))
144 gcptmplpy
145 gcptmplpy gcp.rapidZ(retractheight)
146 gcptmplpy gcp.toolchange(202,10000)
147 gcptmplpy gcp.rapidXY(0, stockYheight/8)
148 gcptmplpy gcp.rapidZ(0)
149 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
                stockYheight/2, -stockZthickness))
150 gcptmplpy
151 gcptmplpy gcp.rapidZ(retractheight)
152 gcptmplpy gcp.toolchange(101,10000)
153 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
154 gcptmplpy gcp.rapidZ(0)
155 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
                stockYheight/2, -stockZthickness))
156 gcptmplpy
157 gcptmplpy gcp.setzpos(retractheight)
158 gcptmplpy gcp.toolchange(390,10000)
159 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
160 gcptmplpy gcp.rapidZ(0)
161 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
                stockYheight/2, -stockZthickness))
```

```
162 gcptmplpy gcp.rapidZ(retractheight)
163 gcptmplpy
164 gcptmplpy gcp.toolchange(301,10000)
165 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
166 gcptmplpy gcp.rapidZ(0)
167 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*2,
               stockYheight/2, -stockZthickness))
168 gcptmplpy
169 gcptmplpy rapids = gcp.rapid(gcp.xpos(),gcp.ypos(),retractheight)
170 gcptmplpy gcp.toolchange(102,10000)
171 gcptmplpy
172 \hspace{0.1cm} \texttt{gcptmplpy rapids = gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight/16)} \\
               /4,0)
173 gcptmplpy
174 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(0,90, gcp.xpos()-
               stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
175 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(90,180, gcp.xpos(), gcp.
               ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
176 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(180,270, gcp.xpos()+
               stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
177 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(270,360, gcp.xpos(), gcp.
               ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
178 gcptmplpy
179 gcptmplpy rapids = gcp.movetosafeZ()
180 \hspace{0.1cm} \texttt{gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight/16)} \\
               /4)
181 gcptmplpy rapids = gcp.rapidZ(0)
182 gcptmplpy
183 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180,90, gcp.xpos()+
               stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
184 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(90,0, gcp.xpos(), gcp.ypos
                ()-stockYheight/16, stockYheight/16, -stockZthickness/4))
185 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(360,270, gcp.xpos()-
               stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
               /4))
186 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(270,180, gcp.xpos(), gcp.
               ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
187 gcptmplpy
188 gcptmplpy rapids = gcp.movetosafeZ()
189 gcptmplpy gcp.toolchange(201,10000)
190 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2, -stockYheight/2)
191 gcptmplpy rapids = gcp.rapidZ(0)
192 gcptmplpy
193 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
               , -stockZthickness))
194 \ \texttt{gcptmplpy} \ \texttt{\#test} \ = \ \texttt{gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)}
195 gcptmplpy
196 gcptmplpy rapids = gcp.movetosafeZ()
197 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
198 gcptmplpy rapids = gcp.rapidZ(0)
199 gcptmplpy
200 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180,90, stockXwidth/2, -
               stockYheight/2, 6.34, -stockZthickness))
201 gcptmplpy
202 gcptmplpy rapids = gcp.movetosafeZ()
203 gcptmplpy gcp.toolchange(814,10000)
204 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
205 gcptmplpy rapids = gcp.rapidZ(0)
206 gcptmplpy
207 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
               , -stockZthickness))
208 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -12.7, -
               stockZthickness))
209 gcptmplpy
210 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
211 gcptmplpy rapids = gcp.movetosafeZ()
212 gcptmplpy gcp.toolchange(374,10000)
213 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight
               /4+stockYheight/16))
214 gcptmplpy rapids = gcp.rapidZ(0)
216 gcptmplpy gcp.rapidZ(retractheight)
217 gcptmplpy gcp.toolchange(374,10000)
218 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+ ^{\prime}
               stockYheight/16))
```

```
219 gcptmplpy gcp.rapidZ(0)
220 gcptmplpy
221 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                                                                      stockZthickness/2))
222 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()+
                                                                      stockYheight/9, gcp.ypos(), gcp.zpos()))
223 gcptmplpy #below should probably be cutlinegc
{\tt 224~gcptmplpy~toolpaths~=~toolpaths.union(gcp.cutline(gcp.xpos()-stockYheight/9,}
                                                                       gcp.ypos(), gcp.zpos()))
225 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
226 gcptmplpv
227 gcptmplpy \#key = gcp.cutkeyholegcdxf(KH\_tool\_num, 0, stockZthickness*0.75, "E" and "E" and "E" and "E" are stock as a stock of the stock of th
                                                                         ", stockYheight/9)
228 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
                                                                     stockYheight/9)
229 gcptmplpy #toolpaths = toolpaths.union(key)
230 gcptmplpy
231 gcptmplpy gcp.rapidZ(retractheight)
 232 \ \texttt{gcptmplpy} \ \texttt{gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+stockXwidth/16, -(stockYwidth/16, -(stockYheight/4+stockXwidth/16, -(stockYheight/4+stockXwidth/16, -(stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+stockYheight/4+s
                                                                       stockYheight/16))
233 gcptmplpy gcp.rapidZ(0)
234 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                                                                     stockZthickness*0.75, "N", stockYheight/9))
235 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                                                                       stockZthickness/2))
236 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                                                                      +stockYheight/9, gcp.zpos()))
237 gcptmplpy #below should probably be cutlinegc
238 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()-
                                                                      stockYheight/9, gcp.zpos()))
239 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
240 gcptmplpy
241 gcptmplpy gcp.rapidZ(retractheight)
242 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-stockXwidth/16), -(stockXwidth/16), -(stockXwidth/16), -(stockXwidt
                                                                       stockYheight/8))
243 gcptmplpy gcp.rapidZ(0)
 244 \ \texttt{gcptmplpy} \ \textit{\#toolpaths} \ = \ \textit{toolpaths.union} (\textit{gcp.cutkeyholegcdxf} (\textit{KH\_tool\_num} \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , \ 0
                                                                       stockZthickness*0.75, "W", stockYheight/9))
245 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                                                                       stockZthickness/2))
246 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()-
                                                                      stockYheight/9, gcp.ypos(), gcp.zpos()))
247 gcptmplpy #below should probably be cutlinegc
248 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()+stockYheight/9,
                                                                      gcp.ypos(), gcp.zpos()))
249 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
250 gcptmplpy
251 gcptmplpy gcp.rapidZ(retractheight)
252~{\tt gcptmplpy}~{\tt gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-stockXwidth/16, -(stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYheight/4-stockYh
                                                                     stockYheight/8))
253 gcptmplpy gcp.rapidZ(0)
254 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                                                                      stockZthickness*0.75, "S", stockYheight/9))
255 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                                                                       stockZthickness/2))
256 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                                                                        -stockYheight/9, gcp.zpos()))
257 gcptmplpy #below should probably be cutlinegc
258 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()+
                                                                      stockYheight/9, gcp.zpos()))
259 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
260 gcptmplpy
261 gcptmplpy gcp.rapidZ(retractheight)
262 gcptmplpy gcp.toolchange(56142,10000)
263 gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
264 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
265 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                                                                         -1.531))
266 \ \texttt{gcptmplpy toolpaths} \ = \ \texttt{toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2, note that the state of the 
                                                                            -(stockYheight/2+0.508/2), -1.531))
267 gcptmplpy
268 gcptmplpy gcp.rapidZ(retractheight)
269 gcptmplpy #gcp.toolchange(56125,10000)
270 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
271 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                                                                         -1.531))
272 \ \texttt{gcptmplpy toolpaths} \ = \ \texttt{toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2, note that the property of the p
                                                                            (stockYheight/2+0.508/2), -1.531))
```

```
274 gcptmplpy gcp.rapidZ(retractheight)
275 gcptmplpy gcp.toolchange(45982, 10000)
276 gcptmplpy gcp.rapidXY(stockXwidth/8,0)
277 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -(
                stockZthickness*7/8)))
278 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(),-
                stockYheight/2, -(stockZthickness*7/8)))
279 gcptmplpy
280 gcptmplpy gcp.rapidZ(retractheight)
281 gcptmplpy
282 gcptmplpy part = gcp.stock.difference(toolpaths)
283 gcptmplpy
284 gcptmplpy output (part)
285 gcptmplpy #output(test)
286 gcptmplpy #output (key)
287 gcptmplpy #output(dt)
288 gcptmplpy #gcp.stockandtoolpaths()
289 gcptmplpy \#gcp.stockandtoolpaths("stock")
290 gcptmplpy #output (gcp.stock)
291 gcptmplpy #output (gcp.toolpaths)
292 gcptmplpy #output (toolpaths)
293 gcptmplpy
294 gcptmplpy \#gcp.makecube(3, 2, 1)
295 gcptmplpy #
296 gcptmplpy #gcp.placecube()
297 gcptmplpy #
298 gcptmplpy \#c = gcp.instantiatecube()
299 gcptmplpy #
300 gcptmplpy #output(c)
301 gcptmplpy
302 gcptmplpy gcp.closegcodefile()
303 gcptmplpy gcp.closedxffiles()
304 gcptmplpy gcp.closedxffile()
```

Which generates a 3D model which previews in PythonSCAD as:



2.3 gcodepreviewtemplate.scad

Since the project began in OpenSCAD, having an implementation in that language has always been a goal. This is quite straight-forward since the Python code when imported into OpenSCAD may be accessed by quite simple modules which are for the most part, a series of decorators/descriptors which wrap up the Python definitions as OpenSCAD modules. Moreover, such an implementation will facilitate usage by tools intended for this application such as OpenSCAD Graph Editor: https://github.com/derkork/openscad-graph-editor. A further consideration worth noting is that when called from OpenSCAD, Python will not halt for errors, but will run through to the end which is an expedient thing for viewing the end result of in-process code.

```
1 gcptmpl //!OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>
```

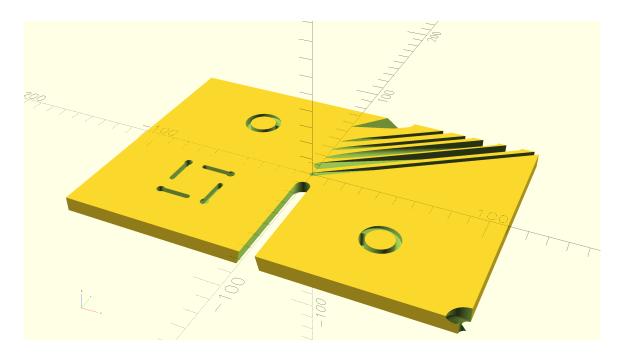
```
4 gcptmpl include <gcodepreview.scad>
5 gcptmpl
6 gcptmpl fa = 2;
7 \text{ gcptmpl } \$fs = 0.125;
8 gcptmpl fa = 2;
9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 219;
13 gcptmpl /* [Stock] */
14 gcptmpl stockYheight = 150;
15 gcptmpl /* [Stock] */
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
25 gcptmpl Base_filename = "export";
26 gcptmpl /* [Export] */
27 gcptmpl generatepaths = true;
28 gcptmpl /* [Export] */
29 gcptmpl generatedxf = true;
30 gcptmpl /* [Export] */
31 gcptmpl generategcode = true;
32 gcptmpl
33 gcptmpl /* [CAM] */
34 gcptmpl toolradius = 1.5875;
35 gcptmpl /* [CAM] */
36 gcptmpl large_square_tool_num = 0; // [0:0,112:112,102:102,201:201]
37 gcptmpl /* [CAM] */
38 gcptmpl small_square_tool_num = 102; // [0:0,122:122,112:112,102:102]
39 gcptmpl /* [CAM] */
40 gcptmpl large_ball_tool_num = 0; // [0:0,111:111,101:101,202:202]
41 gcptmpl /* [CAM] */
42 gcptmpl small_ball_tool_num = 0; // [0:0,121:121,111:111,101:101]
43 gcptmpl /* [CAM] */
44 gcptmpl large_V_tool_num = 0; // [0:0,301:301,690:690]
45 gcptmpl /* [CAM] */
46 gcptmpl small_V_tool_num = 0; // [0:0,390:390,301:301]
47 gcptmpl /* [CAM] */
48 gcptmpl DT_tool_num = 0; // [0:0,814:814]
49 gcptmpl /* [CAM] */
50 gcptmpl KH_tool_num = 0; // [0:0,374:374,375:375,376:376,378:378]
51 gcptmpl /* [CAM] */
52 \text{ gcptmpl Roundover\_tool\_num} = 0; // [56142:56142, 56125:56125, 1570:1570]
53 gcptmpl /* [CAM] */
54 gcptmpl MISC_tool_num = 0; // [648:648, 45982:45982]
55 gcptmpl //648 threadmill_shaft(2.4, 0.75, 18)
56 gcptmpl //45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
              Inch Shank
57 gcptmpl
58 gcptmpl /* [Feeds and Speeds] */
59 gcptmpl plunge = 100;
60 gcptmpl /* [Feeds and Speeds] */
61 gcptmpl feed = 400;
62 gcptmpl /* [Feeds and Speeds] */
63 gcptmpl speed = 16000;
64 gcptmpl /* [Feeds and Speeds] */
65 gcptmpl small_square_ratio = 0.75; // [0.25:2]
66 gcptmpl /* [Feeds and Speeds] */
67 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
68 gcptmpl /* [Feeds and Speeds] */
69 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
70 gcptmpl /* [Feeds and Speeds] */
71 gcptmpl large_V_ratio = 0.875; // [0.25:2]
72 gcptmpl /* [Feeds and Speeds] */
73 gcptmpl small_V_ratio = 0.625; // [0.25:2]
74 gcptmpl /* [Feeds and Speeds] */
75 gcptmpl DT_ratio = 0.75; // [0.25:2]
76 gcptmpl /* [Feeds and Speeds] */
77 gcptmpl KH_ratio = 0.75; // [0.25:2]
78 gcptmpl /* [Feeds and Speeds] */
79 gcptmpl RO_ratio = 0.5; // [0.25:2]
```

```
80 gcptmpl /* [Feeds and Speeds] */
81 gcptmpl MISC_ratio = 0.5; // [0.25:2]
82 gcptmpl
83 gcptmpl thegeneratepaths = generatepaths == true ? 1 : 0;
84 gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
85 gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
86 gcptmpl
87 gcptmpl gcp = gcodepreview(thegeneratepaths,
88 gcptmpl
                                thegenerategcode,
89 gcptmpl
                                thegeneratedxf,
                                );
90 gcptmpl
91 gcptmpl
92 gcptmpl opengcodefile(Base_filename);
93 gcptmpl opendxffile(Base_filename);
94 gcptmpl opendxffiles(Base_filename,
95 gcptmpl
                              large_square_tool_num,
96 gcptmpl
                              small_square_tool_num,
97 gcptmpl
                              large_ball_tool_num,
                              small_ball_tool_num,
98 gcptmpl
99 gcptmpl
                              large_V_tool_num,
                              small_V_tool_num,
100 gcptmpl
101 gcptmpl
                              DT_tool_num,
                              KH tool_num,
102 gcptmpl
103 gcptmpl
                              Roundover_tool_num,
                              MISC_tool_num);
104 gcptmpl
105 gcptmpl
106 \ \texttt{gcptmpl} \ \texttt{setupstock} \\ (\texttt{stockXwidth} \ , \ \texttt{stockYheight} \ , \ \texttt{stockZthickness} \ , \ \texttt{zeroheight} \ , \\ \\
              stockzero);
107 gcptmpl
108 gcptmpl //echo(gcp);
109 gcptmpl //gcpversion();
110 gcptmpl
111 gcptmpl //c = myfunc(4);
112 gcptmpl //echo(c);
113 gcptmpl
114 gcptmpl //echo(getvv());
115 gcptmpl
116 gcptmpl cutline(stockXwidth/2,stockYheight/2,-stockZthickness);
117 gcptmpl
118 gcptmpl rapidZ(retractheight);
119 gcptmpl toolchange(201,10000);
120 gcptmpl rapidXY(0, stockYheight/16);
121 gcptmpl rapidZ(0);
122 gcptmpl cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness);
123 gcptmpl
124 gcptmpl
125 gcptmpl rapidZ(retractheight);
126 gcptmpl toolchange(202,10000);
127 gcptmpl rapidXY(0, stockYheight/8);
128 gcptmpl rapidZ(0);
129 gcptmpl cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
130 gcptmpl
131 gcptmpl rapidZ(retractheight);
132 gcptmpl toolchange(101,10000);
133 gcptmpl rapidXY(0, stockYheight/16*3);
134 gcptmpl rapidZ(0);
135 gcptmpl cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
136 gcptmpl
137 gcptmpl rapidZ(retractheight);
138 gcptmpl toolchange(390,10000);
139 gcptmpl rapidXY(0, stockYheight/16*4);
140 gcptmpl rapidZ(0);
141 gcptmpl
142 gcptmpl cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
143 gcptmpl rapidZ(retractheight);
144 gcptmpl
145 gcptmpl toolchange (301,10000);
146 gcptmpl rapidXY(0, stockYheight/16*6);
147 gcptmpl rapidZ(0);
148 gcptmpl
149 gcptmpl cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
150 gcptmpl
151 gcptmpl
152 gcptmpl movetosafeZ();
153 gcptmpl rapid(gcp.xpos(),gcp.ypos(),retractheight);
154 gcptmpl toolchange(102,10000);
155 gcptmpl
156 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
```

