

The gcodepreview PythonSCAD library*

Author: William F. Adams
willadams at aol dot com

2025/02/14

Abstract

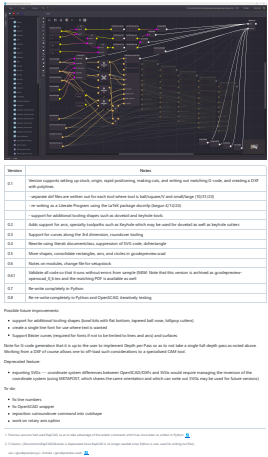
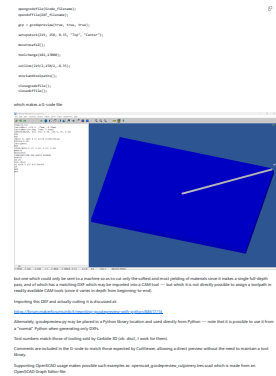
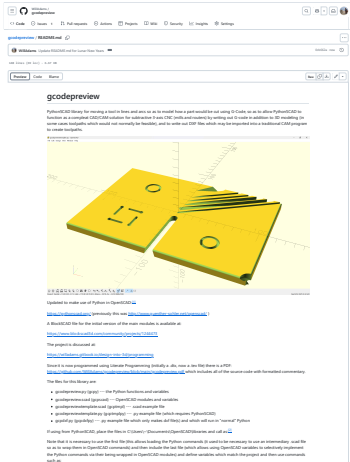
The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.

Contents

1	readme.md	2
2	Usage and Templates	6
2.1	gcpdxf.py	6
2.2	gcodepreviewtemplate.py	9
2.3	gcodepreviewtemplate.scad	14
3	gcodepreview	19
3.1	Module Naming Convention	19
3.1.1	Parameters and Default Values	21
3.2	Implementation files and gcodepreview class	22
3.2.1	Position and Variables	24
3.2.2	Initial Modules	25
3.3	Tools and Changes	29
3.3.1	Numbering for Tools	29
3.3.2	3D Shapes for Tools	31
3.3.2.1	Normal Tooling/toolshapes	31
3.3.2.2	Tooling for Undercutting Toolpaths	32
3.3.2.2.1	Keyhole tools	33
3.3.2.2.2	Thread mills	33
3.3.2.2.3	Dovetails	34
3.3.2.3	Concave toolshapes	34
3.3.2.4	Roundover tooling	34
3.3.3	toolchange	34
3.3.3.1	Selecting Tools	34
3.3.3.2	Square and ball nose (including tapered ball nose)	35
3.3.3.3	Roundover (corner rounding)	35
3.3.3.4	Dovetails	35
3.3.3.5	toolchange routine	35
3.3.4	tooldiameter	36
3.3.5	Feeds and Speeds	38
3.4	Movement and Cutting	38
3.4.1	Lines	41
3.4.2	Arcs for toolpaths and DXFs	42
3.4.3	Cutting shapes and expansion	46
3.4.3.1	Building blocks	46
3.4.3.2	List of shapes	46
3.4.3.2.1	circles	48
3.4.3.2.2	rectangles	48
3.4.3.2.3	Keyhole toolpath and undercut tooling	50
3.4.4	Difference of Stock, Rapids, and Toolpaths	64
3.5	Output files	65
3.5.1	G-code Overview	65
3.5.2	DXF Overview	66
3.5.3	Python and OpenSCAD File Handling	67
3.5.3.1	Writing to DXF files	70
3.5.3.2	Closings	77
4	Notes	80
	Index	85
	Routines	86
	Variables	87

*This file (gcodepreview) has version number vo.802, last revised 2025/02/14.

1 **readme.md**



```
1 rdme # gcodepreview
2 rdme
3 rdme PythonSCAD library for moving a tool in lines and arcs so as to
  model how a part would be cut using G-Code, so as to allow
  PythonSCAD to function as a compleat CAD/CAM solution for
  subtractive 3-axis CNC (mills and routers at this time, 4th-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 DXF files which may be
  imported into a traditional CAM program to create toolpaths.
4 rdme
5 rdme ![OpenSCAD gcodepreview Unit Tests](https://raw.githubusercontent.com/WillAdams/gcodepreview/main/gcodepreview_unittests.png?raw=true)
6 rdme
7 rdme Updated to make use of Python in OpenSCAD:[^rapcad]
8 rdme
9 rdme [^rapcad]: Previous versions had used RapCAD, so as to take
  advantage of the writeln command, which has since been re-
  written in Python.
10 rdme
11 rdme https://pythonscad.org/ (previously this was http://www.guenther-
  sohler.net/openscad/ )
12 rdme
13 rdme A BlockSCAD file for the initial version of the
14 rdme main modules is available at:
15 rdme
16 rdme https://www.blockscad3d.com/community/projects/1244473
17 rdme
18 rdme The project is discussed at:
19 rdme
20 rdme https://willadams.gitbook.io/design-into-3d/programming
21 rdme
22 rdme Since it is now programmed using Literate Programming (initially a
  .dtx, now a .tex file) there is a PDF: https://github.com/
  WillAdams/gcodepreview/blob/main/gcodepreview.pdf which includes
  all of the source code with formatted comments.
23 rdme
24 rdme The files for this library are:
25 rdme
26 rdme - gcodepreview.py (gcpy) --- the Python class/functions and
  variables
27 rdme - gcodepreview.scad (gcpscad) --- OpenSCAD modules and parameters
28 rdme
29 rdme And there several sample/template files which may be used as the
  starting point for a given project:
30 rdme
31 rdme - gcodepreviewtemplate.scad (gcptmpl) --- .scad example file
32 rdme - gcodepreviewtemplate.py (gcptmplpy) --- .py example file
33 rdme - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf
  file(s) and which will run in "normal" Python in addition to
  PythonSCAD
34 rdme
35 rdme If using from PythonSCAD, place the files in C:\Users\\~\Documents
  \OpenSCAD\libraries [^libraries] or, load them from Github using
  the command:
36 rdme
37 rdme nimport("https://raw.githubusercontent.com/WillAdams/
```

```

        gcodepreview/refs/heads/main/gcodepreview.py")
38 rdme
39 rdme [^libraries]: C:\Users\\-\\Documents\RapCAD\libraries is deprecated
        since RapCAD is no longer needed since Python is now used for
        writing out files.
40 rdme
41 rdme If using gcodepreview.scad call as:
42 rdme
43 rdme     use <gcodepreview.py>
44 rdme     include <gcodepreview.scad>
45 rdme
46 rdme Note that it is necessary to use the first file (this allows
        loading the Python commands and then include the last file (
        which allows using OpenSCAD variables to selectively implement
        the Python commands via their being wrapped in OpenSCAD modules)
        and define variables which match the project and then use
        commands such as:
47 rdme
48 rdme    .opengcodefile(Gcode_filename);
49 rdme    .opendxffile(DXF_filename);
50 rdme
51 rdme     gcp = gcodepreview(true, true, true);
52 rdme
53 rdme     setupstock(219, 150, 8.35, "Top", "Center");
54 rdme
55 rdme     movetosafeZ();
56 rdme
57 rdme     toolchange(102, 17000);
58 rdme
59 rdme     cutline(219/2, 150/2, -8.35);
60 rdme
61 rdme     stockandtoolpaths();
62 rdme
63 rdme     closegcodefile();
64 rdme     closedxfile();
65 rdme
66 rdme which makes a G-code file:
67 rdme
68 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
        WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
69 rdme
70 rdme but one which could only be sent to a machine so as to cut only the
        softest and most yielding of materials since it makes a single
        full-depth pass, and 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 which is not
        included in the DXF since few tools make use of that information
        ).
71 rdme
72 rdme Importing this DXF and actually cutting it is discussed at:
73 rdme
74 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
        /88617/14
75 rdme
76 rdme Alternately, gcodepreview.py may be placed in a Python library
        location and used directly from Python --- note that it is
        possible to use it from a "normal" Python when generating only
        DXFs as shown in gcpxdf.py.
77 rdme
78 rdme In the current version, tool numbers match those of tooling sold by
        Carbide 3D (ob. discl., I work for them), but a vendor-neutral
        system is in the process of being developed (the original
        numbers will still be present as 9#### where the #s indicate
        the original tool number with zero padding to fill them out
        where necessary).
79 rdme
80 rdme Comments are included in the G-code to match those expected by
        CutViewer, allowing a direct preview without the need to
        maintain a tool library (for such tooling as that program
        supports).
81 rdme
82 rdme Supporting OpenSCAD usage makes possible such examples as:
        openscad_gcodepreview_cutjoinery.tres.scad which is made from an
        OpenSCAD Graph Editor file:
83 rdme
84 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
        githubusercontent.com/WillAdams/gcodepreview/main/

```

```

OSGE_cutjoinery.png?raw=true)
85 rdme
86 rdme | Version          | Notes          |
87 rdme | ----- | ----- |
88 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. |
89 rdme |              | - separate dxf files are written out for each
           tool where tool is ball/square/V and small/large (10/31/23)

           |
90 rdme |              | - re-writing as a Literate Program using the
           LaTeX package docmfp (begun 4/12/24)

           |
91 rdme |              | - support for additional tooling shapes such as
           dovetail and keyhole tools

           |
92 rdme | 0.2          | Adds support for arcs, specialty toolpaths such
           as Keyhole which may be used for dovetail as well as keyhole
           cutters

           |
93 rdme | 0.3          | Support for curves along the 3rd dimension,
           roundover tooling

           |
94 rdme | 0.4          | Rewrite using literati documentclass, suppression
           of SVG code, dxfrextangle

           |
95 rdme | 0.5          | More shapes, consolidate rectangles, arcs, and
           circles in gcodepreview.scad

           |
96 rdme | 0.6          | Notes on modules, change file for setupstock

           |
97 rdme | 0.61         | 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) |
98 rdme | 0.7          | Re-write completely in Python

           |
99 rdme | 0.8          | Re-re-write completely in Python and OpenSCAD,
           iteratively testing

           |
100 rdme | 0.801        | Add support for bowl bits with flat bottom

           |
101 rdme | 0.802        | Add support for tapered ball-nose and V tools
           with flat bottom

           |
102 rdme | 0.803        | Implement initial colour support and joinery
           modules (dovetail and full blind box joint modules)

           |
103 rdme
104 rdme Possible future improvements:
105 rdme
106 rdme - support for post-processors
107 rdme - support for 4th-axis
108 rdme - support for two-sided machining (import an STL or other file to
           use for stock)
109 rdme - implement tool-numbering scheme
110 rdme - support for additional tooling shapes (lollipop cutters)
111 rdme - create a single line font for use where text is wanted
112 rdme - Support Bézier curves (required for fonts if not to be limited
           to lines and arcs) and surfaces
113 rdme
114 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.
115 rdme

```

```
116 rdme Deprecated feature:
117 rdme
118 rdme - exporting SVGs --- coordinate system differences between
      OpenSCAD/DXFs and SVGs would require managing the inversion of
      the coordinate system (using METAPOST, which shares the same
      orientation and which can write out SVGs may be used for future
      versions)
119 rdme
120 rdme To-do:
121 rdme
122 rdme - add conditional option to toggle between creation of manual and
      Literate Source
123 rdme - fix OpenSCAD wrapper and add any missing commands for Python
124 rdme - reposition cutroundover command into cutshape
125 rdme - re-work architecture so that a tool shape is defined as a list,
      with shaft always defined/included and annotated as such (in a
      different colour so as to identify instances of rubbing)
126 rdme - work on rotary axis option
```

```

7 gcpdxfpyp                                     True    # generatedxf
8 gcpdxfpyp                                     )
9 gcpdxfpyp
10 gcpdxfpyp # [Stock] */
11 gcpdxfpyp stockXwidth = 100
12 gcpdxfpyp # [Stock] */
13 gcpdxfpyp stockYheight = 50
14 gcpdxfpyp
15 gcpdxfpyp # [Export] */
16 gcpdxfpyp Base_filename = "gcpdxf"
17 gcpdxfpyp
18 gcpdxfpyp
19 gcpdxfpyp # [CAM] */
20 gcpdxfpyp large_square_tool_num = 102
21 gcpdxfpyp # [CAM] */
22 gcpdxfpyp small_square_tool_num = 0
23 gcpdxfpyp # [CAM] */
24 gcpdxfpyp large_ball_tool_num = 0
25 gcpdxfpyp # [CAM] */
26 gcpdxfpyp small_ball_tool_num = 0
27 gcpdxfpyp # [CAM] */
28 gcpdxfpyp large_V_tool_num = 0
29 gcpdxfpyp # [CAM] */
30 gcpdxfpyp small_V_tool_num = 0
31 gcpdxfpyp # [CAM] */
32 gcpdxfpyp DT_tool_num = 374
33 gcpdxfpyp # [CAM] */
34 gcpdxfpyp KH_tool_num = 0
35 gcpdxfpyp # [CAM] */
36 gcpdxfpyp Roundover_tool_num = 0
37 gcpdxfpyp # [CAM] */
38 gcpdxfpyp MISC_tool_num = 0
39 gcpdxfpyp
40 gcpdxfpyp # [Design] */
41 gcpdxfpyp inset = 3
42 gcpdxfpyp # [Design] */
43 gcpdxfpyp radius = 6
44 gcpdxfpyp # [Design] */
45 gcpdxfpyp cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
46 gcpdxfpyp
47 gcpdxfpyp gcp.opendxfpfile(Base_filename)
48 gcpdxfpyp #gcp.opendxfpfiles(Base_filename,
49 gcpdxfpyp #                               large_square_tool_num,
50 gcpdxfpyp #                               small_square_tool_num,
51 gcpdxfpyp #                               large_ball_tool_num,
52 gcpdxfpyp #                               small_ball_tool_num,
53 gcpdxfpyp #                               large_V_tool_num,
54 gcpdxfpyp #                               small_V_tool_num,
55 gcpdxfpyp #                               DT_tool_num,
56 gcpdxfpyp #                               KH_tool_num,
57 gcpdxfpyp #                               Roundover_tool_num,
58 gcpdxfpyp #                               MISC_tool_num)
59 gcpdxfpyp
60 gcpdxfpyp gcp.dxfrectangle(large_square_tool_num, 0, 0, stockXwidth,
                               stockYheight)
61 gcpdxfpyp
62 gcpdxfpyp gcp.setdxfcolor("Red")
63 gcpdxfpyp
64 gcpdxfpyp gcp.dxfarc(large_square_tool_num, inset, inset, radius, 0, 90)
65 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth - inset, inset,
                               radius, 90, 180)
66 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
                               - inset, radius, 180, 270)
67 gcpdxfpyp gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
                               radius, 270, 360)
68 gcpdxfpyp
69 gcpdxfpyp gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
                               stockYheight - (inset + radius))
70 gcpdxfpyp gcp.dxfline(large_square_tool_num, inset + radius, inset,
                               stockXwidth - (inset + radius), inset)
71 gcpdxfpyp gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
                               radius, stockXwidth - inset, stockYheight - (inset + radius))
72 gcpdxfpyp gcp.dxfline(large_square_tool_num, inset + radius, stockYheight -
                               inset, stockXwidth - (inset + radius), stockYheight - inset)
73 gcpdxfpyp
74 gcpdxfpyp gcp.setdxfcolor("Blue")
75 gcpdxfpyp
76 gcpdxfpyp gcp.dxfrectangle(large_square_tool_num, radius +inset, radius,

