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

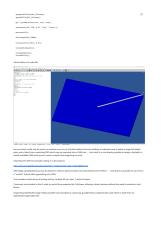
1	read	me.ma	2
2	Usa	ge and Templates	5
	2.1	gcpdxf.py	5
	2.2	gcodepreviewtemplate.py	7
	2.3	gcodepreviewtemplate.scad	12
3	gcoc	lepreview	17
	3.1	Module Naming Convention	17
		3.1.1 Parameters and Default Values	19
	3.2	Implementation files and gcodepreview class	19
		3.2.1 Position and Variables	22
		3.2.2 Initial Modules	23
	3.3	Tools and Changes	25
		3.3.1 3D Shapes for Tools	25
		3.3.1.1 Normal Tooling/toolshapes	25
		3.3.1.2 Tooling for Undercutting Toolpaths	26
		3.3.1.2.1 Keyhole tools	27
		3.3.1.2.2 Thread mills	27
		3.3.1.2.3 Dovetails	27
		3.3.1.3 Concave toolshapes	27
		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)	28
		3.3.2.3 Roundover (corner rounding)	28
		3.3.2.4 Dovetails	28 28
		3.3.2.5 toolchange routine	
		3.3.3 tooldiameter	30
	2.4	3.3.4 Feeds and Speeds	31
	3.4	3.4.1 Lines	31 33
		3.4.2 Arcs for toolpaths and DXFs	35
		3.4.3 Cutting shapes and expansion	<i>3</i> 2
		3.4.3.1 Building blocks	38
		3.4.3.2 List of shapes	38
		3.4.3.2.1 circles	40
		3.4.3.2.2 rectangles	40
		3.4.3.2.3 Keyhole toolpath and undercut tooling	41
		3.4.4 Difference of Stock, Rapids, and Toolpaths	48
	3.5	Output files	48
		3.5.1 G-code Overview	49
		3.5.2 DXF Overview	49
		3.5.3 Python and OpenSCAD File Handling	50
		3.5.3.1 Writing to DXF files	53
		3.5.3.2 Closings	58
	**		_
4	Note	28	60
In	dex		63
		tines	64
		ables	65

^{*}This file (gcodepreview) has version number vo.8, last revised 2025/o1/29.

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{!} [\texttt{OpenSCAD} \ \texttt{gcodepreview} \ \texttt{Unit} \ \texttt{Tests}] (\texttt{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
43 rdme
           opengcodefile (Gcode filename);
           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();
           closedxffile():
59 rdme
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,\ allowing\ a\ direct\ preview\ without\ the\ need\ 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 | ----- | ----- |
83 rdme | 0.1 | Version supports setting up stock, origin, rapid positioning, making cuts, and writing out matching G-code, and
           creating a DXF with polylines. 
 \mid
                - separate dxf files are written out for each tool where
84 rdme
           tool is ball/square/V and small/large (10/31/23)
            | - re-writing as a Literate Program using the LaTeX package
85 rdme |
           docmfp (begun 4/12/24)
            | - support for additional tooling shapes such as dovetail
          and keyhole tools
87 rdme \mid 0.2 \mid Adds support for arcs, specialty toolpaths such as Keyhole
           which may be used for dovetail as well as keyhole cutters |
```

1 readme.md 4

```
88 rdme \mid 0.3 \mid Support for curves along the 3rd dimension, roundover
          tooling
89 rdme | 0.4 | Rewrite using literati documentclass, suppression of SVG
          code, dxfrectangle
90 rdme \mid 0.5 \mid More shapes, consolidate rectangles, arcs, and circles in
          gcodepreview.scad
91 rdme | 0.6 | Notes on modules, change file for setupstock
92 rdme \mid 0.61\mid Validate all code so that it runs without errors 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
94 rdme | 0.8 | Re-re-write completely in Python and OpenSCAD, iteratively
          testing |
96 rdme Possible future improvements:
97 rdme
       - support for additional tooling shapes (bowl bits with flat
98 rdme
           bottom, tapered ball nose, lollipop cutters)
99 rdme - create a single line font for use where text is wanted
100 rdme - Support Bézier curves (required for fonts if not to be limited
           to lines and arcs) and surfaces
102 \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.
103 rdme
104 rdme Deprecated feature:
105 rdme
       - exporting SVGs --- coordinate system differences between
106 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)
```

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 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.

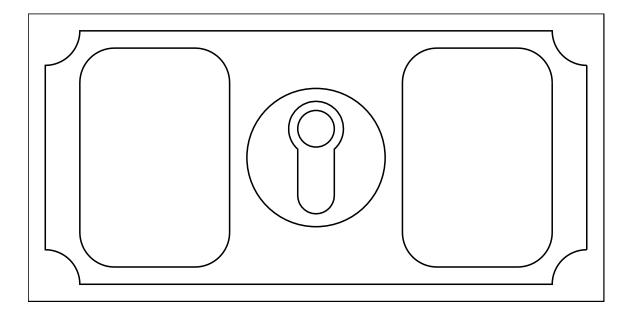
2.1 gcpdxf.py

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]
56 gcptmplpy # [CAM] */
57 gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 1570:1570]
58 gcptmplpy # [CAM] */
59 gcptmplpy MISC_tool_num = 0 #
60 gcptmplpy
61 gcptmplpy # [Feeds and Speeds] */
62 gcptmplpy plunge = 100
63 gcptmplpy # [Feeds and Speeds] */
64 gcptmplpy feed = 400
65 gcptmplpy # [Feeds and Speeds] */
66 gcptmplpy speed = 16000
67 gcptmplpy # [Feeds and Speeds] */
68 gcptmplpy small_square_ratio = 0.75 # [0.25:2]
69 gcptmplpy # [Feeds and Speeds] */
70 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
71 gcptmplpy # [Feeds and Speeds] */
72 gcptmplpy small_ball_ratio = 0.75 \# [0.25:2]
73 gcptmplpy # [Feeds and Speeds] */
74 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
75 gcptmplpy # [Feeds and Speeds] */
76 gcptmplpy small_V_ratio = 0.625 \# [0.25:2]
77 gcptmplpy # [Feeds and Speeds] */
78 gcptmplpy DT_ratio = 0.75 # [0.25:2]
79 gcptmplpy # [Feeds and Speeds] */
80 gcptmplpy KH_ratio = 0.75 \# [0.25:2]
81 gcptmplpy # [Feeds and Speeds] */
82 gcptmplpy RO_ratio = 0.5 # [0.25:2]
83 gcptmplpy # [Feeds and Speeds] */
84 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
85 gcptmplpy
86 gcptmplpy gcp = gcodepreview(generatepaths,
                                generategcode,
87 gcptmplpy
88 gcptmplpy
                                generatedxf,
89 gcptmplpy
90 gcptmplpy
91 gcptmplpy gcp.opengcodefile(Base_filename)
92 gcptmplpy gcp.opendxffile(Base_filename)
```

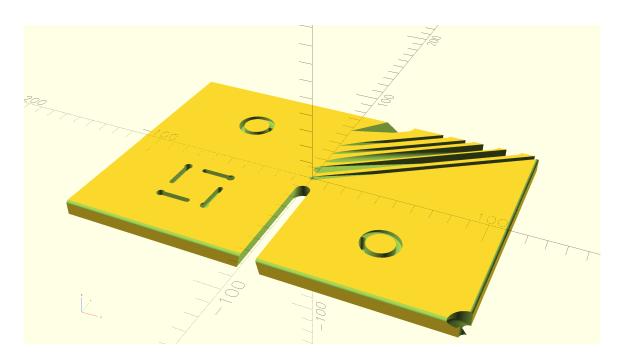
```
93 gcptmplpy gcp.opendxffiles(Base_filename,
                               large_square_tool_num,
 95 gcptmplpy
                               small_square_tool_num,
96 gcptmplpy
                               large_ball_tool_num,
97 gcptmplpy
                               small_ball_tool_num,
                               large_V_tool_num,
 98 gcptmplpy
                               small_V_tool_num,
99 gcptmplpy
                               DT_tool_num,
100 gcptmplpy
101 gcptmplpy
                               KH_tool_num,
102 gcptmplpy
                                Roundover_tool_num,
                               MISC_tool_num)
103 gcptmplpv
104~{\tt gcptmplpy}~{\tt gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top","}
                Center", retractheight)
105 gcptmplpy
106 gcptmplpy #print(pygcpversion())
107 gcptmplpy
108 gcptmplpy #print(gcp.myfunc(4))
109 gcptmplpy
110 gcptmplpy #print(gcp.getvv())
111 gcptmplpy
112 gcptmplpy \#ts = cylinder(12.7, 1.5875, 1.5875)
113 gcptmplpy \#toolpaths = gcp.cutshape(stockXwidth/2,stockYheight/2,-
                stockZthickness)
114 gcptmplpy
115 gcptmplpy gcp.movetosafeZ()
116 gcptmplpy
117 gcptmplpy gcp.toolchange(102,10000)
118 gcptmplpy
119 gcptmplpy #gcp.rapidXY(6,12)
120 gcptmplpy gcp.rapidZ(0)
121 gcptmplpy
122 gcptmplpy #print (gcp.xpos())
123 gcptmplpy #print (gcp.ypos())
124 gcptmplpy #psetzpos(7)
125 gcptmplpy \#gcp.setzpos(-12)
126 gcptmplpy #print (gcp.zpos())
127 gcptmplpy
128 gcptmplpy #print ("X", str(gcp.xpos()))
129 gcptmplpy #print ("Y", str(gcp.ypos()))
130 gcptmplpy #print ("Z", str(gcp.zpos()))
131 gcptmplpy
132 gcptmplpy toolpaths = gcp.currenttool()
133 gcptmplpy
134 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2,stockYheight/2,-
               stockZthickness)
135 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
                stockYheight/2, -stockZthickness))
136 gcptmplpy
137 gcptmplpy gcp.rapidZ(retractheight)
138 gcptmplpy gcp.toolchange(201,10000)
139 gcptmplpy gcp.rapidXY(0, stockYheight/16)
140 gcptmplpy gcp.rapidZ(0)
141 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
                stockYheight/2, -stockZthickness))
142 gcptmplpy
143 gcptmplpy gcp.rapidZ(retractheight)
144 gcptmplpy gcp.toolchange(202,10000)
145 gcptmplpy gcp.rapidXY(0, stockYheight/8)
146 gcptmplpy gcp.rapidZ(0)
147 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
                stockYheight/2, -stockZthickness))
148 gcptmplpy
149 gcptmplpy gcp.rapidZ(retractheight)
150 gcptmplpy gcp.toolchange(101,10000)
151 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
152 gcptmplpy gcp.rapidZ(0)
153 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
                stockYheight/2, -stockZthickness))
154 gcptmplpy
155 gcptmplpy gcp.setzpos(retractheight)
156 gcptmplpy gcp.toolchange(390,10000)
157 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
158 gcptmplpy gcp.rapidZ(0)
159 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
               stockYheight/2, -stockZthickness))
160 gcptmplpy gcp.rapidZ(retractheight)
161 gcptmplpy
162 gcptmplpy gcp.toolchange(301,10000)
```

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

```
stockZthickness/2))
220 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()+
                       stockYheight/9, gcp.ypos(), gcp.zpos()))
221 gcptmplpy #below should probably be cutlinegc
222 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()-stockYheight/9,
                       gcp.ypos(), gcp.zpos()))
223 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
224 gcptmplpv
225 gcptmplpy \#key = gcp.cutkeyholegcdxf(KH_tool_num, 0, stockZthickness*0.75, "E
                        ", stockYheight/9)
226 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
                       stockYheight/9)
227 gcptmplpy #toolpaths = toolpaths.union(key)
228 gcptmplpy
229 gcptmplpy gcp.rapidZ(retractheight)
230~{\rm gcptmplpy}~{\rm gcp.rapidXY} (-{\rm stockXwidth/4} + {\rm stockXwidth/16}\,,~-({\rm stockYheight/4} + {\rm stockXwidth/16}\,))
                       stockYheight/16))
231 gcptmplpy gcp.rapidZ(0)
232 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                       stockZthickness*0.75, "N", stockYheight/9))
233 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                       stockZthickness/2))
234 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                       +stockYheight/9, gcp.zpos()))
235 gcptmplpy #below should probably be cutlinegc
236 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()-
                       stockYheight/9, gcp.zpos()))
237 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
238 gcptmplpy
239 gcptmplpy gcp.rapidZ(retractheight)
240 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-stockXwidth/16), -(stockYwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockYheight/4-stockXwidth/16), -(stockXwidth/16), -(stockXwidth/16), -(stockXwidth/16), -(stockXwidth/16), -(stockXwidth/16), -(st
                       stockYheight/8))
241 gcptmplpy gcp.rapidZ(0)
242 gcptmplpy \#toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0, 1))
                       stockZthickness*0.75, "W", stockYheight/9))
243 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                       stockZthickness/2))
244 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()-
                       stockYheight/9, gcp.ypos(), gcp.zpos()))
245 gcptmplpy #below should probably be cutlinegc
246 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()+stockYheight/9,
                       gcp.ypos(), gcp.zpos()))
247 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
248 gcptmplpy
249 gcptmplpy gcp.rapidZ(retractheight)
250~{\rm gcptmplpy}~{\rm gcp.rapidXY} (-{\rm stockXwidth/4-stockXwidth/16}\,,~-({\rm stockYheight/4-stockXwidth/16}\,)
                       stockYheight/8))
251 gcptmplpy gcp.rapidZ(0)
252 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                       stockZthickness*0.75, "S", stockYheight/9))
253 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                       stockZthickness/2))
254 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                       -stockYheight/9, gcp.zpos()))
255 gcptmplpy \#below\ should\ probably\ be\ cutlinegc
256 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()+
                       stockYheight/9, gcp.zpos()))
257 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
259 gcptmplpy gcp.rapidZ(retractheight)
260 gcptmplpy gcp.toolchange(56142,10000)
261 gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
262 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
263 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                        -1.531))
264 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
                         -(stockYheight/2+0.508/2), -1.531))
265 gcptmplpv
{\tt 266~gcptmplpy~gcp.rapidZ(retractheight)}
267 gcptmplpy \#gcp.toolchange(56125,10000)
268 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
269 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                        -1.531))
270 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
                         (stockYheight/2+0.508/2), -1.531))
271 gcptmplpy
272 gcptmplpy
273 gcptmplpy part = gcp.stock.difference(toolpaths)
```

```
274 gcptmplpy
275 gcptmplpy output (part)
276 gcptmplpy #output(test)
277 gcptmplpy #output (key)
278 gcptmplpy #output(dt)
279 gcptmplpy #gcp.stockandtoolpaths()
280 gcptmplpy #gcp.stockandtoolpaths("stock")
281 gcptmplpy #output (gcp.stock)
282 gcptmplpy #output (gcp.toolpaths)
283 gcptmplpy #output (toolpaths)
284 gcptmplpy
285 gcptmplpy \#gcp.makecube(3, 2, 1)
286 gcptmplpy #
287 gcptmplpy #gcp.placecube()
288 gcptmplpy #
289 gcptmplpy \#c = gcp.instantiatecube()
290 gcptmplpy #
291 gcptmplpy #output(c)
292 gcptmplpy
293 gcptmplpy gcp.closegcodefile()
294 gcptmplpy gcp.closedxffiles()
295 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 gcptmp1 //!OpenSCAD
2 gcptmp1
3 gcptmp1 use <gcodepreview.py>
4 gcptmp1 include <gcodepreview.scad>
5 gcptmp1
6 gcptmp1 $fa = 2;
7 gcptmp1 $fs = 0.125;
8 gcptmp1 fa = 2;
9 gcptmp1 fs = 0.125;
10 gcptmp1
11 gcptmp1 /* [Stock] */
12 gcptmp1 stockXwidth = 219;
13 gcptmp1 /* [Stock] */
14 gcptmp1 stockYheight = 150;
15 gcptmp1 /* [Stock] */
```