```
stockYheight/8);
157 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
             stockYheight/16
158 gcptmpl rapidZ(0);
159 gcptmpl
160 gcptmpl //cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
             stockYheight/16,-stockZthickness);
161 gcptmpl //gcp.cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16))
162 gcptmpl cutarcCC(0,90, gcp.xpos()-stockYheight/16, gcp.ypos(), stockYheight
             /16, -stockZthickness/4);
163 gcptmpl cutarcCC(90,180, gcp.xpos(), gcp.ypos()-stockYheight/16,
             stockYheight/16, -stockZthickness/4);
164 gcptmpl cutarcCC(180,270, gcp.xpos()+stockYheight/16, gcp.ypos(),
             stockYheight/16, -stockZthickness/4);
165 gcptmpl cutarcCC(270,360, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
167 gcptmpl movetosafeZ();
168 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
              stockYheight/8);
169 gcptmpl rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
170 gcptmpl rapidZ(0);
171 gcptmpl
172 gcptmpl cutarcCW(180,90, gcp.xpos()+stockYheight/16, gcp.ypos(),
             stockYheight/16, -stockZthickness/4);
173 \; \texttt{gcptmpl} \; \; \texttt{cutarcCW} \; (90,0, \; \texttt{gcp.xpos}(), \; \texttt{gcp.ypos}() - \texttt{stockYheight} / 16, \; \texttt{stockYheight} )
             /16, -stockZthickness/4);
174 gcptmpl cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
             stockYheight/16, -stockZthickness/4);
175 gcptmpl cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
176 gcptmpl
177 gcptmpl movetosafeZ();
178 gcptmpl toolchange(201, 10000);
179 gcptmpl rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
180 gcptmpl rapidZ(0);
181 gcptmpl cutarcCW(180, 90, stockXwidth /2, -stockYheight/2, 6.34, -
             stockZthickness);
182 gcptmpl
183 gcptmpl movetosafeZ();
184 gcptmpl rapidXY(stockXwidth/2, -stockYheight/2);
185 gcptmpl rapidZ(0);
186 gcptmpl
187 gcptmpl gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);
188 gcptmpl
189 gcptmpl movetosafeZ();
190 gcptmpl toolchange(814, 10000);
191 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
192 gcptmpl rapidZ(0);
193 gcptmpl
194 gcptmpl cutlinedxfgc(xpos(), ypos(), -stockZthickness);
195 gcptmpl cutlinedxfgc(xpos(), -12.7 , -stockZthickness);
196 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
197 gcptmpl
198 gcptmpl //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
199 gcptmpl //rapidZ(0);
200 gcptmpl
201 gcptmpl //movetosafeZ();
202 gcptmpl //toolchange(374, 10000);
203 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 + ^{\prime}
              stockYheight/16))
204 gcptmpl
205 gcptmpl //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
206 gcptmpl //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
207 gcptmpl //cutline(xpos() - stockYheight /9, ypos(), zpos());
208 gcptmpl //cutline(xpos(), ypos(), 0);
209 gcptmpl
210 gcptmpl movetosafeZ();
211 gcptmpl
212 gcptmpl toolchange(374, 10000);
213 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+ ^{\prime}
              stockYheight/16))
214 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
              stockYheight/16))
215 gcptmpl rapidZ(0);
216 gcptmpl
217 gcptmpl cutline(xpos(), ypos(), (stockZthickness/2) * -1);
```

```
218 gcptmpl cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
219 gcptmpl cutline(xpos() - stockYheight /9, ypos(), zpos());
220 gcptmpl cutline(xpos(), ypos(), 0);
221 gcptmpl
222 gcptmpl rapidZ(retractheight);
223 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
             stockYheight/16));
224 gcptmpl rapidZ(0);
225 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
226 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
227 gcptmpl cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
228 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
230 gcptmpl rapidZ(retractheight);
231 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
              stockYheight/8));
232 gcptmpl rapidZ(0);
233 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
234 gcptmpl cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos());
235 gcptmpl cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos());
236 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
237 gcptmpl
238 gcptmpl rapidZ(retractheight);
239 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-stockXwidth/16)
              stockYheight/8));
240 gcptmpl rapidZ(0);
241 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
242 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
243 gcptmpl cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos()); 244 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
245 gcptmpl
246 gcptmpl
247 gcptmpl
248 gcptmpl rapidZ(retractheight);
249 gcptmpl gcp.toolchange(56142,10000);
250 gcptmpl gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2));
251 gcptmpl cutZgcfeed(-1.531,plunge);
252 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);
253 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
              -1.531);
254 gcptmpl
255 gcptmpl rapidZ(retractheight);
256 gcptmpl //#gcp.toolchange(56125,10000)
257 gcptmpl cutZgcfeed(-1.531,plunge);
258 gcptmpl //toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
              -1.531)
259 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
              -1.531);
260 gcptmpl
261 gcptmpl stockandtoolpaths();
262 gcptmpl //stockwotoolpaths();
263 gcptmpl //outputtoolpaths();
264 gcptmpl
265 gcptmpl //makecube(3, 2, 1);
266 gcptmpl
267 gcptmpl //instantiatecube();
268 gcptmpl
269 gcptmpl closegcodefile();
270 gcptmpl closedxffiles();
271 gcptmpl closedxffile();
```

Which generates a 3D model which previews in OpenSCAD as:



Obviously, the use of OpenSCAD to make use of roundover tooling remains to be implemented. Similarly, generate keyhole dxfs needs to be updated to function as expected.

Note that there are several possible ways to work with the 3D models of the cuts, either directly displaying the returned 3D model when explicitly called for after storing it in a variable or calling it up as a calculation (Python command ouput(<foo>) or OpenSCAD returning a model, or calling an appropriate OpenSCAD command):

- generatepaths = true this has the Python code collect toolpath cuts and rapid movements in variables which are then instantiated by appropriate commands/options (shown in the OpenSCAD template gcodepreview.scad)
- generatepaths = false this option affords the user control over how the model elements are handled (shown in the Python template gcodepreview.py)

The templates set up these options as noted, and for OpenSCAD, implement code to ensure that True == true, and a set of commands are provided to output the stock, toolpaths, or part (toolpaths and rapids differenced from stock).

3 gcodepreview

This library for PythonSCAD 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 (note that at least a 4th additional axis may be worked up as a future option) 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 and using a traditional CAM application). There are multiple modes for this, doing so requires two 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 an additional OpenSCAD module which allows accessing it
```

An OpenSCAD file: gcodepreview.scad (gcpscad) — which uses the Python file and which
is included allowing it to access OpenSCAD variables for branching

Note that this architecture requires that many OpenSCAD modules are essentially "Dispatchers" (another term is "Descriptors") which pass information from one aspect of the environment to another, but in some instances it will be necessary to re-write Python definitions in OpenSCAD rather than calling the matching Python function directly.

3.1 Module Naming Convention

The original implementation required three files and used a convention for prefacing commands with \circ or p, but this requirement was obviated in the full Python re-write. The current implentation depends upon the class being instantiated as gcp as a sufficent differentation between the Python and the OpenSCAD versions of commands which will share the same name.

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.

Tool #s where used will be the first argument where possible — this makes it obvious if they are not used — the negative consideration, that it then doesn't allow for a usage where a DEFAULT tool is used is not an issue since the command currenttoolnum() may be used to access that number, and is arguably the preferred mechanism. An exception is when there are multiple tool #s as when opening a file — collecting them all at the end is a more straight-forward approach.

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

- generate (Boolean) used to identify which types of actions will be done
- write (action) used to write to files
- cut (action create 3D object)
- rapid (action create 3D object so as to show a collision)
- open (action (file))
- close (action (file))
- set (action/function) note that the matching get is implicit in functions which return variables, e.g., xpos()
- current

Nouns

- arc
- line
- rectangle
- circle

Suffixes

- feed (parameter)
- gcode/gc (filetype)
- pos position
- tool
- loop
- CC/CW
- 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 dxf	cutline cutlinedxf	dxfline	cutlinegc	cutarc cutarcdxf	dxfarc	cutarcgc	
gcode	cutlinegc		linegc	cutarcgc		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>gc commands was warranted during the initial development of arc-related commands.

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.) note the gcodepreview class which will normally be imported as gcp so that module <foo> will be called as gcp.<foo> from Python and by the same <foo> in OpenSCAD

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

There are multiple modes for programming PythonSCAD:

- Python in gcodepreview this allows writing out dxf files
- $\bullet \ \ OpenSCAD-see: \verb|https://openscad.org/documentation.html||$
- Programming in OpenSCAD with variables and calling Python this requires 3 files and was originally used in the project as written up at: https://github.com/WillAdams/ gcodepreview/blob/main/gcodepreview-openscad_0_6.pdf (for further details see below)
- Programming in OpenSCAD and calling Python where all variables as variables are held in Python classes (this is the technique used as of vo.8)
- Programming in Python and calling OpenSCAD https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/

For reference, structurally, when developing OpenSCAD commands which make use of Python variables this was rendered as:

```
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}
{\tt module\ oFOOBAR(...)\ \{}
    \texttt{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 style of definition might not have been necessary for some later modules since they are in turn calling internal modules which already use this structure.

Lastly note that this style of programming was abandoned in favour of object-oriented dot notation after vo.6 (see below).

3.1.1 Parameters and Default Values

Ideally, there would be *no* hard-coded values — every value used for calculation will be parameterized, and subject to control/modification. Fortunately, Python affords a feature which specifically addresses this, optional arguments with default values:

https://stackoverflow.com/questions/9539921/how-do-i-define-a-function-with-optional-argumen

In short, rather than hard-code numbers, for example in loops, they will be assigned as default stepsizearc values, and thus afford the user/programmer the option of changing them after. See stepsizearc stepsizeroundover and stepsizeroundover.

3.2 Implementation files and gcodepreview class

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

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe" --trust-
          python
3 gcpy #Currently tested with PythonSCAD_nolibfive-2025.01.02-x86-64-
          Installer.exe and Python 3.11
4 gcpy #gcodepreview 0.8, for use with PythonSCAD,
5 gcpy #if using from PythonSCAD using OpenSCAD code, see gcodepreview.
6 дсру
7 gcpy import sys
8 дсру
9 gcpy # getting openscad functions into namespace
10 gcpy #https://github.com/gsohler/openscad/issues/39
11 gcpy try:
12 gcpy
          from openscad import *
13 gcpy except ModuleNotFoundError as e:
14 дсру
          print("OpenSCAD_module_not_loaded.")
15 gcpy
16 gcpy # add math functions (using radians by default, convert to degrees
          where necessary)
17 gcpy import math
18 дсру
19 gcpy def pygcpversion():
20 дсру
          the gcp version = 0.8
21 дсру
          return thegcpversion
```

The OpenSCAD file must use the Python file (note that some test/example code is commented out):

```
1 gcpscad //!OpenSCAD
3 gcpscad //gcodepreview version 0.8
4 gcpscad //
5 gcpscad //used via include <gcodepreview.scad>;
6 gcpscad //
7 gcpscad
8 gcpscad use <gcodepreview.py>
9 gcpscad
10 gcpscad module gcpversion(){
11 gcpscad echo(pygcpversion());
12 gcpscad }
13 gcpscad
14 gcpscad //function myfunc(var) = gcp.myfunc(var);
15 gcpscad //
16 gcpscad //function getvv() = gcp.getvv();
17 gcpscad //
18 gcpscad //module makecube(xdim, ydim, zdim){
19 gcpscad //gcp.makecube(xdim, ydim, zdim);
20 gcpscad //}
21 gcpscad //
22 gcpscad //module placecube(){
23 gcpscad //gcp.placecube();
24 gcpscad //}
25 gcpscad //
26 gcpscad //module instantiatecube(){
27 gcpscad //gcp.instantiatecube();
28 gcpscad //}
29 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, generatepaths, 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 class which is defined is gcodepreview which begins with the init method which allows init passing in and defining the variables which will be used by the other methods in this class. Part of this includes handling various definitions for Boolean values.

```
23 gcpy class gcodepreview:
24 дсру
           def __init__(self, #basefilename = "export",
25 дсру
                         generatepaths = False,
26 дсру
                         generategcode = False,
27 дсру
28 дсру
                         generatedxf = False,
29 gcpy #
                           stockXwidth = 25,
                          stockYheight = 25,
30 gcpy #
31 gcpy #
                           stockZthickness = 1,
                          zeroheight = "Top",
stockzero = "Lower-left" ,
32 gcpy #
33 gcpy #
34 gcpy #
                           retractheight = 6,
35 gcpy #
                           currenttoolnum = 102,
36 gcpy #
                           toolradius = 3.175,
                           plunge = 100,
37 gcpy #
                           feed = 400,
38 gcpy #
                           speed = 10000
39 gcpy #
40 gcpy
                           ):
                self.basefilename = basefilename
41 gcpy #
               if (generatepaths == 1):
42 дсру
43 дсру
                    self.generatepaths = True
               if (generatepaths == 0):
44 дсру
                    self.generatepaths = False
45 дсру
46 дсру
               else:
47 дсру
                    self.generatepaths = generatepaths
48 дсру
               if (generategcode == 1):
49 дсру
                    self.generategcode = True
50 дсру
               if (generategcode == 0):
                    self.generategcode = False
51 дсру
52 дсру
               else:
53 дсру
                    self.generategcode = generategcode
               if (generatedxf == 1):
54 дсру
                    self.generatedxf = True
55 дсру
56 дсру
               if (generatedxf == 0):
                    self.generatedxf = False
57 дсру
58 дсру
               else:
                    self.generatedxf = generatedxf
59 дсру
60 gcpy #
                self.stockXwidth = stockXwidth
                self.stockYheight = stockYheight
61 gcpy #
62 gcpy #
                self.stockZthickness = stockZthickness
                self.zeroheight = zeroheight
63 gcpy #
                self.stockzero = stockzero
64 gcpy #
                self.retractheight = retractheight
65 gcpy #
                self.currenttoolnum = currenttoolnum
66 gcpy #
67 gcpy #
                self.toolradius = toolradius
                self.plunge = plunge
68 gcpy #
                 self.feed = feed
69 gcpy #
70 gcpy #
                 self.speed = speed
```

```
71 gcpy #
                 global toolpaths
                if (openscadloaded == True):
72 gcpy #
                     self.toolpaths = cylinder(0.1, 0.1)
73 gcpy #
               self.generatedxfs = False
74 дсру
75 дсру
76 дсру
           def checkgeneratepaths():
77 дсру
               return self.generatepaths
78 дсру
79 gcpy #
            def myfunc(self, var):
80 gcpy #
                 self.vv = var * var
                return self.vv
81 gcpy #
82 gcpy #
83 gcpy #
            def getvv(self):
84 gcpy #
                 return self.vv
85 gcpy #
            def checkint(self):
86 gcpy #
87 gcpy #
                 return self.mc
88 gcpy #
89 gcpy #
            def makecube(self, xdim, ydim, zdim):
90 gcpy #
                 self.c=cube([xdim, ydim, zdim])
91 gcpy #
92 gcpy #
            def placecube(self):
                 output(self.c)
93 gcpy #
94 gcpy #
            def instantiatecube(self):
95 gcpy #
                 return self.c
96 gcpy #
97 gcpy #
```

3.2.1 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 the current depth in the current toolpath. This will be done using paired functions (which will set and return the matching variable) and a matching variable.

The first such variables are for xyz position:

```
mpxmpxmpympympz
```

Similarly, for some toolpaths it will be necessary to track the depth along the Z-axis as the toolpath tpzinc is cut out, or the increment which a cut advances — this is done using an internal variable, tpzinc. It will further be necessary to have a variable for the current tool:

currenttoolnum

zpos

• currenttoolnum

Note that the currenttoolnum variable should always be accessed and 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.

Similarly, a 3D model of the tool will be available as currenttool itself and used where appropriate.

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

```
98 дсру
            def xpos(self):
99 gcpy #
                  global mpx
100 дсру
                 return self.mpx
101 дсру
            def ypos(self):
102 дсру
103 gcpy #
                  global mpy
104 дсру
                 return self.mpy
105 дсру
            def zpos(self):
106 дсру
107 gcpy #
                  global mpz
108 дсру
                 return self.mpz
109 дсру
110 gcpy #
             def tpzinc(self):
111 gcpy #
                  global tpzinc
112 gcpy #
                  return self.tpzinc
```

Wrapping these in OpenSCAD functions allows use of this positional information from OpenSCAD:

```
30 gcpscad function xpos() = gcp.xpos();
31 gcpscad
32 gcpscad function ypos() = gcp.ypos();
33 gcpscad
34 gcpscad function zpos() = gcp.zpos();
```

setxpos and in turn, functions which set the positions: setxpos, setypos, and setzpos.

```
setypos
setzpos
```

```
def setxpos(self, newxpos):
114 дсру
                  global mpx
115 gcpv #
                 self.mpx = newxpos
116 gcpy
117 gcpy
118 дсру
            def setypos(self, newypos):
                 global mpy
self.mpy = newypos
119 gcpy #
120 дсру
121 дсру
122 дсру
            def setzpos(self, newzpos):
123 gcpy #
                  global mpz
124 дсру
                 self.mpz = newzpos
125 gcpy
126 gcpy #
             def settpzinc(self, newtpzinc):
                  global tpzinc
127 gcpy #
128 gcpy #
                  self.tpzinc = newtpzinc
```

Using the set... routines will afford a single point of control if specific actions are found to be contingent on changes to these positions.

3.2.2 Initial Modules

gcodepreview The first such routine, (actually a subroutine, see gcodepreview) setupstock will be appropriately setupstock 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 version handles all necessary actions.

gcp.setupstock

Since part of a class, it will be called as gcp. setupstock. It requires that the user set parameters for stock dimensions and so forth, and will create comments in the G-code (if generating that file is enabled) which incorporate the stock dimensions and its position relative to the zero as set relative to the stock.

```
130 дсру
            def setupstock(self, stockXwidth,
131 дсру
                         stockYheight,
                         stockZthickness,
132 gcpv
133 дсру
                         zeroheight,
                          stockzero
134 дсру
135 дсру
                         retractheight):
136 дсру
                self.stockXwidth = stockXwidth
                self.stockYheight = stockYheight
137 дсру
138 дсру
                self.stockZthickness = stockZthickness
139 дсру
                self.zeroheight = zeroheight
                self.stockzero = stockzero
140 gcpy
141 gcpy
                self.retractheight = retractheight
                global mpx
142 gcpy #
                self.mpx = float(0)
143 gcpy
                global mpy
144 gcpy #
145 gcpy
                self.mpy = float(0)
                global mpz
146 gcpy #
147 дсру
                self.mpz = float(0)
                global tpz
148 gcpy #
149 gcpy #
                self.tpzinc = float(0)
150 gcpy #
                global currenttoolnum
                self.currenttoolnum = 102
151 дсру
152 gcpy #
                global currenttoolshape
                self.currenttoolshape = cylinder(12.7, 1.5875)
153 gcpy
                self.rapids = self.currenttoolshape
154 дсру
                global stock
155 gcpy #
                self.stock = cube([stockXwidth, stockYheight,
156 gcpy
                   stockZthickness])
157 gcpy #%WRITEGC
                    if self.generategcode == True:
158 gcpy #%WRITEGC
                              self.writegc("(Design File: " + self.
           basefilename + ")")
159 дсру
                self.toolpaths = cylinder(0.1, 0.1)
```

The setupstock command is required if working with a 3D project, creating the block of stock which the following toolpath commands will cut away. Note that since Python in PythonSCAD 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 describes it in using the terms described for CutViewer.