```

```

        stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
        cornerstyle, radius)
77 gcpdxfpyp gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
        2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
        (radius * 2), cornerstyle, radius)
78 gcpdxfpyp #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius
        )
79 gcpdxfpyp #gcp.dxfrectanglechamfer(large_square_tool_num, 64, 7, 24, 36,
        radius)
80 gcpdxfpyp #gcp.dxfrectangleflippedfillet(large_square_tool_num, 64, 7, 24,
        36, radius)
81 gcpdxfpyp
82 gcpdxfpyp gcp.setdxfcolor("Black")
83 gcpdxfpyp
84 gcpdxfpyp gcp.beginpolyline(large_square_tool_num)
85 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
        stockYheight/4)
86 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
        stockYheight*0.75)
87 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*2,
        stockYheight*0.75-radius)
88 gcpdxfpyp gcp.closepolyline(large_square_tool_num)
89 gcpdxfpyp
90 gcpdxfpyp ##gcp.setdxfcolor("White")
91 gcpdxfpyp
92 gcpdxfpyp gcp.setdxfcolor("Yellow")
93 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/4, stockYheight/4,
        radius/2)

94 gcpdxfpyp
95 gcpdxfpyp gcp.setdxfcolor("Green")
96 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75, stockYheight
        *0.75, radius/2)

97 gcpdxfpyp
98 gcpdxfpyp gcp.setdxfcolor("Cyan")
99 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/4, stockYheight
        *0.75, radius/2)

100 gcpdxfpyp
101 gcpdxfpyp gcp.setdxfcolor("Magenta")
102 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75, stockYheight
        /4, radius/2)

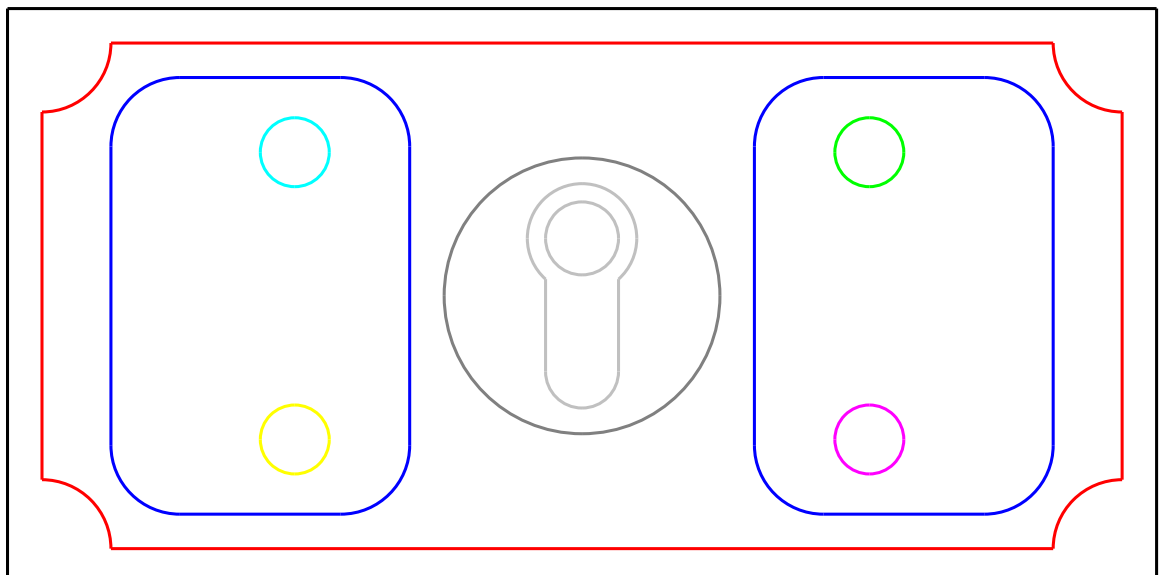
103 gcpdxfpyp
104 gcpdxfpyp gcp.setdxfcolor("Dark_□Gray")
105 gcpdxfpyp
106 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2,
        radius * 2)

107 gcpdxfpyp
108 gcpdxfpyp gcp.setdxfcolor("Light_□Gray")
109 gcpdxfpyp
110 gcpdxfpyp gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270,
        11.5875)

111 gcpdxfpyp
112 gcpdxfpyp #gcp.closedxfiles()
113 gcpdxfpyp gcp.closedxfile()

```

which creates:



and which may be imported into pretty much any CAD or CAM application. Note that the lines referencing multiple files (open/closedxfiles) 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)
- dxfar (specifying arc center, radius, and beginning/ending angles)
- dxfkH (specifying origin, depth, angle, distance)

2.2 gcodepreviewtemplate.py

Note that since the v0.7 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 try:
6 gcptmplpy     if 'gcodepreview' in sys.modules:
7 gcptmplpy         del sys.modules['gcodepreview']
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, 808079:808079]
54 gcptmplpy # [CAM] */

```

```

55 gcptmplpy KH_tool_num = 374 # [0:0, 374:374, 375:375, 376:376, 378:378]
56 gcptmplpy # [CAM] */
57 gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 1570:1570]
58 gcptmplpy # [CAM] */
59 gcptmplpy MISC_tool_num = 0 # [501:501, 502:502, 45982:45982]
60 gcptmplpy #501 https://shop.carbide3d.com/collections/cutters/products/501-
    engraving-bit
61 gcptmplpy #502 https://shop.carbide3d.com/collections/cutters/products/502-
    engraving-bit
62 gcptmplpy #204 tapered ball nose 0.0625", 0.2500", 1.50", 3.6ř
63 gcptmplpy #304 tapered ball nose 0.1250", 0.2500", 1.50", 2.4ř
64 gcptmplpy #648 threadmill_shaft(2.4, 0.75, 18)
65 gcptmplpy #45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
    Inch Shank
66 gcptmplpy #13921 https://www.amazon.com/Yonico-Groove-Bottom-Router-Degree/dp
    /BOCPJPTMP
67 gcptmplpy
68 gcptmplpy # [Feeds and Speeds] */
69 gcptmplpy plunge = 100
70 gcptmplpy # [Feeds and Speeds] */
71 gcptmplpy feed = 400
72 gcptmplpy # [Feeds and Speeds] */
73 gcptmplpy speed = 16000
74 gcptmplpy # [Feeds and Speeds] */
75 gcptmplpy small_square_ratio = 0.75 # [0.25:2]
76 gcptmplpy # [Feeds and Speeds] */
77 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
78 gcptmplpy # [Feeds and Speeds] */
79 gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
80 gcptmplpy # [Feeds and Speeds] */
81 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
82 gcptmplpy # [Feeds and Speeds] */
83 gcptmplpy small_V_ratio = 0.625 # [0.25:2]
84 gcptmplpy # [Feeds and Speeds] */
85 gcptmplpy DT_ratio = 0.75 # [0.25:2]
86 gcptmplpy # [Feeds and Speeds] */
87 gcptmplpy KH_ratio = 0.75 # [0.25:2]
88 gcptmplpy # [Feeds and Speeds] */
89 gcptmplpy RO_ratio = 0.5 # [0.25:2]
90 gcptmplpy # [Feeds and Speeds] */
91 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
92 gcptmplpy
93 gcptmplpy gcp = gcodepreview(generatepaths,
94 gcptmplpy                                generategcode,
95 gcptmplpy                                generatedxf,
96 gcptmplpy                                )
97 gcptmplpy
98 gcptmplpy gcp.opengcodefile(Base_filename)
99 gcptmplpy gcp.opendxffile(Base_filename)
100 gcptmplpy gcp.opendxxfiles(Base_filename,
101 gcptmplpy                                large_square_tool_num,
102 gcptmplpy                                small_square_tool_num,
103 gcptmplpy                                large_ball_tool_num,
104 gcptmplpy                                small_ball_tool_num,
105 gcptmplpy                                large_V_tool_num,
106 gcptmplpy                                small_V_tool_num,
107 gcptmplpy                                DT_tool_num,
108 gcptmplpy                                KH_tool_num,
109 gcptmplpy                                Roundover_tool_num,
110 gcptmplpy                                MISC_tool_num)
111 gcptmplpy gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
    zeroheight, stockzero, retractheight)
112 gcptmplpy
113 gcptmplpy #print(pygcpcversion())
114 gcptmplpy
115 gcptmplpy #print(gcp.myfunc(4))
116 gcptmplpy
117 gcptmplpy #print(gcp.getvv())
118 gcptmplpy
119 gcptmplpy #ts = cylinder(12.7, 1.5875, 1.5875)
120 gcptmplpy #toolpaths = gcp.cutshape(stockXwidth/2, stockYheight/2, -
    stockZthickness)
121 gcptmplpy
122 gcptmplpy gcp.movetosafeZ()
123 gcptmplpy
124 gcptmplpy gcp.toolchange(102, 10000)
125 gcptmplpy
126 gcptmplpy #gcp.rapidXY(6, 12)

```

```

127 gcptmplpy gcp.rapidZ(0)
128 gcptmplpy
129 gcptmplpy #print (gcp.xpos())
130 gcptmplpy #print (gcp.ypos())
131 gcptmplpy #psetzpos(7)
132 gcptmplpy #gcp.setzpos(-12)
133 gcptmplpy #print (gcp.zpos())
134 gcptmplpy
135 gcptmplpy #print ("X", str(gcp.xpos()))
136 gcptmplpy #print ("Y", str(gcp.ypos()))
137 gcptmplpy #print ("Z", str(gcp.zpos()))
138 gcptmplpy
139 gcptmplpy toolpaths = gcp.currenttool()
140 gcptmplpy
141 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2, stockYheight/2, -
      stockZthickness)
142 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
      stockYheight/2, -stockZthickness))
143 gcptmplpy
144 gcptmplpy gcp.rapidZ(retractheight)
145 gcptmplpy gcp.toolchange(201, 10000)
146 gcptmplpy gcp.rapidXY(0, stockYheight/16)
147 gcptmplpy gcp.rapidZ(0)
148 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
      stockYheight/2, -stockZthickness))
149 gcptmplpy
150 gcptmplpy gcp.rapidZ(retractheight)
151 gcptmplpy gcp.toolchange(202, 10000)
152 gcptmplpy gcp.rapidXY(0, stockYheight/8)
153 gcptmplpy gcp.rapidZ(0)
154 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
      stockYheight/2, -stockZthickness))
155 gcptmplpy
156 gcptmplpy gcp.rapidZ(retractheight)
157 gcptmplpy gcp.toolchange(101, 10000)
158 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
159 gcptmplpy gcp.rapidZ(0)
160 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
      stockYheight/2, -stockZthickness))
161 gcptmplpy
162 gcptmplpy gcp.setzpos(retractheight)
163 gcptmplpy gcp.toolchange(390, 10000)
164 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
165 gcptmplpy gcp.rapidZ(0)
166 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
      stockYheight/2, -stockZthickness))
167 gcptmplpy gcp.rapidZ(retractheight)
168 gcptmplpy
169 gcptmplpy gcp.toolchange(301, 10000)
170 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
171 gcptmplpy gcp.rapidZ(0)
172 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*2,
      stockYheight/2, -stockZthickness))
173 gcptmplpy
174 gcptmplpy rapids = gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight)
175 gcptmplpy gcp.toolchange(102, 10000)
176 gcptmplpy
177 gcptmplpy rapids = gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight/4,
      0)
178 gcptmplpy
179 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(0, 90, gcp.xpos()-
      stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
      /4))
180 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(90, 180, gcp.xpos(), gcp.
      ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
181 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(180, 270, gcp.xpos()+
      stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
      /4))
182 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(270, 360, gcp.xpos(), gcp.
      ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
183 gcptmplpy
184 gcptmplpy rapids = gcp.movetosafeZ()
185 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight
      /4)
186 gcptmplpy rapids = gcp.rapidZ(0)
187 gcptmplpy
188 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180, 90, gcp.xpos()+
      stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness

```

```

/4))
189 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(90, 0, gcp.xpos(), gcp.
      ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
190 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(360, 270, gcp.xpos()-
      stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
      /4))
191 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(270, 180, gcp.xpos(), gcp.
      ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
192 gcptmplpy
193 gcptmplpy rapids = gcp.movetosafeZ()
194 gcptmplpy gcp.toolchange(201, 10000)
195 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2, -stockYheight/2)
196 gcptmplpy rapids = gcp.rapidZ(0)
197 gcptmplpy
198 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
      , -stockZthickness))
199 gcptmplpy #test = gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
200 gcptmplpy
201 gcptmplpy rapids = gcp.movetosafeZ()
202 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
203 gcptmplpy rapids = gcp.rapidZ(0)
204 gcptmplpy
205 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180, 90, stockXwidth/2, -
      stockYheight/2, 6.34, -stockZthickness))
206 gcptmplpy
207 gcptmplpy rapids = gcp.movetosafeZ()
208 gcptmplpy gcp.toolchange(814, 10000)
209 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
210 gcptmplpy rapids = gcp.rapidZ(0)
211 gcptmplpy
212 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
      , -stockZthickness))
213 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -12.7, -
      stockZthickness))
214 gcptmplpy
215 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
216 gcptmplpy rapids = gcp.movetosafeZ()
217 gcptmplpy gcp.toolchange(374, 10000)
218 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight
      /4+stockYheight/16))
219 gcptmplpy rapids = gcp.rapidZ(0)
220 gcptmplpy
221 gcptmplpy gcp.rapidZ(retractheight)
222 gcptmplpy gcp.toolchange(374, 10000)
223 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
      stockYheight/16))
224 gcptmplpy gcp.rapidZ(0)
225 gcptmplpy
226 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
      stockZthickness/2))
227 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()+
      stockYheight/9, gcp.ypos(), gcp.zpos()))
228 gcptmplpy #below should probably be cutlinegc
229 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()-stockYheight/9,
      gcp.ypos(), gcp.zpos()))
230 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
231 gcptmplpy
232 gcptmplpy #key = gcp.cutkeyholegcdxf(KH_tool_num, 0, stockZthickness*0.75, "E
      ", stockYheight/9)
233 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
      stockYheight/9)
234 gcptmplpy #toolpaths = toolpaths.union(key)
235 gcptmplpy
236 gcptmplpy gcp.rapidZ(retractheight)
237 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
      stockYheight/16))
238 gcptmplpy gcp.rapidZ(0)
239 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
      stockZthickness*0.75, "N", stockYheight/9))
240 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
      stockZthickness/2))
241 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
      +stockYheight/9, gcp.zpos()))
242 gcptmplpy #below should probably be cutlinegc
243 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()-
      stockYheight/9, gcp.zpos()))
244 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
245 gcptmplpy

```