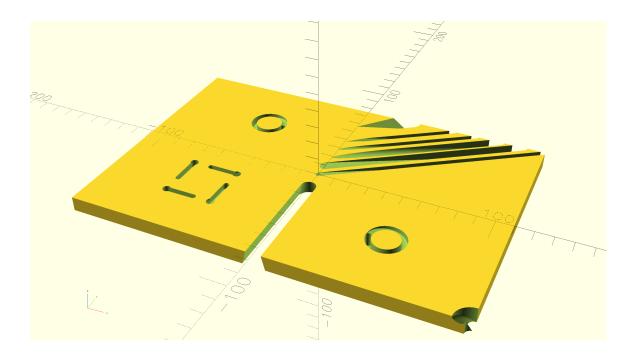
```
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
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]
51 gcptmpl /* [CAM] */
52 gcptmpl Roundover_tool_num = 0; // [56142:56142, 56125:56125, 1570:1570] 53 gcptmpl /* [CAM] */
54 gcptmpl MISC_tool_num = 0; // [648:648]
55 gcptmpl
56 gcptmpl /* [Feeds and Speeds] */
57 gcptmpl plunge = 100;
58 gcptmpl /* [Feeds and Speeds] */
59 gcptmpl feed = 400;
60 gcptmpl /* [Feeds and Speeds] */
61 gcptmpl speed = 16000;
62 gcptmpl /* [Feeds and Speeds] */
63 gcptmpl small_square_ratio = 0.75; // [0.25:2]
64 gcptmpl /* [Feeds and Speeds] */
65 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
66 gcptmpl /* [Feeds and Speeds] */
67 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
68 gcptmpl /* [Feeds and Speeds] */
69 gcptmpl large_V_ratio = 0.875; // [0.25:2]
70 gcptmpl /* [Feeds and Speeds] */
71 gcptmpl small_V_ratio = 0.625; // [0.25:2]
72 gcptmpl /* [Feeds and Speeds] */
73 gcptmpl DT_ratio = 0.75; // [0.25:2]
74 gcptmpl /* [Feeds and Speeds] */
75 gcptmpl KH_ratio = 0.75; // [0.25:2]
76 gcptmpl /* [Feeds and Speeds] */
77 gcptmpl RO_ratio = 0.5; // [0.25:2]
78 gcptmpl /* [Feeds and Speeds] */
79 gcptmpl MISC_ratio = 0.5; // [0.25:2]
80 gcptmpl
81 gcptmpl thegeneratepaths = generatepaths == true ? 1 : 0;
82 gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
83 gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
84 gcptmpl
85 gcptmpl gcp = gcodepreview(thegeneratepaths,
86 gcptmpl
                               thegenerategcode,
87 gcptmpl
                               thegeneratedxf,
88 gcptmpl
89 gcptmpl
90 gcptmpl opengcodefile(Base_filename);
91 gcptmpl opendxffile(Base_filename);
92 gcptmpl opendxffiles(Base_filename,
```

```
93 gcptmpl
                             large_square_tool_num,
94 gcptmpl
                             small_square_tool_num,
                            large_ball_tool_num,
small_ball_tool_num,
95 gcptmpl
96 gcptmpl
97 gcptmpl
                            large_V_tool_num,
                             small_V_tool_num,
98 gcptmpl
                             DT_tool_num,
99 gcptmpl
                             KH_tool_num ,
100 gcptmpl
101 gcptmpl
                             Roundover_tool_num,
102 gcptmpl
                             MISC_tool_num);
103 gcptmpl
104 gcptmpl setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
             stockzero);
105 gcptmpl
106 gcptmpl //echo(gcp);
107 gcptmpl //gcpversion();
108 gcptmpl
109 gcptmpl //c = myfunc(4);
110 gcptmpl //echo(c);
111 gcptmpl
112 gcptmpl //echo(getvv());
113 gcptmpl
114 gcptmpl cutline(stockXwidth/2,stockYheight/2,-stockZthickness);
115 gcptmpl
116 gcptmpl rapidZ(retractheight);
117 gcptmpl toolchange(201,10000);
118 gcptmpl rapidXY(0, stockYheight/16);
119 gcptmpl rapidZ(0);
120 gcptmpl cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness);
121 gcptmpl
122 gcptmpl
123 gcptmpl rapidZ(retractheight);
124 gcptmpl toolchange(202,10000);
125 gcptmpl rapidXY(0, stockYheight/8);
126 gcptmpl rapidZ(0);
127 gcptmpl cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
128 gcptmpl
129 gcptmpl rapidZ(retractheight);
130 gcptmpl toolchange(101,10000);
131 gcptmpl rapidXY(0, stockYheight/16*3);
132 gcptmpl rapidZ(0);
133 gcptmpl cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
134 gcptmpl
135 gcptmpl rapidZ(retractheight);
136 gcptmpl toolchange(390,10000);
137 gcptmpl rapidXY(0, stockYheight/16*4);
138 gcptmpl rapidZ(0);
139 gcptmpl
140 gcptmpl cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
141 gcptmpl rapidZ(retractheight);
142 gcptmpl
143 gcptmpl toolchange(301,10000);
144 gcptmpl rapidXY(0, stockYheight/16*6);
145 gcptmpl rapidZ(0);
146 gcptmpl
147 gcptmpl cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
148 gcptmpl
149 gcptmpl
150 gcptmpl movetosafeZ();
151 gcptmpl rapid(gcp.xpos(),gcp.ypos(),retractheight);
152 gcptmpl toolchange(102,10000);
153 gcptmpl
154 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
             stockYheight/8);
155 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
              stockYheight/16
156 gcptmpl rapidZ(0);
157 gcptmpl
158 gcptmpl //cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16,-stockZthickness);
159 gcptmpl //gcp.cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16))
160 gcptmpl cutarcCC(0,90, gcp.xpos()-stockYheight/16, gcp.ypos(), stockYheight
              /16, -stockZthickness/4);
161 gcptmpl cutarcCC(90,180, gcp.xpos(), gcp.ypos()-stockYheight/16,
             stockYheight/16, -stockZthickness/4);
162 gcptmpl cutarcCC(180,270, gcp.xpos()+stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
```

```
163 gcptmpl cutarcCC(270,360, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
164 gcptmpl
165 gcptmpl movetosafeZ();
166 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
              stockYheight/8);
167 gcptmpl rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
168 gcptmpl rapidZ(0);
169 gcptmpl
170 gcptmpl cutarcCW(180,90, gcp.xpos()+stockYheight/16, gcp.ypos(),
             stockYheight/16, -stockZthickness/4);
171~\text{gcptmpl cutarcCW} (90,0,~\text{gcp.xpos()},~\text{gcp.ypos()} - \text{stockYheight}/16\,,~\text{stockYheight}/10\,)
              /16, -stockZthickness/4);
172 gcptmpl cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
             stockYheight/16, -stockZthickness/4);
173 gcptmpl cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
174 gcptmpl
175 gcptmpl movetosafeZ();
176 gcptmpl toolchange(201, 10000);
177 gcptmpl rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
178 gcptmpl rapidZ(0);
179 gcptmpl cutarcCW(180, 90, stockXwidth /2 , -stockYheight/2, 6.34, -
              stockZthickness);
180 gcptmpl
181 gcptmpl movetosafeZ();
182 gcptmpl rapidXY(stockXwidth/2, -stockYheight/2);
183 gcptmpl rapidZ(0);
184 gcptmpl
185 gcptmpl gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);
186 gcptmpl
187 gcptmpl movetosafeZ();
188 gcptmpl toolchange(814, 10000);
189 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
190 gcptmpl rapidZ(0);
191 gcptmpl
192 gcptmpl cutlinedxfgc(xpos(), ypos(), -stockZthickness);
193 gcptmpl cutlinedxfgc(xpos(), -12.7, -stockZthickness);
194 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
196 gcptmpl //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
197 gcptmpl //rapidZ(0);
198 gcptmpl
199 gcptmpl //movetosafeZ();
200 gcptmpl //toolchange(374, 10000);
201 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 + ^{2}
              stockYheight/16))
202 gcptmpl
203 gcptmpl //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
204 gcptmpl //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
205 gcptmpl //cutline(xpos() - stockYheight /9, ypos(), zpos());
206 gcptmpl //cutline(xpos(), ypos(), 0);
207 gcptmpl
208 gcptmpl movetosafeZ();
209 gcptmpl
210 gcptmpl toolchange(374, 10000);
211 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
              stockYheight/16))
212 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 + ^{2}
             stockYheight/16))
213 gcptmpl rapidZ(0);
214 gcptmpl
215 gcptmpl cutline(xpos(), ypos(), (stockZthickness/2) * -1);
216 gcptmpl cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
217 gcptmpl cutline(xpos() - stockYheight /9, ypos(), zpos());
218 gcptmpl cutline(xpos(), ypos(), 0);
219 gcptmpl
220 gcptmpl rapidZ(retractheight);
221 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+ ^{\circ}
              stockYheight/16));
222 gcptmpl rapidZ(0);
223 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
224 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
225 gcptmpl cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
226 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
227 gcptmpl
228 gcptmpl rapidZ(retractheight);
229 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
```

```
stockYheight/8));
230 gcptmpl rapidZ(0);
231 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
232 gcptmpl cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos());
233 gcptmpl cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos());
234 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
235 gcptmpl
236 gcptmpl rapidZ(retractheight);
237 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
              stockYheight/8));
238 gcptmpl rapidZ(0);
239 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
240 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
241 gcptmpl cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos()); 242 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
243 gcptmpl
244 gcptmpl
245 gcptmpl
246 gcptmpl rapidZ(retractheight);
247 gcptmpl gcp.toolchange(56142,10000);
248 gcptmpl gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2));
249 gcptmpl cutZgcfeed(-1.531,plunge);
250 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);
251 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
               -1.531);
252 gcptmpl
253 gcptmpl rapidZ(retractheight);
254 gcptmpl //#gcp.toolchange(56125,10000)
255 gcptmpl cutZgcfeed(-1.531,plunge);
256 gcptmpl //toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
              -1.531))
257 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
              -1.531);
258 gcptmpl
259 gcptmpl stockandtoolpaths();
260 gcptmpl //stockwotoolpaths();
261 gcptmpl //outputtoolpaths();
262 gcptmpl
263 gcptmpl //makecube(3, 2, 1);
264 gcptmpl
265 gcptmpl //instantiatecube();
266 gcptmpl
267 gcptmpl closegcodefile();
268 gcptmpl closedxffiles();
269 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 <code>cut...</code> 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
- OpenSCAD see: 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}
\begin{writecode}{a}{pygcodepreview.scad}{scad}
module oFOOBAR(...) {
   pFOOBAR(...);
\end{writecode}
\addtocounter{pyscad}{4}
which in turn calls the internal Python definitioon \DescribeSubroutine{FOOBAR}{pFOOBAR}
\lstset{firstnumber=\thegcpy}
\begin{writecode}{a}{gcodepreview.py}{python}
def pFOOBAR (...)
\end{writecode}
\addtocounter{gcpy}{3}
```

Further note that this 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.
          scad
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:
          from openscad import *
12 gcpy
13 gcpy except ModuleNotFoundError as e:
14 дсру
          print("OpenSCAD_module_not_loaded.")
15 дсру
16 gcpy # add math functions (using radians by default, convert to degrees
          where necessary)
17 gcpy {\tt import} math
18 дсру
19 gcpy def pygcpversion():
          the gcp version = 0.8
20 дсру
          return thegcpversion
21 дсру
```

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

```
1 gcpscad //!OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview version 0.8
4 gcpscad //
5 gcpscad //used via include <gcodepreview.scad>;
6 gcpscad //
7 gcpscad
8 gcpscad use <gcodepreview.py>
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
26 дсру
                         generatepaths = False,
                         generategcode = False,
27 дсру
28 дсру
                         generatedxf = False,
                          stockXwidth = 25,
29 gcpy #
30 gcpy #
                          stockYheight = 25,
31 gcpy #
                          stockZthickness = 1,
                          zeroheight = "Top",
stockzero = "Lower-left" ,
32 gcpy #
33 gcpy #
34 gcpy #
                          retractheight = 6,
                          currenttoolnum = 102,
35 gcpy #
36 gcpy #
                          toolradius = 3.175,
37 gcpy #
                          plunge = 100,
38 gcpy #
                          feed = 400,
39 gcpy #
                          speed = 10000
                          ):
40 дсру
                self.basefilename = basefilename
41 gcpy #
               if (generatepaths == 1):
42 дсру
43 дсру
                   self.generatepaths =
               if (generatepaths == 0):
44 дсру
45 дсру
                    self.generatepaths = False
46 дсру
               else:
47 дсру
                   self.generatepaths = generatepaths
               48 дсру
49 дсру
                    self.generategcode = True
               if (generategcode == 0):
50 дсру
                   self.generategcode = False
51 дсру
52 дсру
               else:
53 дсру
                    self.generategcode = generategcode
               if (generatedxf == 1):
54 дсру
                    self.generatedxf = True
55 дсру
               if (generatedxf == 0):
56 дсру
                   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
63 gcpy #
                self.zeroheight = zeroheight
                self.stockzero = stockzero
64 gcpy #
65 gcpy #
                self.retractheight = retractheight
66 gcpy #
                self.currenttoolnum = currenttoolnum
                self.toolradius = toolradius
67 gcpy #
68 gcpy #
                self.plunge = plunge
69 gcpy #
                self.feed = feed
                self.speed = speed
70 gcpy #
                global toolpaths
71 gcpy #
                if (openscadloaded == True):
72 gcpy #
                    self.toolpaths = cylinder(0.1, 0.1)
73 gcpy #
74 дсру
               self.generatedxfs = False
75 дсру
76 gcpy
           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 #
            {\tt def getvv(self):}
83 gcpy #
84 gcpy #
                return self.vv
85 gcpy #
            def checkint(self):
86 gcpy #
87 gcpy #
                return self.mc
88 gcpy #
            def makecube(self, xdim, ydim, zdim):
89 gcpy #
90 gcpy #
                self.c=cube([xdim, ydim, zdim])
91 gcpy #
            def placecube(self):
92 gcpy #
                output(self.c)
93 gcpy #
94 gcpy #
95 gcpy #
            def instantiatecube(self):
96 gcpy #
                return self.c
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 gcpy 103 gcpy # global mpy 104 дсру return self.mpy 105 дсру 106 дсру def zpos(self): 107 gcpy # global mpz 108 дсру return self.mpz 109 дсру 110 gcpy # def tpzinc(self): global tpzinc 111 gcpy # 112 gcpy # return self.tpzinc

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

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

 ${\tt setxpos} \ \ {\tt and} \ \ {\tt in} \ \ {\tt turn}, \ {\tt functions} \ \ {\tt which} \ \ {\tt set} \ \ {\tt the} \ \ {\tt positions}; \ \ {\tt setxpos}, \ \ {\tt setxpos}, \ \ {\tt and} \ \ {\tt setzpos}.$

setypos setzpos

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

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 $\verb"gcp.setupstock". It requires that the user set parameters$ 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.