```
if self.zeroheight == "Top":
160 gcpy
                                    if self.stockzero == "Lower-Left":
161 дсру
                                           self.stock = stock.translate([0,0,-self.
162 gcpy
                                                  stockZthickness])
                                           if self.generategcode == True:
163 дсру
                                                   self.writegc("(stockMin:0.00mm,_{\sqcup}0.00mm,_{\sqcup}-",str(
164 дсру
                                                          self.stockZthickness),"mm)")
                                                   self.writegc("(stockMax:",str(self.stockXwidth)
165 gcpv
                                                   ,"mm,_{\square}", str(stockYheight),"mm,_{\square}0.00mm)") self.writegc("(STOCK/BLOCK,_{\square}", str(self.
166 gcpy
                                                          stockXwidth),",u",str(self.stockYheight),",u
                                                          ", str(self.stockZthickness), ", \u0.00, \u0.0
                                    if self.stockzero == "Center-Left":
167 gcpy
168 дсру
                                           self.stock = self.stock.translate([0,-stockYheight
                                                  / 2,-stockZthickness])
                                           if self.generategcode == True:
169 дсру
                                                   self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
170 дсру
                                                          stockYheight/2), "mm, __-", str(self.
                                                          stockZthickness), "mm)")
                                                   self.writegc("(stockMax:",str(self.stockXwidth)
171 gcpy
                                                          ,"mm,_{\sqcup}", {\tt str} (self.stockYheight/2),"mm,_{\sqcup}0.00mm
                                                          )")
                                                   self.writegc("(STOCK/BLOCK,_{\sqcup}",str(self.
172 gcpy
                                                          stockXwidth),",",str(self.stockYheight),",
                                                          ", str(self.stockZthickness), ", \square 0.00, \square ", str(
                                                          self.stockYheight/2),",\square",str(self.
                                                          stockZthickness),")");
                                    if self.stockzero == "Top-Left":
173 дсру
                                           self.stock = self.stock.translate([0,-self.
174 gcpy
                                                  stockYheight,-self.stockZthickness])
                                           if self.generategcode == True:
175 gcpy
176 дсру
                                                   self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
                                                          stockYheight), "mm, __-", str(self.
                                                          stockZthickness),"mm)")
                                                   self.writegc("(stockMax:",str(self.stockXwidth)
,"mm, _ _ 0.00mm, _ _ 0.00mm)")
177 gcpy
                                                   self.writegc("(STOCK/BLOCK, □", str(self.
178 дсру
                                                          stockXwidth),",",",str(self.stockYheight),","
                                                          ", str(self.stockZthickness), ", u0.00, u", str(
                                                          self.stockYheight), ", ", ", str(self.
                                                          stockZthickness),")")
                                    if self.stockzero == "Center":
179 дсру
                                            self.stock = self.stock.translate([-self.
180 дсру
                                                  stockXwidth / 2,-self.stockYheight / 2,-self.
                                                  stockZthickness])
                                           if self.generategcode == True:
181 дсру
                                                   self.writegc\tilde{(}"(stockMin:_{\sqcup}-",str(self.
182 дсру
                                                          stockXwidth/2), ", u-", str(self.stockYheight)
                                                          /2), "mm, u-", str(self.stockZthickness), "mm)")
                                                   self.writegc("(stockMax:",str(self.stockXwidth
183 дсру
                                                          /2), "mm, _{\sqcup}", str(self.stockYheight/2), "mm, _{\sqcup}
                                                          0.00mm)")
                                                   self.writegc("(STOCK/BLOCK, ", str(self.
184 gcpv
                                                          stockXwidth), ", ", str(self.stockYheight), ", u
                                                          ", str(self.stockZthickness), ", ", ", str(self.
                                                          stockXwidth/2),", ", str(self.stockYheight
                           /2),",",str(self.stockZthickness),")")

if self.zeroheight == "Bottom":
185 gcpy
                                    if self.stockzero == "Lower-Left":
186 дсру
187 дсру
                                             self.stock = self.stock.translate([0,0,0])
                                             if self.generategcode == True:
188 дсру
                                                     self.writegc("(stockMin:0.00mm,_{\sqcup}0.00mm,_{\sqcup}0.00mm
189 дсру
                                                           )")
190 gcpy
                                                     self.writegc("(stockMax:",str(self.stockXwidth
                                                           ), "mm, _{\sqcup}", str(self.stockYheight), "mm, _{\sqcup\sqcup}", str
                                                           (self.stockZthickness),"mm)")
                                                     self.writegc("(STOCK/BLOCK,_{\sqcup}",str(self.
191 дсру
                                                            stockXwidth),",",str(self.stockYheight),",
                                                            \square", str(self.stockZthickness), ", \square0.00, \square0.00,
                                                           ۵.00)")
                                    if self.stockzero == "Center-Left":
192 gcpy
                                           self.stock = self.stock.translate([0,-self.
193 дсру
                                                  stockYheight / 2,0])
194 дсру
                                           if self.generategcode == True:
```

```
195 дсру
                               self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
                                   stockYheight/2), "mm, u0.00mm)")
                               self.writegc("(stockMax:",str(self.stockXwidth)
196 gcpy
                                    , "mm, _{\sqcup}", str(self.stockYheight/2), "mm, _{\sqcup}-", str
                                   (self.stockZthickness),"mm)")
                               self.writegc("(STOCK/BLOCK, ", str(self.
197 дсру
                                   stockXwidth), ", u", str(self.stockYheight), ", u
                                   ", str(self.stockZthickness), ", u0.00, u", str(
                                   self.stockYheight/2),", \( 0.00mm )");
                      if self.stockzero == "Top-Left":
198 дсру
                          self.stock = self.stock.translate([0,-self.
199 дсру
                              stockYheight,0])
                          if self.generategcode == True:
200 дсру
                               self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
201 дсру
                                   stockYheight), "mm, u0.00mm)")
                               self.writegc("(stockMax:",str(self.stockXwidth)
202 дсру
                                   ,"mm,_{\sqcup}0.00mm,_{\sqcup}",str(self.stockZthickness),"
                               self.writegc("(STOCK/BLOCK, __", str(self.
203 gcpy
                                   stockXwidth), ", u ", str(self.stockYheight), ", u
                                   ", str(self.stockZthickness), ", u0.00, u", str(self.stockYheight), ", u0.00)")
                      if self.stockzero == "Center":
204 дсру
205 дсру
                          self.stock = self.stock.translate([-self.
                               stockXwidth / 2,-self.stockYheight / 2,0])
                          if self.generategcode == True:
206 дсру
                               self.writegc("(stockMin:_{\sqcup}-", str(self.
207 дсру
                                   stockXwidth/2), ", ", str(self.stockYheight
                                   /2),"mm,<sub>\u0.00mm</sub>)")
208 gcpy
                               self.writegc("(stockMax:",str(self.stockXwidth
                                   /2), "mm, \square", str(self.stockYheight/2), "mm, \square",
                                   str(self.stockZthickness),"mm)")
                               self.writegc("(STOCK/BLOCK, _ ", str(self.
209 дсру
                                   stockXwidth),",",",str(self.stockYheight),","
                                   ", str(self.stockZthickness), ", ", ", str(self.
                                   stockXwidth/2), ", ", str(self.stockYheight
                                   /2),", 🗆 0.00)")
                 if self.generategcode == True:
210 gcpy
                      self.writegc("G90");
211 дсру
                      self.writegc("G21");
212 дсру
```

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. The OpenSCAD version is simply a descriptor:

```
{\tt 36~gcpscad~\textbf{module}~setupstock(stockXwidth,~stockYheight,~stockZthickness,}
             zeroheight, stockzero, retractheight) {
37 gcpscad
              gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
                 zeroheight, stockzero, retractheight);
38 gcpscad }
```

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()
```

Tools and Changes

currenttoolnumber Similarly Python functions and variables will be used in: currenttoolnumber (note that it is imsettool portant to use a different name than the variable currenttoolnum and settool to track and set and return the current tool:

```
214 дсру
            def settool(self,tn):
215 gcpy #
                 global currenttoolnum
216 дсру
                self.currenttoolnum = tn
217 дсру
218 дсру
            def currenttoolnumber(self):
219 gcpy #
                 global currenttoolnum
220 дсру
                return self.currenttoolnum
221 дсру
222 gcpy #
             def currentroundovertoolnumber(self):
```

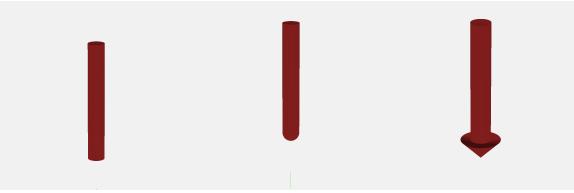
```
223 gcpy # global Roundover_tool_num
224 gcpy # return self.Roundover_tool_num
```

The settool command will normally be set using one of the variables as defined in the template, and the gcodepreview object is hard-coded to use the tool numbers which Carbide 3D uses for their tooling.

3.3.1 3D Shapes for Tools

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

3.3.1.1 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:

ballnose The ballnose is modeled as a hemisphere joined with a cylinder:

endmill v The 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):

bowl tool The bowl tool is modeled as a series of cylinders stacked on top of each other and hull()ed together:

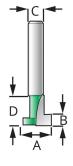
```
242 gcpy def bowl_tool(self, radius, diameter, height):
243 gcpy bts = cylinder(height - radius, diameter / 2, diameter / 2, center=False)
```

```
244 дсру
               bts = bts.translate([0, 0, radius])
               bts = bts.union(cylinder(height, diameter / 2 - radius,
245 дсру
                   diameter / 2 - radius, center=False))
               for i in range (90):
246 gcpy
247 gcpy #
                    print(math.sin(math.radians(i)))
                    slice = cylinder((radius / 90), ((diameter / 2 - radius
248 дсру
                       ) + radius * math.sin(math.radians(i))), ((diameter
                       / 2 - radius) + radius * math.sin(math.radians(i +
                       1))), center=False)
                       = hull(bts, slice.translate([0, 0, (radius - radius
249 дсру
                        * math.cos(math.radians(i)))]))
250 дсру
               return bts
```

3.3.1.2 Tooling for Undercutting Toolpaths There are several notable candidates for undercutting tooling.

- 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.

3.3.1.2.1 Keyhole tools Keyhole toolpaths (see: subsection **3.4.3.2.3** 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.



Keyhole Router Bits

#	Α	В	С	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 The keyhole is modeled in two parts, first the cutting base:

```
def keyhole(self, es_diameter, es_flute_length):
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:

```
def keyhole_shaft(self, es_diameter, es_flute_length):
return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
h=es_flute_length, center=False)
```

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

```
def threadmill(self, minor_diameter, major_diameter, cut_height
):
```

```
btm = cylinder(r1=(minor_diameter / 2), r2=(major_diameter
259 дсру
                   / 2), h=cut_height, center = False)
                top = cylinder(r1=(major_diameter / 2), r2=(minor_diameter
260 gcpy
                   / 2), h=cut_height, center = False)
                top = top.translate([0, 0, cut_height/2])
261 gcpy
262 дсру
                tm = btm.union(top)
263 дсру
                return tm
264 дсру
265 дсру
            def threadmill_shaft(self, diameter, cut_height, height):
                shaft = cylinder(r1=(diameter / 2), r2=(diameter / 2), h=
266 дсру
                   height, center = False)
                shaft = shaft.translate([0, 0, cut_height/2])
267 дсру
                {\tt return} \ {\tt shaft}
268 дсру
```

3.3.1.2.3 Dovetails The 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)

```
270 gcpy
             def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height
                  , dt_angle):
                  \textbf{return} \  \, \texttt{cylinder(r1=(dt\_bottomdiameter \ / \ 2), r2=(}
271 gcpy
                      {\tt dt\_topdiameter} / 2), h= dt_height, center=False)
```

3.3.1.3 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 sections or even slices of the tool shape to be modeled separately which are then hulled together. Something of this can be seen in the manual work-around for previewing them: https://community.carbide3d.com/t/ using-unsupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723.

Because it is necessary to divide the tooling into vertical slices and call the hull operation for each slice the tool definitions have to be called separately in the cut... modules.

3.3.1.4 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 3.3.1.3.

```
40 gcpscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) { 41 gcpscad if (radiustn == 56125) {
41 gcpscad
42 gcpscad
                  cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
              } else if (radiustn == 56142) {
43 gcpscad
44 gcpscad
                   cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
45 gcpscad //
                } else if (radiustn == 312) {
                    cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
46 gcpscad //
              } else if (radiustn == 1570) {
47 gcpscad
                  cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
48 gcpscad
49 gcpscad
50 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.

3.3.2 toolchange

toolchange Then 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.

> Note that the comments written out in G-code correspond to those 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 there are many varieties of tooling and not all will be implemented, especially in the early iterations of this project.

3.3.2.1 Selecting Tools The original implementation created the model for the tool at the current position, and a duplicate at the end 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 as part of the cutting command indirectly by first storing currenttoolshape 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.

```
273 gcpy def currenttool(self):
274 gcpy # global currenttoolshape
275 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:

3.3.2.2 Square and ball nose (including tapered ball nose)

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

3.3.2.3 Roundover (corner rounding)

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

3.3.2.4 Dovetails Unfortunately, tools which support undercuts such as dovetails are not supported by CutViewer (CAMotics will work for such tooling, at least dovetails which may be defined as "stub" endmills with a bottom diameter greater than upper diameter).

3.3.2.5 toolchange routine The Python definition for toolchange requires the tool number (used to write out the G-code comment description for CutViewer and also expects the speed for the current tool since this is passed into the G-code tool change command as part of the spindle on command.

```
def toolchange(self,tool_number,speed = 10000):
277 дсру
278 gcpy #
                  global currenttoolshape
                 self.currenttoolshape = self.endmill_square(0.001, 0.001)
279 дсру
280 gcpy
                 self.settool(tool_number)
281 дсру
282 дсру
                 if (self.generategcode == True):
283 дсру
                      self.writegc("(Toolpath)")
                      self.writegc("M05")
284 дсру
                 if (tool_number == 201):
285 дсру
                      self.writegc("(TOOL/MILL,6.35,\square0.00,\square0.00,\square0.00)")
286 дсру
287 дсру
                      self.currenttoolshape = self.endmill_square(6.35,
                          19.05)
                 elif (tool_number == 102):
288 дсру
                      self.writegc("(TOOL/MILL,3.175,_{\square}0.00,_{\square}0.00,_{\square}0.00)")
289 дсру
290 дсру
                      self.currenttoolshape = self.endmill_square(3.175,
                          12.7)
                 elif (tool_number == 112):
    self.writegc("(TOOL/MILL,1.5875, _0.00, _0.00, _0.00)")
291 дсру
292 дсру
293 дсру
                      self.currenttoolshape = self.endmill_square(1.5875,
                          6.35)
                 elif (tool_number == 122):
294 дсру
                      self.writegc("(TOOL/MILL,0.79375,_{\square}0.00,_{\square}0.00,_{\square}0.00)")
295 дсру
296 дсру
                      self.currenttoolshape = self.endmill_square(0.79375,
                          1.5875)
297 дсру
                 elif (tool_number == 202):
298 дсру
                      self.writegc("(TOOL/MILL,6.35,_{\sqcup}3.175,_{\sqcup}0.00,_{\sqcup}0.00)")
299 дсру
                      self.currenttoolshape = self.ballnose(6.35, 19.05)
                 elif (tool_number == 101):
300 дсру
                      \texttt{self.writegc("(TOOL/MILL,3.175,\_1.5875,\_0.00,\_0.00)")}
301 дсру
                      self.currenttoolshape = self.ballnose(3.175, 12.7)
302 дсру
                 elif (tool_number == 111):
303 дсру
                      self.writegc("(TOOL/MILL,1.5875, _0.79375, _0.00, _0.00)")
304 дсру
305 дсру
                      self.currenttoolshape = self.ballnose(1.5875, 6.35)
306 дсру
                 elif (tool number == 121):
                      self.writegc("(TOOL/MILL,3.175, _0.79375, _0.00, _0.00)")
307 дсру
                      self.currenttoolshape = self.ballnose(0.79375, 1.5875)
308 дсру
309 дсру
                 elif (tool number == 327):
                      self.writegc("(TOOL/MILL,0.03,_{\square}0.00,_{\square}13.4874,_{\square}30.00)")
310 дсру
                      self.currenttoolshape = self.endmill_v(60, 26.9748)
311 gcpy
                 elif (tool number == 301):
312 дсру
313 дсру
                      self.writegc("(TOOL/MILL,0.03,_{\sqcup}0.00,_{\sqcup}6.35,_{\sqcup}45.00)")
                      self.currenttoolshape = self.endmill_v(90, 12.7)
314 gcpy
315 дсру
                 elif (tool number == 302):
                      \texttt{self.writegc("(TOOL/MILL,0.03,\_0.00,\_10.998,\_30.00)")}
316 дсру
                      self.currenttoolshape = self.endmill_v(60, 12.7)
317 дсру
318 gcpy
                 elif (tool_number == 390):
                      self.writegc("(TOOL/MILL,0.03, __0.00, __1.5875, __45.00)")
319 gcpy
320 дсру
                      self.currenttoolshape = self.endmill_v(90, 3.175)
```

```
elif (tool_number == 374):
321 gcpy
                                               self.writegc("(TOOL/MILL,9.53,_{\square}0.00,_{\square}3.17,_{\square}0.00)")
322 дсру
                                      elif (tool_number == 375):
323 gcpy
                                                self.writegc("(TOOL/MILL,9.53, _0.00, _3.17, _0.00)")
324 gcpy
325 gcpy
                                      elif (tool_number == 376):
326 дсру
                                                self.writegc("(TOOL/MILL, 12.7, \_0.00, \_4.77, \_0.00)")
                                      elif (tool_number == 378):
327 gcpy
                                                self.writegc("(TOOL/MILL,12.7,_{\square}0.00,_{\square}4.77,_{\square}0.00)")
328 дсру
329 дсру
                                      elif (tool_number == 814):
                                                self.writegc("(TOOL/MILL,12.7,_{\sqcup}6.367,_{\sqcup}12.7,_{\sqcup}0.00)")
330 дсру
                                                \verb|#dt_bottomdiameter|, | dt_topdiameter|, | dt_height|, | dt_angle|
331 дсру
                                                #https://www.leevalley.com/en-us/shop/tools/power-tool-
332 дсру
                                                         accessories/router-bits/30172-dovetail-bits?item=18
                                                         J1607
333 дсру
                                                self.currenttoolshape = self.dovetail(12.7, 6.367,
                                                         12.7, 14)
334 дсру
                                      elif (tool_number == 56125):#0.508/2, 1.531
                                                self.writegc("(TOOL/CRMILL,_{\square}0.508,_{\square}6.35,_{\square}3.175,_{\square}7.9375,
335 дсру
                                                         ⊔3.175)")
                                      elif (tool_number == 56142):#0.508/2, 2.921
336 дсру
                                                self.writegc("(TOOL/CRMILL, _0.508, _3.571875, _1.5875, _
337 дсру
                                                         5.55625, _1.5875)")
338 gcpy #
                                         elif (tool_number == 312): #1.524/2, 3.175
                                                   self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
339 gcpy #
                          Radius, Height, Length)")
                                      elif (tool_number == 1570):#0.507/2, 4.509
340 gcpy
                                                self.writegc("(TOOL/CRMILL,_{\sqcup}0.17018,_{\sqcup}9.525,_{\sqcup}4.7625,_{\sqcup}
341 gcpy
                                                        12.7, 4.7625)")
342 gcpy \#https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warder-tray-1-4-warde
                          radius - x - 3 - 4 - dia - x - 5 - 8 - x - 1 - 4 - inch - shank. html
343 дсру
                                      elif (tool_number == 45982):#0.507/2, 4.509
                                                self.writegc("(TOOL/MILL,_{\square}15.875,_{\square}6.35,_{\square}19.05,_{\square}0.00)")
344 дсру
                                                self.currenttoolshape = self.bowl_tool(6.35, 19.05,
345 дсру
                                                         15.875)
```

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.