```

246 gcptmplpy gcp.rapidZ(retractheight)
247 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
    stockYheight/8))
248 gcptmplpy gcp.rapidZ(0)
249 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
    stockZthickness*0.75, "W", stockYheight/9))
250 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
    stockZthickness/2))
251 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos() -
    stockYheight/9, gcp.ypos(), gcp.zpos()))
252 gcptmplpy #below should probably be cutlinegc
253 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()+stockYheight/9,
    gcp.ypos(), gcp.zpos()))
254 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
255 gcptmplpy
256 gcptmplpy gcp.rapidZ(retractheight)
257 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
    stockYheight/8))
258 gcptmplpy gcp.rapidZ(0)
259 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
    stockZthickness*0.75, "S", stockYheight/9))
260 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
    stockZthickness/2))
261 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
    -stockYheight/9, gcp.zpos()))
262 gcptmplpy #below should probably be cutlinegc
263 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()+
    stockYheight/9, gcp.zpos()))
264 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
265 gcptmplpy
266 gcptmplpy gcp.rapidZ(retractheight)
267 gcptmplpy gcp.toolchange(56142, 10000)
268 gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
269 gcptmplpy #gcp.cutlineZgcfeed(-1.531, plunge)
270 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
    -1.531))
271 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
    -(stockYheight/2+0.508/2), -1.531))
272 gcptmplpy
273 gcptmplpy gcp.rapidZ(retractheight)
274 gcptmplpy #gcp.toolchange(56125, 10000)
275 gcptmplpy #gcp.cutlineZgcfeed(-1.531, plunge)
276 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
    -1.531))
277 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
    (stockYheight/2+0.508/2), -1.531))
278 gcptmplpy
279 gcptmplpy gcp.rapidZ(retractheight)
280 gcptmplpy gcp.toolchange(45982, 10000)
281 gcptmplpy gcp.rapidXY(stockXwidth/8, 0)
282 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -(
    stockZthickness*7/8)))
283 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -
    stockYheight/2, -(stockZthickness*7/8)))
284 gcptmplpy
285 gcptmplpy gcp.rapidZ(retractheight)
286 gcptmplpy gcp.toolchange(204, 10000)
287 gcptmplpy gcp.rapidXY(stockXwidth*0.3125, 0)
288 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -(
    stockZthickness*7/8)))
289 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -
    stockYheight/2, -(stockZthickness*7/8)))
290 gcptmplpy
291 gcptmplpy gcp.rapidZ(retractheight)
292 gcptmplpy gcp.toolchange(502, 10000)
293 gcptmplpy gcp.rapidXY(stockXwidth*0.375, 0)
294 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
    -4.24))
295 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -
    stockYheight/2, -4.24))
296 gcptmplpy
297 gcptmplpy gcp.rapidZ(retractheight)
298 gcptmplpy gcp.toolchange(13921, 10000)
299 gcptmplpy gcp.rapidXY(-stockXwidth*0.375, 0)
300 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
    stockZthickness/2))
301 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -
    stockYheight/2, -stockZthickness/2))

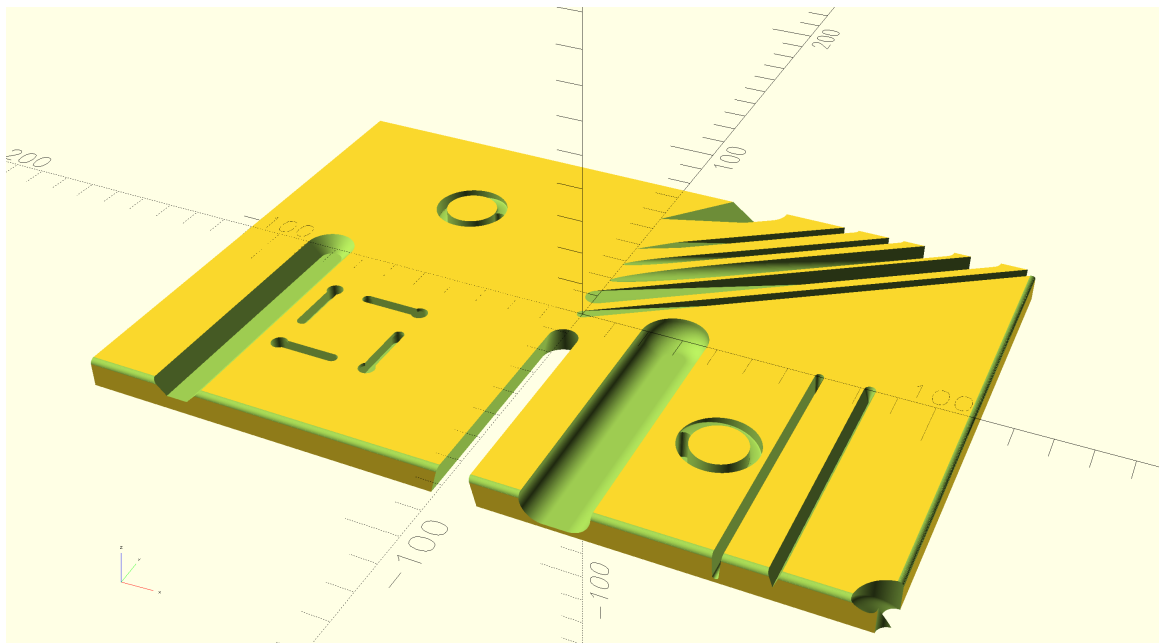
```

```

302 gcptmplpy
303 gcptmplpy gcp.rapidZ(retractheight)
304 gcptmplpy
305 gcptmplpy part = gcp.stock.difference(toolpaths)
306 gcptmplpy
307 gcptmplpy show(part)
308 gcptmplpy #show(test)
309 gcptmplpy #show(key)
310 gcptmplpy #show(dt)
311 gcptmplpy #gcp.stockandtoolpaths()
312 gcptmplpy #gcp.stockandtoolpaths("stock")
313 gcptmplpy #output (gcp.stock)
314 gcptmplpy #output (gcp.toolpaths)
315 gcptmplpy #output (toolpaths)
316 gcptmplpy
317 gcptmplpy #gcp.makecube(3, 2, 1)
318 gcptmplpy #
319 gcptmplpy #gcp.placecube()
320 gcptmplpy #
321 gcptmplpy #c = gcp.instantiatecube()
322 gcptmplpy #
323 gcptmplpy #show(c)
324 gcptmplpy
325 gcptmplpy gcp.closegcodefile()
326 gcptmplpy gcp.closedxfiles()
327 gcptmplpy gcp.closedxfile()

```

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>.

```

1 gcptmpl //! OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>
4 gcptmpl include <gcodepreview.scad>
5 gcptmpl
6 gcptmpl $fa = 2;
7 gcptmpl $fs = 0.125;
8 gcptmpl fa = 2;
9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 219;
13 gcptmpl /* [Stock] */

```

```

14 gcptmpl stockYheight = 150;
15 gcptmpl /* [Stock] */
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
    ]
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
25 gcptmpl Base_filename = "export";
26 gcptmpl /* [Export] */
27 gcptmpl generatepaths = true;
28 gcptmpl /* [Export] */
29 gcptmpl generatedxf = true;
30 gcptmpl /* [Export] */
31 gcptmpl generategcode = true;
32 gcptmpl
33 gcptmpl /* [CAM] */
34 gcptmpl toolradius = 1.5875;
35 gcptmpl /* [CAM] */
36 gcptmpl large_square_tool_num = 0; // [0:0, 112:112, 102:102, 201:201]
37 gcptmpl /* [CAM] */
38 gcptmpl small_square_tool_num = 102; // [0:0, 122:122, 112:112, 102:102]
39 gcptmpl /* [CAM] */
40 gcptmpl large_ball_tool_num = 0; // [0:0, 111:111, 101:101, 202:202]
41 gcptmpl /* [CAM] */
42 gcptmpl small_ball_tool_num = 0; // [0:0, 121:121, 111:111, 101:101]
43 gcptmpl /* [CAM] */
44 gcptmpl large_V_tool_num = 0; // [0:0, 301:301, 690:690]
45 gcptmpl /* [CAM] */
46 gcptmpl small_V_tool_num = 0; // [0:0, 390:390, 301:301]
47 gcptmpl /* [CAM] */
48 gcptmpl DT_tool_num = 0; // [0:0, 814:814, 808079:808079]
49 gcptmpl /* [CAM] */
50 gcptmpl KH_tool_num = 0; // [0:0, 374:374, 375:375, 376:376, 378:378]
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, 45982:45982]
55 gcptmpl //648 threadmill_shaft(2.4, 0.75, 18)
56 gcptmpl //45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
    Inch Shank
57 gcptmpl
58 gcptmpl /* [Feeds and Speeds] */
59 gcptmpl plunge = 100;
60 gcptmpl /* [Feeds and Speeds] */
61 gcptmpl feed = 400;
62 gcptmpl /* [Feeds and Speeds] */
63 gcptmpl speed = 16000;
64 gcptmpl /* [Feeds and Speeds] */
65 gcptmpl small_square_ratio = 0.75; // [0.25:2]
66 gcptmpl /* [Feeds and Speeds] */
67 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
68 gcptmpl /* [Feeds and Speeds] */
69 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
70 gcptmpl /* [Feeds and Speeds] */
71 gcptmpl large_V_ratio = 0.875; // [0.25:2]
72 gcptmpl /* [Feeds and Speeds] */
73 gcptmpl small_V_ratio = 0.625; // [0.25:2]
74 gcptmpl /* [Feeds and Speeds] */
75 gcptmpl DT_ratio = 0.75; // [0.25:2]
76 gcptmpl /* [Feeds and Speeds] */
77 gcptmpl KH_ratio = 0.75; // [0.25:2]
78 gcptmpl /* [Feeds and Speeds] */
79 gcptmpl R0_ratio = 0.5; // [0.25:2]
80 gcptmpl /* [Feeds and Speeds] */
81 gcptmpl MISC_ratio = 0.5; // [0.25:2]
82 gcptmpl
83 gcptmpl thegeneratepaths = generatepaths == true ? 1 : 0;
84 gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
85 gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
86 gcptmpl
87 gcptmpl gcp = gcodepreview(thegeneratepaths,
88 gcptmpl thegenerategcode,
89 gcptmpl thegeneratedxf,

```

```

90 gcptmpl );
91 gcptmpl
92 gcptmpl.opengcodefile(Base_filename);
93 gcptmpl.opendxfile(Base_filename);
94 gcptmpl.opendxfiles(Base_filename,
95 gcptmpl         large_square_tool_num,
96 gcptmpl         small_square_tool_num,
97 gcptmpl         large_ball_tool_num,
98 gcptmpl         small_ball_tool_num,
99 gcptmpl         large_V_tool_num,
100 gcptmpl        small_V_tool_num,
101 gcptmpl        DT_tool_num,
102 gcptmpl        KH_tool_num,
103 gcptmpl        Roundover_tool_num,
104 gcptmpl        MISC_tool_num);
105 gcptmpl
106 gcptmpl.setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
        stockzero);