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

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 comments using the terms described for CutViewer.

```
if self.zeroheight == "Top":
160 дсру
                                                          if self.stockzero == "Lower-Left":
161 gcpy
                                                                       self.stock = stock.translate([0,0,-self.
162 gcpy
                                                                                  stockZthickness])
                                                                       if self.generategcode == True:
163 gcpy
                                                                                   self.writegc("(stockMin:0.00mm,_{\sqcup}0.00mm,_{\sqcup}-",str(
164 дсру
                                                                                             self.stockZthickness),"mm)")
                                                                                    self.writegc("(stockMax:",str(self.stockXwidth)
165 gcpy
                                                                                                ,"mm,_{\sqcup}", str(stockYheight),"mm,_{\sqcup}0.00mm)")
                                                                                    self.writegc("(STOCK/BLOCK, ", str(self.
166 gcpy
                                                                                               stockXwidth),",",str(self.stockYheight),",
                                                                                               ", str(self.stockZthickness), ", \u0.00, \u0.0
                                                          if self.stockzero == "Center-Left":
167 дсру
                                                                       self.stock = self.stock.translate([0,-stockYheight
168 gcpy
                                                                                   / 2,-stockZthickness])
                                                                       if self.generategcode == True:
169 gcpy
                                                                                    self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
170 дсру
                                                                                               stockYheight/2), "mm, _-", str(self.
                                                                                              stockZthickness), "mm)")
                                                                                    self.writegc("(stockMax:",str(self.stockXwidth)
171 дсру
                                                                                               ,"mm,_{\sqcup}", {\tt str} (self.stockYheight/2),"mm,_{\sqcup}0.00mm
                                                                                              )")
172 gcpy
                                                                                    \verb|self.writegc("(STOCK/BLOCK, ", \verb|str(self.)|)|)| \\
                                                                                               stockXwidth), ", u", str(self.stockYheight), ", u
                                                                                               ", str(self.stockZthickness), ", \u0.00, \u0.7", str(
                                                                                               self.stockYheight/2),",",str(self.
```

```
stockZthickness),")");
                     if self.stockzero == "Top-Left":
173 дсру
                          self.stock = self.stock.translate([0,-self.
174 дсру
                              stockYheight, -self.stockZthickness])
175 gcpy
                          if self.generategcode == True:
                               self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
176 дсру
                                   stockYheight), "mm, _{\sqcup}-", str(self.
                                   stockZthickness),"mm)")
                               self.writegc("(stockMax:",str(self.stockXwidth)
177 gcpy
                                   ,"mm, _0.00mm, _0.00mm)")
                               self.writegc("(STOCK/BLOCK,_{\sqcup}",str(self.
178 gcpv
                                   stockXwidth),",u",str(self.stockYheight),",u
                                   ", str(self.stockZthickness), ", \u0.00, \u0.str(
                                   self.stockYheight),",",str(self.
stockZthickness),")")
                     if self.stockzero == "Center":
179 gcpy
180 дсру
                          self.stock = self.stock.translate([-self.
                              stockXwidth / 2,-self.stockYheight / 2,-self.
                              stockZthickness1)
                          if self.generategcode == True:
181 дсру
                               self.writegc("(stockMin:_{\sqcup}-",\mathbf{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,_{\sqcup}",\mathbf{str}(self.
184 gcpy
                                   stockXwidth),",",",str(self.stockYheight),","
                                   ", str(self.stockZthickness), ", ", ", str(self.
                                   \verb|stockXwidth/2|, ", ", " | str(self.stockYheight)|
                                   /2),",_{\sqcup}",str(self.stockZthickness),")")
                 if self.zeroheight == "Bottom":
185 дсру
                      if self.stockzero == "Lower-Left":
186 дсру
                           self.stock = self.stock.translate([0,0,0])
187 дсру
                           if self.generategcode == True:
188 дсру
189 дсру
                                self.writegc("(stockMin:0.00mm,_{\sqcup}0.00mm,_{\sqcup}0.00mm
                                   )")
                                self.writegc("(stockMax:",str(self.stockXwidth
190 gcpv
                                    ), "mm, _{\sqcup}", str(self.stockYheight), "mm, _{\sqcup\sqcup}", str
                                    (self.stockZthickness),"mm)")
                                self.writegc("(STOCK/BLOCK, ", str(self.
191 gcpy
                                    stockXwidth),",",str(self.stockYheight),",
                                    \square", str(self.stockZthickness), ", \square0.00, \square0.00,
                                    ۵.00)")
                     if self.stockzero == "Center-Left":
192 дсру
                          193 дсру
                          if self.generategcode == True:
194 gcpy
                              self.writegc("(stockMin:0.00mm, _-", str(self.
195 дсру
                                   stockYheight/2),"mm, u0.00mm)")
196 дсру
                               self.writegc("(stockMax:",str(self.stockXwidth)
                                   ,"mm,_{\sqcup}",str(self.stockYheight/2),"mm,_{\sqcup}-",str
                                   (self.stockZthickness),"mm)")
                               self.writegc("(STOCK/BLOCK, ", str(self.
197 gcpy
                                   stockXwidth),",u",str(self.stockYheight),",u
                                   ", str(self.stockZthickness), ", u0.00, u", str(
                                   self.stockYheight/2),",\square0.00mm)");
                     if self.stockzero == "Top-Left":
198 gcpy
                          self.stock = self.stock.translate([0,-self.
199 дсру
                              stockYheight,0])
                          if self.generategcode == True:
200 gcpy
                               self.writegc("(stockMin:0.00mm,_{\sqcup}-",str(self.
201 gcpy
                                   stockYheight), "mm, _{\square}0.00mm)")
                               self.writegc("(stockMax:",str(self.stockXwidth)
202 gcpy
                                   ,"mm,_{\sqcup}0.00mm,_{\sqcup}",str(self.stockZthickness),"
                                   mm)")
                               self.writegc("(STOCK/BLOCK, ", str(self.
203 дсру
                                   stockXwidth), ", ", str(self.stockYheight), ", "
                                   ",str(self.stockZthickness),",u0.00,u",str(
                                   self.stockYheight),",_{\sqcup}0.00)")
                     if self.stockzero == "Center":
204 дсру
                          self.stock = self.stock.translate([-self.
205 дсру
                              stockXwidth / 2,-self.stockYheight / 2,0])
                          if self.generategcode == True:
206 дсру
                               self.writegc("(stockMin:_{\sqcup}-", str(self.
207 дсру
                                   stockXwidth/2),",u-",str(self.stockYheight
/2),"mm,u0.00mm)")
208 дсру
                               self.writegc("(stockMax:",str(self.stockXwidth
```

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:

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()
```

3.3 Tools and Changes

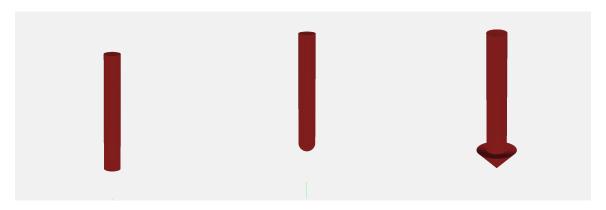
currenttoolnumber Similarly Python functions and variables will be used in: currenttoolnumber (note that it is imsettool portant to use a different name than the variable currenttoolnum and settool to track and set and return the current tool:

```
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
                return self.currenttoolnum
220 gcpy
221 gcpy
             def\ current round over tool number (self):
222 gcpy #
                  global Roundover_tool_num
223 gcpy #
224 gcpy #
                  return self.Roundover_tool_num
```

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:

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):

```
def endmill_v(self, es_v_angle, es_diameter):

236 gcpy
237 gcpy
v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter / 2) / math.tan((es_v_angle / 2))), center=False)

238 gcpy
s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h
=((es_diameter * 8)), center=False)

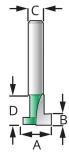
239 gcpy
sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((es_v_angle / 2)))])

240 gcpy
return union(v,sh)
```

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
248 gcpy
               ):
                btm = cylinder(r1=(minor_diameter / 2), r2=(major_diameter
249 дсру
                   / 2), h=cut_height, center = False)
                top = cylinder(r1=(major_diameter / 2), r2=(minor_diameter
250 gcpy
                   / 2), h=cut_height, center = False)
251 дсру
                top = top.translate([0, 0, cut_height/2])
                tm = btm.union(top)
252 дсру
               return tm
253 gcpy
254 дсру
           def threadmill_shaft(self, diameter, cut_height, height):
255 дсру
                shaft = cylinder(r1=(diameter / 2), r2=(diameter / 2), h=
256 gcpy
                   height, center = False)
257 дсру
                shaft = shaft.translate([0, 0, cut_height/2])
                return shaft
258 дсру
```

dovetail 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)

```
def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle):
return cylinder(r1=(dt_bottomdiameter / 2), r2=(
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 {\bf module} cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
41 gcpscad
             if (radiustn == 56125) {
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
                 \verb|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.

```
263 дсру
            def currenttool(self):
264 gcpy #
                 global currenttoolshape
                return self.currenttoolshape
265 дсру
```

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)

```
{\tt 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.

```
267 дсру
          def toolchange(self,tool_number,speed = 10000):
268 gcpy #
              global currenttoolshape
              self.currenttoolshape = self.endmill_square(0.001, 0.001)
269 дсру
270 дсру
             self.settool(tool_number)
271 gcpy
             272 дсру
```

```
273 дсру
                      self.writegc("(Toolpath)")
                      self.writegc("M05")
274 дсру
                 if (tool_number == 201):
275 дсру
                      self.writegc("(TOOL/MILL,6.35,\square0.00,\square0.00,\square0.00)")
276 gcpy
277 дсру
                      self.currenttoolshape = self.endmill_square(6.35,
                          19.05)
                 elif (tool_number == 102):
278 gcpy
                      self.writegc("(TOOL/MILL,3.175, _0.00, _0.00, _0.00)")
279 gcpy
280 дсру
                      self.currenttoolshape = self.endmill_square(3.175,
                 elif (tool_number == 112):
281 gcpv
                      self.writegc("(TOOL/MILL,1.5875, _0.00, _0.00, _0.00)")
282 дсру
                      self.currenttoolshape = self.endmill_square(1.5875,
283 дсру
                          6.35)
284 дсру
                 elif (tool number == 122):
                      self.writegc("(TOOL/MILL,0.79375,_{\square}0.00,_{\square}0.00,_{\square}0.00)")
285 дсру
286 gcpy
                      self.currenttoolshape = self.endmill_square(0.79375,
                          1.5875)
                 elif (tool_number == 202):
287 gcpy
                      self.writegc("(TOOL/MILL,6.35,\square3.175,\square0.00,\square0.00)")
288 дсру
                      self.currenttoolshape = self.ballnose(6.35, 19.05)
289 дсру
290 дсру
                 elif (tool_number == 101):
                     self.writegc("(TOOL/MILL,3.175, __1.5875, __0.00, __0.00)")
291 gcpy
292 дсру
                      self.currenttoolshape = self.ballnose(3.175, 12.7)
293 дсру
                 elif (tool number == 111):
                     self.writegc("(TOOL/MILL,1.5875, _0.79375, _0.00, _0.00)")
294 дсру
                      self.currenttoolshape = self.ballnose(1.5875, 6.35)
295 дсру
296 дсру
                 elif (tool_number == 121):
297 дсру
                      self.writegc("(TOOL/MILL,3.175, _0.79375, _0.00, _0.00)")
298 дсру
                      self.currenttoolshape = self.ballnose(0.79375, 1.5875)
                 elif (tool_number == 327):
299 дсру
                      self.writegc("(TOOL/MILL,0.03,_{\square}0.00,_{\square}13.4874,_{\square}30.00)")
300 дсру
301 дсру
                      self.currenttoolshape = self.endmill_v(60, 26.9748)
                 elif (tool number == 301):
302 дсру
                      self.writegc("(TOOL/MILL,0.03,\square0.00,\square6.35,\square45.00)")
303 дсру
                      self.currenttoolshape = self.endmill_v(90, 12.7)
304 дсру
305 дсру
                 elif (tool_number == 302):
                      self.writegc("(TOOL/MILL,0.03, _0.00, _10.998, _30.00)")
306 дсру
307 дсру
                      self.currenttoolshape = self.endmill_v(60, 12.7)
                 elif (tool_number == 390):
308 дсру
309 дсру
                      self.writegc("(TOOL/MILL,0.03, _0.00, _1.5875, _45.00)")
                      self.currenttoolshape = self.endmill v(90, 3.175)
310 gcpy
311 дсру
                 elif (tool_number == 374):
                      self.writegc("(TOOL/MILL,9.53,_{\square}0.00,_{\square}3.17,_{\square}0.00)")
312 дсру
                 elif (tool_number == 375):
313 gcpy
                      self.writegc("(TOOL/MILL,9.53,_{\sqcup}0.00,_{\sqcup}3.17,_{\sqcup}0.00)")
314 дсру
315 дсру
                 elif (tool_number == 376):
                     self.writegc("(TOOL/MILL,12.7,_{\sqcup}0.00,_{\sqcup}4.77,_{\sqcup}0.00)")
316 дсру
                 elif (tool_number == 378):
317 дсру
                      self.writegc("(TOOL/MILL,12.7, _0.00, _4.77, _0.00)")
318 дсру
319 дсру
                 elif (tool_number == 814):
                      self.writegc("(TOOL/MILL, 12.7, \_6.367, \_12.7, \_0.00)")
320 дсру
                      #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
321 дсру
                      #https://www.leevalley.com/en-us/shop/tools/power-tool-
322 gcpy
                          accessories/router-bits/30172-dovetail-bits?item=18
                          J1607
                      self.currenttoolshape = self.dovetail(12.7, 6.367,
323 gcpy
                          12.7, 14)
                 elif (tool_number == 56125):#0.508/2, 1.531
324 gcpy
                      self.writegc("(TOOL/CRMILL,_{\square}0.508,_{\square}6.35,_{\square}3.175,_{\square}7.9375,
325 gcpy
                          ⊔3.175)<sup>"</sup>)
                 elif (tool_number == 56142):#0.508/2, 2.921
326 gcpy
                      self.writegc("(TOOL/CRMILL, _0.508, _3.571875, _1.5875, _
327 gcpy
                          5.55625, _1.5875)")
                  elif (tool_number == 312): #1.524/2, 3.175
328 gcpy #
                       self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
329 gcpy #
            Radius, Height, Length)")
                 elif (tool_number == 1570):#0.507/2, 4.509
330 дсру
                      self.writegc("(TOOL/CRMILL,_{\square}0.17018,_{\square}9.525,_{\square}4.7625,_{\square}
331 дсру
                          12.7, 4.7625)")
```