```
346 gcpy self.writegc("M6T",str(tool_number))
347 gcpy self.writegc("M03S",str(speed))
```

Note that the if...else constructs will need to be extended into the command cutline for those toolshapes (keyhole, roundover, &c.) which will not work with a straight-forward hull... implementation.

As per usual, the OpenSCAD command is simply a dispatcher:

```
52 gcpscad module toolchange(tool_number,speed){
53 gcpscad gcp.toolchange(tool_number,speed);
54 gcpscad }
```

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 and feed rates/ratios being calculated to provide the correct chipload at that setting.)

3.3.3 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:

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

```
def tool_diameter(self, ptd_tool, ptd_depth):
349 дсру
350 gcpy # Square 122,112,102,201
                if ptd_tool == 122:
351 дсру
352 дсру
                     return 0.79375
353 дсру
                 if ptd_tool == 112:
354 дсру
                     return 1.5875
                 if ptd_tool == 102:
355 дсру
356 дсру
                     return 3.175
                 if ptd_tool == 201:
357 дсру
                     return 6.35
358 дсру
359 gcpy # Ball 121,111,101,202
360 дсру
                 if ptd_tool == 122:
361 дсру
                     if ptd_depth > 0.396875:
                         return 0.79375
362 дсру
363 дсру
                     else:
364 дсру
                         return ptd_tool
365 дсру
                 if ptd_tool == 112:
                     if ptd_depth > 0.79375:
366 дсру
367 gcpy
                          return 1.5875
                     else:
368 дсру
369 дсру
                         return ptd_tool
                 if ptd_tool == 101:
370 дсру
                     if ptd_depth > 1.5875:
371 дсру
372 дсру
                          return 3.175
373 дсру
                     else:
                         return ptd_tool
374 дсру
375 дсру
                 if ptd_tool == 202:
                     if ptd_depth > 3.175:
376 дсру
377 дсру
                          return 6.35
378 дсру
                     else:
379 дсру
                          return ptd_tool
380 gcpy # V 301, 302, 390
                if ptd_tool == 301:
381 дсру
382 дсру
                     return ptd_tool
383 дсру
                 if ptd_tool == 302:
384 дсру
                     return ptd_tool
                 if ptd_tool == 390:
385 дсру
                     return ptd_tool
386 дсру
387 gcpy # Keyhole
                if ptd_tool == 374:
388 дсру
                     if ptd_depth < 3.175:</pre>
389 дсру
390 дсру
                          return 9.525
391 дсру
392 дсру
                         return 6.35
                if ptd_tool == 375:
393 дсру
394 дсру
                     if ptd_depth < 3.175:
                          return 9.525
395 дсру
396 дсру
                     else:
397 дсру
                          return 8
398 дсру
                if ptd_tool == 376:
                     if ptd_depth < 4.7625:</pre>
399 дсру
400 gcpy
                          return 12.7
401 gcpy
                     else:
402 gcpy
                         return 6.35
403 дсру
                if ptd_tool == 378:
                     if ptd_depth < 4.7625:</pre>
404 gcpy
405 gcpy
                          return 12.7
406 дсру
                      else:
407 дсру
                         return 8
408 gcpy # Dovetail
                if ptd_tool == 814:
409 gcpy
410 дсру
                     if ptd_depth > 12.7:
411 дсру
                          return 6.35
412 дсру
                     else:
413 gcpy
                          return 12.7
414 gcpy #https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
           radius - x - 3 - 4 - dia - x - 5 - 8 - x - 1 - 4 - inch - shank.html
                if ptd_tool == 45982:
415 gcpy
416 дсру
                     if ptd_depth > 6.35:
                          return 15.875
417 дсру
418 дсру
                     else:
419 дсру
                          return 0
```

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:

```
421 gcpy def tool_radius(self, ptd_tool, ptd_depth):
```

```
422 gcpy tr = self.tool_diameter(ptd_tool, ptd_depth)/2
423 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.)

3.3.4 Feeds and Speeds

feed There are several possibilities for handling feeds and speeds. Currently, base values for feed, plunge plunge, and speed are used, which may then be adjusted using various <tooldescriptor>_ratio speed 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.

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

3.4 Movement and Cutting

With all the scaffolding in place, it is possible to model the tool and hull() between copies of the cut... 3D model of the tool, or a cross-section of it for both cut... and rapid... operations.

Note that the variables self.rapids and self.toolpaths are used to hold the accumulated (unioned) 3D models of the rapid motions and cuts so that they may be differenced from the stock when the value generatepaths is set to True.

In order to manage the various options when cutting it will be necessary to have a command where the actual cut is made, passing in the shape used for the cut as a parameter. Since the 3D rcs aspect of rapid and cut operations are fundamentally the same, the command rcs which returns the hull of the begin (the current machine position as accessed by the x/y/zpos() commands and end positioning (provided as arguments ex, ey, and ez) of the tool shape/cross-section will be defined for the common aspects:

```
425 gcpy def rcs(self,ex, ey, ez, shape):
426 gcpy start = shape
427 gcpy end = shape
428 gcpy toolpath = hull(start.translate([self.xpos(), self.ypos(), self.zpos()]),
429 gcpy end.translate([ex,ey,ez]))
430 gcpy return toolpath
```

Diagramming this is quite straight-forward — there is simply a movement made from the current position to the end. If we start at the origin, X0, Y0, Z0, then it is simply a straight-line movement (rapid)/cut (possibly a partial cut in the instance of a keyhole or roundover tool), and no variables change value.

The code for diagramming this is quite straight-forward. A BlockSCAD implementation is available at: https://www.blockscad3d.com/community/projects/1894400, and the OpenSCAD version is only a little more complex (adding code to ensure positioning):

rapid...



Note that this routine does *not* alter the machine position variables since it may be called multiple times for a given toolpath. This command will then be called in the definitions for rapid and cutshape which only differ in which variable the 3D model is unioned with:

There are three different movements in G-code which will need to be handled. Rapid commands will be used for GO 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.

```
432 gcpy
            def rapid(self,ex, ey, ez):
433 дсру
                cts = self.currenttoolshape
                toolpath = self.rcs(ex, ey, ez, cts)
434 дсру
435 дсру
                self.setxpos(ex)
                self.setypos(ey)
436 gcpy
437 дсру
                self.setzpos(ez)
                if self.generatepaths == True:
438 дсру
439 дсру
                     self.rapids = self.rapids.union(toolpath)
                      return cylinder (0.01, 0, 0.01, center = False, fn = 3)
440 gcpy #
                     return cube([0.001,0.001,0.001])
441 gcpy
442 gcpy
                else:
443 дсру
                     return toolpath
444 дсру
            def cutshape(self,ex, ey, ez):
445 gcpy
446 gcpy
                cts = self.currenttoolshape
                toolpath = self.rcs(ex, ey, ez, cts)
447 gcpy
448 дсру
                if self.generatepaths == True:
                     self.toolpaths = self.toolpaths.union(toolpath)
449 gcpy
                     return cube([0.001,0.001,0.001])
450 дсру
451 gcpy
                else:
452 gcpy
                     return toolpath
```

Note that it is necessary to return a shape so that modules which use a $\$ variable>.union command will function as expected even when the 3D model created is stored in a variable.

It is then possible to add specific rapid... commands to match typical usages of G-code. 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:

```
def movetosafeZ(self):
454 gcpy
455 gcpy
                rapid = self.rapid(self.xpos(),self.ypos(),self.
                    retractheight)
456 gcpy #
                 if self.generatepaths == True:
                     rapid = self.rapid(self.xpos(),self.ypos(),self.
457 gcpy #
           retractheight)
458 gcpy #
                     self.rapids = self.rapids.union(rapid)
459 gcpy #
                 else:
          if (generategoode == true) \{
460 gcpy #
                 writecomment("PREPOSITION FOR RAPID PLUNGE"); Z25.650
461 gcpy #
          //G1Z24.663F381.0 ,"F",str(plunge)
462 gcpy #
463 дсру
                if self.generatepaths == False:
```

```
464 дсру
                    return rapid
465 gcpy
466 дсру
                    return cube([0.001,0.001,0.001])
467 gcpy
468 gcpy
            def rapidXY(self, ex, ey):
469 дсру
                rapid = self.rapid(ex,ey,self.zpos())
                 if self.generatepaths == True:
470 gcpy #
                      self.rapids = self.rapids.union(rapid)
471 gcpy #
472 gcpy #
                 else:
473 дсру
                if self.generatepaths == False:
474 gcpy
                    return rapid
475 gcpy
476 gcpy
            def rapidZ(self, ez):
                rapid = self.rapid(self.xpos(),self.ypos(),ez)
477 gcpy
                 if self.generatepaths == True:
478 gcpy #
                      self.rapids = self.rapids.union(rapid)
479 gcpy #
480 gcpy #
                 else:
481 дсру
                if self.generatepaths == False:
                    return rapid
482 gcpy
```

Note that rather than re-create the matching OpenSCAD commands as descriptors, due to the issue of redirection and return values and the possibility for errors it is more expedient to simply re-create the matching command (at least for the rapids):

```
58 gcpscad module movetosafeZ(){
59 gcpscad
              gcp.rapid(gcp.xpos(),gcp.ypos(),retractheight);
60 gcpscad }
61 gcpscad
62 gcpscad module rapid(ex, ey, ez) {
63 gcpscad
             gcp.rapid(ex, ey, ez);
64 gcpscad }
65 gcpscad
66 gcpscad module rapidXY(ex, ey) {
67 gcpscad
              gcp.rapid(ex, ey, gcp.zpos());
68 gcpscad }
69 gcpscad
70 gcpscad module rapidZ(ez) {
71 gcpscad
              gcp.rapid(gcp.xpos(),gcp.ypos(),ez);
72 gcpscad }
```

3.4.1 Lines

cut... The Python commands cut... add the currenttool to the toolpath hulled together at the curcutline rent position and the end position of the move. For cutline, this is a straight-forward connection of the current (beginning) and ending coordinates:

```
{\tt def} \ {\tt cutline}({\tt self}, {\tt ex}, \ {\tt ey}, \ {\tt ez}): \setminus
484 gcpv
485 gcpy \#below will need to be integrated into if/then structure not yet
486 gcpy #
                   cts = self.currenttoolshape
487 gcpy
                  if (self.currenttoolnumber() == 374):
                        {\tt self.writegc("(TOOL/MILL, 9.53, \ 0.00, \ 3.17, \ 0.00)")}
488 gcpy #
489 gcpy
                       self.currenttoolshape = self.keyhole(9.53/2, 3.175)
                       toolpath = self.cutshape(ex, ey, ez)
490 gcpy
                       \verb|self.currenttoolshape| = \verb|self.keyhole_shaft| (6.35/2,
491 дсру
                          12.7)
                       toolpath = toolpath.union(self.cutshape(ex, ey, ez))
492 gcpy
                   elif (self.currenttoolnumber() == 375):
493 gcpy #
                        \verb|self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")| \\
494 gcpy #
495 gcpy #
                   elif (self.currenttoolnumber() == 376):
                        self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
496 gcpy #
497 gcpy #
                   elif (self.currenttoolnumber() == 378):
                   self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
498 gcpy #
499 gcpy #
                       self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
500 gcpy #
            7.9375, 3.175)")
                  elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
501 gcpy
                        {\tt self.writegc("(TOOL/CRMILL,~0.508,~3.571875,~1.5875,~
502 gcpy #
            5.55625, 1.5875)")
                       toolpath = self.cutroundovertool(self.xpos(), self.ypos
503 дсру
                   (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
504 gcpy #
                        self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
505 gcpy #
            12.7, 4.7625)")
506 дсру
                  else:
                       toolpath = self.cutshape(ex, ey, ez)
507 дсру
```

```
508 дсру
                self.setxpos(ex)
               self.setypos(ey)
509 дсру
510 дсру
               self.setzpos(ez)
                if self.generatepaths == True:
511 gcpy #
512 gcpy #
                     self.toolpaths = union([self.toolpaths, toolpath])
513 gcpy #
                 else:
514 gcpy
                if self.generatepaths == False:
515 gcpy
                    return toolpath
516 gcpy
                else:
                    return cube([0.001,0.001,0.001])
517 дсру
518 дсру
           def cutlinedxfgc(self,ex, ey, ez):
519 дсру
                self.dxfline(self.currenttoolnumber(), self.xpos(), self.
520 gcpy
                   ypos(), ex, ey)
                self.writegc("G01_{\square}X", str(ex), "_{\square}Y", str(ey), "_{\square}Z", str(ez)
521 gcpy
522 gcpy #
                 if self.generate paths == False:
                return self.cutline(ex, ey, ez)
523 gcpy
524 gcpy
525 gcpy
           def cutroundovertool(self, bx, by, bz, ex, ey, ez,
               tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
                 n = 90 + fn*3
526 gcpy #
                 print("Tool dimensions", tool_radius_tip,
527 gcpy #
           tool_radius_width, "begin ",bx, by, bz, "end ", ex, ey, ez)
                step = 4 #360/n
528 gcpy
                shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
529 gcpy
                toolpath = hull(shaft.translate([bx,by,bz]), shaft.
530 gcpy
                   translate([ex,ey,ez]))
531 дсру
                shaft = cylinder(tool_radius_width*2,tool_radius_tip+
                   tool_radius_width,tool_radius_tip+tool_radius_width)
                toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
532 gcpy
                    tool_radius_width]), shaft.translate([ex,ey,ez+
                    tool_radius_width])))
                for i in range(1, 90, stepsizeroundover):
533 дсру
534 дсру
                    angle = i
                    dx = tool_radius_width*math.cos(math.radians(angle))
535 дсру
                    dxx = tool_radius_width*math.cos(math.radians(angle+1))
536 gcpy
                    dzz = tool_radius_width*math.sin(math.radians(angle))
537 gcpy
538 дсру
                    dz = tool_radius_width*math.sin(math.radians(angle+1))
                    dh = abs(dzz-dz)+0.0001
539 gcpy
540 gcpy
                    slice = cylinder(dh,tool_radius_tip+tool_radius_width-
                       dx,tool_radius_tip+tool_radius_width-dxx)
541 gcpy
                    toolpath = toolpath.union(hull(slice.translate([bx,by,
                        bz+dz]), slice.translate([ex,ey,ez+dz])))
                if self.generatepaths == True:
542 gcpy
543 gcpy
                    self.toolpaths = self.toolpaths.union(toolpath)
544 дсру
                else:
545 gcpy
                    return toolpath
546 дсру
           def cutZgcfeed(self, ez, feed):
547 gcpy
548 gcpy
                self.writegc("G01_{\square}Z", str(ez), "F",str(feed))
                 if self.generatepaths == False:
549 gcpy #
                return self.cutline(self.xpos(),self.ypos(),ez)
550 gcpy
```

The matching OpenSCAD command is a descriptor:

```
74 gcpscad module cutline(ex, ey, ez){
75 gcpscad gcp.cutline(ex, ey, ez);
76 gcpscad }
77 gcpscad
78 gcpscad module cutlinedxfgc(ex, ey, ez){
79 gcpscad gcp.cutlinedxfgc(ex, ey, ez);
80 gcpscad }
81 gcpscad module cutZgcfeed(ez, feed){
83 gcpscad gcp.cutZgcfeed(ez, feed);
84 gcpscad }
```

3.4.2 Arcs for toolpaths and DXFs

A further consideration here is that G-code and DXF support arcs in addition to the lines already implemented. Implementing arcs wants at least the following options for quadrant and direction:

- cutarcCW cut a partial arc described in a clock-wise direction
- cutarcCC counter-clock-wise

- cutarcNWCW cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCC while it wont matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCW
- cutcircleCCdxf
- cutcircleCWdxf

It will be necessary to have two separate representations of arcs — the G-code and 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 cutting in each direction and at changing Z-heights so as to allow for threading and similar operations. 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) in both clock-wise and counter-clock-wise directions

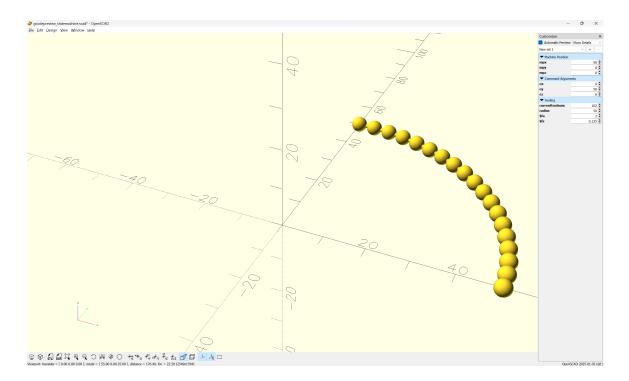
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

- ex note that the matching origins (bx, by, bz) as well as the (current) toolnumber are
 accessed using the appropriate commands
- 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
- tpzreldim the relative depth (or increase in height) of the current cutting motion

Since OpenSCAD does not have an arc movement command it is necessary to iterate through a cutarcCW loop: cutarcCW (clockwise) or cutarcCC (counterclockwise) to handle the drawing and processing cutarcCC of the cutline() 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 (the line version is used rather than shape so as to capture the changing machine positions with each step through the loop). 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.