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

```



```

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

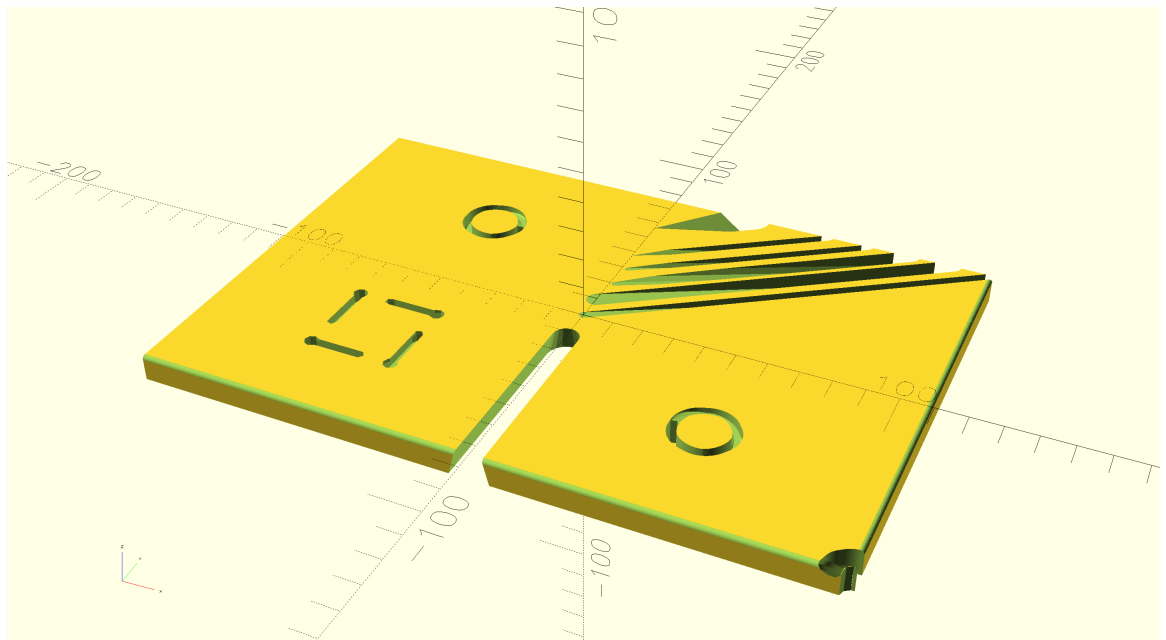
```

```

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

```

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



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 `output(<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`), one typical approach is to collect the toolpaths (and rapids) into variables and then subtract them from the stock for output

`generatepaths` This behaviour is handled by the `generatepaths` Boolean. If set to `True` then each toolpath/cut
`toolpaths` will be added to a `toolpaths` variable (identified as either `self` or `gcp` depending on the context)
`stockandtoolpaths` which then will be used in the command `stockandtoolpaths`. If this variable is set to `False`, then it will be the responsibility of the user to manage the return of the 3D model by the module/routine.

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 supporting the work-around of two-sided (flip) machining by using an imported file as the Stock seems promising) 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 may require at least 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 and that there is an option for loading directly from the Github repository implemented in PythonSCAD
- 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.

PYTHON CODING CONSIDERATIONS: Python style may be checked using a tool such as: <https://www.codewof.co.nz/style/python3/>. Not all conventions will necessarily be adhered to — limiting line length in particular conflicts with the flexibility of Literate Programming. Note that `numpydoc`-style docstrings will be added to help define the functionality of each defined module in Python. <https://numpydoc.readthedocs.io/en/latest/>.

3.1 Module Naming Convention

The original implementation required three files and used a convention for prefacing commands with `o` or `p`, but this requirement was obviated in the full Python re-write. The current implementation depends upon the class being instantiated as `gcp` as a sufficient differentiation between the Python and the OpenSCAD versions of commands which will otherwise 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 complete 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 tool movement removing volume from 3D object)
 - rapid (action — create tool movement of 3D object so as to show any collision or rubbing)
 - 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 (shapes)
 - 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, while number will be used for module/function names, making it straight-forward to ensure that functions and variables have different names for purposes of scope

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

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

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

Note that certain commands (dxflinegc, dxfarcdxfgc, linegc, arcdxfgc) 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.

A further consideration is that when processing G-code it is typical for a given command to be minimal and only include the axis of motion for the end-position, so for each of the above which is likely to appear in a .nc file, it will be necessary to have a matching command for the combinatorial possibilities, hence:

cutlineXYZ	cutlineXYZwithfeed
cutlineXY	cutlineXYwithfeed
cutlineXZ	cutlineXZwithfeed
cutlineYZ	cutlineYZwithfeed
cutlineX	cutlineXwithfeed
cutlineY	cutlineYwithfeed
cutlineZ	cutlineZwithfeed

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 v0.8)
- Programming in Python and calling OpenSCAD — https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/

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

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

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

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

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

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

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

which in turn calls the internal Python definition `\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 v0.6 (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-arguments>

In short, rather than hard-code numbers, for example in loops, they will be assigned as default 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 https://www.pythonscad.org/downloads/
   PythonSCAD-2024.12.29-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 gcpy
7 gcpy import sys
8 gcpy
9 gcpy # add math functions (using radians by default, convert to degrees
   where necessary)
10 gcpy import math
11 gcpy
12 gcpy # getting openscad functions into namespace
13 gcpy #https://github.com/gsohler/openscad/issues/39
14 gcpy try:
15 gcpy     from openscad import *
16 gcpy except ModuleNotFoundError as e:
17 gcpy     print("OpenSCAD module not loaded.")
18 gcpy
19 gcpy def pygcpversion():
20 gcpy     thegcpversion = 0.8
21 gcpy     return thegcpversion

```

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>
9 gcpscad
10 gcpscad module gcpversion(){
11 gcpscad echo(pygcpversion());
12 gcpscad }
13 gcpscad
14 gcpscad //function myfunc(var) = gcp.myfunc(var);
15 gcpscad //
16 gcpscad //function getvv() = gcp.getvv();
17 gcpscad //
18 gcpscad //module makecube(xdim, ydim, zdim){
19 gcpscad //gcp.makecube(xdim, ydim, zdim);
20 gcpscad //}
21 gcpscad //
22 gcpscad //module placecube(){
23 gcpscad //gcp.placecube();
24 gcpscad //}
25 gcpscad //
26 gcpscad //module instantiatecube(){
27 gcpscad //gcp.instantiatecube();
28 gcpscad //}
29 gcpscad //

```

If all functions are to be handled within Python, then they will need to be gathered into a class which contains them and which is initialized so as to define shared variables and initial program state, 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
 - mistool:
 - * 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 gcpy
25 gcpy     def __init__(self, #basefilename = "export",
26 gcpy                     generatepaths = False,
27 gcpy                     generategcode = False,
28 gcpy                     generatedxf = False,
29 gcpy #                     stockXwidth = 25,
30 gcpy #                     stockYheight = 25,
31 gcpy #                     stockZthickness = 1,
32 gcpy #                     zeroheight = "Top",
33 gcpy #                     stockzero = "Lower-left",
34 gcpy #                     retractheight = 6,
35 gcpy #                     currenttoolnum = 102,
36 gcpy #                     toolradius = 3.175,
37 gcpy #                     plunge = 100,
38 gcpy #                     feed = 400,
39 gcpy #                     speed = 10000
40 gcpy                     ):
41 gcpy         """
42 gcpy         Initialize gcodepreview object.
43 gcpy
44 gcpy         Parameters
45 gcpy         -----
46 gcpy         generatepaths : boolean
47 gcpy                         Determines if toolpaths will be stored
48 gcpy                         internally or returned directly
49 gcpy         generategcode : boolean
50 gcpy                         Enables writing out G-code.
51 gcpy         generatedxf   : boolean
52 gcpy                         Enables writing out DXF file(s).
53 gcpy
54 gcpy         Returns
55 gcpy         -----
56 gcpy         object
57 gcpy         The initialized gcodepreview object.
58 gcpy         """
59 gcpy #         self.basefilename = basefilename
60 gcpy if (generatepaths == 1):
61 gcpy     self.generatepaths = True
62 gcpy if (generatepaths == 0):
63 gcpy     self.generatepaths = False
64 gcpy else:
65 gcpy     self.generatepaths = generatepaths
66 gcpy if (generategcode == 1):

```

```
66 gcpy          self.generategcode = True
67 gcpy          if (generategcode == 0):
68 gcpy              self.generategcode = False
69 gcpy          else:
70 gcpy              self.generategcode = generategcode
71 gcpy          if (generatedxf == 1):
72 gcpy              self.generatedxf = True
73 gcpy          if (generatedxf == 0):
74 gcpy              self.generatedxf = False
75 gcpy          else:
76 gcpy              self.generatedxf = generatedxf
77 gcpy #          self.stockXwidth = stockXwidth
78 gcpy #          self.stockYheight = stockYheight
79 gcpy #          self.stockZthickness = stockZthickness
80 gcpy #          self.zeroheight = zeroheight
81 gcpy #          self.stockzero = stockzero
82 gcpy #          self.retractheight = retractheight
83 gcpy #          self.currenttoolnum = currenttoolnum
84 gcpy #          self.toolradius = toolradius
85 gcpy #          self.plunge = plunge
86 gcpy #          self.feed = feed
87 gcpy #          self.speed = speed
88 gcpy #          global toolpaths
89 gcpy #          if (openscadloaded == True):
90 gcpy #              self.toolpaths = cylinder(0.1, 0.1)
91 gcpy          self.generatedxfs = False
92 gcpy
93 gcpy          def checkgeneratepaths():
94 gcpy              return self.generatepaths
95 gcpy
96 gcpy #          def myfunc(self, var):
97 gcpy #              self.vv = var * var
98 gcpy #              return self.vv
99 gcpy #
100 gcpy #          def getvv(self):
101 gcpy #              return self.vv
102 gcpy #
103 gcpy #          def checkint(self):
104 gcpy #              return self.mc
105 gcpy #
106 gcpy #          def makecube(self, xdim, ydim, zdim):
107 gcpy #              self.c=cube([xdim, ydim, zdim])
108 gcpy #
109 gcpy #          def placecube(self):
110 gcpy #              show(self.c)
111 gcpy #
112 gcpy #          def instantiatecube(self):
113 gcpy #              return self.c
```

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:

- mpx
- mpx
- mpy
- mpy
- mpz
- mpz

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


```
115 gcpy      def xpos(self):
116 gcpy #          global mpx
117 gcpy      return self.mpx
118 gcpy
119 gcpy      def ypos(self):
120 gcpy #          global mpy
121 gcpy      return self.mpy
122 gcpy
123 gcpy      def zpos(self):
124 gcpy #          global mpz
125 gcpy      return self.mpz
126 gcpy
127 gcpy #      def tpzinc(self):
128 gcpy #          global tpzinc
129 gcpy #      return self.tpzinc
```

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

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

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

setypos

setzpos

```
131 gcpy      def setxpos(self, newxpos):
132 gcpy #          global mpx
133 gcpy      self.mpx = newxpos
134 gcpy
135 gcpy      def setypos(self, newypos):
136 gcpy #          global mpy
137 gcpy      self.mpy = newypos
138 gcpy
139 gcpy      def setzpos(self, newzpos):
140 gcpy #          global mpz
141 gcpy      self.mpz = newzpos
142 gcpy
143 gcpy #      def settpzinc(self, newtpzinc):
144 gcpy #          global tpzinc
145 gcpy #      self.tpzinc = newtpzinc
```

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

3.2.2 Initial Modules

initializemachinestate()

The first routine, actually a subroutine, is initializemachinestate() which is necessary because there are multiple routines for setting up the cut, depending on the context (processing G-code or no) and the type of project (3-axis mill (or possibly in the future, lathe)).

```
147 gcpy      def initializemachinestate(self):
148 gcpy #          global mpx
149 gcpy      self.mpx = float(0)
150 gcpy #          global mpy
151 gcpy      self.mpy = float(0)
152 gcpy #          global mpz
153 gcpy      self.mpz = float(0)
154 gcpy #          global tpz
155 gcpy #          self.tpzinc = float(0)
156 gcpy #          global currenttoolnum
157 gcpy      self.currenttoolnum = 102
158 gcpy #          global currenttoolshape
159 gcpy      self.currenttoolshape = cylinder(12.7, 1.5875)
160 gcpy      self.rapids = self.currenttoolshape
161 gcpy      self.retractheight = 53.0
```

gcodepreview The first such setup subroutine is gcodepreview setupstock which is appropriately enough, setupstock 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 (see initializemachinestate() above), but the rewritten standalone version handles all necessary actions.

gcp.setupstock Since part of a class, it will be called as gcp.setupstock. It requires that the user set parameters for stock dimensions and so forth, and will create comments in the G-code (if generating that file

is enabled) which incorporate the stock dimensions and its position relative to the zero as set relative to the stock.

```
163 gcpy      def setupstock(self, stockXwidth,
164 gcpy                      stockYheight,
165 gcpy                      stockZthickness,
166 gcpy                      zeroheight,
167 gcpy                      stockzero,
168 gcpy                      retractheight):
169 gcpy      """
170 gcpy      Set up blank/stock for material and position/zero.
171 gcpy
172 gcpy      Parameters
173 gcpy      -----
174 gcpy      stockXwidth : float
175 gcpy                      X extent/dimension
176 gcpy      stockYheight : float
177 gcpy                      Y extent/dimension
178 gcpy      stockZthickness : boolean
179 gcpy                      Z extent/dimension
180 gcpy      zeroheight : string
181 gcpy                      Top or Bottom, determines if Z extent will
                        be positive or negative
182 gcpy      stockzero : string
183 gcpy                      Lower-Left, Center-Left, Top-Left, Center,
                        determines XY position of stock
184 gcpy      retractheight : float
185 gcpy                      Distance which tool retracts above surface
                        of stock.
186 gcpy
187 gcpy      Returns
188 gcpy      -----
189 gcpy      none
190 gcpy      """
191 gcpy      self.initializemachinestate()
192 gcpy      self.stockXwidth = stockXwidth
193 gcpy      self.stockYheight = stockYheight
194 gcpy      self.stockZthickness = stockZthickness
195 gcpy      self.zeroheight = zeroheight
196 gcpy      self.stockzero = stockzero
197 gcpy      self.retractheight = retractheight
198 gcpy      # global stock
199 gcpy      self.stock = cube([stockXwidth, stockYheight,
                        stockZthickness])
200 gcpy      %%WRITEGC      if self.generategcode == True:
201 gcpy      %%WRITEGC      self.writegc("(Design File: " + self.
                        basefilename + ")")
202 gcpy      self.toolpaths = cylinder(0.1, 0.1)
```