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

```
332 gcpy self.writegc("M6T", str(tool_number))
333 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:

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

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

```
335 дсру
            def tool_diameter(self, ptd_tool, ptd_depth):
336 gcpy # Square 122,112,102,201
                if ptd_tool == 122:
337 дсру
                     return 0.79375
338 дсру
339 дсру
                if ptd_tool == 112:
340 дсру
                    return 1.5875
                if ptd_tool == 102:
341 gcpy
342 дсру
                    return 3.175
343 дсру
                if ptd_tool == 201:
                     return 6.35
344 дсру
345 gcpy # Ball 121,111,101,202
346 дсру
                if ptd_tool == 122:
                     if ptd_depth > 0.396875:
347 gcpy
348 дсру
                         return 0.79375
349 дсру
                     else:
350 дсру
                         return ptd_tool
                if ptd_tool == 112:
351 дсру
                     if ptd_depth > 0.79375:
352 дсру
353 дсру
                         return 1.5875
354 дсру
                     else:
355 дсру
                         return ptd_tool
                if ptd tool == 101:
356 дсру
                     if ptd_depth > 1.5875:
357 дсру
                         return 3.175
358 дсру
359 дсру
360 дсру
                         return ptd_tool
361 дсру
                if ptd_tool == 202:
                     if ptd_depth > 3.175:
362 gcpy
363 дсру
                         return 6.35
                     else:
364 дсру
365 дсру
                         return ptd_tool
366 gcpy # V 301, 302, 390
               if ptd_tool == 301:
367 gcpy
368 gcpy
                     return ptd_tool
369 дсру
                if ptd_tool == 302:
                    return ptd_tool
370 дсру
                if ptd_tool == 390:
371 дсру
372 дсру
                     return ptd_tool
373 gcpy # Keyhole
               if ptd_tool == 374:
374 дсру
                     if ptd_depth < 3.175:</pre>
375 gcpy
```

```
return 9.525
376 дсру
377 дсру
378 дсру
                           return 6.35
                 if ptd tool == 375:
379 gcpy
380 дсру
                      if ptd_depth < 3.175:</pre>
381 дсру
                           return 9.525
382 дсру
                      else:
383 дсру
                           return 8
384 дсру
                 if ptd_tool == 376:
385 дсру
                      if ptd_depth < 4.7625:</pre>
386 дсру
                           return 12.7
387 дсру
                      else:
388 дсру
                           return 6.35
389 дсру
                  if ptd_tool == 378:
                      if ptd_depth < 4.7625:</pre>
390 дсру
391 дсру
                           return 12.7
                       else:
392 дсру
393 дсру
                           return 8
394 gcpy # Dovetail
                 if ptd_tool == 814:
395 дсру
                      if ptd_depth > 12.7:
396 дсру
397 дсру
                           return 6.35
                      else:
398 дсру
399 дсру
                           return 12.7
```

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

```
def tool_radius(self, ptd_tool, ptd_depth):
tr = self.tool_diameter(ptd_tool, ptd_depth)/2
gcpy
403 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

rapid...

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:

```
410 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):



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.

```
412 gcpy
            def rapid(self,ex, ey, ez):
                cts = self.currenttoolshape
413 дсру
                toolpath = self.rcs(ex, ey, ez, cts)
414 дсру
415 дсру
                self.setxpos(ex)
                self.setypos(ey)
416 gcpy
417 дсру
                self.setzpos(ez)
                if self.generatepaths == True:
418 дсру
                     self.rapids = self.rapids.union(toolpath)
419 дсру
420 gcpy
                     return cylinder(0.01, 0, 0.01, center = False, fn = 3)
421 gcpy
                    return cube([0.001,0.001,0.001])
422 gcpy
                else:
423 дсру
                     return toolpath
424 дсру
            def cutshape(self,ex, ey, ez):
425 gcpy
                cts = self.currenttoolshape
426 gcpy
427 gcpy
                toolpath = self.rcs(ex, ey, ez, cts)
                if self.generatepaths == True:
428 дсру
                    self.toolpaths = self.toolpaths.union(toolpath)
429 gcpy
                    return cube([0.001,0.001,0.001])
430 gcpy
431 gcpy
                else:
432 дсру
                    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) \ensuremath{\texttt{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:

```
434 gcpy def movetosafeZ(self):
```

```
435 дсру
                rapid = self.rapid(self.xpos(),self.ypos(),self.
                   retractheight)
                 if self.generatepaths == True:
436 gcpy #
                     rapid = self.rapid(self.xpos(),self.ypos(),self.
437 gcpy #
           retractheight)
438 gcpy #
                     self.rapids = self.rapids.union(rapid)
439 gcpy #
                 else:
          if (generategoode == true) \{
440 gcpy #
                 writecomment("PREPOSITION FOR RAPID PLUNGE"); Z25.650
441 gcpy #
          //G1Z24.663F381.0 ,"F",str(plunge)
442 gcpy #
443 дсру
                if self.generatepaths == False:
444 дсру
                    return rapid
445 gcpy
                else:
                    return cube([0.001,0.001,0.001])
446 gcpy
447 gcpy
448 gcpy
            def rapidXY(self, ex, ey):
449 дсру
                rapid = self.rapid(ex,ey,self.zpos())
                 if self.generatepaths == True:
450 gcpy #
451 gcpy #
                     self.rapids = self.rapids.union(rapid)
452 gcpy #
                 else:
453 дсру
                if self.generatepaths == False:
454 дсру
                    return rapid
455 gcpy
456 gcpy
            def rapidZ(self, ez):
457 gcpy
                rapid = self.rapid(self.xpos(),self.ypos(),ez)
                 if self.generatepaths == True:
458 gcpy #
                     self.rapids = self.rapids.union(rapid)
459 gcpy #
460 gcpy #
                 else:
461 gcpy
                if self.generatepaths == False:
462 gcpy
                    return rapid
```

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(){
              gcp.rapid(gcp.xpos(),gcp.ypos(),retractheight);
59 gcpscad
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:

```
def cutline(self,ex, ey, ez):\
464 gcpy
465 gcpy \#below will need to be integrated into if/then structure not yet
            copied
466 gcpy #
                  cts = self.currenttoolshape
467 дсру
                 if (self.currenttoolnumber() == 374):
                       self.writegc("(TOOL/MILL,9.53, 0.00, 3.17, 0.00)")
468 gcpy #
469 gcpy
                      self.currenttoolshape = self.keyhole(9.53/2, 3.175)
                      toolpath = self.cutshape(ex, ey, ez)
470 gcpy
                      self.currenttoolshape = self.keyhole_shaft(6.35/2,
471 gcpy
                          12.7)
                      toolpath = toolpath.union(self.cutshape(ex, ey, ez))
472 gcpy
                  elif (self.currenttoolnumber() == 375):
473 gcpy #
                  self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")\\ elif (self.currenttoolnumber() == 376):
474 gcpy #
475 gcpy #
                       self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
476 gcpy #
                  elif (self.currenttoolnumber() == 378):
477 gcpy #
                  self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
478 gcpy #
479 gcpy #
                       self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
480 gcpy #
```

```
7.9375, 3.175)")
                         elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
481 gcpy
482 gcpy #
                                     self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
                   5.55625, 1.5875)")
483 gcpy
                                   toolpath = self.cutroundovertool(self.xpos(), self.ypos
                                         (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
                              elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
484 gcpy #
                                     self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
485 gcpy #
                   12.7, 4.7625)")
486 дсру
                           else:
                                  toolpath = self.cutshape(ex, ey, ez)
487 дсру
                           self.setxpos(ex)
488 дсру
489 дсру
                           self.setypos(ey)
490 дсру
                           self.setzpos(ez)
                            if self.generatepaths == True:
491 gcpy #
492 gcpy #
                                    self.toolpaths = union([self.toolpaths, toolpath])
493 gcpy #
                             else:
494 дсру
                           if self.generatepaths == False:
495 gcpy
                                  return toolpath
496 дсру
                            else:
497 дсру
                                   return cube([0.001,0.001,0.001])
498 дсру
499 дсру
                    def cutlinedxfgc(self,ex, ey, ez):
500 дсру
                            self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                                  ypos(), ex, ey)
                            self.writegc("G01_{\square}X", str(ex), "_{\square}Y", str(ey), "_{\square}Z", str(ez)
501 gcpy
                                 )
502 gcpy #
                             if self.generatepaths == False:
503 дсру
                            return self.cutline(ex, ey, ez)
504 gcpy
505 gcpy
                    {\tt def} cutroundovertool(self, bx, by, bz, ex, ey, ez,
                           tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
506 gcpy #
                             n = 90 + fn*3
                             print("Tool dimensions", tool_radius_tip,
507 gcpy #
                    tool_radius_width, "begin ",bx, by, bz,"end ", ex, ey, ez)
508 дсру
                            step = 4 #360/n
                            shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
509 дсру
                            toolpath = hull(shaft.translate([bx,by,bz]), shaft.
510 gcpy
                                 translate([ex,ey,ez]))
                            shaft = cylinder(tool_radius_width*2,tool_radius_tip+
511 дсру
                                  tool_radius_width,tool_radius_tip+tool_radius_width)
                            toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
512 gcpy
                                  tool_radius_width]), shaft.translate([ex,ey,ez+
                                  tool_radius_width])))
513 дсру
                            for i in range(1, 90, stepsizeroundover):
                                   angle = i
514 gcpy
515 gcpy
                                   dx = tool_radius_width*math.cos(math.radians(angle))
516 дсру
                                   dxx = tool_radius_width*math.cos(math.radians(angle+1))
                                   dzz = tool_radius_width*math.sin(math.radians(angle))
517 gcpy
518 дсру
                                   \texttt{dz = tool\_radius\_width*math.sin(math.radians(angle+1))}
                                   dh = abs(dzz-dz)+0.0001
519 gcpy
                                   slice = cylinder(dh,tool_radius_tip+tool_radius_width-
520 дсру
                                        dx,tool_radius_tip+tool_radius_width-dxx)
                                   toolpath = toolpath.union(hull(slice.translate([bx,by,
521 gcpy
                                         bz+dz]), slice.translate([ex,ey,ez+dz])))
522 gcpy
                            if self.generatepaths == True:
                                  self.toolpaths = self.toolpaths.union(toolpath)
523 gcpy
524 дсру
                            else:
                                   return toolpath
525 gcpy
526 дсру
527 gcpy
                    \begin{tabular}{ll} \beg
                            self.writegc("G01_{\sqcup}Z", str(ez), "F",str(feed))
528 gcpy
                             if self.generatepaths == False:
529 gcpy #
530 дсру
                           return self.cutline(self.xpos(),self.ypos(),ez)
```

The matching OpenSCAD command is a descriptor:

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) $G2/3 \ldots$
- DXF dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)
- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

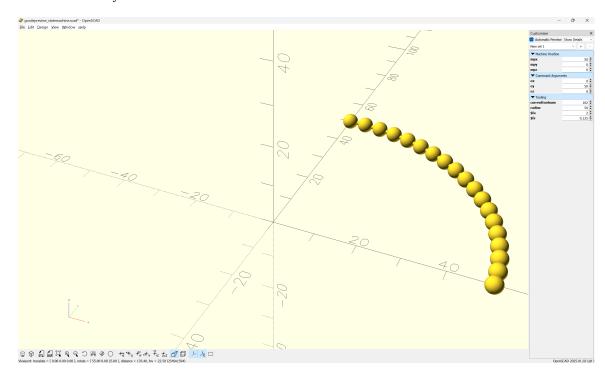
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

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);
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(){
translate([-200, -200, 20]){
 cube([0.001, 0.001, 0.001], center=true);
translate([200, 200, 20]){
  cube([0.001, 0.001, 0.001], center=true);
}
module plot_cut(bx, by, bz, ex, ey, ez) {
  union(){
    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,
532 gcpv
               tpzreldim, stepsizearc=1):
                tpzinc = ez - self.zpos() / (earc - barc)
533 gcpy #
                tpzinc = tpzreldim / (earc - barc)
534 дсру
535 дсру
               cts = self.currenttoolshape
                toolpath = cts
536 gcpy
537 дсру
                toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                   .zpos()])
               i = barc
538 дсру
                while i < earc:</pre>
539 дсру
540 gcpy
                    toolpath = toolpath.union(self.cutline(xcenter + radius
                         * math.cos(math.radians(i)), ycenter + radius *
                       math.sin(math.radians(i)), self.zpos()+tpzinc))
                    i += stepsizearc
541 gcpy
                if self.generatepaths == False:
542 gcpy
543 gcpy
                    return toolpath
544 дсру
                else:
                    return cube([0.01,0.01,0.01])
545 gcpy
546 gcpy
547 gcpy
           def cutarcCW(self, barc,earc, xcenter, ycenter, radius,
               tpzreldim, stepsizearc=1):
                 print(str(self.zpos()))
548 gcpy #
                print(str(ez))
549 gcpy #
                 print(str(barc - earc))
550 gcpy #
                 tpzinc = ez - self.zpos() / (barc - earc)
551 gcpy #
                print(str(tzinc))
552 gcpy #
                global toolpath
553 gcpy #
                print("Entering n toolpath")
554 gcpy #
               tpzinc = tpzreldim / (barc - earc)
555 дсру
556 дсру
               cts = self.currenttoolshape
557 gcpy
               toolpath = cts
               toolpath = toolpath.translate([self.xpos(),self.ypos(),self
558 gcpy
                    .zpos()])
               i = barc
559 дсру
560 дсру
                while i > earc:
                    toolpath = toolpath.union(self.cutline(xcenter + radius
561 gcpy
                         * math.cos(math.radians(i)), ycenter + radius *
                        math.sin(math.radians(i)), self.zpos()+tpzinc))
```