This brings to the fore the fact that at its heart, this program is simply graphing math in 3D using tools (as presaged by the book series *Make:Geometry/Trigonometry/Calculus*). This is clear in a depiction of the algorithm for the cutarccc/cw commands, where the x value is the cos of the radius and the y value the sin:



The code for which makes this obvious:

```
/* [Machine Position] */
mpx = 0;
/* [Machine Position] */
mpy = 0;
/* [Machine Position] */
mpz = 0;
/* [Command Arguments] */
ex = 50;
/* [Command Arguments] */
ey = 25;
/* [Command Arguments] */
ez = -10;
/* [Tooling] */
currenttoolnum = 102;
machine_extents();
radius = 50;
$fa = 2;
fs = 0.125;
plot arc(radius, 0, 0, 0, radius, 0, 0,0, radius, 0,90, 5);
\verb|module plot_arc(bx, by, bz, ex, ey, ez, acx, acy, radius, barc, earc, inc)| \\
for (i = [barc : inc : earc-inc]) {
  union(){
    hull()
    {
      translate([acx + cos(i)*radius,
                 acy + sin(i)*radius,
                 0]){
        sphere(r=0.5);
      translate([acx + cos(i+inc)*radius,
                 acy + sin(i+inc)*radius,
                 0]){
        sphere(r=0.5);
    }
      translate([acx + cos(i)*radius,
                 acy + sin(i)*radius,
                 0]){
      sphere(r=2);
      translate([acx + cos(i+inc)*radius,
                 acy + sin(i+inc)*radius,
                 0]){
      sphere(r=2);
```

```
}
module machine_extents(){
{\tt translate([-200, -200, 20])\{}
  cube([0.001, 0.001, 0.001], center=true);
{\tt translate([200,\ 200,\ 20])\{}
  cube([0.001, 0.001, 0.001], center=true);
module plot_cut(bx, by, bz, ex, ey, ez) {
    translate([bx, by, bz]){
      sphere(r=5);
    translate([ex, ey, ez]){
     sphere(r=5);
    hull(){
      translate([bx, by, bz]){
       sphere(r=1);
      }
      translate([ex, ey, ez]){
       sphere(r=1);
     }
    }
 }
```

Note that it is necessary to move to the beginning cutting position before calling, and that it is necessary to pass in the relative change in Z position/depth. (Previous iterations calculated the increment of change outside the loop, but it is more workable to do so inside.)

```
def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
552 gcpy
                tpzreldim, stepsizearc=1):
553 gcpy #
                 tpzinc = ez - self.zpos() / (earc - barc)
                tpzinc = tpzreldim / (earc - barc)
554 дсру
555 gcpy
                cts = self.currenttoolshape
556 gcpy
                toolpath = cts
                toolpath = toolpath.translate([self.xpos(),self.ypos(),self
557 gcpy
                    .zpos()])
                i = barc
558 дсру
                while i < earc:</pre>
559 дсру
                     toolpath = toolpath.union(self.cutline(xcenter + radius
560 дсру
                        * math.cos(math.radians(i)), ycenter + radius * math.sin(math.radians(i)), self.zpos()+tpzinc))
                     i += stepsizearc
561 gcpy
                if self.generatepaths == False:
562 дсру
563 дсру
                    return toolpath
564 gcpy
                else:
                     return cube([0.01,0.01,0.01])
565 дсру
566 дсру
            def cutarcCW(self, barc,earc, xcenter, ycenter, radius,
567 дсру
                tpzreldim, stepsizearc=1):
568 gcpy #
                 print(str(self.zpos()))
569 gcpy #
                 print(str(ez))
                 print(str(barc - earc))
570 gcpy #
                 tpzinc = ez - self.zpos() / (barc - earc)
571 gcpy #
572 gcpy #
                 print(str(tzinc))
573 gcpy #
                 global toolpath
                 print("Entering n toolpath")
574 gcpy #
575 gcpy
                tpzinc = tpzreldim / (barc - earc)
                cts = self.currenttoolshape
576 gcpy
577 дсру
                toolpath = cts
                toolpath = toolpath.translate([self.xpos(),self.ypos(),self
578 gcpy
                    .zpos()])
579 gcpy
                i = barc
                while i > earc:
580 дсру
581 дсру
                     toolpath = toolpath.union(self.cutline(xcenter + radius
                          * math.cos(math.radians(i)), ycenter + radius *
                         math.sin(math.radians(i)), self.zpos()+tpzinc))
                      self.setxpos(xcenter + radius * math.cos(math.radians(
582 gcpy #
           i)))
583 gcpy #
                      self.setypos(ycenter + radius * math.sin(math.radians(
           i)))
```

```
print(str(self.xpos()), str(self.ypos(), str(self.zpos
584 gcpy #
           ())))
585 gcpy #
                     self.setzpos(self.zpos()+tpzinc)
                    i += abs(stepsizearc) * -1
586 дсру
587 gcpy #
                 self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
           radius, barc, earc)
                if self.generatepaths == True:
588 gcpy #
                     print ("{\it Unioning n toolpath"})
589 gcpy #
590 gcpy #
                     self.toolpaths = self.toolpaths.union(toolpath)
591 gcpy #
592 дсру
                if self.generatepaths == False:
593 дсру
                    return toolpath
594 дсру
                else:
                    return cube([0.01,0.01,0.01])
595 дсру
```

Matching OpenSCAD modules are easily made:

```
86 gcpscad module cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim){
87 gcpscad gcp.cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim);
88 gcpscad }
89 gcpscad module cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim){
91 gcpscad gcp.cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim);
92 gcpscad }
```

3.4.3 Cutting shapes and expansion

Certain basic shapes (arcs, circles, rectangles), will be incorporated in the main code. Other shapes will be added 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 3.3.2) 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.

3.4.3.1 Building blocks The outlines of shapes will be defined using:

- lines dxfline
- arcs dxfarc

It may be that splines or Bézier curves will be added as well.

3.4.3.2 List of shapes 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.

```
• o
```

- circle dxfcircle
- ellipse (oval) (requires some sort of non-arc curve)
 - * egg-shaped
- annulus (one circle within another, forming a ring) handled by nested circles
- 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 sector¹
- two convex, one concave arc
- 4
- rectangle (including square) dxfrectangle, dxfrectangleround
- 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

Note that most shapes will also exist in a rounded form where sharp angles/points are replaced by arcs/portions of circles, with the most typical being dxfrectangleround.

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

- two straight edges, one concave arc—oddly, 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 arc—with 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?

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

- $\bullet\,$ 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., dxfrectangleround
- 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, the command for this also models the areas which should be cleared for the sake of reducing wear on the tool and ensuring chip clearance

Some further considerations:

relationship of geometry to toolpath — arguably there should be an option for each toolpath
(we will use Carbide Create as a reference implementation) which is to be supported. Note
that there are several possibilities: modeling the tool movement, describing the outline which
the tool will cut, modeling a reference shape for the toolpath

- tool geometry it should be possible to include support for specialty tooling such as dovetail cutters and to get an accurate 3D model, esp. for tooling which undercuts since they cannot be modeled in Carbide Create.
- 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)

3.4.3.2.1 circles Circles are made up of a series of arcs:

```
def dxfcircle(self, tool_num, xcenter, ycenter, radius):
self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
self.dxfarc(tool_num, xcenter, ycenter, radius, 270, 360)
```

A Drill toolpath is a simple plunge operation will will have a matching circle to define it.

3.4.3.2.2 rectangles There are two forms for rectangles, square cornered and rounded:

```
def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
    yheight, corners = "Square", radius = 6):
603 gcpy
                 if corners == "Square":
604 дсру
                     self.dxfline(tool_num, xorigin, yorigin, xorigin +
605 дсру
                         xwidth, yorigin)
606 дсру
                     self.dxfline(tool_num, xorigin + xwidth, yorigin,
                         xorigin + xwidth, yorigin + yheight)
                     self.dxfline(tool_num, xorigin + xwidth, yorigin +
607 gcpy
                         yheight, xorigin, yorigin + yheight)
                     self.dxfline(tool_num, xorigin, yorigin + yheight,
608 дсру
                 xorigin, yorigin)
elif corners == "Fillet":
609 gcpv
                     self.dxfrectangleround(tool_num, xorigin, yorigin,
610 gcpy
                         xwidth, yheight, radius)
                 elif corners == "Chamfer":
611 gcpy
                     self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
612 gcpy
                         xwidth, yheight, radius)
                 elif corners == "Flipped | Fillet":
613 gcpy
                     self.dxfrectangleflippedfillet(tool_num, xorigin,
614 gcpy
                         yorigin, xwidth, yheight, radius)
```

Note that the rounded shape below would be described as a rectangle with the "Fillet" corner treatment in Carbide Create.

```
def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
616 дсру
              yheight, radius):
              self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
617 gcpy
                                           0, 90)
                 yheight - radius, radius,
              self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
618 дсру
                  radius, radius, 90, 180)
              self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
619 gcpy
                 radius, 180, 270)
              self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
620 дсру
                 radius, radius, 270, 360)
621 gcpy
              self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
622 gcpy
                  xwidth - radius, yorigin)
              self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
623 gcpy
                 xorigin + xwidth, yorigin + yheight - radius)
              self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
624 gcpy
                  yheight, xorigin + radius, yorigin + yheight)
              625 gcpy
```

So we add the balance of the corner treatments which are decorative (and easily implemented), Chamfer:

```
def dxfrectanglechamfer(self, tool_num, xorigin, yorigin, xwidth, yheight, radius):

self.dxfline(tool_num, xorigin + radius, yorigin, xorigin, yorigin + radius)
```

```
629 gcpy
                  self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                       xorigin + radius, yorigin + yheight)
                  self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
    yheight, xorigin + xwidth, yorigin + yheight - radius)
630 gcpy
                  self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
631 gcpy
                      xorigin + xwidth, yorigin + radius)
632 дсру
                  self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
633 gcpy
                       xwidth - radius, yorigin)
                  self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
634 дсру
                  xorigin + xwidth, yorigin + yheight - radius)
self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
635 дсру
                       yheight, xorigin + radius, yorigin + yheight)
                  self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
636 дсру
                       xorigin, yorigin + radius)
```

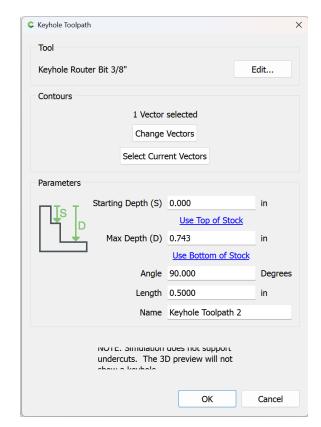
Flipped Fillet:

```
def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
638 gcpv
                xwidth, yheight, radius):
639 дсру
                self.dxfarc(tool_num, xorigin, yorigin, radius,
                                                                   0, 90)
               self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
640 gcpy
                   90, 180)
641 gcpy
               self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                   radius, 180, 270)
                self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
642 gcpy
                   270, 360)
643 gcpy
               self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
644 дсру
                    xwidth - radius, yorigin)
               self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
645 gcpy
                   xorigin + xwidth, yorigin + yheight - radius)
                self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
646 дсру
                    yheight, xorigin + radius, yorigin + yheight)
               \verb|self.dxfline(tool_num, xorigin, yorigin + yheight - radius, \\
647 gcpy
                    xorigin, yorigin + radius)
```

3.4.3.2.3 Keyhole toolpath and undercut tooling 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 3.3.1.2

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



- 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:

```
def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
 649 gcpy
                  kh_max_depth, kht_direction, kh_distance):
                  if (kht_direction == "N"):
 650 дсру
 651 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                           kh_max_depth, 90, kh_distance)
                  elif (kht_direction == "S"):
 652 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
 653 дсру
                  kh_max_depth, 270, kh_distance)
elif (kht_direction == "E"):
 654 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
 655 дсру
                           kh_max_depth, 0, kh_distance)
                  elif (kht_direction == "W"):
 656 дсру
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
 657 gcpy
                  kh_max_depth, 180, kh_distance)
if self.generatepaths == True:
 658 gcpy
                      self.toolpaths = union([self.toolpaths, toolpath])
 659 gcpy
 660 дсру
                      return toolpath
 661 дсру
                  else:
                      return cube([0.01,0.01,0.01])
 662 gcpy
94 gcpscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
             kht_direction, kh_distance){
95 gcpscad
             gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
                 kht_direction, kh_distance);
96 gcpscad }
```

cutKHgcdxf

The original version of the command, <code>cutKHgcdxf</code> 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:

```
664 дсру
           def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
                kh_angle, kh_distance):
                oXpos = self.xpos()
665 gcpy
                oYpos = self.ypos()
666 дсру
                self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
667 дсру
                   kh_start_depth, kh_max_depth, kh_angle, kh_distance)
                toolpath = self.cutline(self.xpos(), self.ypos(), -
668 дсру
                   kh_max_depth)
                self.setxpos(oXpos)
669 дсру
670 gcpy
                self.setypos(oYpos)
                if self.generatepaths == False:
671 gcpy
672 gcpy
                    return toolpath
673 gcpy
                else:
674 gcpy
                    return cube([0.001,0.001,0.001])
```

```
def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
676 gcpv
               kh_max_depth, kh_angle, kh_distance):
                oXpos = self.xpos()
oYpos = self.ypos()
677 gcpy #
678 gcpy #
679 gcpy #Circle at entry hole
               self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
    kh_tool_num, 7), 0, 90)
680 дсру
               681 gcpy
                   kh_tool_num, 7), 90,180)
               self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
682 gcpy
                   kh_tool_num, 7),180,270)
               self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
683 дсру
                   kh_tool_num, 7),270,360)
```