The **setupstock** command is required if working with a 3D project, creating the block of stock which the following toolpath commands will cut away. Note that since Python in PythonSCAD defers output of the 3D model, it is possible to define it once, then set up all the specifics for each possible positioning of the stock in terms of origin. The internal variable stockzero is used in an <if then else> structure to position the 3D model of the stock and write out the G-code comment which describes it in using the terms described for CutViewer.

```
203 gcpy      if self.zeroheight == "Top":
204 gcpy          if self.stockzero == "Lower-Left":
205 gcpy              self.stock = self.stock.translate([0, 0, -self.
                        stockZthickness])
206 gcpy          if self.generategcode == True:
207 gcpy              self.writegc("(stockMin:0.00mm,␣0.00mm,␣-", str
                        (self.stockZthickness), "mm)")
208 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
                        ), "mm,␣", str(stockYheight), "mm,␣0.00mm)")
209 gcpy              self.writegc("(STOCK/BLOCK,␣", str(self.
                        stockXwidth), ",␣", str(self.stockYheight),
                        ",␣", str(self.stockZthickness), ",␣0.00,␣
                        0.00,␣", str(self.stockZthickness), ")")
210 gcpy          if self.stockzero == "Center-Left":
211 gcpy              self.stock = self.stock.translate([0, -stockYheight
                        / 2, -stockZthickness])
212 gcpy          if self.generategcode == True:
213 gcpy              self.writegc("(stockMin:0.00mm,␣-", str(self.
                        stockYheight/2), "mm,␣-", str(self.
                        stockZthickness), "mm)")
214 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
```

```

        ), "mm,␣", str(self.stockYheight/2), "mm,␣
        0.00mm)")
215 gcpy        self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣0.00,␣",
                str(self.stockYheight/2), ",␣", str(self.
                stockZthickness), ")");
216 gcpy    if self.stockzero == "Top-Left":
217 gcpy        self.stock = self.stock.translate([0, -self.
                stockYheight, -self.stockZthickness])
218 gcpy        if self.generategcode == True:
219 gcpy            self.writegc("(stockMin:0.00mm,␣-", str(self.
                stockYheight), "mm,␣-", str(self.
                stockZthickness), "mm)")
220 gcpy            self.writegc("(stockMax:", str(self.stockXwidth
                ), "mm,␣0.00mm,␣0.00mm)")
221 gcpy            self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣0.00,␣",
                str(self.stockYheight), ",␣", str(self.
                stockZthickness), ")");
222 gcpy    if self.stockzero == "Center":
223 gcpy        self.stock = self.stock.translate([-self.
                stockXwidth / 2, -self.stockYheight / 2, -self.
                stockZthickness])
224 gcpy        if self.generategcode == True:
225 gcpy            self.writegc("(stockMin:␣-", str(self.
                stockXwidth/2), ",␣-", str(self.stockYheight
                /2), "mm,␣-", str(self.stockZthickness), "mm
                )");
226 gcpy            self.writegc("(stockMax:", str(self.stockXwidth
                /2), "mm,␣", str(self.stockYheight/2), "mm,␣
                0.00mm)")
227 gcpy            self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣", str(
                self.stockXwidth/2), ",␣", str(self.
                stockYheight/2), ",␣", str(self.
                stockZthickness), ")");
228 gcpy    if self.zeroheight == "Bottom":
229 gcpy        if self.stockzero == "Lower-Left":
230 gcpy            self.stock = self.stock.translate([0, 0, 0])
231 gcpy            if self.generategcode == True:
232 gcpy                self.writegc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
                )");
233 gcpy                self.writegc("(stockMax:", str(self.
                stockXwidth), "mm,␣", str(self.stockYheight
                ), "mm,␣", str(self.stockZthickness), "mm)"
                );
234 gcpy                self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣0.00,␣
                0.00,␣0.00)")
235 gcpy    if self.stockzero == "Center-Left":
236 gcpy        self.stock = self.stock.translate([0, -self.
                stockYheight / 2, 0])
237 gcpy        if self.generategcode == True:
238 gcpy            self.writegc("(stockMin:0.00mm,␣-", str(self.
                stockYheight/2), "mm,␣0.00mm)")
239 gcpy            self.writegc("(stockMax:", str(self.stockXwidth
                ), "mm,␣", str(self.stockYheight/2), "mm,␣-",
                str(self.stockZthickness), "mm)")
240 gcpy            self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣0.00,␣",
                str(self.stockYheight/2), ",␣0.00mm");
241 gcpy    if self.stockzero == "Top-Left":
242 gcpy        self.stock = self.stock.translate([0, -self.
                stockYheight, 0])
243 gcpy        if self.generategcode == True:
244 gcpy            self.writegc("(stockMin:0.00mm,␣-", str(self.
                stockYheight), "mm,␣0.00mm)")
245 gcpy            self.writegc("(stockMax:", str(self.stockXwidth
                ), "mm,␣0.00mm,␣", str(self.stockZthickness)
                , "mm)")
246 gcpy            self.writegc("(STOCK/BLOCK,␣", str(self.
                stockXwidth), ",␣", str(self.stockYheight),
                ",␣", str(self.stockZthickness), ",␣0.00,␣",

```

```

                str(self.stockYheight), ",_0.00)")
247 gcpy          if self.stockzero == "Center":
248 gcpy              self.stock = self.stock.translate([-self.
                    stockXwidth / 2, -self.stockYheight / 2, 0])
249 gcpy          if self.generategcode == True:
250 gcpy              self.writegc("(stockMin:_", str(self.
                    stockXwidth/2), ",_-", str(self.stockYheight
                    /2), "mm,_0.00mm)")
251 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
                    /2), "mm,_", str(self.stockYheight/2), "mm,_
                    ", str(self.stockZthickness), "mm)")
252 gcpy              self.writegc("(STOCK/BLOCK,_", str(self.
                    stockXwidth), ",_", str(self.stockYheight),
                    ",_", str(self.stockZthickness), ",_", str(
                    self.stockXwidth/2), ",_", str(self.
                    stockYheight/2), ",_0.00)")
253 gcpy          if self.generategcode == True:
254 gcpy              self.writegc("G90");
255 gcpy              self.writegc("G21");
```

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

```

37 gcpscad module setupstock(stockXwidth, stockYheight, stockZthickness,
    zeroheight, stockzero, retractheight) {
38 gcpscad     gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
    zeroheight, stockzero, retractheight);
39 gcpscad }
```

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

```

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

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

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

If processing G-code, the parameters passed in are necessarily different, and there is of course, no need to write out G-code.

```

257 gcpy      def setupcuttingarea(self, sizeX, sizeY, sizeZ, extentleft,
    extentfb, extentd):
258 gcpy          self.initializemachinestate()
259 gcpy          c=cube([sizeX,sizeY,sizeZ])
260 gcpy          c = c.translate([extentleft,extentfb,extentd])
261 gcpy          self.stock = c
262 gcpy          self.toolpaths = cylinder(0.1, 0.1)
263 gcpy          return c
```

Adjustments and Additions

For certain projects and toolpaths it will be helpful to shift the stock, and to add additional pieces to the project. Shifting the stock is simple:

```

265 gcpy      def shiftstock(self, shiftX, shiftY, shiftZ):
266 gcpy          self.stock = self.stock.translate([shiftX, shiftY, shiftZ
    ])

41 gcpscad module shiftstock(shiftX, shiftY, shiftZ) {
42 gcpscad     gcp.shiftstock(shiftX, shiftY, shiftZ);
43 gcpscad }
```

adding stock is similar, but adds the requirement that it include options for shifting the stock:

```

268 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
    ,
269 gcpy          shiftX = 0,
270 gcpy          shiftY = 0,
```

```
271 gcpy                                shiftZ = 0):
272 gcpy                                addedpart = cube([stockXwidth, stockYheight,
                                                stockZthickness])
273 gcpy                                addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
274 gcpy                                self.stock = self.stock.union(addedpart)

45 gcpscad module addtostock(stockXwidth, stockYheight, stockZthickness,
                                shiftX, shiftY, shiftZ) {
46 gcpscad     gcp.addtostock(stockXwidth, stockYheight, stockZthickness,
                                shiftX, shiftY, shiftZ);
47 gcpscad }
```

3.3 Tools and Changes

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

```
276 gcpy     def settool(self, tn):
277 gcpy #         global currenttoolnum
278 gcpy         self.currenttoolnum = tn
279 gcpy
280 gcpy     def currenttoolnumber(self):
281 gcpy #         global currenttoolnum
282 gcpy         return self.currenttoolnum
283 gcpy
284 gcpy #     def currentroundovertoolnumber(self):
285 gcpy #         global Roundover_tool_num
286 gcpy #         return self.Roundover_tool_num
```

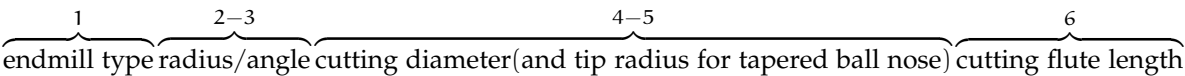
The **settool** command will normally be set using one of the variables as defined in the template, and the gcodepreview object is currently hard-coded to use the tool numbers which Carbide 3D uses for their tooling.

3.3.1 Numbering for Tools

Originally, the numbering scheme used was that of the various manufacturers of the tools used (with a disclosure that the author is a Carbide 3D employee).

Creating any numbering scheme is like most things in life, a tradeoff, balancing length and expressiveness/compleatness against simplicity and usability. The software application Carbide Create (as released by an employer of the main author) has a limit of six digits, which seems a reasonable length from a complexity/simplicity standpoint, but also potentially reasonably ex-pressible.

It will be desirable to track the following characteristics and measurements, apportioned over the digits as follows:



- 1st digit: endmill type:
 - 0 - “O”-flute
 - 1 - square
 - 2 - ball
 - 3 - V
 - 4 - bowl
 - 5 - tapered ball
 - 6 - roundover
 - 7 - thread-cutting
 - 8 - dovetail
 - 9 - other (e.g., lollipop, or manufacturer number — if manufacturer number is used, then the 9 and any padding zeroes will be removed from the G-code or DXF when writing out file(s))
- 2nd and 3rd digits shape radius (ball/roundover) or angle (V), 2nd and 3rd digit together 10-99 indicate measurement in tenth of a millimeter. 2nd digit:
 - 0 - Imperial (oo indicates n/a or square)

- any other value for both the 2nd and 3rd digits together indicate a metric measurement or an angle in degrees
- 3rd digit (if 2nd is 0 indicating Imperial)
 - 1 - $1/32^{\text{nd}}$
 - 2 - $1/16$
 - 3 - $1/8$
 - 4 - $1/4$
 - 5 - $5/16$
 - 6 - $3/8$
 - 7 - $1/2$
 - 8 - $3/4$
 - 9 - $>1''$ or other
- 4th and 5th digits cutting diameter as 2nd and 3rd above except 4th digit indicates tip radius for tapered ball nose and such tooling is only represented in Imperial measure:
- 4th digit (tapered ball nose)
 - 1 - 0.0025 in
 - 2 - 0.015625 in ($1/64^{\text{th}}$)
 - 3 - 0.0295
 - 4 - 0.03125 in ($1/32^{\text{nd}}$)
 - 5 - 0.0335
 - 6 - 0.0354
 - 7 - 0.0625 in ($1/16^{\text{th}}$)
 - 8 - 0.125 in ($1/8^{\text{th}}$)
 - 9 - 0.25 in ($1/4$)
- 6th digit cutting flute length:
 - 0 - other
 - 1 - calculate based on V angle
 - 2 - $1/16$
 - 3 - $1/8$
 - 4 - $1/4$
 - 5 - $5/16$
 - 6 - $1/2$
 - 7 - $3/4$
 - 8 - "long reach" $1''$ or greater
 - 9 - calculate based on radius

Using this technique to create tool numbers for Carbide 3D tooling we arrive at:

- Square
 - #122 == 100012
 - #112 == 100024
 - #102 == 100036
 - #201 == 100047
- Ball
 - #121 == 201012
 - #111 == 202024
 - #101 == 203036
 - #202 == 204047
- V
 - #301 == 390074
 - #302 == 360071
- Single (O) flute
 - #282 == 000204
 - #274 == 000036
 - #278 == 000047

- Tapered Ball Nose
#501 == 530131
#502 == 540131

(note that some dimensions were rounded off/approximated)
Extending that to the non-Carbide 3D tooling thus implemented:

- Dovetail
814 == 814###
45828 == 808079
- Keyhole Tool
374
375
376
378
- Roundover Tool
56142 == 6
56125 == 6
1570 == 6
- Tapered Ball Nose
204 == 2
304 == 2
- Threadmill
648 == 7
- Bowl bit
45982 == 4

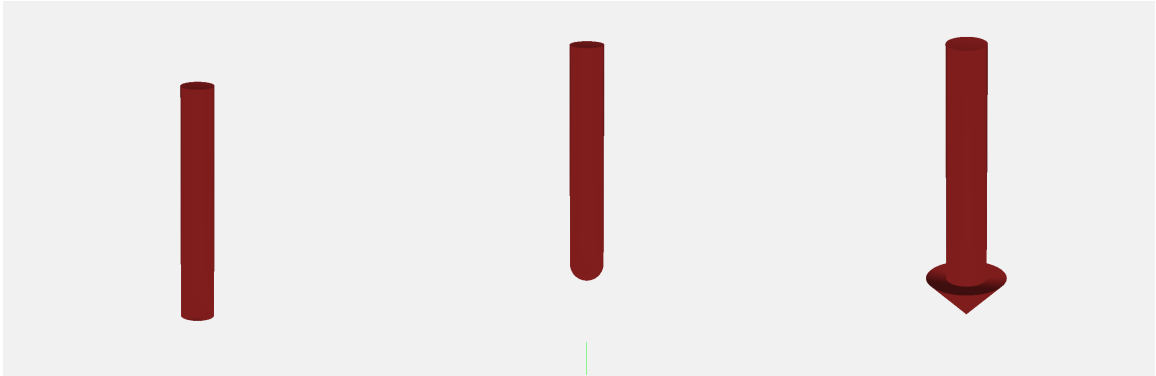
All of which reveals some notable limitations:

- No way to indicate flute geometry beyond O-flute
- Lack of precision for metric tooling/limited support for Imperial sizes
- No way to indicate flat-bottomed V/chamfer tools

3.3.2 3D Shapes for Tools

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

3.3.2.1 Normal Tooling/toolshapes Most tooling has quite standard shapes and are defined by their profile as defined in a module which simply defines/declares their shape:



- 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
- 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

Most tools are easily implemented with concise 3D descriptions which may be connected with a simple hull operation. Note that extending the normal case to a pair of such operations, one for the shaft, the other for the cutting shape will markedly simplify the code, and will make it possible to colour-code the shaft which may afford indication of instances of it rubbing against the stock.

endmill square The endmill square is a simple cylinder:

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

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

```
291 gcpy      def ballnose(self, es_diameter, es_flute_length):
292 gcpy          b = sphere(r=(es_diameter / 2))
293 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                        es_flute_length, center=False)
294 gcpy          p = union(b, s)
295 gcpy          return p.translate([0, 0, (es_diameter / 2)])
```

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):

```
297 gcpy      def endmill_v(self, es_v_angle, es_diameter):
298 gcpy          es_v_angle = math.radians(es_v_angle)
299 gcpy          v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                        2) / math.tan((es_v_angle / 2))), center=False)
300 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                        ((es_diameter * 8) ), center=False)
301 gcpy          sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                        es_v_angle / 2)))]])
302 gcpy          return union(v, sh)
```

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

```
304 gcpy      def bowl_tool(self, radius, diameter, height):
305 gcpy          bts = cylinder(height - radius, diameter / 2, diameter / 2,
                                center=False)
306 gcpy          bts = bts.translate([0, 0, radius])
307 gcpy          bts = bts.union(cylinder(height, diameter / 2 - radius,
                                diameter / 2 - radius, center=False))
308 gcpy          for i in range(90):
309 gcpy              # print(math.radians(i))
310 gcpy              slice = cylinder((radius / 90), ((diameter / 2 - radius
                        ) + radius * math.sin(math.radians(i))), ((diameter
                        / 2 - radius) + radius * math.sin(math.radians(i +
                        1))), center=False)
311 gcpy          bts = hull(bts, slice.translate([0, 0, (radius - radius
                        * math.cos(math.radians(i)))]])
312 gcpy          return bts
```

tapered ball The tapered ball nose tool is modeled as a sphere at the tip and a pair of cylinders, where one (a cone) describes the taper, while the other represents the shaft.
One vendor which provides such tooling is Precise Bits: <https://www.precisebits.com/products/carbidebits/taperedcarve250b2f.asp&filter=7>, but unfortunately, their tool numbering is ambiguous, the version of each major number (204 and 304) for their 1/4" shank tooling which is sufficiently popular to also be offered in a ZRN coating will be used. Similarly, the #501 and #502 PCB engravers from Carbide 3D will also be supported.