```
self.setxpos(xcenter + radius * math.cos(math.radians(
562 gcpy #
           i)))
                     self.setypos(ycenter + radius * math.sin(math.radians(
563 gcpy #
           i)))
                     print(str(self.xpos()), str(self.ypos(), str(self.zpos
564 gcpy #
           ())))
565 gcpy #
                     self.setzpos(self.zpos()+tpzinc)
                    i += abs(stepsizearc) * -1
566 дсру
567 gcpy #
                 self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
           radius, barc, earc)
568 gcpy #
                if self.generatepaths == True:
                     print("Unioning n toolpath")
569 gcpy #
570 gcpy #
                     self.toolpaths = self.toolpaths.union(toolpath)
571 gcpy #
                if self.generatepaths == False:
572 gcpy
573 gcpy
                    return toolpath
574 gcpy
575 gcpy
                    return cube([0.01,0.01,0.01])
```

Matching OpenSCAD modules are easily made:

```
86 gcpscad module cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim){
87 gcpscad gcpscad }
88 gcpscad }
89 gcpscad
90 gcpscad module cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim){
91 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:

- 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

 $^{{}^{1}\}text{en.wikipedia.org/wiki/Hyperbolic_sector} \quad and \quad \text{www.reddit.com/r/Geometry/comments/bkbzgh/is_there_a_name_for_a_3_pointed_figure_with_two}$

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, 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,
583 gcpy
                yheight, corners = "Square", radius = 6):
                if corners == "Square":
584 дсру
                     self.dxfline(tool_num, xorigin, yorigin, xorigin +
585 дсру
                         xwidth, yorigin)
                     self.dxfline(tool_num, xorigin + xwidth, yorigin,
586 дсру
                        xorigin + xwidth, yorigin + yheight)
                     self.dxfline(tool_num, xorigin + xwidth, yorigin +
587 дсру
                        yheight, xorigin, yorigin + yheight)
                     self.dxfline(tool_num, xorigin, yorigin + yheight,
588 gcpy
                xorigin, yorigin)
elif corners == "Fillet":
589 дсру
590 дсру
                     self.dxfrectangleround(tool_num, xorigin, yorigin,
                xwidth, yheight, radius)
elif corners == "Chamfer":
591 gcpy
                     self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
592 дсру
                        xwidth, yheight, radius)
                elif corners == "Flipped_{\sqcup}Fillet":
593 gcpy
594 дсру
                     self.dxfrectangleflippedfillet(tool_num, xorigin,
                        yorigin, xwidth, yheight, radius)
```

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

```
596 gcpy
            def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
                 yheight, radius):
                 self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
597 дсру
                     yheight - radius, radius,
                                                    0, 90)
                 self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
radius, radius, 90, 180)
598 дсру
                 self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
599 дсру
                     radius, 180, 270)
                 self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
600 дсру
                     radius, radius, 270, 360)
601 дсру
602 gcpy
                 self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                      xwidth - radius, yorigin)
603 дсру
                 self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                     xorigin + xwidth, yorigin + yheight - radius)
                 self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
    yheight, xorigin + radius, yorigin + yheight)
604 gcpy
                 self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
605 gcpy
                      xorigin, yorigin + radius)
```

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

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

```
608 дсру
                self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
                    yorigin + radius)
                self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
609 gcpy
                     xorigin + radius, yorigin + yheight)
                self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
610 gcpy
                yheight, xorigin + xwidth, yorigin + yheight - radius;
self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
611 gcpy
                    xorigin + xwidth, yorigin + radius)
612 gcpy
                self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
613 дсру
                     xwidth - radius, yorigin)
                self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
614 gcpy
                    xorigin + xwidth, yorigin + yheight - radius)
                self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
615 дсру
                     yheight, xorigin + radius, yorigin + yheight)
                self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
616 gcpy
                     xorigin, yorigin + radius)
```

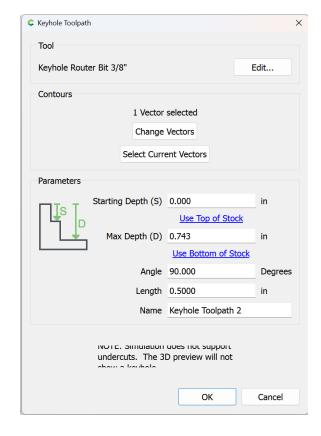
Flipped Fillet:

```
618 gcpy
           def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
                xwidth, yheight, radius):
               self.dxfarc(tool_num, xorigin, yorigin, radius,
                                                                  0, 90)
619 gcpy
620 gcpy
               self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                   90, 180)
621 gcpy
               self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                   radius, 180, 270)
622 gcpy
               self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                   270, 360)
623 gcpy
               self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
624 gcpy
                    xwidth - radius, yorigin)
               self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
625 gcpy
                   xorigin + xwidth, yorigin + yheight - radius)
               self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
626 gcpy
                    yheight, xorigin + radius, yorigin + yheight)
               self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
627 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:



Hence the parameters:

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

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

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

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

```
629 gcpy
                  cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
                  kh_max_depth, kht_direction, kh_distance):
                   \mbox{\bf if} \ (\mbox{\tt kht\_direction} \ \mbox{\tt ==} \ \mbox{\tt "N"}): 
 630 gcpy
 631 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                  kh_max_depth, 90, kh_distance)
elif (kht_direction == "S"):
 632 дсру
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
 633 дсру
                            kh_max_depth, 270, kh_distance)
                  elif (kht_direction == "E"):
 634 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
 635 дсру
                            kh_max_depth, 0, kh_distance)
                  elif (kht_direction == "W"):
 636 дсру
 637 gcpy
                       toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                            kh_max_depth, 180, kh_distance)
 638 gcpy
                  {\tt if} self.generatepaths == True:
                       self.toolpaths = union([self.toolpaths, toolpath])
 639 дсру
 640 gcpy
                       return toolpath
 641 gcpy
                  else:
                       return cube([0.01,0.01,0.01])
 642 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:

```
def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
644 gcpy
                kh_angle, kh_distance):
                oXpos = self.xpos()
645 gcpy
                oYpos = self.ypos()
646 gcpy
                self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
647 gcpy
                   kh_start_depth, kh_max_depth, kh_angle, kh_distance)
                toolpath = self.cutline(self.xpos(), self.ypos(),
648 дсру
                   kh_max_depth)
                self.setxpos(oXpos)
649 gcpy
650 дсру
                self.setypos(oYpos)
651 gcpy
                if self.generatepaths == False:
                    return toolpath
652 gcpy
653 gcpy
                else:
654 дсру
                    return cube([0.001,0.001,0.001])
```

```
def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
656 gcpv
                kh_max_depth, kh_angle, kh_distance):
                  oXpos = self.xpos()
oYpos = self.ypos()
657 gcpy #
658 gcpy #
659 gcpy #Circle at entry hole
                 self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
    kh_tool_num, 7), 0, 90)
660 дсру
                 self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
661 дсру
                     kh_tool_num, 7), 90,180)
                 self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
662 gcpy
                     kh_tool_num, 7),180,270)
                 self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
663 gcpy
                     kh_tool_num, 7),270,360)
```

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

```
665 gcpy #pre-calculate needed values
666 дсру
               r = self.tool_radius(kh_tool_num, 7)
667 gcpy #
                 print(r)
668 дсру
                rt = self.tool_radius(kh_tool_num, 1)
669 gcpy #
                print(rt)
670 gcpy
                ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                   tool_radius(kh_tool_num, 7))**2)
                 print(ro)
671 gcpy #
                angle = math.degrees(math.acos(ro/rt))
672 gcpy
673 gcpy #Outlines of entry hole and slot
                if (kh_angle == 0):
675 gcpy #Lower left of entry hole
                    self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
676 gcpy
                        tool_radius(kh_tool_num, 1),180,270)
677 gcpy \#Upper left of entry hole
678 gcpy
                    self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                        tool_radius(kh_tool_num, 1),90,180)
679 gcpy \#Upper\ right\ of\ entry\ hole
                     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
680 gcpy #
            41.810, 90)
                    self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
681 gcpy
                        angle, 90)
682 gcpy #Lower right of entry hole
683 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.|| \\
684 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))))
685 gcpy #Actual line of cut
                     self.dxfline(kh_tool_num, self.xpos(),self.ypos(),self
686 gcpy #
           .xpos()+kh_distance,self.ypos())
687 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
688 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)
689 gcpy #lower right of end of slot
690 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)
691 gcpy #upper right slot
692 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.
693 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, (
           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)
694 \ \mathrm{gcpy} \ \text{\#end} \ position \ \mathrm{at} \ top \ \mathrm{of} \ \mathrm{slot}
695 gcpy #lower right slot
696 дсру
                    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
697 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)
698 gcpy #end position at top of slot
699 gcpy #
                           hull(){
                               translate([xpos(), ypos(), zpos()]){
700 gcpy #
                                   keyhole_shaft(6.35, 9.525);
701 gcpy #
702 gcpy #
703 gcpy #
                                translate([xpos(), ypos(), zpos()-kh_max_depth]){
                                    keyhole_shaft(6.35, 9.525);
704 gcpy #
705 gcpy #
706 gcpy #
707 gcpy #
                           hull(){
                                translate([xpos(), ypos(), zpos()-kh_max_depth]){
708 gcpy #
                                   keyhole_shaft(6.35, 9.525);
709 gcpy #
710 gcpy #
711 gcpy #
                                translate\left( \texttt{[xpos()+kh\_distance, ypos(), zpos()-kh\_max\_depth]} \right)
                                    keyhole shaft (6.35, 9.525);
712 gcpy #
713 gcpy #
                           7
714 gcpy #
715 gcpy #
                           cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
716 gcpy #
                           cut with feed (\texttt{getxpos}() + \texttt{kh\_distance}, \texttt{getypos}(), -\texttt{kh\_max\_depth}, \texttt{feed}
                           setxpos(getxpos()-kh_distance);
717 gcpy #
                      } else if (kh_angle > 0 && kh_angle < 90) {
718 gcpy #
719 gcpy #//echo(kh_angle);
720 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\,,} \ \ (
721 gcpy #
                        \verb|kh_max_depth|)/2,180+\verb|kh_angle|,270+\verb|kh_angle|, KH_tool_num||;
722 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);
723 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);
724 gcpy \#dxfarc(getxpos()+(kh\_distance*cos(kh\_angle)),
                     getypos()+(kh_distance*sin(kh_angle)),tool_diameter(KH_tool_num,
725 gcpy #
                          (kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle,KH_tool_num);
726 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
                        kh_distance*sin(kh_angle)),tool_diameter(KH_tool_num, (
                        kh_max_depth+4.36))/2,270+kh_angle,360+kh_angle, KH_tool_num);
727 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))),
728 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))),
729 gcpy # getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
                        , (kh_max_depth+4.36))/2)*sin(kh_angle)),
730 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_tool_num_angle))+(tool_diameter(KH_tool_num_angle))
                            (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
731 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
732 gcpy #//echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
733 gcpy #echo("Aangle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36)
                        )/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
734 gcpy #//echo(kh_angle);
735 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()+
736 gcpy #
                        kh_distance, self.ypos(), -kh_max_depth))
                                 elif (kh_angle == 90):
737 gcpy
738 gcpy #Lower left of entry hole
739 дсру
                                           self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                                                   kh_tool_num, 1),180,270)
740 gcpy #Lower right of entry hole
741 gcpy
                                           self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                                                   kh_tool_num, 1),270,360)
742 gcpy #left slot
743 gcpy
                                           self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
                                                   oYpos+kh_distance)
744 gcpy #right slot
745 gcpy
                                           self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
                                                  oYpos+kh_distance)
746 gcpy #upper left of end of slot
```

```
747 gcpy
                    \verb|self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r|\\
,90,180)
748 gcpy #upper right of end of slot
                    self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r
749 gcpy
,0,90)
750 gcpy #Upper right of entry hole
751 gcpy
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
752 gcpy #Upper left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
753 дсру
                       180)
                     toolpath = toolpath.union(self.cutline(oXpos, oYpos+
754 gcpy #
           \verb|kh_distance, -kh_max_depth|)|
                elif (kh_angle == 180):
755 gcpy
756 gcpy #Lower right of entry hole
757 gcpy
                    \verb|self.dxfarc(kh_tool_num|, oXpos,oYpos,self.tool_radius(|
                       kh_tool_num, 1),270,360)
758 gcpy #Upper right of entry hole
759 gcpy
                    self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                        kh_tool_num, 1),0,90)
760 gcpy #Upper left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
761 gcpy
                       angle)
762 gcpy #Lower left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
                        270)
764 gcpy #upper slot
                    self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
765 gcpy
                       kh_distance, oYpos-r)
766 gcpy #lower slot
767 gcpy
                    self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                       kh distance, oYpos+r)
768 gcpy #upper left of end of slot
                    self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
                       ,90,180)
770 gcpy #lower left of end of slot
                    self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
771 gcpy
                       ,180,270)
                     toolpath = toolpath.union(self.cutline(oXpos-
772 gcpy #
           kh_distance, oYpos, -kh_max_depth))
                elif (kh_angle == 270):
773 gcpy
774 gcpy #Upper left of entry hole
                    \verb|self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(|
775 gcpy
                        kh_tool_num, 1),90,180)
776 gcpy #Upper right of entry hole
                    self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
777 gcpy
                        kh_tool_num, 1),0,90)
778 gcpy #left slot
                    self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
779 gcpy
                       oYpos-kh_distance)
780 gcpy \#right slot
                    self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                       oYpos-kh_distance)
782 gcpy #lower left of end of slot
                    \verb|self.dxfarc(kh_tool_num|, oXpos,oYpos-kh_distance,r|\\
783 gcpy
                        ,180,270)
784 gcpy #lower right of end of slot
785 дсру
                    self.dxfarc(kh_tool_num, oXpos,oYpos-kh_distance,r
                        ,270,360)
786 gcpy #lower right of entry hole
787 дсру
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                       angle)
788 gcpy #lower left of entry hole
                    self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
789 gcpy
                       360)
                     toolpath = toolpath.union(self.cutline(oXpos, oYpos-
790 gcpy #
           kh_distance, -kh_max_depth))
                 print(self.zpos())
791 gcpy #
792 gcpy #
                 self.setxpos(oXpos)
                 self.setypos(oYpos)
793 gcpy #
794 gcpy #
                 if self.generate paths == False:
795 gcpy #
                     return toolpath
796 дсру
797 gcpy #
          } else if (kh_angle == 90) {
            //Lower left of entry hole
798 gcpy #
            dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
799 gcpy #
800 gcpy #
            //Lower right of entry hole
            {\tt dxfarc(getxpos(),getypos(),9.525/2,270,360,~KH\_tool\_num);}
801 gcpy #
802 gcpy #
            //Upper right of entry hole
```