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

```
685 gcpy #pre-calculate needed values
686 дсру
               r = self.tool_radius(kh_tool_num, 7)
687 gcpy #
                print(r)
688 дсру
               rt = self.tool_radius(kh_tool_num, 1)
689 gcpy #
                print(rt)
690 gcpy
               ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                   tool_radius(kh_tool_num, 7))**2)
                print(ro)
691 gcpy #
               angle = math.degrees(math.acos(ro/rt))
692 gcpy
693 gcpy #Outlines of entry hole and slot
694 gcpy
               if (kh_angle == 0):
695 gcpy #Lower left of entry hole
                    self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
696 gcpy
                        tool_radius(kh_tool_num, 1),180,270)
697 gcpy \#Upper left of entry hole
698 дсру
                    self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                       tool_radius(kh_tool_num, 1),90,180)
699 gcpy #Upper right of entry hole
                     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
700 gcpy #
            41.810, 90)
701 gcpy
                    self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                       angle, 90)
702 gcpy #Lower right of entry hole
703 дсру
                    self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        270, 360-angle)
                     \verb|self.dxfarc(kh_tool_num|, \verb|self.xpos()|, \verb|self.ypos()|, \verb|self.|| \\
704 gcpy #
           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))))
705 gcpy #Actual line of cut
                     self.dxfline(kh_tool_num, self.xpos(),self.ypos(),self
706 gcpy #
           .xpos()+kh_distance,self.ypos())
707 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
708 дсру
                    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)
709 gcpy #lower right of end of slot
710 gcpy
                    \verb|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)
711 gcpy #upper right slot
712 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))
                     \verb|self.dxfline(kh_tool_num, self.xpos()+(sqrt((self.
713 gcpy #
           tool_diameter(kh_tool_num,1)^2)-(self.tool_diameter(kh_tool_num
           ,5)^2))/2), self.ypos()+self.tool_diameter(kh_tool_num, (
           \label{lem:hhmax_depth} $$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)
714 gcpy \#end position at top of slot
715 gcpy #lower right slot
716 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))
                 dxfline(kh_tool_num, self.xpos()+(sqrt((self.tool_diameter
717 gcpv #
           (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)
718 gcpy #end position at top of slot
719 gcpy #
                           hull(){
                                translate([xpos(), ypos(), zpos()]){
720 gcpy #
                                   keyhole_shaft(6.35, 9.525);
721 gcpy #
722 gcpy #
723 gcpy #
                                translate([xpos(), ypos(), zpos()-kh_max_depth]){
                                    keyhole_shaft(6.35, 9.525);
724 gcpy #
725 gcpy #
726 gcpy #
727 gcpy #
                           hull(){
                                translate([xpos(), ypos(), zpos()-kh_max_depth]){
728 gcpy #
                                   keyhole_shaft(6.35, 9.525);
729 gcpy #
730 gcpy #
731 gcpy #
                                translate\left( \texttt{[xpos()+kh\_distance, ypos(), zpos()-kh\_max\_depth]} \right)
                                    keyhole shaft (6.35, 9.525);
732 gcpy #
733 gcpy #
                           7
734 gcpy #
735 gcpy #
                           cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
736 gcpy #
                           cut with feed (\texttt{getxpos}() + \texttt{kh\_distance}, \texttt{getypos}(), -\texttt{kh\_max\_depth}, \texttt{feed}
                           setxpos(getxpos()-kh_distance);
737 gcpy #
                      } else if (kh_angle > 0 && kh_angle < 90) {
738 gcpy #
739 gcpy #//echo(kh_angle);
740 gcpy #
                     dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
                        kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_num);
                      {\tt dxfarc\,(getxpos\,()\,,getypos\,()\,,tool\_diameter\,(KH\_tool\_num\,,} \ \ (
741 gcpy #
                        \verb|kh_max_depth|)/2,180+\verb|kh_angle|,270+\verb|kh_angle|, KH_tool_num||;
742 gcpy #dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
                        kh_max_depth))/2,kh_angle+asin((tool_diameter(KH_tool_num, (
                        \verb|kh_max_depth+4.36|)/2)/(\verb|tool_diameter(KH_tool_num, (kh_max_depth)|)/2)/(\verb|tool_diameter(KH_tool_num, (kh_max_depth)|)/2)/(\verb|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, (kh_max_depth)|)/2)/(\|tool_num, 
                        ))/2)),90+kh_angle, KH_tool_num);
743 gcpy #dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
                        kh_max_depth))/2,270+kh_angle,360+kh_angle-asin((tool_diameter(
                        {\it KH\_tool\_num}\;,\;\;({\it kh\_max\_depth+4.36}))/2)/({\it tool\_diameter(KH\_tool\_num}\;,\;
                          (kh_max_depth))/2)), KH_tool_num);
744 gcpy \#dxfarc(getxpos()+(kh\_distance*cos(kh\_angle)),
                     getypos()+(kh_distance*sin(kh_angle)),tool_diameter(KH_tool_num,
745 gcpy #
                          (kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle,KH_tool_num);
746 gcpy \#dxfarc(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(kh_distance*cos(kh_angle))
                        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);
747 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))),
748 gcpy # getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(
                        \verb|kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))||
                        /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
749 gcpy # getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
                        , (kh_max_depth+4.36))/2)*sin(kh_angle)),
750 gcpy # getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(KH_tool_num_angle))+((tool_diameter(
                            (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
751 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
752 gcpy #//echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
753 gcpy #echo("Aangle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
                        )/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
754 gcpy #//echo(kh_angle);
755 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
                        kh_distance*sin(kh_angle)),-kh_max_depth,feed);
                                              toolpath = toolpath.union(self.cutline(self.xpos()+
756 gcpy #
                        kh_distance, self.ypos(), -kh_max_depth))
                                 elif (kh_angle == 90):
757 gcpy
758 gcpy #Lower left of entry hole
                                           self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
759 gcpy
                                                   kh_tool_num, 1),180,270)
760 gcpy #Lower right of entry hole
761 gcpy
                                           self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                                                   kh_tool_num, 1),270,360)
762 gcpy #left slot
763 gcpy
                                           self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
                                                   oYpos+kh_distance)
764 gcpy #right slot
765 gcpy
                                           self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
                                                   oYpos+kh_distance)
766 gcpy #upper left of end of slot
```

```
767 gcpy
                    \verb|self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r|\\
,90,180) 768 gcpy #upper right of end of slot
                    self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r
769 gcpy
,0,90)
770 gcpy #Upper right of entry hole
771 gcpy
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
772 gcpy #Upper left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
773 дсру
                       180)
                     toolpath = toolpath.union(self.cutline(oXpos, oYpos+
774 gcpy #
           \verb|kh_distance, -kh_max_depth|)|
               elif (kh_angle == 180):
775 gcpy
776 gcpy #Lower right of entry hole
                    \verb|self.dxfarc(kh_tool_num|, oXpos,oYpos,self.tool_radius(|
777 gcpy
                       kh_tool_num, 1),270,360)
778 gcpy #Upper right of entry hole
779 gcpy
                    self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                        kh_tool_num, 1),0,90)
780 gcpy #Upper left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
781 gcpy
                       angle)
782 gcpy #Lower left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
                        270)
784 gcpy #upper slot
                    self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
785 gcpy
                       kh_distance, oYpos-r)
786 gcpy #lower slot
787 дсру
                    self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                       kh distance, oYpos+r)
788 gcpy #upper left of end of slot
                    self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
                       ,90,180)
790 gcpy #lower left of end of slot
                    self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
791 gcpy
                       ,180,270)
                     toolpath = toolpath.union(self.cutline(oXpos-
792 gcpy #
           kh_distance, oYpos, -kh_max_depth))
               elif (kh_angle == 270):
793 gcpy
794 gcpy #Upper left of entry hole
                    self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
795 gcpy
                        kh_tool_num, 1),90,180)
796 gcpy #Upper right of entry hole
                    self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
797 gcpy
                        kh_tool_num, 1),0,90)
798 gcpy #left slot
799 дсру
                    self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                       oYpos-kh_distance)
800 gcpy \#right slot
                    self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                       oYpos-kh_distance)
802 gcpy #lower left of end of slot
                    \verb|self.dxfarc(kh_tool_num|, oXpos,oYpos-kh_distance,r|\\
803 gcpy
                        ,180,270)
804 gcpy #lower right of end of slot
805 дсру
                    self.dxfarc(kh_tool_num, oXpos,oYpos-kh_distance,r
                        ,270,360)
806 gcpy #lower right of entry hole
807 дсру
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                       angle)
808 gcpy #lower left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
809 gcpy
                       360)
                     toolpath = toolpath.union(self.cutline(oXpos, oYpos-
810 gcpy #
           kh_distance, -kh_max_depth))
                print(self.zpos())
811 gcpy #
812 gcpy #
                 self.setxpos(oXpos)
                 self.setypos(oYpos)
813 gcpy #
814 gcpy #
                 if self.generate paths == False:
815 gcpy #
                     return toolpath
816 gcpy
817 gcpy #
          } else if (kh_angle == 90) {
            //Lower left of entry hole
818 gcpy #
            dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
819 gcpy #
820 gcpy #
            //Lower right of entry hole
            {\tt dxfarc(getxpos(),getypos(),9.525/2,270,360,~KH\_tool\_num);}
821 gcpy #
822 gcpy #
            //Upper right of entry hole
```

```
{\tt dxfarc\,(getxpos\,()\,,getypos\,()\,,9.525/2\,,0\,,acos\,(tool\_diameter\,(}
823 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
824 gcpy #
             //Upper left of entry hole
             dxfarc(getxpos(),getypos(),9.525/2,180-acos(tool_diameter()
825 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180,KH_tool_num)
            //Actual line of cut
826 gcpy #
             {\tt dxfline}\,({\tt getxpos}\,()\,,{\tt getypos}\,()\,,{\tt getxpos}\,()\,,{\tt getypos}\,()\,+{\tt kh\_distance})\,;
827 gcpy #
828 gcpy #
             //upper right of slot
             dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
829 gcpy #
           //upper left of slot
830 gcpy #
             dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
831 gcpy #
           KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
             //right of slot
832 gcpy #
833 gcpy #
             dxfline(
834 gcpy #
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
835 gcpy #
                 getypos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
           tool\_diameter(\textit{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
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
836 gcpy #
837 gcpy #
             //end position at top of slot
                 getypos()+kh_distance,
838 gcpy #
839 gcpy #
                 KH_tool_num);
             dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))
840 gcpy #
           /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);
841 gcpy #
             hull(){
               translate([xpos(), ypos(), zpos()])\{
842 gcpy #
843 gcpy #
                 keyhole_shaft(6.35, 9.525);
844 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
845 gcpy #
                 keyhole_shaft(6.35, 9.525);
846 gcpy #
847 gcpy #
848 gcpy #
849 gcpy #
             hull(){
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
850 gcpy #
851 gcpy #
                 keyhole_shaft(6.35, 9.525);
852 gcpy #
               translate\left( \texttt{[xpos(), ypos()+kh\_distance, zpos()-kh\_max\_depth]} \right)
853 gcpy #
                 keyhole_shaft(6.35, 9.525);
854 gcpy #
               }
855 gcpy #
856 gcpy #
857 gcpy #
             cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
             cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed
858 gcpy #
           ):
            setypos(getypos()-kh_distance);
859 gcpy #
860 gcpy #
          } else if (kh_angle == 180) {
             //Lower right of entry hole
861 gcpy #
             dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
862 gcpy #
             //Upper right of entry hole
863 gcpy #
             dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
864 gcpy #
             //Upper left of entry hole
865 gcpy #
             dxfarc(getxpos(),getypos(),9.525/2,90, 90+acos(tool_diameter()
866 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
             //Lower left of entry hole
867 gcpy #
868 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
869 gcpy #
             //upper left of slot
             dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
870 gcpy #
           KH\_tool\_num, (kh\_max\_depth+6.35))/2,90,180, KH\_tool\_num);
871 gcpy #
             //lower left of slot
             dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
872 gcpy #
           KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
             //Actual line of cut
873 gcpy #
874 gcpy #
             \tt dxfline(getxpos(),getxpos()-kh\_distance,getypos());\\
875 gcpy #
             //upper left slot
876 gcpy #
            dxfline(
                 getxpos()-(sqrt((tool\_diameter(KH\_tool\_num,1)^2)-(
877 gcpy #
           tool_diameter(KH_tool_num, 5)^2))/2),
                 getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
878 gcpy #
            (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
           kh_{max_depth-6.34})/2)^2,
                 getxpos()-kh_distance,
879 gcpy #
```

```
880 gcpy #
             getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
881 gcpy #
882 gcpy #
                 KH_tool_num);
             //lower right slot
883 gcpy #
884 gcpy #
             dxfline(
                 getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
885 gcpy #
           tool_diameter(KH_tool_num,5)^2))/2),
                 \tt getypos()-tool\_diameter(KH\_tool\_num\,,\,\,(kh\_max\_depth))/2,//(
886 gcpy #
            (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
           kh_{max_depth-6.34})/2)^2,
887 gcpy #
                 getxpos()-kh_distance,
888 gcpy #
             //end position at top of slot
889 gcpy #
                 getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
890 gcpy #
                 KH_tool_num);
             hull(){
891 gcpy #
               translate([xpos(), ypos(), zpos()])\{
892 gcpy #
893 gcpy #
                 keyhole_shaft(6.35, 9.525);
894 gcpy #
895 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
896 gcpy #
                 keyhole_shaft(6.35, 9.525);
897 gcpy #
898 gcpy #
             hull(){
899 gcpy #
900 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth])\{
                 keyhole_shaft(6.35, 9.525);
901 gcpy #
902 gcpy #
903 gcpy #
               translate ([xpos()-kh\_distance, ypos(), zpos()-kh\_max\_depth])
           {
904 gcpy #
                 keyhole_shaft(6.35, 9.525);
905 gcpy #
906 gcpy #
907 gcpy #
             cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
908 gcpy #
             cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed
909 gcpy #
             setxpos(getxpos()+kh_distance);
          } else if (kh_angle == 270) {
910 gcpy #
             //Upper right of entry hole
911 gcpy #
             dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
912 gcpy #
913 gcpy #
             //Upper left of entry hole
             {\tt dxfarc\,(getxpos\,()\,,getypos\,()\,,9.525/2\,,90\,,180\,,\ KH\_tool\_num\,)}\,;
914 gcpy #
915 gcpy #
             //lower right of slot
             {\tt dxfarc\,(getxpos\,()\,,getypos\,()\,-kh\_distance\,,tool\_diameter\,(}
916 gcpy #
           {\tt KH\_tool\_num}\;,\;\;({\tt kh\_max\_depth+4.36}))/2\;,270\;,360\;,\;\;{\tt KH\_tool\_num})\;;
917 gcpy #
             //lower left of slot
             dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
918 gcpy #
           KH_tool_num, (kh_max_depth+4.36))/2,180,270, KH_tool_num);
919 gcpy #
             //Actual line of cut
             dxfline(getxpos(),getypos(),getxpos(),getypos()-kh_distance);
920 gcpy #
921 gcpy #
             //right of slot
922 gcpy #
             dxfline(
923 gcpy #
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
924 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,
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
925 gcpy #
926 gcpy #
             //end position at top of slot
927 gcpy #
                 getypos()-kh_distance,
928 gcpy #
                 KH_tool_num);
             //left of slot
929 gcpy #
930 gcpy #
             dxfline(
                 getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
931 gcpy #
932 gcpy #
                 \tt getypos()-(sqrt((tool\_diameter(KH\_tool\_num,1)^2)-(
           tool\_diameter(\textit{KH\_tool\_num}, 5) \, \hat{} \, 2))/2), //((kh\_max\_depth - 6.34))/2)
            ^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
                 getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
933 gcpy #
934 gcpy #
             //end position at top of slot
                 getypos()-kh_distance,
935 gcpy #
936 gcpy #
                 KH tool num);
             //Lower right of entry hole
937 gcpy #
938 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
939 gcpy #
             //Lower left of entry hole
             dxfarc(getxpos(),getypos(),9.525/2,180, 180+acos(tool_diameter
940 gcpy #
            (KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
             hull(){
941 gcpy #
942 gcpy #
               translate([xpos(), ypos(), zpos()]){
                 keyhole_shaft(6.35, 9.525);
943 gcpy #
```

```
944 gcpy #
              translate([xpos(), ypos(), zpos()-kh_max_depth]){
945 gcpy #
                keyhole_shaft(6.35, 9.525);
946 gcpy #
947 gcpy #
            7
948 gcpy #
            hull(){
949 gcpy #
950 gcpy #
              translate([xpos(), ypos(), zpos()-kh_max_depth]){
                keyhole_shaft(6.35, 9.525);
951 gcpy #
952 gcpy #
              translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
953 gcpy #
                 keyhole_shaft(6.35, 9.525);
954 gcpy #
              }
955 gcpy #
956 gcpy #
957 gcpy #
            cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
958 gcpy #
            cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed
959 gcpy #
            setypos(getypos()+kh_distance);
960 gcpy #
961 gcpy #}
```

3.4.4 Difference of Stock, Rapids, and Toolpaths

At the end of cutting it will be necessary to subtract the accumulated toolpaths and rapids from the stock. If in OpenSCAD, the 3D model is returned, causing it to be instantiated on the 3D stage unless the Boolean generatepaths is True.

```
963 дсру
            def stockandtoolpaths(self, option = "stockandtoolpaths"):
                if option == "stock":
964 дсру
965 gcpy
                    if self.generatepaths == False:
                         output(self.stock)
966 дсру
967 gcpy #
                         print("Outputting stock")
                    else:
968 gcpv
                        return self.stock
969 gcpy
970 gcpy
                elif option == "toolpaths":
971 gcpy
                    if self.generatepaths == False:
                        output(self.toolpaths)
972 gcpy
973 дсру
                    else:
974 дсру
                         return self.toolpaths
                elif option == "rapids":
975 gcpy
976 gcpy
                    if self.generatepaths == False:
                        output(self.rapids)
977 gcpy
978 gcpy
                    else:
979 gcpy
                         return self.rapids
980 дсру
                else:
                    part = self.stock.difference(self.toolpaths)
981 gcpy
                    if self.generatepaths == False:
982 gcpy
983 дсру
                         output(part)
984 дсру
                     else:
985 gcpy
                         return part
```

```
98 gcpscad module stockandtoolpaths(){
              gcp.stockandtoolpaths();
99 gcpscad
100 gcpscad }
101 gcpscad
102 gcpscad module stockwotoolpaths(){
             gcp.stockandtoolpaths("stock");
103 gcpscad
104 gcpscad }
105 gcpscad
106 gcpscad module outputtoolpaths(){
              gcp.stockandtoolpaths("toolpaths");
107 gcpscad
108 gcpscad }
109 gcpscad
110 gcpscad module outputrapids(){
              gcp.stockandtoolpaths("rapids");
111 gcpscad
112 gcpscad }
```

3.5 Output files

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

3.5.1 G-code Overview

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) G53GOZ-5.000
<pre>toolchange();</pre>	(TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
<pre>cutoneaxis_setfeed();</pre>	(PREPOSITION FOR RAPID PLUNGE) GOXOYO Z0.25 G1ZOF100 G1 X109.5 Y75 Z-8.35F400 Z9
<pre>cutwithfeed();</pre>	
<pre>closegcodefile();</pre>	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	<pre>rapid() rapid()</pre>
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	<pre>cutwithfeed(); cutwithfeed();</pre>
M05 M02	<pre>closegcodefile();</pre>

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).

3.5.2 DXF Overview

```
SECTION
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
```

3.5.3 Python and OpenSCAD File Handling

The class gcodepreview will need additional commands for opening files. The original implementation in RapSCAD used a command writeln — fortunately, this command is easily re-created in Python, though it is made as a separate file for each sort of file which may be opened. 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.

```
987 дсру
            def writegc(self, *arguments):
988 дсру
                 if self.generategcode == True:
989 дсру
                     line_to_write = ""
                     for element in arguments:
990 дсру
991 дсру
                         line_to_write += element
                     self.gc.write(line_to_write)
992 дсру
                     self.gc.write("\n")
993 дсру
994 дсру
995 дсру
            def writedxf(self, toolnumber, *arguments):
                 global dxfclosed
996 gcpy #
997 дсру
                 line_to_write = ""
                for element in arguments:
998 дсру
999 дсру
                     line_to_write += element
1000 дсру
                 if self.generatedxf == True:
                     if self.dxfclosed == False:
1001 дсру
                         self.dxf.write(line_to_write)
1002 gcpy
1003 дсру
                         self.dxf.write("\n")
                 if self.generatedxfs == True:
1004 дсру
                     self.writedxfs(toolnumber, line_to_write)
1005 дсру
1006 дсру
1007 дсру
            def writedxfs(self, toolnumber, line_to_write):
                  print("Processing writing toolnumber", toolnumber)
1008 gcpy #
                  line_to_write =
1009 gcpy #
1010 gcpy #
                  for element in arguments:
                      line_to_write += element
1011 gcpy #
                 if (toolnumber == 0):
1012 дсру
1013 gcpy
                     return
1014 gcpy
                 elif self.generatedxfs == True:
                     if (self.large_square_tool_num == toolnumber):
1015 дсру
1016 дсру
                         self.dxflgsq.write(line_to_write)
                          self.dxflgsq.write("\n")
1017 gcpy
1018 дсру
                     if (self.small_square_tool_num == toolnumber):
                          self.dxfsmsq.write(line_to_write)
1019 дсру
                         self.dxfsmsq.write("\n")
1020 gcpy
```

```
1021 дсру
                     if (self.large_ball_tool_num == toolnumber):
                          self.dxflgbl.write(line_to_write)
1022 gcpy
1023 дсру
                          self.dxflgbl.write("\n")
                     if (self.small_ball_tool_num == toolnumber):
1024 gcpy
1025 gcpy
                          self.dxfsmbl.write(line_to_write)
1026 дсру
                          self.dxfsmbl.write("\n")
                     if (self.large_V_tool_num == toolnumber):
1027 gcpy
                          self.dxflgV.write(line_to_write)
1028 gcpy
1029 дсру
                          self.dxflgV.write("\n")
                     if (self.small_V_tool_num == toolnumber):
1030 дсру
                          self.dxfsmV.write(line_to_write)
1031 gcpy
                          self.dxfsmV.write("\n")
1032 gcpy
                     if (self.DT_tool_num == toolnumber):
1033 дсру
                          self.dxfDT.write(line_to_write)
1034 дсру
1035 дсру
                          self.dxfDT.write("\n")
                     if (self.KH_tool_num == toolnumber):
1036 gcpy
                          self.dxfKH.write(line_to_write)
1037 gcpy
                          self.dxfKH.write("\n")
1038 дсру
                     if (self.Roundover_tool_num == toolnumber):
1039 дсру
1040 gcpy
                          self.dxfRt.write(line_to_write)
                          self.dxfRt.write("\n")
1041 gcpy
1042 gcpy
                     if (self.MISC_tool_num == toolnumber):
                          self.dxfMt.write(line_to_write)
1043 дсру
                          \verb|self.dxfMt.write("\n")|\\
1044 дсру
```

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).

opengcodefile

For writing to files it will be necessary to have commands for opening the files opengcodefile opendxffile and opendxffile and setting the associated defaults. 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.