```
314 gcpy      def tapered_ball(self, es_tip, es_diameter, es_flute_length,
                        es_angle):
315 gcpy          b = sphere(r=(es_tip / 2))
316 gcpy          s = cylinder(r1=(es_tip / 2), r2=(es_diameter / 2), h=
                        es_flute_length, center=False)
317 gcpy          p = union(b, s)
318 gcpy          return p.translate([0, 0, (es_tip / 2)])
```

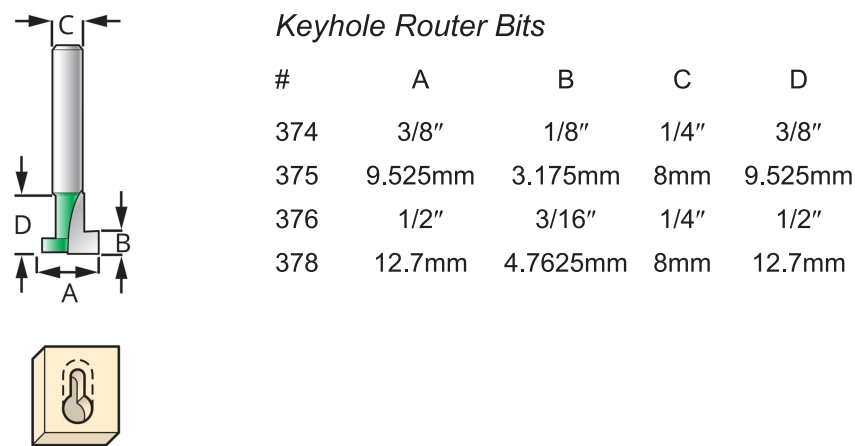
flat V The flat V tool is modeled as a cylinder with two different diameters, forming a truncated cone.

```
320 gcpy      def flat_V(self, es_tip, es_diameter, es_flute_length, es_angle
):
321 gcpy          c = cylinder(r1=(es_tip / 2), r2=(es_diameter / 2), h=
                        es_flute_length, center=False)
322 gcpy          return c
```

3.3.2.2 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 thrice, once for the actual keyhole cutting, second for the fluted portion of the shaft, and then the shaft should be modeled for collision <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 completeness’ sake and are not (at this time) implemented
- Threadmill — used for cutting threads, normally a single form geometry is used on a CNC.

3.3.2.2.1 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 The keyhole is modeled in two parts, first the cutting base:

```
324 gcpy      def keyhole(self, es_diameter, es_flute_length):
325 gcpy      return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                             h=es_flute_length, center=False)
```

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

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

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

```
330 gcpy      def threadmill(self, minor_diameter, major_diameter, cut_height):
331 gcpy      ):
332 gcpy      btm = cylinder(r1=(minor_diameter / 2), r2=(major_diameter / 2), h=cut_height, center = False)
333 gcpy      top = cylinder(r1=(major_diameter / 2), r2=(minor_diameter / 2), h=cut_height, center = False)
334 gcpy      top = top.translate([0, 0, cut_height/2])
335 gcpy      tm = btm.union(top)
336 gcpy      return tm
337 gcpy      def threadmill_shaft(self, diameter, cut_height, height):
338 gcpy      shaft = cylinder(r1=(diameter / 2), r2=(diameter / 2), h=height, center = False)
339 gcpy      shaft = shaft.translate([0, 0, cut_height/2])
340 gcpy      return shaft
```

dovetail **3.3.2.2.3 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)

```
342 gcpy      def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height
              , dt_angle):
343 gcpy      return cylinder(r1=(dt_bottomdiameter / 2), r2=(
                  dt_topdiameter / 2), h= dt_height, center=False)
```

3.3.2.3 Concave toolshapes While normal tooling may be represented with a one (or more) hull operation(s) betwixt two 3D toolshapes (or six 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, or integrated at the lowest level.

3.3.2.4 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 3.3.2.3.

```
49 gpcscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
50 gpcscad     if (radiustn == 56125) {
51 gpcscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
52 gpcscad     } else if (radiustn == 56142) {
53 gpcscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
54 gpcscad //     } else if (radiustn == 312) {
55 gpcscad //         cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
56 gpcscad     } else if (radiustn == 1570) {
57 gpcscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
58 gpcscad     }
59 gpcscad }
```

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.

3.3.3 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). Similarly, the G-code previewing functionality in this library expects that such comments will be in place so as to model the stock.

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 directly supported, and that at need, additional tool shape support may be added under misc.

3.3.3.1 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 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.

currenttoolshape

```
345 gcpy      def currenttool(self):
346 gcpy #      global currenttoolshape
347 gcpy      return self.currenttoolshape
```

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

3.3.3.2 Square and ball nose (including tapered ball nose)

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

3.3.3.3 Roundover (corner rounding)

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

3.3.3.4 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).

3.3.3.5 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.

```

349 gcpy      def toolchange(self, tool_number, speed = 10000):
350 gcpy      #          global currenttoolshape
351 gcpy          self.currenttoolshape = self.endmill_square(0.001, 0.001)
352 gcpy
353 gcpy          self.settool(tool_number)
354 gcpy          if (self.generategcode == True):
355 gcpy              self.writegc("(Toolpath)")
356 gcpy              self.writegc("M05")
357 gcpy          if (tool_number == 201):
358 gcpy              self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
359 gcpy              self.currenttoolshape = self.endmill_square(6.35,
360 gcpy                  19.05)
361 gcpy          elif (tool_number == 102):
362 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
363 gcpy              self.currenttoolshape = self.endmill_square(3.175,
364 gcpy                  12.7)
365 gcpy          elif (tool_number == 112):
366 gcpy              self.writegc("(TOOL/MILL,␣1.5875,␣0.00,␣0.00,␣0.00)")
367 gcpy              self.currenttoolshape = self.endmill_square(1.5875,
368 gcpy                  6.35)
369 gcpy          elif (tool_number == 122):
370 gcpy              self.writegc("(TOOL/MILL,␣0.79375,␣0.00,␣0.00,␣0.00)")
371 gcpy              self.currenttoolshape = self.endmill_square(0.79375,
372 gcpy                  1.5875)
373 gcpy          elif (tool_number == 202):
374 gcpy              self.writegc("(TOOL/MILL,␣6.35,␣3.175,␣0.00,␣0.00)")
375 gcpy              self.currenttoolshape = self.ballnose(6.35, 19.05)
376 gcpy          elif (tool_number == 101):
377 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣1.5875,␣0.00,␣0.00)")
378 gcpy              self.currenttoolshape = self.ballnose(3.175, 12.7)
379 gcpy          elif (tool_number == 111):
380 gcpy              self.writegc("(TOOL/MILL,␣1.5875,␣0.79375,␣0.00,␣0.00)"
381 gcpy                  )
382 gcpy              self.currenttoolshape = self.ballnose(1.5875, 6.35)
383 gcpy          elif (tool_number == 121):
384 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣0.79375,␣0.00,␣0.00)")
385 gcpy              self.currenttoolshape = self.ballnose(0.79375, 1.5875)
386 gcpy          elif (tool_number == 327):
387 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
388 gcpy              self.currenttoolshape = self.endmill_v(60, 26.9748)
389 gcpy          elif (tool_number == 301):
390 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣6.35,␣45.00)")
391 gcpy              self.currenttoolshape = self.endmill_v(90, 12.7)
392 gcpy          elif (tool_number == 302):
393 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣10.998,␣30.00)")
394 gcpy              self.currenttoolshape = self.endmill_v(60, 12.7)
395 gcpy          elif (tool_number == 390):
396 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
397 gcpy              self.currenttoolshape = self.endmill_v(90, 3.175)
398 gcpy          elif (tool_number == 374):
399 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
400 gcpy          elif (tool_number == 375):
401 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
402 gcpy          elif (tool_number == 376):
403 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
404 gcpy          elif (tool_number == 378):
405 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
406 gcpy          elif (tool_number == 814):
407 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣6.367,␣12.7,␣0.00)")
408 gcpy          #          dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
409 gcpy          )

```

```
404 gcpy          #      https://www.leevalley.com/en-us/shop/tools/power-tool-
                        accessories/router-bits/30172-dovetail-bits?item=18J1607
405 gcpy          self.currenttoolshape = self.dovetail(12.7, 6.367,
                        12.7, 14)
406 gcpy          #      45828
407 gcpy          elif (tool_number == 808079):
408 gcpy          self.writegc("(TOOL/MILL,␣12.7,␣6.816,␣20.95,␣0.00)")
409 gcpy          #      http://www.amanatool.com/45828-carbide-tipped-dovetail
                        -8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
410 gcpy          self.currenttoolshape = self.dovetail(12.7, 6.816,
                        20.95, 8)
411 gcpy          elif (tool_number == 56125):#0.508/2, 1.531
412 gcpy          self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
                        ␣3.175)")
413 gcpy          elif (tool_number == 56142):#0.508/2, 2.921
414 gcpy          self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
                        5.55625,␣1.5875)")
415 gcpy          #      elif (tool_number == 312):#1.524/2, 3.175
416 gcpy          #      self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
                        Radius, Height, Length)")
417 gcpy          elif (tool_number == 1570):#0.507/2, 4.509
418 gcpy          self.writegc("(TOOL/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
                        12.7,␣4.7625)")
419 gcpy          #https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
                        radius-x-3-4-dia-x-5-8-x-1-4-inch-shank.html
420 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
421 gcpy          self.writegc("(TOOL/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
422 gcpy          self.currenttoolshape = self.bowl_tool(6.35, 19.05,
                        15.875)
423 gcpy          elif (tool_number == 204):#
424 gcpy          self.writegc("()")
425 gcpy          self.currenttoolshape = self.tapered_ball(1.5875, 6.35,
                        38.1, 3.6)
426 gcpy          elif (tool_number == 304):#
427 gcpy          self.writegc("()")
428 gcpy          self.currenttoolshape = self.tapered_ball(3.175, 6.35,
                        38.1, 2.4)
429 gcpy          elif (tool_number == 501):#
430 gcpy          self.writegc("()")
431 gcpy          self.currenttoolshape = self.tapered_ball(0.127, 3.175,
                        2.688, 60)
432 gcpy          elif (tool_number == 502):#
433 gcpy          self.writegc("()")
434 gcpy          self.currenttoolshape = self.tapered_ball(0.127, 3.175,
                        4.25, 40)
435 gcpy          elif (tool_number == 13921):#
436 gcpy          self.writegc("()")
437 gcpy          self.currenttoolshape = self.flat_V(6.35, 31.75, 12.7,
                        45)
```

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.

```
438 gcpy          self.writegc("M6T", str(tool_number))
439 gcpy          self.writegc("M03S", str(speed))
```

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

```
61 gcpscad module toolchange(tool_number, speed){
62 gcpscad     gcp.toolchange(tool_number, speed);
63 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.)

3.3.4 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.

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

```
65 gpcscad 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:

```
441 gcpy      def tool_diameter(self, ptd_tool, ptd_depth):
442 gcpy # Square 122, 112, 102, 201
443 gcpy      if ptd_tool == 122:
444 gcpy          return 0.79375
445 gcpy      if ptd_tool == 112:
446 gcpy          return 1.5875
447 gcpy      if ptd_tool == 102:
448 gcpy          return 3.175
449 gcpy      if ptd_tool == 201:
450 gcpy          return 6.35
451 gcpy # Ball 121, 111, 101, 202
452 gcpy      if ptd_tool == 122:
453 gcpy          if ptd_depth > 0.396875:
454 gcpy              return 0.79375
455 gcpy          else:
456 gcpy              return ptd_tool
457 gcpy      if ptd_tool == 112:
458 gcpy          if ptd_depth > 0.79375:
459 gcpy              return 1.5875
460 gcpy          else:
461 gcpy              return ptd_tool
462 gcpy      if ptd_tool == 101:
463 gcpy          if ptd_depth > 1.5875:
464 gcpy              return 3.175
465 gcpy          else:
466 gcpy              return ptd_tool
467 gcpy      if ptd_tool == 202:
468 gcpy          if ptd_depth > 3.175:
469 gcpy              return 6.35
470 gcpy          else:
471 gcpy              return ptd_tool
472 gcpy # V 301, 302, 390
473 gcpy      if ptd_tool == 301:
474 gcpy          return ptd_tool
475 gcpy      if ptd_tool == 302:
476 gcpy          return ptd_tool
477 gcpy      if ptd_tool == 390:
478 gcpy          return ptd_tool
479 gcpy # Keyhole
480 gcpy      if ptd_tool == 374:
481 gcpy          if ptd_depth < 3.175:
482 gcpy              return 9.525
483 gcpy          else:
484 gcpy              return 6.35
485 gcpy      if ptd_tool == 375:
486 gcpy          if ptd_depth < 3.175:
487 gcpy              return 9.525
488 gcpy          else:
489 gcpy              return 8
490 gcpy      if ptd_tool == 376:
491 gcpy          if ptd_depth < 4.7625:
492 gcpy              return 12.7
493 gcpy          else:
494 gcpy              return 6.35
495 gcpy      if ptd_tool == 378:
496 gcpy          if ptd_depth < 4.7625:
497 gcpy              return 12.7
498 gcpy          else:
499 gcpy              return 8
500 gcpy # Dovetail
501 gcpy      if ptd_tool == 814:
502 gcpy          if ptd_depth > 12.7:
503 gcpy              return 6.35
504 gcpy          else:
505 gcpy              return ptd_tool
506 gcpy      if ptd_tool == 808079:
```