```
{\tt dxfarc\,(getxpos\,()\,,getypos\,()\,,9.525/2\,,0\,,acos\,(tool\_diameter\,(}
803 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
804 gcpy #
             //Upper left of entry hole
            dxfarc(getxpos(),getypos(),9.525/2,180-acos(tool_diameter()
805 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180,KH_tool_num)
            //Actual line of cut
806 gcpy #
            \tt dxfline\,(getxpos\,()\,,getypos\,()\,,getxpos\,()\,,getypos\,()\,+kh\_distance)\,;
807 gcpy #
808 gcpy #
             //upper right of slot
             dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
809 gcpy #
           //upper left of slot
810 gcpy #
            dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
811 gcpy #
           KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
            //right of slot
812 gcpy #
813 gcpy #
             dxfline(
814 gcpy #
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
815 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,
816 gcpy #
817 gcpy #
             //end position at top of slot
                 getypos()+kh_distance,
818 gcpy #
819 gcpy #
                 KH_tool_num);
             dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))
820 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);
821 gcpy #
            hull(){
               translate([xpos(), ypos(), zpos()])\{
822 gcpy #
823 gcpy #
                 keyhole_shaft(6.35, 9.525);
824 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
825 gcpy #
                 keyhole_shaft(6.35, 9.525);
826 gcpy #
827 gcpy #
828 gcpy #
829 gcpy #
            hull(){
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
830 gcpy #
831 gcpy #
                 keyhole_shaft(6.35, 9.525);
832 gcpy #
               translate\left( \texttt{[xpos(), ypos()+kh\_distance, zpos()-kh\_max\_depth]} \right)
833 gcpy #
                 keyhole_shaft(6.35, 9.525);
834 gcpy #
              }
835 gcpy #
836 gcpy #
837 gcpy #
             cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
             cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed
838 gcpy #
           ):
            setypos(getypos()-kh_distance);
839 gcpy #
840 gcpy #
          } else if (kh_angle == 180) {
             //Lower right of entry hole
841 gcpy #
            dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
842 gcpy #
             //Upper right of entry hole
843 gcpy #
            dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
844 gcpy #
            //Upper left of entry hole
845 gcpy #
            dxfarc(getxpos(),getypos(),9.525/2,90, 90+acos(tool_diameter()
846 gcpy #
           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
847 gcpy #
             //Lower left of entry hole
848 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
849 gcpy #
             //upper left of slot
            dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
850 gcpy #
           KH\_tool\_num, (kh\_max\_depth+6.35))/2,90,180, KH\_tool\_num);
851 gcpy #
             //lower left of slot
             dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
852 gcpy #
           KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
            //Actual line of cut
853 gcpy #
854 gcpy #
             \tt dxfline(getxpos(),getxpos()-kh\_distance,getypos());\\
855 gcpy #
             //upper left slot
856 gcpy #
            dxfline(
                 getxpos()-(sqrt((tool\_diameter(KH\_tool\_num,1)^2)-(
857 gcpy #
           tool_diameter(KH_tool_num, 5)^2))/2),
                 getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,//(
858 gcpy #
            (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
           kh_{max_depth-6.34})/2)^2,
                 getxpos()-kh_distance,
859 gcpy #
```

```
860 gcpy #
             getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
861 gcpy #
862 gcpy #
                 KH_tool_num);
863 gcpy #
             //lower right slot
864 gcpy #
             dxfline(
                 getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
865 gcpy #
            tool_diameter(KH_tool_num,5)^2))/2),
                 \tt getypos()-tool\_diameter(KH\_tool\_num\,,\,\,(kh\_max\_depth))/2,//(
866 gcpy #
            (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
           kh_{max_depth-6.34})/2)^2,
867 gcpy #
                 getxpos()-kh_distance,
868 gcpy #
             //end position at top of slot
869 gcpy #
                 getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
870 gcpy #
                 KH_tool_num);
             hull(){
871 gcpy #
               translate([xpos(), ypos(), zpos()])\{
872 gcpy #
873 gcpy #
                 keyhole_shaft(6.35, 9.525);
874 gcpy #
               translate ([xpos(), ypos(), zpos()-kh\_max\_depth]) \{
875 gcpy #
876 gcpy #
                 keyhole_shaft(6.35, 9.525);
877 gcpy #
878 gcpy #
             hull(){
879 gcpy #
880 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth])\{
                 keyhole_shaft(6.35, 9.525);
881 gcpy #
882 gcpy #
883 gcpy #
               translate ([xpos()-kh\_distance, ypos(), zpos()-kh\_max\_depth])
           {
884 gcpy #
                 keyhole_shaft(6.35, 9.525);
885 gcpy #
886 gcpy #
887 gcpy #
             cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
888 gcpy #
             cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed
889 gcpy #
             setxpos(getxpos()+kh_distance);
           } else if (kh_angle == 270) {
890 gcpy #
             //Upper right of entry hole
891 gcpy #
             dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
892 gcpy #
             //Upper left of entry hole
893 gcpy #
             {\tt dxfarc\,(getxpos\,()\,,getypos\,()\,,9.525/2\,,90\,,180\,,\ KH\_tool\_num\,)}\,;
894 gcpy #
895 gcpy #
             //lower right of slot
             {\tt dxfarc\,(getxpos\,()\,,getypos\,()\,-kh\_distance\,,tool\_diameter\,(}
896 gcpy #
           {\tt KH\_tool\_num}\;,\;\;({\tt kh\_max\_depth+4.36}))/2\;,270\;,360\;,\;\;{\tt KH\_tool\_num})\;;
897 gcpy #
             //lower left of slot
             dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
898 gcpy #
           KH_tool_num, (kh_max_depth+4.36))/2,180,270, KH_tool_num);
899 gcpy #
             //Actual line of cut
             dxfline(getxpos(),getypos(),getxpos(),getypos()-kh_distance);
900 gcpy #
901 gcpy #
             //right of slot
902 gcpy #
             dxfline(
903 gcpy #
                 getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
904 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,
905 gcpy #
906 gcpy #
             //end position at top of slot
907 gcpy #
                 getypos()-kh_distance,
908 gcpy #
                 KH_tool_num);
             //left of slot
909 gcpy #
910 gcpy #
             dxfline(
                 getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
911 gcpy #
912 gcpy #
                 \tt getypos()-(sqrt((tool\_diameter(KH\_tool\_num,1)^2)-(
            tool\_diameter(\texttt{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,
913 gcpy #
914 gcpy #
             //end position at top of slot
                 getypos()-kh_distance,
915 gcpy #
916 gcpy #
                 KH tool num);
             //Lower right of entry hole
917 gcpy #
918 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
919 gcpy #
             //Lower left of entry hole
             dxfarc(getxpos(),getypos(),9.525/2,180, 180+acos(tool_diameter
920 gcpy #
            (KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
             hull(){
921 gcpy #
922 gcpy #
               translate([xpos(), ypos(), zpos()]){
                 keyhole_shaft(6.35, 9.525);
923 gcpy #
```

```
924 gcpy #
              translate([xpos(), ypos(), zpos()-kh_max_depth]){
925 gcpy #
                keyhole_shaft(6.35, 9.525);
926 gcpy #
927 gcpy #
            7
928 gcpy #
            hull(){
929 gcpy #
930 gcpy #
              translate([xpos(), ypos(), zpos()-kh_max_depth]){
                keyhole_shaft(6.35, 9.525);
931 gcpy #
932 gcpy #
              translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
933 gcpy #
                 keyhole_shaft(6.35, 9.525);
934 gcpy #
              }
935 gcpy #
936 gcpy #
937 gcpy #
            cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
938 gcpy #
            cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed
939 gcpy #
            setypos(getypos()+kh_distance);
940 gcpy #
941 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.

```
def stockandtoolpaths(self, option = "stockandtoolpaths"):
943 gcpy
                if option == "stock":
944 дсру
945 gcpy
                    if self.generatepaths == False:
                         output(self.stock)
946 дсру
947 gcpy #
                         print("Outputting stock")
                    else:
948 gcpv
                        return self.stock
949 gcpy
950 дсру
                elif option == "toolpaths":
951 gcpy
                    if self.generatepaths == False:
                        output(self.toolpaths)
952 gcpy
953 дсру
                    else:
954 дсру
                         return self.toolpaths
                elif option == "rapids":
955 gcpy
956 дсру
                    if self.generatepaths == False:
                        output(self.rapids)
957 дсру
958 дсру
                    else:
959 gcpy
                         return self.rapids
960 дсру
                else:
                    part = self.stock.difference(self.toolpaths)
961 gcpy
                    if self.generatepaths == False:
962 gcpy
963 дсру
                         output(part)
964 дсру
                     else:
965 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
toolchange();	(TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
<pre>cutoneaxis_setfeed();</pre>	(PREPOSITION FOR RAPID PLUNGE) GOXOYO ZO.25 G1ZOF100 G1 X109.5 Y75 Z-8.35F400 Z9
<pre>cutwithfeed();</pre>	
closegcodefile();	M05 M02

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

G-code	Command/Module
(Design File:) (stockMin:0.00mm, -152.40mm, -34.92mm) (stockMax:109.50mm, -77.40mm, 0.00mm) (STOCK/BLOCK,109.50, 75.00, 34.92,0.00, 152.40, 34.92) G90 G21	opengcodefile(s)(); setupstock(
(Move to safe Z to avoid workholding) G53G0Z-5.000	movetosafez()
(Toolpath: Contour Toolpath 1) M05 (TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S10000	toolchange();
(PREPOSITION FOR RAPID PLUNGE)	writecomment()
G0X0.000Y-152.400 Z0.250	<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

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

```
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.

```
967 gcpy
            def writegc(self, *arguments):
968 дсру
                 if self.generategcode == True:
969 дсру
                     line_to_write = ""
                     for element in arguments:
970 gcpy
971 gcpy
                         line_to_write += element
                     self.gc.write(line_to_write)
972 gcpy
                     self.gc.write("\n")
973 дсру
974 gcpy
975 gcpy
            def writedxf(self, toolnumber, *arguments):
                 global dxfclosed
976 gcpy #
977 gcpy
                 line_to_write = ""
                 for element in arguments:
978 gcpy
979 gcpy
                     line_to_write += element
980 дсру
                 if self.generatedxf == True:
                     if self.dxfclosed == False:
981 дсру
                         self.dxf.write(line_to_write)
982 gcpy
983 дсру
                         self.dxf.write("\n")
                 if self.generatedxfs == True:
984 дсру
                     self.writedxfs(toolnumber, line_to_write)
985 дсру
986 дсру
987 дсру
            def writedxfs(self, toolnumber, line_to_write):
                  print("Processing writing toolnumber", toolnumber)
988 gcpy #
                  line_to_write =
989 gcpy #
990 gcpy #
                  for element in arguments:
                      line_to_write += element
991 gcpy #
                 if (toolnumber == 0):
992 дсру
993 дсру
                     return
994 дсру
                 elif self.generatedxfs == True:
                     if (self.large_square_tool_num == toolnumber):
995 дсру
996 дсру
                         self.dxflgsq.write(line_to_write)
                         self.dxflgsq.write("\n")
997 дсру
998 дсру
                     if (self.small_square_tool_num == toolnumber):
                          self.dxfsmsq.write(line_to_write)
999 дсру
                         self.dxfsmsq.write("\n")
1000 gcpy
```

```
1001 дсру
                     if (self.large_ball_tool_num == toolnumber):
                          self.dxflgbl.write(line_to_write)
1002 gcpy
1003 дсру
                          self.dxflgbl.write("\n")
                     if (self.small_ball_tool_num == toolnumber):
1004 gcpy
1005 gcpy
                          self.dxfsmbl.write(line_to_write)
1006 дсру
                          self.dxfsmbl.write("\n")
1007 дсру
                     if (self.large_V_tool_num == toolnumber):
                          self.dxflgV.write(line_to_write)
1008 gcpy
1009 дсру
                          self.dxflgV.write("\n")
                     if (self.small_V_tool_num == toolnumber):
1010 дсру
                          self.dxfsmV.write(line_to_write)
1011 gcpy
                          self.dxfsmV.write("\n")
1012 gcpy
                     if (self.DT_tool_num == toolnumber):
1013 дсру
                          self.dxfDT.write(line_to_write)
1014 дсру
1015 дсру
                          self.dxfDT.write("\n")
                     if (self.KH_tool_num == toolnumber):
1016 gcpy
                          self.dxfKH.write(line_to_write)
1017 gcpy
                          self.dxfKH.write("\n")
1018 дсру
                     if (self.Roundover_tool_num == toolnumber):
1019 gcpy
1020 gcpy
                          self.dxfRt.write(line_to_write)
                          self.dxfRt.write("\n")
1021 gcpy
1022 gcpy
                     if (self.MISC_tool_num == toolnumber):
                          self.dxfMt.write(line_to_write)
1023 gcpy
                          \verb|self.dxfMt.write("\n")|\\
1024 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

```
{\tt 122~gcpscad~\textbf{module}~opengcodefile(basefilename\,,~current tool num\,,~tool radius\,,}
               plunge, feed, speed) {
               gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
123 gcpscad
                   plunge, feed, speed);
124 gcpscad }
```

and Python:

```
def opengcodefile(self, basefilename = "export",
1026 gcpy
1027 gcpy
                                 currenttoolnum = 102,
                                 toolradius = 3.175,
1028 дсру
                                 plunge = 400,
1029 дсру
1030 дсру
                                 feed = 1600.
                                 speed = 10000
1031 gcpy
1032 gcpy
                 self.basefilename = basefilename
1033 gcpy
1034 дсру
                 self.currenttoolnum = currenttoolnum
                 self.toolradius = toolradius
1035 дсру
1036 дсру
                 self.plunge = plunge
                 self.feed = feed
1037 дсру
                 self.speed = speed
1038 дсру
                 if self.generategcode == True:
1039 gcpy
```