There will need to be matching OpenSCAD modules for the Python functions:

```
114 gcpscad module opendxffile(basefilename){
115 gcpscad
              gcp.opendxffile(basefilename);
116 gcpscad }
117 gcpscad
118 gcpscad module opendxffiles(Base_filename, large_square_tool_num,
              small_square_tool_num, large_ball_tool_num, small_ball_tool_num,
              large_V_tool_num, small_V_tool_num, DT_tool_num, KH_tool_num,
Roundover_tool_num, MISC_tool_num) {
              gcp.opendxffiles(Base_filename, large_square_tool_num,
119 gcpscad
                  small_square_tool_num, large_ball_tool_num,
                  small_ball_tool_num, large_V_tool_num, small_V_tool_num,
                  DT_tool_num, KH_tool_num, Roundover_tool_num, MISC_tool_num)
120 gcpscad }
```

With matching OpenSCAD commands: opengcodefile for OpenSCAD: opengcodefile

```
122\ {\tt gcpscad}\ {\tt module}\ {\tt opengcodefile} ({\tt basefilename}\ ,\ {\tt currenttoolnum}\ ,\ {\tt toolradius}\ ,
                plunge, feed, speed) {
                 gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
123 gcpscad
                     plunge, feed, speed);
124 gcpscad }
```

and Python:

```
def opengcodefile(self, basefilename = "export",
1046 gcpy
1047 gcpy
                                 currenttoolnum = 102,
                                 toolradius = 3.175,
1048 дсру
                                 plunge = 400,
1049 дсру
1050 дсру
                                 feed = 1600.
                                 speed = 10000
1051 gcpy
1052 gcpy
                 self.basefilename = basefilename
1053 gcpy
1054 gcpy
                 self.currenttoolnum = currenttoolnum
                 self.toolradius = toolradius
1055 gcpy
1056 дсру
                 self.plunge = plunge
                 self.feed = feed
1057 gcpy
                 self.speed = speed
1058 дсру
                 if self.generategcode == True:
1059 gcpy
```

```
self.gcodefilename = basefilename + ".nc"
1060 дсру
                                    self.gc = open(self.gcodefilename, "w")
1061 gcpy
1062 дсру
                     def opendxffile(self, basefilename = "export"):
1063 gcpy
1064 gcpy
                            self.basefilename = basefilename
1065 gcpy #
                             global generatedxfs
                              global dxfclosed
1066 gcpy #
1067 дсру
                             self.dxfclosed = False
                            if self.generatedxf == True:
1068 дсру
                                    self.generatedxfs = False
1069 дсру
1070 дсру
                                    self.dxffilename = basefilename + ".dxf"
                                    self.dxf = open(self.dxffilename, "w")
1071 gcpy
1072 gcpy
                                    self.dxfpreamble(-1)
1073 дсру
1074 дсру
                     def opendxffiles(self, basefilename = "export",
1075 gcpy
                                                     large_square_tool_num = 0,
1076 gcpy
                                                     small_square_tool_num = 0,
1077 дсру
                                                     large_ball_tool_num = 0,
                                                     small_ball_tool_num = 0,
1078 дсру
                                                    large_V_tool_num = 0,
1079 gcpy
                                                     small_V_tool_num = 0,
1080 gcpy
1081 дсру
                                                    DT_tool_num = 0,
                                                    KH_tool_num = 0,
1082 gcpy
1083 дсру
                                                    Roundover_tool_num = 0,
                                                    {\tt MISC\_tool\_num} = {\tt O)}:
1084 дсру
                             global generatedxfs
1085 gcpy #
1086 дсру
                            self.basefilename = basefilename
1087 gcpy
                            self.generatedxfs = True
                            self.large_square_tool_num = large_square_tool_num
1088 дсру
                            self.small_square_tool_num = small_square_tool_num
1089 дсру
                            self.large_ball_tool_num = large_ball_tool_num
1090 gcpy
                            self.small_ball_tool_num = small_ball_tool_num
1091 gcpy
1092 дсру
                            self.large_V_tool_num = large_V_tool_num
                            self.small_V_tool_num = small_V_tool_num
1093 дсру
                            self.DT_tool_num = DT_tool_num
self.KH_tool_num = KH_tool_num
1094 дсру
1095 gcpy
                            self.Roundover_tool_num = Roundover_tool_num
1096 gcpy
                            self.MISC_tool_num = MISC_tool_num
if self.generatedxf == True:
1097 дсру
1098 дсру
1099 дсру
                                    if (large_square_tool_num > 0):
1100 gcpy
                                           self.dxflgsqfilename = basefilename + str(
                                                  large_square_tool_num) + ".dxf"
1101 gcpy #
                                             print("Opening ", str(self.dxflgsqfilename))
1102 gcpy
                                            self.dxflgsq = open(self.dxflgsqfilename, "w")
1103 дсру
                                    if (small_square_tool_num > 0):
                                           print("Opening small square")
self.dxfsmsqfilename = basefilename + str(
1104 gcpy #
1105 gcpy
                                                  small_square_tool_num) + ".dxf"
                                           self.dxfsmsq = open(self.dxfsmsqfilename, "w")
1106 gcpy
                                    if (large_ball_tool_num > 0):
1107 gcpy
                                            print("Opening large ball")
1108 gcpy #
                                           self.dxflgblfilename = basefilename + str(
1109 дсру
                                                  large_ball_tool_num) + ".dxf"
                                           self.dxflgbl = open(self.dxflgblfilename, "w")
1110 gcpy
                                    if (small_ball_tool_num > 0):
1111 gcpy
                                            print("Opening small ball")
1112 gcpy #
                                           self.dxfsmblfilename = basefilename + str(
1113 дсру
                                                  small_ball_tool_num) + ".dxf"
1114 дсру
                                            self.dxfsmbl = open(self.dxfsmblfilename, "w")
                                    if (large_V_tool_num > 0):
    print("Opening large V")
1115 дсру
1116 gcpy #
                                            self.dxflgVfilename = basefilename + str(
1117 gcpy
                                                  large_V_tool_num) + ".dxf"
                                           self.dxflgV = open(self.dxflgVfilename, "w")
1118 дсру
                                    if (small_V_tool_num > 0):
    print("Opening small V")
1119 дсру
1120 gcpy #
                                            self.dxfsmVfilename = basefilename + str(
1121 gcpy
                                                  small_V_tool_num) + ".dxf"
                                            self.dxfsmV = open(self.dxfsmVfilename, "w")
1122 gcpy
                                     \begin{tabular}{ll} \be
1123 дсру
                                            print("Opening DT")
1124 gcpy #
                                            self.dxfDTfilename = basefilename + str(DT_tool_num
1125 gcpy
                                                  ) + ".dxf"
1126 gcpy
                                            self.dxfDT = open(self.dxfDTfilename, "w")
                                    if (KH_tool_num > 0):
1127 дсру
1128 gcpy #
                                            print("Opening KH")
                                            self.dxfKHfilename = basefilename + str(KH_tool_num
1129 дсру
                                                  ) + ".dxf"
```

```
1130 дсру
                         self.dxfKH = open(self.dxfKHfilename, "w")
                     if (Roundover_tool_num > 0):
1131 gcpy
1132 gcpy #
                          print("Opening Rt")
                         self.dxfRtfilename = basefilename + str(
1133 gcpy
                             Roundover_tool_num) + ".dxf"
                         self.dxfRt = open(self.dxfRtfilename, "w")
1134 дсру
1135 дсру
                     if (MISC_tool_num > 0):
                          print("Opening Mt")
1136 gcpy #
1137 gcpy
                         self.dxfMtfilename = basefilename + str(
                             MISC_tool_num) + ".dxf"
                         self.dxfMt = open(self.dxfMtfilename, "w")
1138 дсру
```

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

```
1139 дсру
                                                                          if (large_square_tool_num > 0):
1140 gcpy
                                                                                          self.dxfpreamble(large_square_tool_num)
                                                                          if (small square tool num > 0):
1141 gcpy
1142 gcpy
                                                                                          self.dxfpreamble(small_square_tool_num)
                                                                          if (large_ball_tool_num > 0):
1143 дсру
1144 дсру
                                                                                         self.dxfpreamble(large_ball_tool_num)
                                                                          if (small_ball_tool_num > 0):
1145 gcpy
1146 дсру
                                                                                          self.dxfpreamble(small_ball_tool_num)
1147 дсру
                                                                          if (large_V_tool_num > 0):
1148 дсру
                                                                                         self.dxfpreamble(large_V_tool_num)
                                                                          if (small_V_tool_num > 0):
1149 gcpy
1150 gcpy
                                                                                          self.dxfpreamble(small_V_tool_num)
1151 gcpy
                                                                          if (DT_tool_num > 0):
1152 gcpy
                                                                                          self.dxfpreamble(DT_tool_num)
                                                                           \begin{tabular}{ll} \be
1153 gcpy
1154 дсру
                                                                                          self.dxfpreamble(KH_tool_num)
1155 gcpy
                                                                                   (Roundover_tool_num > 0):
                                                                                         self.dxfpreamble(Roundover_tool_num)
1156 gcpy
1157 дсру
                                                                          if (MISC_tool_num > 0):
1158 дсру
                                                                                          self.dxfpreamble(MISC_tool_num)
```

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

3.5.3.1 Writing to DXF 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 dxfwrite current tool. It will be necessary for the 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).

Each tool has a matching command for each tool/size combination:

```
• Ball nose, large (lgbl) writedxflgbl
writedxflgbl
writedxfsmbl
                 • Ball nose, small (smbl) writedxfsmbl
writedxflgsq
                 • Square, large (lgsq) writedxflgsq
                 • Square, small (smsq) writedxfsmsq
writedxfsmsq
                 • V, large (lgV) writedxflgV
 writedxflgV
                 ullet V, small (smV) writedxfsmV
 writedxfsmV
  writedxfKH
                 • Keyhole (KH) writedxfKH
                 • Dovetail (DT) writedxfDT
  writedxfDT
```

dxfpreamble 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.

```
def dxfpreamble(self, tn):

1161 gcpy # self.writedxf(tn,str(tn))

1162 gcpy self.writedxf(tn,"0")

1163 gcpy self.writedxf(tn,"SECTION")

1164 gcpy self.writedxf(tn,"2")

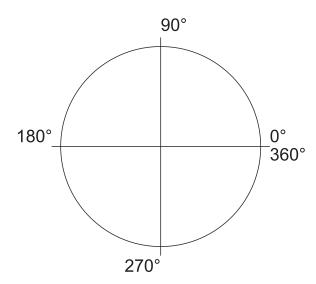
1165 gcpy self.writedxf(tn,"ENTITIES")
```

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

dxfline

- a line dvfline
- dxfarc
- ARC a notable option would be for the arc to close on itself, creating a circle: 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.

```
1167 gcpy
              def dxfline(self, tn, xbegin,ybegin,xend,yend):
                  self.writedxf(tn,"0")
1168 gcpy
                  self.writedxf(tn,"LWPOLYLINE")
self.writedxf(tn,"90")
1169 gcpy
1170 gcpy
                  self.writedxf(tn,"2")
1171 gcpy
                  self.writedxf(tn,"70")
1172 gcpy
                  self.writedxf(tn,"0")
1173 gcpy
1174 дсру
                  self.writedxf(tn,"43")
                  self.writedxf(tn,"0")
1175 gcpy
                  self.writedxf(tn,"10")
1176 gcpy
                  self.writedxf(tn,str(xbegin))
1177 gcpy
                  self.writedxf(tn,"20")
1178 дсру
                  self.writedxf(tn,str(ybegin))
1179 gcpy
                  self.writedxf(tn,"10")
1180 gcpy
                  \verb|self.writedxf(tn, \verb|str(xend)|)|
1181 gcpy
1182 дсру
                  self.writedxf(tn,"20")
1183 дсру
                  self.writedxf(tn,str(yend))
```

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, and to determine if the arc is clockwise or no $(G_2 \text{ vs. } G_3)$.

```
1188 дсру
                      self.writedxf(tn, "ARC")
                     self.writedxf(tn, "10")
1189 gcpy
1190 gcpy
                      self.writedxf(tn, str(xcenter))
                     self.writedxf(tn, "20")
1191 gcpy
1192 gcpy
                      self.writedxf(tn, str(ycenter))
1193 дсру
                      self.writedxf(tn, "40")
1194 дсру
                     self.writedxf(tn, str(radius))
                      self.writedxf(tn, "50")
self.writedxf(tn, str(anglebegin))
1195 gcpy
1196 дсру
                      self.writedxf(tn, "51")
1197 дсру
                      self.writedxf(tn, str(endangle))
1198 gcpy
1199 дсру
1200 gcpy
             def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
                 endangle):
                 if (self.generategcode == True):
1201 gcpy
                      self.writegc(tn, "(0)")
1202 gcpy
```