```
507 gcpy          if ptd_depth > 20.95:
508 gcpy              return 6.816
509 gcpy          else:
510 gcpy              return ptd_tool
511 gcpy # Bowl Bit
512 gcpy #https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
           radius-x-3-4-dia-x-5-8-x-1-4-inch-shank.html
513 gcpy          if ptd_tool == 45982:
514 gcpy              if ptd_depth > 6.35:
515 gcpy                  return 15.875
516 gcpy          else:
517 gcpy              return ptd_tool
518 gcpy # Tapered Ball Nose
519 gcpy          if ptd_tool == 204:
520 gcpy              if ptd_depth > 6.35:
521 gcpy                  return ptd_tool
522 gcpy          if ptd_tool == 304:
523 gcpy              if ptd_depth > 6.35:
524 gcpy                  return ptd_tool
525 gcpy          else:
526 gcpy              return ptd_tool
```

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:

```
528 gcpy          def tool_radius(self, ptd_tool, ptd_depth):
529 gcpy              tr = self.tool_diameter(ptd_tool, ptd_depth)/2
530 gcpy              return tr
```

(Note that where values are not fully calculated values currently the passed in tool number (ptd tool)is returned which will need to be replaced with code which calculates the appropriate values.)

3.3.5 Feeds and Speeds

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

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

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

3.4 Movement and Cutting

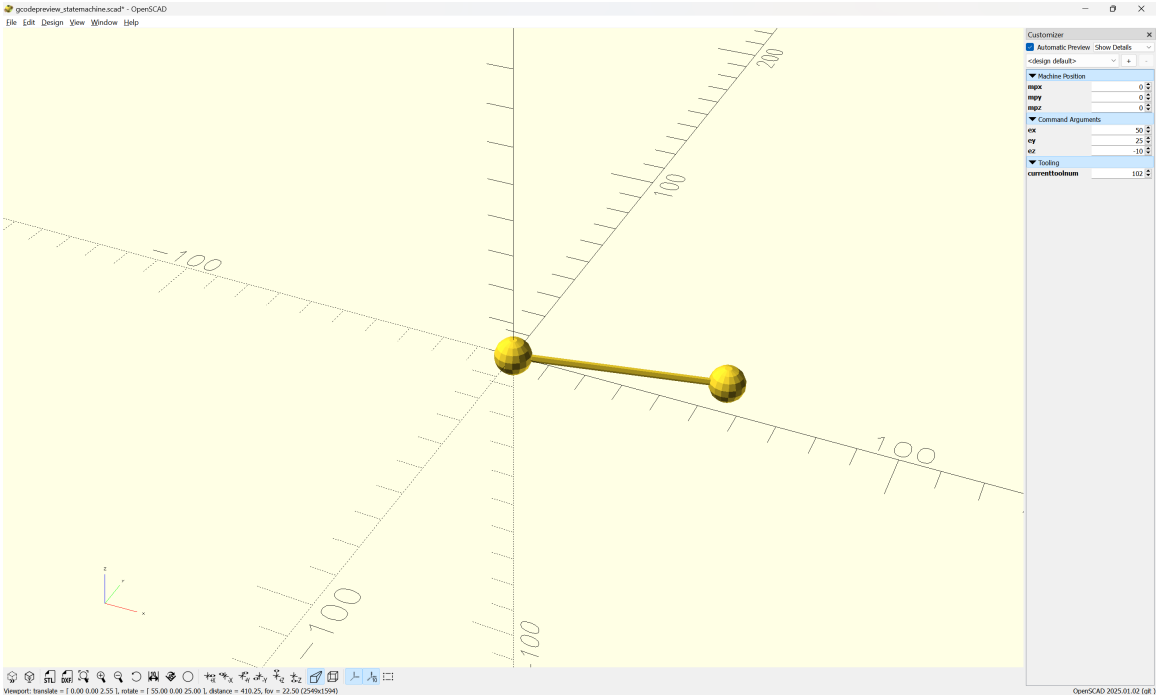
cut... With all the scaffolding in place, it is possible to model the tool and hull() between copies of the 3D model of the tool, or a cross-section of it for both cut... and rapid... operations.
rapid... Note that the variables self.rapids and self.toolpaths are used to hold the accumulated (unioned) 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 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:

```
532 gcpy          def rcs(self, ex, ey, ez, shape):
533 gcpy              start = shape
534 gcpy              end = shape
535 gcpy              toolpath = hull(start.translate([self.xpos(), self.ypos(),
           self.zpos()]),
536 gcpy                  end.translate([ex, ey, ez]))
537 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 G0 movements and will not appear in DXFs but will appear in G-code files, while straight line cut (G1) and arc (G2/G3) commands may appear in both G-code and DXF files, depending on the specific command invoked.

```
539 gcpy      def rapid(self, ex, ey, ez):
540 gcpy      cts = self.currenttoolshape
541 gcpy      toolpath = self.rcs(ex, ey, ez, cts)
542 gcpy      self.setxpos(ex)
543 gcpy      self.setypos(ey)
544 gcpy      self.setzpos(ez)
545 gcpy      if self.generatepaths == True:
546 gcpy          self.rapids = self.rapids.union(toolpath)
547 gcpy      #         return cylinder(0.01, 0, 0.01, center = False, fn = 3)
548 gcpy          return cube([0.001, 0.001, 0.001])
549 gcpy      else:
550 gcpy          return toolpath
551 gcpy
552 gcpy      def cutshape(self, ex, ey, ez):
553 gcpy      cts = self.currenttoolshape
554 gcpy      toolpath = self.rcs(ex, ey, ez, cts)
555 gcpy      if self.generatepaths == True:
556 gcpy          self.toolpaths = self.toolpaths.union(toolpath)
557 gcpy          return cube([0.001, 0.001, 0.001])
558 gcpy      else:
559 gcpy          return toolpath
```

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

It is then possible to add specific rapid... commands to match typical usages of G-code. The first command needs to be a move to/from the safe Z height. In G-code this would be:

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

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

```
561 gcpy      def movetosafeZ(self):
562 gcpy          rapid = self.rapid(self.xpos(), self.ypos(), self.
                                retractheight)
```

```

563 gcpy #         if self.generatepaths == True:
564 gcpy #             rapid = self.rapid(self.xpos(), self.ypos(), self.
                    retractheight)
565 gcpy #             self.rapids = self.rapids.union(rapid)
566 gcpy #         else:
567 gcpy #         if (generategcode == true) {
568 gcpy #             // writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
569 gcpy #             //G1Z24.663F381.0, "F", str(plunge)
570 gcpy #             if self.generatepaths == False:
571 gcpy #                 return rapid
572 gcpy #             else:
573 gcpy #                 return cube([0.001, 0.001, 0.001])
574 gcpy
575 gcpy     def rapidXYZ(self, ex, ey, ez):
576 gcpy         rapid = self.rapid(ex, ey, ez)
577 gcpy         if self.generatepaths == False:
578 gcpy             return rapid
579 gcpy
580 gcpy     def rapidXY(self, ex, ey):
581 gcpy         rapid = self.rapid(ex, ey, self.zpos())
582 gcpy         if self.generatepaths == True:
583 gcpy             self.rapids = self.rapids.union(rapid)
584 gcpy         #
585 gcpy         else:
586 gcpy         if self.generatepaths == False:
587 gcpy             return rapid
588 gcpy
589 gcpy     def rapidXZ(self, ex, ez):
590 gcpy         rapid = self.rapid(ex, self.ypos(), ez)
591 gcpy         if self.generatepaths == False:
592 gcpy             return rapid
593 gcpy
594 gcpy     def rapidYZ(self, ey, ez):
595 gcpy         rapid = self.rapid(self.xpos(), ey, ez)
596 gcpy         if self.generatepaths == False:
597 gcpy             return rapid
598 gcpy
599 gcpy     def rapidX(self, ex):
600 gcpy         rapid = self.rapid(ex, self.ypos(), self.zpos())
601 gcpy         if self.generatepaths == False:
602 gcpy             return rapid
603 gcpy
604 gcpy     def rapidY(self, ey):
605 gcpy         rapid = self.rapid(self.xpos(), ey, self.zpos())
606 gcpy         if self.generatepaths == False:
607 gcpy             return rapid
608 gcpy
609 gcpy     def rapidZ(self, ez):
610 gcpy         rapid = self.rapid(self.xpos(), self.ypos(), ez)
611 gcpy         if self.generatepaths == True:
612 gcpy             self.rapids = self.rapids.union(rapid)
613 gcpy         #
614 gcpy         else:
615 gcpy         if self.generatepaths == False:
616 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):

```

67 gcpscad module movetosafeZ(){
68 gcpscad     gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight);
69 gcpscad }
70 gcpscad
71 gcpscad module rapid(ex, ey, ez) {
72 gcpscad     gcp.rapid(ex, ey, ez);
73 gcpscad }
74 gcpscad
75 gcpscad module rapidXY(ex, ey) {
76 gcpscad     gcp.rapid(ex, ey, gcp.zpos());
77 gcpscad }
78 gcpscad
79 gcpscad module rapidXZ(ex, ez) {
80 gcpscad     gcp.rapid(ex, gcp.zpos(), ez);
81 gcpscad }
82 gcpscad
83 gcpscad module rapidZ(ez) {
84 gcpscad     gcp.rapid(gcp.xpos(), gcp.ypos(), ez);
85 gcpscad }

```

3.4.1 Lines

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

```

616 gcpy      def cutline(self, ex, ey, ez):\
617 gcpy      #below will need to be integrated into if/then structure not yet
              copied
618 gcpy      #          cts = self.currenttoolshape
619 gcpy      if (self.currenttoolnumber() == 374):
620 gcpy      #          self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
621 gcpy      #          self.currenttoolshape = self.keyhole(9.53/2, 3.175)
622 gcpy      #          toolpath = self.cutshape(ex, ey, ez)
623 gcpy      #          self.currenttoolshape = self.keyhole_shaft(6.35/2,
              12.7)
624 gcpy      #          toolpath = toolpath.union(self.cutshape(ex, ey, ez))
625 gcpy      #      elif (self.currenttoolnumber() == 375):
626 gcpy      #          self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
627 gcpy      #      elif (self.currenttoolnumber() == 376):
628 gcpy      #          self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
629 gcpy      #      elif (self.currenttoolnumber() == 378):
630 gcpy      #          self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
631 gcpy      #      elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
632 gcpy      #          self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
              7.9375, 3.175)")
633 gcpy      #      elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
634 gcpy      #          self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
              5.55625, 1.5875)")
635 gcpy      #          toolpath = self.cutroundovertool(self.xpos(), self.ypos
              (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
636 gcpy      #      elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
637 gcpy      #          self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
              12.7, 4.7625)")
638 gcpy      #      else:
639 gcpy      #          toolpath = self.cutshape(ex, ey, ez)
640 gcpy      #          self.setxpos(ex)
641 gcpy      #          self.setypos(ey)
642 gcpy      #          self.setzpos(ez)
643 gcpy      #      if self.generatepaths == True:
644 gcpy      #          self.toolpaths = union([self.toolpaths, toolpath])
645 gcpy      #      else:
646 gcpy      #      if self.generatepaths == False:
647 gcpy      #          return toolpath
648 gcpy      #      else:
649 gcpy      #          return cube([0.001, 0.001, 0.001])
650 gcpy
651 gcpy      def cutlinedxfgc(self, ex, ey, ez):
652 gcpy      #          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
              ypos(), ex, ey)
653 gcpy      #          self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
              )
654 gcpy      #      if self.generatepaths == False:
655 gcpy      #          return self.cutline(ex, ey, ez)
656 gcpy
657 gcpy      def cutvertexdxf(self, ex, ey, ez):
658 gcpy      #          self.addvertex(self.currenttoolnumber(), ex, ey)
659 gcpy      #          self.writegc("G01 X", str(ex), " Y", str(ey), " Z", str(ez)
              ))
660 gcpy      #      if self.generatepaths == False:
661 gcpy      #          return self.cutline(ex, ey, ez)
662 gcpy
663 gcpy      def cutroundovertool(self, bx, by, bz, ex, ey, ez,
              tool_radius_tip, tool_radius_width, stepsizeroundover =1):
664 gcpy      #          n = 90 + fn*3
665 gcpy      #          print("Tool dimensions", tool_radius_tip,
              tool_radius_width, "begin ", bx, by, bz, "end ", ex, ey, ez)
666 gcpy      #          step = 4 #360/n
667 gcpy      #          shaft = cylinder(step, tool_radius_tip, tool_radius_tip)
668 gcpy      #          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
              translate([ex, ey, ez]))
669 gcpy      #          shaft = cylinder(tool_radius_width*2, tool_radius_tip+
              tool_radius_width, tool_radius_tip+tool_radius_width)
670 gcpy      #          toolpath = toolpath.union(hull(shaft.translate([bx, by, bz+
              tool_radius_width]), shaft.translate([ex, ey, ez+
              tool_radius_width])))
671 gcpy      #      for i in range(1, 90, stepsizeroundover):
672 gcpy      #          angle = i

```