```
self.gcodefilename = basefilename + ".nc"
1040 gcpy
                      self.gc = open(self.gcodefilename, "w")
1041 gcpy
1042 gcpy
             def opendxffile(self, basefilename = "export"):
1043 gcpy
1044 gcpy
                 self.basefilename = basefilename
1045 gcpy #
                  global generatedxfs
                   global dxfclosed
1046 gcpy #
                  self.dxfclosed = False
1047 gcpy
                 if self.generatedxf == True:
1048 дсру
                      self.generatedxfs = False
1049 дсру
                      self.dxffilename = basefilename + ".dxf"
1050 gcpy
                      self.dxf = open(self.dxffilename, "w")
1051 gcpy
1052 gcpy
                      self.dxfpreamble(-1)
1053 дсру
1054 дсру
             def opendxffiles(self, basefilename = "export",
1055 gcpy
                                 large_square_tool_num = 0,
1056 gcpy
                                 small_square_tool_num = 0,
1057 gcpy
                                 large_ball_tool_num = 0,
                                 small_ball_tool_num = 0,
1058 дсру
1059 дсру
                                large_V_tool_num = 0,
                                 small_V_tool_num = 0,
1060 gcpy
1061 дсру
                                DT_tool_num = 0,
                                KH_tool_num = 0,
1062 gcpy
1063 дсру
                                Roundover_tool_num = 0,
                                {\tt MISC\_tool\_num} = {\tt O)}:
1064 дсру
                  global generatedxfs
1065 gcpy #
                 self.basefilename = basefilename
1066 gcpy
1067 gcpy
                 self.generatedxfs = True
                 self.large_square_tool_num = large_square_tool_num
1068 дсру
                 self.small_square_tool_num = small_square_tool_num
1069 дсру
                 self.large_ball_tool_num = large_ball_tool_num
1070 дсру
                 self.small_ball_tool_num = small_ball_tool_num
1071 gcpy
1072 дсру
                 self.large_V_tool_num = large_V_tool_num
                 self.small_V_tool_num = small_V_tool_num
1073 дсру
                 self.DT_tool_num = DT_tool_num
self.KH_tool_num = KH_tool_num
1074 gcpy
1075 gcpy
1076 дсру
                 self.Roundover_tool_num = Roundover_tool_num
                 self.MISC_tool_num = MISC_tool_num
if self.generatedxf == True:
1077 gcpy
1078 дсру
1079 дсру
                      if (large_square_tool_num > 0):
1080 дсру
                           self.dxflgsqfilename = basefilename + str(
                               large_square_tool_num) + ".dxf"
1081 gcpy #
                            print("Opening ", str(self.dxflgsqfilename))
1082 gcpy
                           self.dxflgsq = open(self.dxflgsqfilename, "w")
1083 дсру
                      if (small_square_tool_num > 0):
                           print("Opening small square")
self.dxfsmsqfilename = basefilename + str(
1084 gcpy #
1085 дсру
                               small_square_tool_num) + ".dxf"
                           self.dxfsmsq = open(self.dxfsmsqfilename, "w")
1086 gcpv
1087 дсру
                      if (large_ball_tool_num > 0):
                           print("Opening large ball")
1088 gcpy #
                           self.dxflgblfilename = basefilename + str(
1089 дсру
                               large_ball_tool_num) + ".dxf"
                           self.dxflgbl = open(self.dxflgblfilename, "w")
1090 gcpy
                      if (small_ball_tool_num > 0):
1091 gcpy
                           print("Opening small ball")
1092 gcpy #
1093 дсру
                           self.dxfsmblfilename = basefilename + str(
                               small_ball_tool_num) + ".dxf"
                           self.dxfsmbl = open(self.dxfsmblfilename, "w")
1094 gcpy
                      if (large_V_tool_num > 0):
    print("Opening large V")
1095 дсру
1096 gcpy #
                           self.dxflgVfilename = basefilename + str(
1097 gcpy
                               large_V_tool_num) + ".dxf"
                           self.dxflgV = open(self.dxflgVfilename, "w")
1098 дсру
                      if (small_V_tool_num > 0):
    print("Opening small V")
1099 дсру
1100 gcpy #
                           self.dxfsmVfilename = basefilename + str(
1101 gcpy
                               small_V_tool_num) + ".dxf"
                           self.dxfsmV = open(self.dxfsmVfilename, "w")
1102 gcpy
                       \mbox{if} \mbox{ (DT_tool_num > 0):} 
1103 дсру
                           print("Opening DT")
1104 gcpy #
1105 дсру
                           self.dxfDTfilename = basefilename + str(DT_tool_num
                               ) + ".dxf"
1106 дсру
                           self.dxfDT = open(self.dxfDTfilename, "w")
                      if (KH_tool_num > 0):
1107 дсру
1108 gcpy #
                           print("Opening KH")
                           self.dxfKHfilename = basefilename + str(KH_tool_num
1109 дсру
                               ) + ".dxf"
```

```
1110 дсру
                         self.dxfKH = open(self.dxfKHfilename, "w")
                     if (Roundover_tool_num > 0):
1111 gcpy
1112 gcpy #
                          print("Opening Rt")
                         self.dxfRtfilename = basefilename + str(
1113 gcpy
                             Roundover_tool_num) + ".dxf"
                         self.dxfRt = open(self.dxfRtfilename, "w")
1114 дсру
1115 дсру
                     if (MISC_tool_num > 0):
                          print("Opening Mt")
1116 gcpy #
1117 gcpy
                         self.dxfMtfilename = basefilename + str(
                             MISC_tool_num) + ".dxf"
                         self.dxfMt = open(self.dxfMtfilename, "w")
1118 дсру
```

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

```
1119 дсру
                                                                          if (large_square_tool_num > 0):
1120 gcpy
                                                                                          self.dxfpreamble(large_square_tool_num)
                                                                          if (small square tool num > 0):
1121 gcpy
1122 gcpy
                                                                                          self.dxfpreamble(small_square_tool_num)
                                                                          if (large_ball_tool_num > 0):
1123 gcpy
                                                                                          self.dxfpreamble(large_ball_tool_num)
1124 gcpy
                                                                          if (small_ball_tool_num > 0):
1125 gcpy
1126 дсру
                                                                                          self.dxfpreamble(small_ball_tool_num)
1127 дсру
                                                                          if (large_V_tool_num > 0):
1128 дсру
                                                                                          self.dxfpreamble(large_V_tool_num)
                                                                          if (small_V_tool_num > 0):
1129 gcpy
1130 gcpy
                                                                                          self.dxfpreamble(small_V_tool_num)
1131 дсру
                                                                          if (DT_tool_num > 0):
1132 gcpy
                                                                                          self.dxfpreamble(DT_tool_num)
                                                                           \begin{tabular}{ll} \be
1133 gcpy
1134 дсру
                                                                                          self.dxfpreamble(KH_tool_num)
1135 gcpy
                                                                                   (Roundover_tool_num > 0):
                                                                                          self.dxfpreamble(Roundover_tool_num)
1136 gcpy
1137 дсру
                                                                          if (MISC_tool_num > 0):
1138 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.

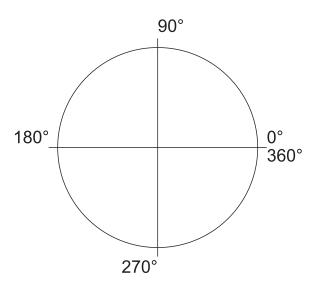
```
1140 gcpydefdxfpreamble(self, tn):1141 gcpy #self.writedxf(tn,str(tn))1142 gcpyself.writedxf(tn,"0")1143 gcpyself.writedxf(tn,"SECTION")1144 gcpyself.writedxf(tn,"2")1145 gcpyself.writedxf(tn,"ENTITIES")
```

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

dxfline

- a line dvfline
- 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.

```
1147 gcpy
              def dxfline(self, tn, xbegin,ybegin,xend,yend):
                  self.writedxf(tn,"0")
1148 дсру
                  self.writedxf(tn,"LWPOLYLINE")
self.writedxf(tn,"90")
1149 gcpy
1150 gcpy
                  self.writedxf(tn,"2")
1151 gcpy
                  self.writedxf(tn,"70")
1152 gcpy
                  self.writedxf(tn,"0")
1153 gcpy
                  self.writedxf(tn,"43")
1154 gcpy
                  self.writedxf(tn,"0")
1155 gcpy
                  self.writedxf(tn,"10")
1156 gcpy
                  self.writedxf(tn,str(xbegin))
1157 gcpy
                  self.writedxf(tn,"20")
1158 дсру
                  self.writedxf(tn,str(ybegin))
1159 gcpy
                  self.writedxf(tn,"10")
1160 gcpy
                  \verb|self.writedxf(tn, \verb|str(xend)|)|
1161 gcpy
1162 gcpy
                  self.writedxf(tn,"20")
                  self.writedxf(tn,str(yend))
1163 gcpy
```

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 $(G_2 \text{ vs. } G_3)$.

```
self.writedxf(tn, "ARC")
1168 дсру
                     self.writedxf(tn, "10")
1169 gcpy
1170 дсру
                     self.writedxf(tn, str(xcenter))
                     self.writedxf(tn, "20")
1171 gcpy
1172 gcpy
                     self.writedxf(tn, str(ycenter))
1173 дсру
                     self.writedxf(tn, "40")
1174 дсру
                     self.writedxf(tn, str(radius))
                     self.writedxf(tn, "50")
self.writedxf(tn, str(anglebegin))
1175 дсру
1176 gcpy
                      self.writedxf(tn, "51")
1177 дсру
                      self.writedxf(tn, str(endangle))
1178 gcpy
1179 дсру
             def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
1180 gcpy
                 endangle):
                 if (self.generategcode == True):
1181 gcpy
                      self.writegc(tn, "(0)")
1182 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.

```
1184 дсру
            def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1185 gcpy #
                  global toolpath
                  toolpath = self.currenttool()
1186 gcpy #
1187 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1188 gcpy
                    radius,0,90)
                 if (self.zpos == ez):
1189 gcpy
1190 дсру
                    self.settzpos(0)
                 else:
1191 gcpy
1192 gcpy
                     self.settzpos((self.zpos()-ez)/90)
1193 gcpy #
                 self.setxpos(ex)
1194 gcpy #
                 self.setypos(ey)
                  self.setzpos(ez)
1195 gcpy #
1196 дсру
                 if self.generatepaths == True:
                     print("Unioning ucutarcNECCdxf utoolpath")
1197 дсру
1198 дсру
                     self.arcloop(1,90, xcenter, ycenter, radius)
                      self.toolpaths = self.toolpaths.union(toolpath)
1199 gcpy #
                 \verb"else":
1200 gcpy
1201 gcpy
                     toolpath = self.arcloop(1,90, xcenter, ycenter, radius)
                      print("Returning cutarcNECCdxf toolpath")
1202 gcpy #
                     return toolpath
1203 gcpy
1204 дсру
1205 дсру
            def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
                  global toolpath
1206 gcpy #
1207 gcpy #
                  toolpath = self.currenttool()
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1208 gcpy #
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1209 gcpy
                    radius, 90, 180)
                 if (self.zpos == ez):
1210 gcpy
                     self.settzpos(0)
1211 gcpy
1212 gcpy
                 else:
1213 дсру
                     self.settzpos((self.zpos()-ez)/90)
1214 gcpy #
                  self.setxpos(ex)
                 self.setypos(ey)
1215 gcpy #
1216 gcpy #
                 self.setzpos(ez)
1217 gcpy
                 if self.generatepaths == True:
                     self.arcloop(91,180, xcenter, ycenter, radius)
1218 дсру
1219 gcpy #
                      self.toolpaths = self.toolpaths.union(toolpath)
1220 gcpy
                 else:
                     toolpath = self.arcloop(91,180, xcenter, ycenter,
1221 gcpy
                         radius)
1222 gcpy
                     return toolpath
1223 gcpy
            def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1224 gcpy
1225 gcpy #
                  global toolpath
1226 gcpy #
                  toolpath = self.currenttool()
1227 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
1228 дсру
                 self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius, 180, 270)
1229 gcpy
                 if (self.zpos == ez):
1230 дсру
                     self.settzpos(0)
1231 дсру
                 else:
                     self.settzpos((self.zpos()-ez)/90)
1232 gcpy
1233 gcpy #
                  self.setxpos(ex)
                 self.setypos(ey)
1234 gcpy #
```

```
1235 gcpy #
                  self.setzpos(ez)
                 if self.generatepaths == True:
1236 gcpy
1237 дсру
                     self.arcloop(181,270, xcenter, ycenter, radius)
                      self.toolpaths = self.toolpaths.union(toolpath)
1238 gcpy #
1239 gcpy
                 else:
                     toolpath = self.arcloop(181,270, xcenter, ycenter,
1240 дсру
                         radius)
1241 gcpy
                      return toolpath
1242 gcpy
             def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1243 gcpy
                  global toolpath
1244 gcpy #
                   toolpath = self.currenttool()
1245 gcpy #
1246 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(). xcenter.vcenter.
1247 gcpy
                     radius,270,360)
1248 дсру
                 if (self.zpos == ez):
1249 дсру
                     self.settzpos(0)
                 else:
1250 gcpy
                     self.settzpos((self.zpos()-ez)/90)
1251 gcpy
1252 gcpy #
                 self.setxpos(ex)
1253 gcpy #
                  self.setypos(ey)
                  self.setzpos(ez)
1254 gcpy #
1255 gcpy
                 if self.generatepaths == True:
                      self.arcloop(271,360, xcenter, ycenter, radius)
1256 gcpy
                      self.toolpaths = self.toolpaths.union(toolpath)
1257 gcpy #
                 else:
1258 gcpy
1259 gcpy
                     toolpath = self.arcloop(271,360, xcenter, ycenter,
                         radius)
1260 gcpy
                      return toolpath
1261 gcpy
1262 gcpy
             def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1263 gcpy #
                  global toolpath
                   toolpath = self.currenttool()
1264 gcpy #
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1265 gcpy #
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1266 gcpy
                     radius,0,90)
                 if (self.zpos == ez):
1267 gcpy
                     self.settzpos(0)
1268 дсру
1269 gcpy
                 else:
                     self.settzpos((self.zpos()-ez)/90)
1270 gcpy
1271 gcpy #
                  self.setxpos(ex)
                  self.setypos(ey)
1272 gcpy #
1273 gcpy #
                  self.setzpos(ez)
1274 gcpy
                 if self.generatepaths == True:
                      self.narcloop(89,0, xcenter, ycenter, radius)
1275 gcpy
                      self.toolpaths = self.toolpaths.union(toolpath)
1276 gcpy #
                 else:
1277 gcpy
                     toolpath = self.narcloop(89,0, xcenter, ycenter, radius
1278 gcpy
1279 дсру
                      return toolpath
1280 gcpy
1281 дсру
             \begin{center} \textbf{def} \ \texttt{cutarcSECWdxf} (\texttt{self}, \ \texttt{ex}, \ \texttt{ey}, \ \texttt{ez}, \ \texttt{xcenter}, \ \texttt{ycenter}, \ \texttt{radius}) : \\ \end{center}
                  global toolpath
1282 gcpy #
1283 gcpy #
                   toolpath = self.currenttool()
                  toolpath = toolpath.translate([self.xpos(),self.ypos(),
1284 gcpy #
            self.zpos()])
                 self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1285 gcpy
                     radius,270,360)
                 if (self.zpos == ez):
1286 gcpy
1287 gcpy
                     self.settzpos(0)
1288 gcpy
                 else:
1289 дсру
                     self.settzpos((self.zpos()-ez)/90)
1290 gcpy #
                  self.setxpos(ex)
1291 gcpy #
                  self.setypos(ey)
                  self.setzpos(ez)
1292 gcpy #
1293 gcpv
                 if self.generatepaths == True:
                      self.narcloop(359,270, xcenter, ycenter, radius)
1294 дсру
1295 gcpy #
                      self.toolpaths = self.toolpaths.union(toolpath)
1296 дсру
                 else:
1297 дсру
                     toolpath = self.narcloop(359,270, xcenter, ycenter,
                         radius)
1298 gcpy
                      return toolpath
1299 дсру
             def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1300 дсру
                  global toolpath
1301 gcpy #
                  toolpath = self.currenttool()
1302 gcpy #
```