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

```
1204 дсру
            def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
                  global toolpath
1205 gcpy #
                  toolpath = self.currenttool()
1206 gcpy #
1207 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1208 gcpy
                    radius,0,90)
                 if (self.zpos == ez):
1209 gcpy
1210 дсру
                    self.settzpos(0)
                 else:
1211 gcpy
1212 gcpy
                     self.settzpos((self.zpos()-ez)/90)
1213 gcpy #
                 self.setxpos(ex)
1214 gcpy #
                 self.setypos(ey)
                  self.setzpos(ez)
1215 gcpy #
1216 gcpy
                 if self.generatepaths == True:
                     print("Unioning ucutarcNECCdxf utoolpath")
1217 gcpy
1218 дсру
                     self.arcloop(1,90, xcenter, ycenter, radius)
                      self.toolpaths = self.toolpaths.union(toolpath)
1219 gcpy #
                 \verb"else":
1220 gcpy
1221 gcpy
                     toolpath = self.arcloop(1,90, xcenter, ycenter, radius)
                      print("Returning cutarcNECCdxf toolpath")
1222 gcpy #
                     return toolpath
1223 gcpy
1224 дсру
1225 дсру
            def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
                  global toolpath
1226 gcpy #
1227 gcpy #
                  toolpath = self.currenttool()
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1228 gcpy #
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1229 gcpy
                    radius, 90, 180)
                 if (self.zpos == ez):
1230 gcpy
1231 дсру
                     self.settzpos(0)
1232 gcpy
                 else:
1233 дсру
                     self.settzpos((self.zpos()-ez)/90)
1234 gcpy #
                  self.setxpos(ex)
                 self.setypos(ey)
1235 gcpy #
1236 gcpy #
                 self.setzpos(ez)
1237 gcpy
                 if self.generatepaths == True:
                     self.arcloop(91,180, xcenter, ycenter, radius)
1238 дсру
                      self.toolpaths = self.toolpaths.union(toolpath)
1239 gcpy #
1240 gcpy
                 else:
                     toolpath = self.arcloop(91,180, xcenter, ycenter,
1241 gcpy
                         radius)
1242 gcpy
                     return toolpath
1243 gcpy
            def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1244 дсру
1245 gcpy #
                  global toolpath
                  toolpath = self.currenttool()
1246 gcpy #
1247 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
1248 дсру
                 self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius, 180, 270)
1249 gcpy
                 if (self.zpos == ez):
1250 дсру
                     self.settzpos(0)
1251 дсру
                 else:
                     self.settzpos((self.zpos()-ez)/90)
1252 gcpy
1253 gcpy #
                  self.setxpos(ex)
                 self.setypos(ey)
1254 gcpy #
```

```
1255 gcpy #
                  self.setzpos(ez)
                 if self.generatepaths == True:
1256 gcpy
1257 дсру
                     self.arcloop(181,270, xcenter, ycenter, radius)
                      self.toolpaths = self.toolpaths.union(toolpath)
1258 gcpy #
1259 gcpy
                 else:
                     toolpath = self.arcloop(181,270, xcenter, ycenter,
1260 дсру
                         radius)
                      return toolpath
1261 gcpy
1262 gcpy
             def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1263 gcpy
                  global toolpath
1264 gcpy #
                   toolpath = self.currenttool()
1265 gcpy #
1266 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
                 self.dxfarc(self.currenttoolnumber(). xcenter.vcenter.
1267 gcpy
                     radius,270,360)
1268 дсру
                 if (self.zpos == ez):
1269 дсру
                     self.settzpos(0)
                 else:
1270 gcpy
                     self.settzpos((self.zpos()-ez)/90)
1271 gcpy
1272 gcpy #
                 self.setxpos(ex)
1273 gcpy #
                  self.setypos(ey)
                  self.setzpos(ez)
1274 gcpy #
1275 gcpy
                 if self.generatepaths == True:
                      self.arcloop(271,360, xcenter, ycenter, radius)
1276 gcpy
1277 gcpy #
                      self.toolpaths = self.toolpaths.union(toolpath)
                 else:
1278 gcpy
1279 gcpy
                     toolpath = self.arcloop(271,360, xcenter, ycenter,
                         radius)
1280 gcpy
                      return toolpath
1281 gcpy
1282 gcpy
             def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1283 gcpy #
                  global toolpath
                   toolpath = self.currenttool()
1284 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1285 gcpy #
            self.zpos()])
                 self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1286 gcpy
                     radius,0,90)
                 if (self.zpos == ez):
1287 gcpy
                     self.settzpos(0)
1288 дсру
1289 дсру
                 else:
                     self.settzpos((self.zpos()-ez)/90)
1290 gcpy
1291 gcpy #
                  self.setxpos(ex)
                  self.setypos(ey)
1292 gcpy #
                  self.setzpos(ez)
1293 gcpy #
1294 дсру
                 if self.generatepaths == True:
                      self.narcloop(89,0, xcenter, ycenter, radius)
1295 gcpy
                       self.toolpaths = self.toolpaths.union(toolpath)
1296 gcpy #
                 else:
1297 gcpy
                     toolpath = self.narcloop(89,0, xcenter, ycenter, radius
1298 gcpy
1299 дсру
                      return toolpath
1300 дсру
1301 дсру
             \label{eq:def_def} \textbf{def} \ \texttt{cutarcSECWdxf} \ (\texttt{self} \ , \ \texttt{ex} \ , \ \texttt{ey} \ , \ \texttt{ez} \ , \ \texttt{xcenter} \ , \ \texttt{ycenter} \ , \ \texttt{radius}) :
                  global toolpath
1302 gcpy #
1303 gcpy #
                   toolpath = self.currenttool()
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1304 gcpy #
            self.zpos()])
                 self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1305 gcpy
                     radius,270,360)
                 if (self.zpos == ez):
1306 gcpy
1307 gcpy
                     self.settzpos(0)
1308 gcpy
                 else:
1309 дсру
                     self.settzpos((self.zpos()-ez)/90)
1310 gcpy #
                  self.setxpos(ex)
1311 gcpy #
                  self.setypos(ey)
                  self.setzpos(ez)
1312 gcpy #
1313 gcpv
                 if self.generatepaths == True:
                      self.narcloop(359,270, xcenter, ycenter, radius)
1314 дсру
1315 gcpy #
                       self.toolpaths = self.toolpaths.union(toolpath)
1316 дсру
                 else:
1317 дсру
                     toolpath = self.narcloop(359,270, xcenter, ycenter,
                         radius)
1318 gcpy
                      return toolpath
1319 дсру
             def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1320 дсру
                  global toolpath
1321 gcpy #
                  toolpath = self.currenttool()
1322 gcpy #
```

```
toolpath = toolpath.translate([self.xpos(),self.ypos(),
1323 gcpy #
            self.zpos()])
1324 gcpy
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius, 180, 270)
                if (self.zpos == ez):
1325 gcpy
1326 дсру
                    self.settzpos(0)
1327 дсру
                else:
                    self.settzpos((self.zpos()-ez)/90)
1328 gcpy
1329 gcpy #
                 self.setxpos(ex)
1330 gcpy #
                 self.setypos(ey)
1331 gcpy #
                 self.setzpos(ez)
                if self.generatepaths == True:
1332 gcpy
                     self.narcloop(269,180, xcenter, ycenter, radius)
1333 дсру
                     self.toolpaths = self.toolpaths.union(toolpath)
1334 gcpy #
1335 дсру
                else:
                     toolpath = self.narcloop(269,180, xcenter, ycenter,
1336 дсру
                        radius)
1337 дсру
                     return toolpath
1338 gcpy
            def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1339 gcpy
1340 gcpy #
                 global toolpath
1341 gcpy #
                  toolpath = self.currenttool()
                 toolpath = toolpath.translate([self.xpos(),self.ypos(),
1342 gcpy #
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1343 дсру
                    radius,90,180)
                if (self.zpos == ez):
1344 gcpy
1345 gcpy
                     self.settzpos(0)
1346 дсру
                else:
1347 дсру
                    self.settzpos((self.zpos()-ez)/90)
1348 gcpy #
                 self.setxpos(ex)
1349 gcpy #
                 self.setypos(ey)
1350 gcpy #
                 self.setzpos(ez)
                if self.generatepaths == True:
1351 gcpy
                     self.narcloop(179,90, xcenter, ycenter, radius)
1352 gcpy
                      self.toolpaths = self.toolpaths.union(toolpath)
1353 gcpy #
1354 дсру
                 else:
                     toolpath = self.narcloop(179,90, xcenter, ycenter,
1355 дсру
                        radius)
                     return toolpath
1356 дсру
```

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

cutarcNECCdxf(-stockXwidth/4, stockYheight/4+stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYheight/4, stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/16), stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYheight/16)

```
def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
    self.writegc("G03_\(\Delta\X\)", str(ex), "_\(\Delta\Y\)", str(ey), "_\(\Delta\Z\)", str(ez)
    , "_\(\Delta\R\)", str(radius))

def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
    self.writegc("G02_\(\Delta\X\)", str(ex), "_\(\Delta\Y\)", str(ey), "_\(\Delta\Z\)", str(ez)
    , "_\(\Delta\R\)", str(radius))
```

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

```
1364 дсру
            def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
1365 gcpy
                 if self.generatepaths == True:
1366 дсру
1367 дсру
                     self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
1368 gcpy
                 else:
                     return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
1369 дсру
                          radius)
1370 gcpv
            def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1371 gcpy
1372 gcpy
                \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
                 if self.generatepaths == False:
1373 дсру
1374 дсру
                     return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                          radius)
1375 gcpy
1376 gcpy
            def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
```

```
1377 дсру
                   self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1378 дсру
 1379 дсру
                       return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                             radius)
 1380 gcpy
               {\tt def} \ {\tt cutarcSECCdxfgc(self,\ ex,\ ey,\ ez,\ xcenter,\ ycenter,\ radius)}
 1381 дсру
                   \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
 1382 gcpy
                   if self.generatepaths == False:
 1383 gcpy
                       return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
 1384 дсру
                             radius)
 1385 gcpy
 1386 дсру
               def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1387 дсру
                   \verb|self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)|\\
 1388 gcpy
                   if self.generatepaths == False:
 1389 дсру
                       return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                             radius)
 1390 дсру
              def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1391 gcpy
 1392 дсру
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1393 дсру
 1394 дсру
                       \textbf{return} \ \texttt{self.cutarcSECWdxf} (\texttt{ex}, \ \texttt{ey}, \ \texttt{ez}, \ \texttt{xcenter}, \ \texttt{ycenter},
                             radius)
 1395 дсру
              def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1396 дсру
 1397 дсру
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1398 gcpy
                       return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
 1399 gcpy
                            radius)
 1400 дсру
              def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1401 gcpy
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
 1402 gcpy
 1403 gcpy
                   if self.generatepaths == False:
 1404 дсру
                       return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
126 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
127 gcpscad
              gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
128 gcpscad }
129 gcpscad
130 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
131 gcpscad
             gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
132 gcpscad }
133 gcpscad
134 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
              gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
135 gcpscad
136 gcpscad }
137 gcpscad
138 gcpscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
              gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
139 gcpscad
140 gcpscad }
```

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

```
def dxfpostamble(self,tn):
1406 gcpy
                   self.writedxf(tn,str(tn))
1407 gcpy #
1408 дсру
                  self.writedxf(tn,"0")
                  self.writedxf(tn,"ENDSEC")
self.writedxf(tn,"0")
1409 gcpy
1410 gcpy
                  self.writedxf(tn,"EOF")
1411 gcpy
1413 дсру
              def gcodepostamble(self):
1414 дсру
                  self.writegc("Z12.700")
                  self.writegc("M05")
1415 дсру
1416 дсру
                  self.writegc("M02")
```

dxfpostamble 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.

```
1418 дсру
            def closegcodefile(self):
                 {\tt self.gcodepostamble} ()
1419 gcpy
1420 дсру
                 self.gc.close()
1421 gcpy
1422 gcpy
            def closedxffile(self):
1423 gcpy
                 if self.generatedxf == True:
                      global dxfclosed
1424 gcpy #
                     self.dxfpostamble(-1)
1425 gcpv
                      self.dxfclosed = True
1426 gcpy #
1427 gcpy
                     self.dxf.close()
1428 gcpy
1429 gcpy
            def closedxffiles(self):
                 if self.generatedxfs == True:
1430 gcpy
1431 gcpy
                     if (self.large_square_tool_num > 0):
                          self.dxfpostamble(self.large_square_tool_num)
1432 gcpy
                     if (self.small_square_tool_num > 0):
1433 gcpy
                          self.dxfpostamble(self.small_square_tool_num)
1434 gcpy
1435 gcpy
                     if (self.large_ball_tool_num > 0):
1436 gcpy
                         self.dxfpostamble(self.large_ball_tool_num)
                     if (self.small_ball_tool_num > 0):
1437 gcpy
1438 дсру
                          self.dxfpostamble(self.small_ball_tool_num)
1439 дсру
                     if (self.large_V_tool_num > 0):
1440 gcpy
                          self.dxfpostamble(self.large_V_tool_num)
                     if (self.small_V_tool_num > 0):
1441 gcpy
1442 gcpy
                          self.dxfpostamble(self.small_V_tool_num)
                     if (self.DT_tool_num > 0):
1443 gcpy
1444 дсру
                         self.dxfpostamble(self.DT_tool_num)
1445 дсру
                     if (self.KH_tool_num > 0):
1446 дсру
                          self.dxfpostamble(self.KH_tool_num)
                     if (self.Roundover_tool_num > 0):
1447 gcpy
                         self.dxfpostamble(self.Roundover_tool_num)
1448 дсру
                     if (self.MISC_tool_num > 0):
1449 gcpy
1450 gcpy
                          self.dxfpostamble(self.MISC_tool_num)
1451 gcpy
1452 gcpy
                     if (self.large_square_tool_num > 0):
                          self.dxflgsq.close()
1453 gcpy
                     if (self.small_square_tool_num > 0):
1454 gcpy
1455 gcpy
                         self.dxfsmsq.close()
                     if (self.large_ball_tool_num > 0):
1456 gcpy
1457 gcpy
                          self.dxflgbl.close()
1458 дсру
                     if (self.small_ball_tool_num > 0):
1459 gcpy
                         self.dxfsmbl.close()
                     if (self.large_V_tool_num > 0):
1460 gcpy
                          self.dxflgV.close()
1461 gcpy
                     1462 gcpy
                         self.dxfsmV.close()
1463 gcpy
                     if (self.DT_tool_num > 0):
1464 gcpy
1465 дсру
                          self.dxfDT.close()
                     if (self.KH_tool_num > 0):
1466 gcpy
1467 gcpy
                         self.dxfKH.close()
1468 дсру
                     if (self.Roundover_tool_num > 0):
1469 gcpy
                         self.dxfRt.close()
                     if (self.MISC_tool_num > 0):
1470 gcpy
                         self.dxfMt.close()
1471 gcpy
```

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

4 Notes 61

4 Notes

Other Resources

Coding Style

A notable influence on the coding style in this project is John Ousterhout's *A Philosophy of Software Design*[SoftwareDesign]. Complexity is managed by the overall design and structure of the code, structuring it so that each component may be worked with on an individual basis, hiding the maximum information, and exposing the maximum functionality, with names selected so as to express their functionality/usage.

Red Flags to avoid include:

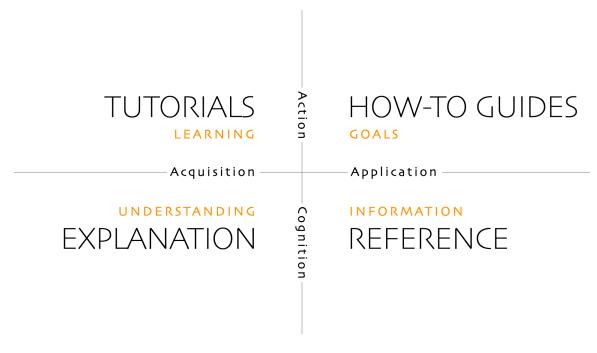
- Shallow Module
- Information Leakage
- Temporal Decomposition
- Overexposure
- Pass-Through Method
- Repetition
- Special-General Mixture
- Conjoined Methods
- Comment Repeats Code
- Implementation Documentation Contaminates Interface
- Vague Name
- Hard to Pick Name
- Hard to Describe
- Nonobvious Code

Documentation Style

https://diataxis.fr/ (originally developed at: https://docs.divio.com/documentation-system/) — divides documentation along two axes:

- Action (Practical) vs. Cognition (Theoretical)
- Acquisition (Studying) vs. Application (Working)

resulting in a matrix of:



where:

- 1. readme.md (Overview) Explanation (understanding-oriented)
- 2. Templates Tutorials (learning-oriented)
- 3. gcodepreview How-to Guides (problem-oriented)
- 4. Index Reference (information-oriented)

4 Notes 62

Holidays

Holidays are from https://nationaltoday.com/

DXFs

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

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/
https://ciechanow.ski/curves-and-surfaces/
https://www.youtube.com/watch?v=aVwxzDHniEw
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
- **Z**Y

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:

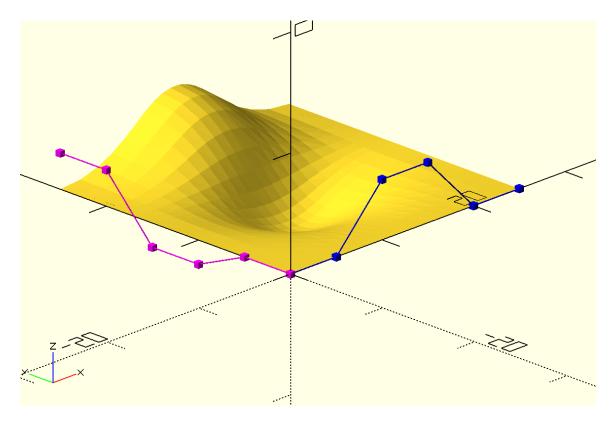
```
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs
```

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/

References

[ConstGeom]

[1981.
[MkCalc]	Horvath, Joan, and Rich Cameron. <i>Make: Calculus: Build models to learn, visualize, and explore.</i> First edition., Make: Community LLC, 2022.
[MkGeom]	Horvath, Joan, and Rich Cameron. <i>Make: Geometry: Learn by 3D Printing, Coding and Exploring</i> . First edition., Make: Community LLC, 2021.
[MkTrig]	Horvath, Joan, and Rich Cameron. <i>Make: Trigonometry: Build your way from triangles to analytic geometry.</i> First edition., Make: Community LLC, 2023.
[PractShopMath]	Begnal, Tom. <i>Practical Shop Math: Simple Solutions to Workshop Fractions, Formulas + Geometric Shapes.</i> 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. <i>A Philosophy of Software Design</i> . First Edition., Yaknyam Press, Palo Alto, Ca., 2018

Walmsley, Brian. Construction Geometry. 2d ed., Centennial College Press,

Command Glossary

```
settool settool(102). 25
setupstock setupstock(200, 100, 8.35, "Top", "Lower-left", 8.35). 23
```

Index

ballnose, 26	opengcodefile, 52
bowl tool, 26	•
	plunge, 32
closedxffile, 59, 60	. 1
closegcodefile, 59, 60	rapid, <mark>32</mark>
currenttoolnum, 22	rcs, 32
currenttoolnumber, 25	
currenttoolshape, 28	settool, 25
cut, 32, 34	setupstock, 23
cutarcCC, 36	gcodepreview, 23
cutarcCW, 36	setxpos, 23
cutkeyhole toolpath, 42	setypos, 23
cutKHgcdxf, 43	setzpos, 23
cutline, 34	speed, 32
	stepsizearc, 19
dovetail, 28	stepsizeroundover, 19
dxfarc, 55	subroutine
dxfline, 55	gcodepreview, 23
dxfpostamble, 60	writeln, 51
dxfpreamble, 54	
dxfwrite, 54	threadmill, 27
	tool diameter, 30
endmill square, 26	tool radius, 31
endmill v, 26	toolchange, 28
	tpzinc, 22
feed, 32	: 1 (DE
	writedxfDT, 54
gcodepreview, 21	writedxfKH, 54
writeln, 51	writedxflgbl, 54
gcp.setupstock, 23	writedxflgsq, 54
	writedxflgV, 54
init, 21	writedxfsmbl, 54
111.	writedxfsmsq, 54
keyhole, 27	writedxfsmV, 54
mpx, 22	vnos as
mpy, 22	xpos, 22
mpz, 22	Vnos 22
mpe, at	ypos, 22
opendxffile, 52	zpos, 22

Routines

ballnose, 26	opengcodefile, 52
bowl tool, 26	rapid as
closedxffile, 59, 60	rapid, 32 rcs, 32
closegcodefile, 59, 60 currenttoolnumber, 25	settool, 25
cut, 32, 34	setupstock, 23
cutarcCC, 36	setxpos, 23
cutarcCW, 36	setypos, 23 setzpos, 23
cutkeyhole toolpath, 42 cutKHgcdxf, 43	3ct2p03, 25
cutline, 34	threadmill, 27
dovetail, 28	tool diameter, 30 tool radius, 31
dxfarc, 55	toolchange, 28
dxfline, 55	
dxfpostamble, 60	writedxfDT, 54 writedxfKH, 54
dxfpreamble, 54 dxfwrite, 54	writedxflgbl, 54
uxiwiite, 54	writedxflgsq, 54
endmill square, 26	writedxflgV, 54
endmill v, 26	writedxfsmbl, 54 writedxfsmsq, 54
gcodepreview, 21, 23	writedxfsmV, 54
gcp.setupstock, 23	writeln, 51
init, 21	xpos, 22
loubala	-
keyhole, 27	ypos, 22
opendxffile, 52	zpos, 22

Variables

```
currenttoolnum, 22
currenttoolshape, 28

feed, 32

mpx, 22

mpy, 22

mpz, 22

tpzinc, 22

stepsizearc, 19

stepsizeroundover, 19
```