```
toolpath = toolpath.translate([self.xpos(),self.ypos(),
1303 gcpy #
            self.zpos()])
1304 gcpy
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius, 180, 270)
                if (self.zpos == ez):
1305 gcpy
1306 дсру
                    self.settzpos(0)
1307 дсру
                else:
                    self.settzpos((self.zpos()-ez)/90)
1308 gcpy
1309 gcpy #
                 self.setxpos(ex)
1310 gcpy #
                 self.setypos(ey)
1311 gcpy #
                 self.setzpos(ez)
                if self.generatepaths == True:
1312 gcpy
                     self.narcloop(269,180, xcenter, ycenter, radius)
1313 дсру
                     self.toolpaths = self.toolpaths.union(toolpath)
1314 gcpy #
1315 дсру
                else:
                     toolpath = self.narcloop(269,180, xcenter, ycenter,
1316 gcpy
                        radius)
1317 дсру
                     return toolpath
1318 gcpy
            def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1319 gcpy
                 global toolpath
1320 gcpy #
1321 gcpy #
                  toolpath = self.currenttool()
                 toolpath = toolpath.translate([self.xpos(),self.ypos(),
1322 gcpy #
            self.zpos()])
                self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1323 gcpy
                    radius,90,180)
                if (self.zpos == ez):
1324 gcpy
1325 gcpy
                     self.settzpos(0)
1326 дсру
                else:
1327 дсру
                    self.settzpos((self.zpos()-ez)/90)
1328 gcpy #
                 self.setxpos(ex)
1329 gcpy #
                 self.setypos(ey)
1330 gcpy #
                 self.setzpos(ez)
                if self.generatepaths == True:
1331 дсру
                     self.narcloop(179,90, xcenter, ycenter, radius)
1332 gcpy
                      self.toolpaths = self.toolpaths.union(toolpath)
1333 gcpy #
1334 дсру
                 else:
                     toolpath = self.narcloop(179,90, xcenter, ycenter,
1335 дсру
                        radius)
                     return toolpath
1336 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:

```
1344 дсру
            def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
1345 gcpy
                 if self.generatepaths == True:
1346 дсру
1347 дсру
                     self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
1348 gcpy
                 else:
                     return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
1349 дсру
                          radius)
1350 gcpv
            def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1351 gcpy
                \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
1352 gcpv
                 if self.generatepaths == False:
1353 gcpy
                     return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
1354 gcpy
                          radius)
1355 дсру
1356 gcpy
            def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
```

```
1357 дсру
                   self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1358 дсру
 1359 дсру
                        return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                             radius)
 1360 gcpy
               {\tt def} \ {\tt cutarcSECCdxfgc(self,\ ex,\ ey,\ ez,\ xcenter,\ ycenter,\ radius)}
 1361 gcpy
                   \verb|self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)|\\
 1362 gcpy
                   if self.generatepaths == False:
 1363 gcpy
                       return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
 1364 дсру
                             radius)
 1365 gcpy
 1366 дсру
               def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1367 дсру
                   \verb|self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)|\\
 1368 gcpy
                   if self.generatepaths == False:
 1369 дсру
                        return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                             radius)
 1370 gcpy
              def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1371 gcpy
 1372 дсру
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1373 дсру
 1374 дсру
                       \textbf{return} \ \texttt{self.cutarcSECWdxf} (\texttt{ex}, \ \texttt{ey}, \ \texttt{ez}, \ \texttt{xcenter}, \ \texttt{ycenter},
                             radius)
 1375 дсру
              def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1376 gcpy
 1377 дсру
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
                   if self.generatepaths == False:
 1378 gcpy
                        return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
 1379 gcpy
                            radius)
 1380 дсру
              def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
 1381 gcpy
                   self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
 1382 gcpy
 1383 дсру
                   if self.generatepaths == False:
 1384 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){
              {\tt 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):
1386 дсру
                   self.writedxf(tn,str(tn))
1387 gcpy #
1388 дсру
                  self.writedxf(tn,"0")
                  self.writedxf(tn,"ENDSEC")
self.writedxf(tn,"0")
1389 дсру
1390 gcpy
                  self.writedxf(tn,"EOF")
1391 дсру
1393 дсру
              def gcodepostamble(self):
1394 дсру
                  self.writegc("Z12.700")
                  self.writegc("M05")
1395 дсру
1396 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.

```
1398 дсру
             def closegcodefile(self):
                 {\tt self.gcodepostamble} ()
1399 gcpy
1400 дсру
                 self.gc.close()
1401 gcpy
1402 gcpy
             def closedxffile(self):
1403 gcpy
                 if self.generatedxf == True:
                      global dxfclosed
1404 gcpy #
                     self.dxfpostamble(-1)
1405 gcpv
                      self.dxfclosed = True
1406 gcpy #
1407 gcpy
                     self.dxf.close()
1408 gcpy
1409 gcpy
            def closedxffiles(self):
                 if self.generatedxfs == True:
1410 gcpy
1411 gcpy
                     if (self.large_square_tool_num > 0):
                          self.dxfpostamble(self.large_square_tool_num)
1412 gcpy
                     if (self.small_square_tool_num > 0):
1413 gcpy
                          self.dxfpostamble(self.small_square_tool_num)
1414 gcpy
                     if (self.large_ball_tool_num > 0):
1415 gcpy
1416 gcpy
                          self.dxfpostamble(self.large_ball_tool_num)
                     if (self.small_ball_tool_num > 0):
1417 gcpy
1418 gcpy
                          self.dxfpostamble(self.small_ball_tool_num)
1419 дсру
                     if (self.large_V_tool_num > 0):
1420 gcpy
                          self.dxfpostamble(self.large_V_tool_num)
                     if (self.small_V_tool_num > 0):
1421 gcpy
1422 gcpy
                          self.dxfpostamble(self.small_V_tool_num)
                     if (self.DT_tool_num > 0):
1423 gcpy
1424 gcpy
                          self.dxfpostamble(self.DT_tool_num)
1425 дсру
                     if (self.KH_tool_num > 0):
1426 дсру
                          self.dxfpostamble(self.KH_tool_num)
                     if (self.Roundover_tool_num > 0):
1427 gcpy
                          self.dxfpostamble(self.Roundover_tool_num)
1428 gcpy
                     if (self.MISC_tool_num > 0):
1429 gcpy
1430 gcpy
                          self.dxfpostamble(self.MISC_tool_num)
1431 дсру
1432 gcpy
                     if (self.large_square_tool_num > 0):
                          self.dxflgsq.close()
1433 дсру
                     if (self.small_square_tool_num > 0):
1434 gcpy
1435 gcpy
                          self.dxfsmsq.close()
                     if (self.large_ball_tool_num > 0):
1436 gcpy
1437 дсру
                          self.dxflgbl.close()
1438 дсру
                     if (self.small_ball_tool_num > 0):
1439 gcpy
                          self.dxfsmbl.close()
                     if (self.large_V_tool_num > 0):
1440 gcpy
                          self.dxflgV.close()
1441 gcpy
                     if (self.small_V_tool_num > 0):
1442 gcpy
                          self.dxfsmV.close()
1443 дсру
                     if (self.DT_tool_num > 0):
1444 gcpy
1445 дсру
                          self.dxfDT.close()
                     if (self.KH_tool_num > 0):
1446 gcpy
1447 gcpy
                          self.dxfKH.close()
                     if (self.Roundover_tool_num > 0):
1448 gcpy
1449 gcpy
                          self.dxfRt.close()
1450 gcpy
                     if (self.MISC_tool_num > 0):
1451 gcpy
                          self.dxfMt.close()
```

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 60

4 Notes

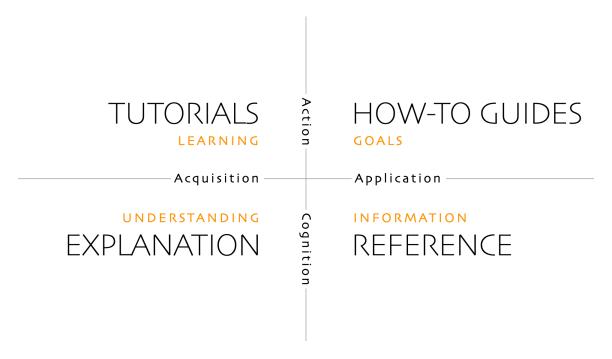
Other Resources

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:

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

Adding a Command Glossary may be a useful addition or alternative to the Index.

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

4 Notes 61

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
- ZY

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

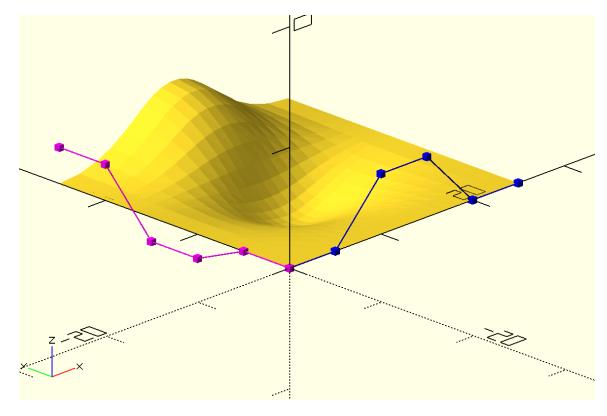
```
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs
```

which is a marked contrast to representations such as:

https://github.com/DavidPhillipOster/Teapot

and regions which could not be so represented could be sub-divided until the representation is workable.

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



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

References 62

References

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

1981.

[MkCalc] Horvath, Joan, and Rich Cameron. Make: Calculus: Build models to learn,

visualize, and explore. First edition., Make: Community LLC, 2022.

[MkGeom] Horvath, Joan, and Rich Cameron. Make: Geometry: Learn by 3D Printing,

Coding and Exploring. First edition., Make: Community LLC, 2021.

[MkTrig] Horvath, Joan, and Rich Cameron. Make: Trigonometry: Build your way from

triangles to analytic geometry. First edition., Make: Community LLC, 2023.

[PractShopMath] Begnal, Tom. Practical Shop Math: Simple Solutions to Workshop Fractions,

Formulas + Geometric Shapes. Updated edition, Spring House Press, 2018.

[RS274] Thomas R. Kramer, Frederick M. Proctor, Elena R. Messina.

https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374

https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3

[Software Design] Ousterhout, John K. A Philosophy of Software Design. First Edition., Yaknyam

Press, Palo Alto, Ca., 2018

Index

ballnose, 26	plunge, 31
closedxffile, 58, 59	rapid, 31
closegcodefile, 58, 59	rcs, 31
currenttoolnum, 22	
currenttoolnumber, 25	settool, 25
currenttoolshape, 28	setupstock, 23
cut, 31, 33	gcodepreview, 23
cutarcCC, 35	setxpos, 22
cutarcCW, 35	setypos, 22
cutkeyhole toolpath, 41	setzpos, 22
cutKHgcdxf, 42	speed, 31
cutline, 33	stepsizearc, 19
1	stepsizeroundover, 19
dovetail, 27	subroutine
dxfarc, 54	gcodepreview, 23
dxfline, 54	writeln, 50
dxfpostamble, 59	, , , , , , , , , , , , , , , , , , , ,
dxfpreamble, 53	threadmill, 27
dxfwrite, 53	tool diameter, 30
andmill square 26	tool radius, 31
endmill square, 26 endmill v, 26	toolchange, 28
endinii v, 20	tpzinc, 22
feed, 31	-r ====, <u>==</u>
reca, gr	writedxfDT, 53
gcodepreview, 20	writedxfKH, 53
writeln, 50	writedxflgbl, 53
gcp.setupstock, 23	writedxflgsq, 53
	writedxflgV, 53
init, 20	writedxfsmbl, 53
	writedxfsmsq, 53
keyhole, 27	writedxfsmV, 53
mpx, 22	, 33
mpy, 22	xpos, 22
mpz, 22	_
1 '	ypos, <mark>22</mark>
opendxffile, 51	
opengcodefile, 51	zpos, <mark>22</mark>
	_

Routines

ballnose, 26	rapid, 31 rcs, 31
closedxffile, 58, 59	105, 31
closegcodefile, 58, 59	settool, 25
currenttoolnumber, 25	setupstock, 23
cut, 31, 33	setxpos, 22
cutarcCC, 35	setypos, 22
cutarcCW, 35	setzpos, 22
cutkeyhole toolpath, 41	1
cutKHgcdxf, 42	threadmill, 27
cutline, 33	tool diameter, 30
dovetail, 27	tool radius, 31
dxfarc, 54	toolchange, 28
dxfline, 54	
dxfpostamble, 59	writedxfDT, 53
dxfpreamble, 53	writedxfKH, 53
dxfwrite, 53	writedxflgbl, 53
	writedxflgsq, 53
endmill square, 26	writedxflgV, 53
endmill v, <mark>26</mark>	writedxfsmbl, 53
gcodepreview, 20, 23	writedxfsmsq, 53
gcp.setupstock, 23	writedxfsmV, 53
gep setup stocky 25	writeln, 50
init, 20	V1206 22
	xpos, 22
keyhole, 27	ypos, 22
opendxffile, 51	JP 30, 22
opengcodefile, 51	zpos, 22
or	-r,

Variables

```
currenttoolnum, 22
currenttoolshape, 28

feed, 31

mpx, 22

mpy, 22

mpz, 22

mpz, 22

plunge, 31

speed, 31

stepsizearc, 19

stepsizeroundover, 19

tpzinc, 22
```