```

673 gcpy          dx = tool_radius_width*math.cos(math.radians(angle))
674 gcpy          dxx = tool_radius_width*math.cos(math.radians(angle+1))
675 gcpy          dzz = tool_radius_width*math.sin(math.radians(angle))
676 gcpy          dz = tool_radius_width*math.sin(math.radians(angle+1))
677 gcpy          dh = abs(dzz-dz)+0.0001
678 gcpy          slice = cylinder(dh, tool_radius_tip+tool_radius_width-
                                dx, tool_radius_tip+tool_radius_width-dxx)
679 gcpy          toolpath = toolpath.union(hull(slice.translate([bx, by,
                                bz+dz]), slice.translate([ex, ey, ez+dz])))
680 gcpy          if self.generatepaths == True:
681 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
682 gcpy          else:
683 gcpy              return toolpath
684 gcpy
685 gcpy          def cutlineXYZwithfeed(self, ex, ey, ez, feed):
686 gcpy              return self.cutline(ex, ey, ez)
687 gcpy
688 gcpy          def cutlineXYZ(self, ex, ey, ez):
689 gcpy              return self.cutline(ex, ey, ez)
690 gcpy
691 gcpy          def cutlineXYwithfeed(self, ex, ey, feed):
692 gcpy              return self.cutline(ex, ey, self.zpos())
693 gcpy
694 gcpy          def cutlineXY(self, ex, ey):
695 gcpy              return self.cutline(ex, ey, self.zpos())
696 gcpy
697 gcpy          def cutlineXZwithfeed(self, ex, ez, feed):
698 gcpy              return self.cutline(ex, self.ypos(), ez)
699 gcpy
700 gcpy          def cutlineXZ(self, ex, ez):
701 gcpy              return self.cutline(ex, self.ypos(), ez)
702 gcpy
703 gcpy          def cutlineXwithfeed(self, ex, feed):
704 gcpy              return self.cutline(ex, self.ypos(), self.zpos())
705 gcpy
706 gcpy          def cutlineX(self, ex):
707 gcpy              return self.cutline(ex, self.ypos(), self.zpos())
708 gcpy
709 gcpy          def cutlineYZ(self, ey, ez):
710 gcpy              return self.cutline(self.xpos(), ey, ez)
711 gcpy
712 gcpy          def cutlineYwithfeed(self, ey, feed):
713 gcpy              return self.cutline(self.xpos(), ey, self.zpos())
714 gcpy
715 gcpy          def cutlineY(self, ey):
716 gcpy              return self.cutline(self.xpos(), ey, self.zpos())
717 gcpy
718 gcpy          def cutlineZgcfeed(self, ez, feed):
719 gcpy              self.writegc("G01_Z", str(ez), "F", str(feed))
720 gcpy          # if self.generatepaths == False:
721 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)
722 gcpy
723 gcpy          def cutlineZwithfeed(self, ez, feed):
724 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)
725 gcpy
726 gcpy          def cutlineZ(self, ez):
727 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)

```

The matching OpenSCAD command is a descriptor:

```

87 gcpscad module cutline(ex, ey, ez){
88 gcpscad     gcp.cutline(ex, ey, ez);
89 gcpscad }
90 gcpscad
91 gcpscad module cutlinedxfgc(ex, ey, ez){
92 gcpscad     gcp.cutlinedxfgc(ex, ey, ez);
93 gcpscad }
94 gcpscad
95 gcpscad module cutlineZgcfeed(ez, feed){
96 gcpscad     gcp.cutlineZgcfeed(ez, feed);
97 gcpscad }

```

3.4.2 Arcs for toolpaths and DXFs

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

- cutarcCW — cut a partial arc described in a clock-wise direction
- cutarcCC — counter-clock-wise
- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC — upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCC — while it won't 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/G3 ...
- DXF — dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)
- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

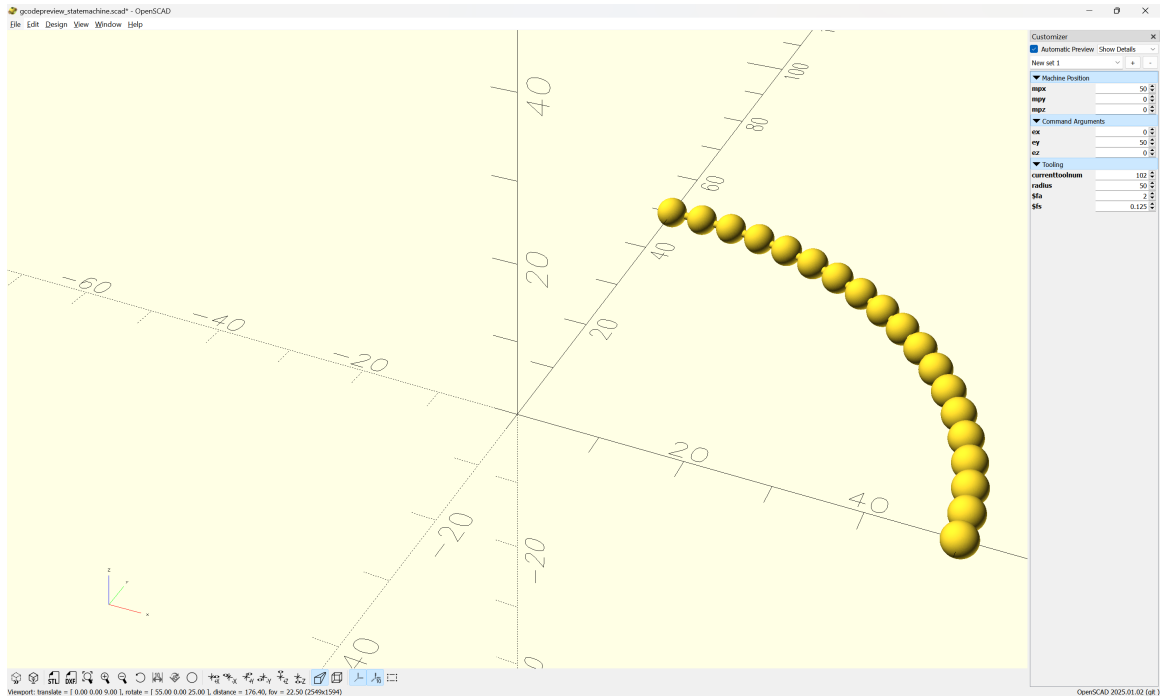
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

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 for machine position
- ey
- ez — allowing a different Z position will make possible threading and similar helical tool-paths
- 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 (potentially) could be used as a check on the other parameters
- tpreldim — 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 loop: cutarcCW (clockwise) or cutarcCC (counterclockwise) to handle the drawing and processing 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);
}
}
}

```

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.)

```

723 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
724 gcpy #      tpzinc = ez - self.zpos() / (earc - barc)
725 gcpy      tpzinc = tpzreldim / (earc - barc)
726 gcpy      cts = self.currenttoolshape
727 gcpy      toolpath = cts
728 gcpy      toolpath = toolpath.translate([self.xpos(), self.ypos(),
                self.zpos()])
729 gcpy      i = barc
730 gcpy      while i < earc:
731 gcpy          toolpath = toolpath.union(self.cutline(xcenter + radius
                * math.cos(math.radians(i)), ycenter + radius *
                math.sin(math.radians(i)), self.zpos()+tpzinc))
732 gcpy          i += stepsizearc
733 gcpy      self.setxpos(xcenter + radius * math.cos(math.radians(earc)
                ))
734 gcpy      self.setypos(ycenter + radius * math.sin(math.radians(earc)
                ))
735 gcpy      if self.generatepaths == False:
736 gcpy          return toolpath
737 gcpy      else:
738 gcpy          return cube([0.01, 0.01, 0.01])
739 gcpy
740 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
741 gcpy #          print(str(self.zpos()))
742 gcpy #          print(str(ez))
743 gcpy #          print(str(barc - earc))
744 gcpy #          tpzinc = ez - self.zpos() / (barc - earc)
745 gcpy #          print(str(tpzinc))
746 gcpy #          global toolpath
747 gcpy #          print("Entering n toolpath")
748 gcpy      tpzinc = tpzreldim / (barc - earc)
749 gcpy      cts = self.currenttoolshape
750 gcpy      toolpath = cts
751 gcpy      toolpath = toolpath.translate([self.xpos(), self.ypos(),
                self.zpos()])
752 gcpy      i = barc
753 gcpy      while i > earc:
754 gcpy          toolpath = toolpath.union(self.cutline(xcenter + radius
                * math.cos(math.radians(i)), ycenter + radius *
                math.sin(math.radians(i)), self.zpos()+tpzinc))
755 gcpy #          self.setxpos(xcenter + radius * math.cos(math.radians(
                i)))
756 gcpy #          self.setypos(ycenter + radius * math.sin(math.radians(
                i)))
757 gcpy #          print(str(self.xpos()), str(self.ypos()), str(self.zpos
                ()))
758 gcpy #          self.setzpos(self.zpos()+tpzinc)
759 gcpy          i += abs(stepsizearc) * -1
760 gcpy #          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                radius, barc, earc)
761 gcpy #          if self.generatepaths == True:
762 gcpy #              print("Unioning n toolpath")
763 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
764 gcpy #          else:
765 gcpy      self.setxpos(xcenter + radius * math.cos(math.radians(earc)
                ))
766 gcpy      self.setypos(ycenter + radius * math.sin(math.radians(earc)
                ))
767 gcpy      if self.generatepaths == False:
768 gcpy          return toolpath

```

```
769 gcpy          else:
770 gcpy          return cube([0.01, 0.01, 0.01])
```

Note that it will be necessary to add versions which write out a matching DXF element:

```
772 gcpy      def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
773 gcpy          tpzreldim, stepsizearc=1):
774 gcpy          toolpath = self.cutarcCW(barc, earc, xcenter, ycenter,
775 gcpy              radius, tpzreldim, stepsizearc=1)
776 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
777 gcpy              radius, earc, barc)
778 gcpy          if self.generatepaths == False:
779 gcpy              return toolpath
780 gcpy          else:
781 gcpy              return cube([0.01, 0.01, 0.01])
```

Matching OpenSCAD modules are easily made:

```
99 gpcscad module cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim){
100 gpcscad     gcp.cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim);
101 gpcscad }
102 gpcscad
103 gpcscad module cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim){
104 gpcscad     gcp.cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim);
105 gpcscad }
```

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 be consolidated in 3.3.3) 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 — dxflines
- arcs — dxfarcs

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. Shapes for which code exists (or is trivially coded) are indicated by Forest Green — for those which have sub-classes, if all are feasible only the higher level is so called out.

- 0
 - 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. (It’s 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 (it’s 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?

These shapes will then be used in constructing toolpaths. The program Carbide Create has toolpath types and options which are as follows:

- Contour — No Offset — the default, this is already supported in the existing code
- Contour — Outside Offset
- Contour — Inside Offset
- Pocket — such toolpaths/geometry should include the rounding of the tool at the corners, c.f., `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

¹en.wikipedia.org/wiki/Hyperbolic_sector and 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 — support is included for specialty tooling such as dovetail cutters allowing one to to get an accurate 3D model, including 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) — would supporting layers be an option?

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

```
780 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
781 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
782 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
783 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
784 gcpy          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 obvious forms for rectangles, square cornered and rounded:

```
786 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
787 gcpy          yheight, corners = "Square", radius = 6):
788 gcpy          if corners == "Square":
789 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
790 gcpy                  xwidth, yorigin)
791 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
792 gcpy                  xorigin + xwidth, yorigin + yheight)
793 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
794 gcpy                  yheight, xorigin, yorigin + yheight)
795 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
796 gcpy                  xorigin, yorigin)
797 gcpy          elif corners == "Fillet":
798 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
799 gcpy                  xwidth, yheight, radius)
800 gcpy          elif corners == "Chamfer":
801 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
802 gcpy                  xwidth, yheight, radius)
803 gcpy          elif corners == "Flipped_Fillet":
804 gcpy              self.dxfrectangleflippedfillet(tool_num, xorigin,
805 gcpy                  yorigin, xwidth, yheight, radius)
```

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

```
799 gcpy      def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
800 gcpy          yheight, radius):
801 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
802 gcpy                  xorigin + xwidth, yorigin + yheight - radius)
803 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
804 gcpy                  yheight - radius, radius, 0, 90)
805 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin
806 gcpy                  + yheight, xorigin + radius, yorigin + yheight)
807 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
808 gcpy                  radius, radius, 90, 180)
809 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius
810 gcpy                  , xorigin, yorigin + radius)
811 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
812 gcpy                  radius, 180, 270)
813 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin
814 gcpy                  + xwidth - radius, yorigin)
815 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
816 gcpy                  radius, radius, 270, 360)
```

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


```
810 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
811 gcpy          xwidth, yheight, radius):
812 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
813 gcpy              yorigin + radius)
814 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
815 gcpy              xorigin + radius, yorigin + yheight)
816 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
817 gcpy              yheight, xorigin + xwidth, yorigin + yheight - radius)
818 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
819 gcpy              xorigin + xwidth, yorigin + radius)
```

Flipped Fillet:

```
821 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
822 gcpy          xwidth, yheight, radius):
823 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
824 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
825 gcpy              90, 180)
826 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
827 gcpy              radius, 180, 270)
828 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
829 gcpy              270, 360)
830 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
831 gcpy              xwidth - radius, yorigin)
832 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
833 gcpy              xorigin + xwidth, yorigin + yheight - radius)
834 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
835 gcpy              yheight, xorigin + radius, yorigin + yheight)
836 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
837 gcpy              xorigin, yorigin + radius)
```

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

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

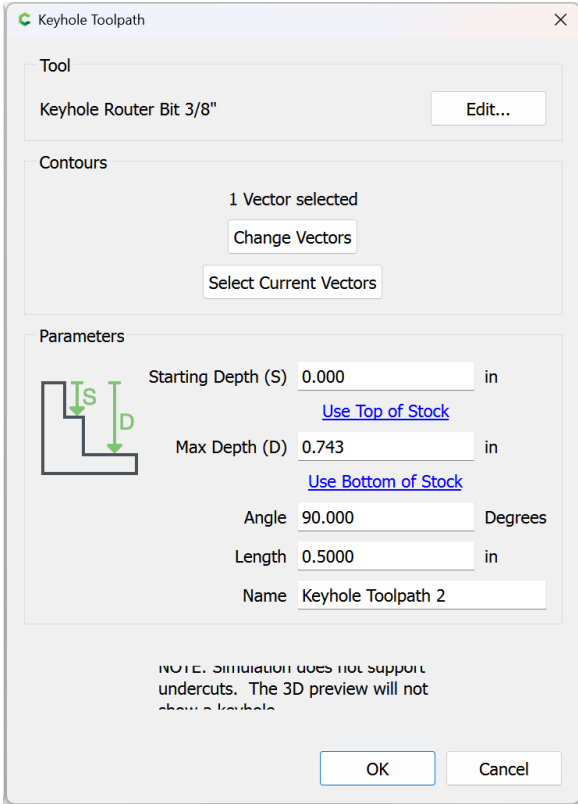
cutrectangleround The routine cutrectangleround cuts the outline of a rectangle creating rounded corners.

```
832 gcpy      def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
833 gcpy          yheight, zdepth, radius):
834 gcpy          tool = self.currenttool()
835 gcpy          toolpath = hull(
836 gcpy              tool.translate([bx+radius,by+radius,bz-zdepth]),
837 gcpy              tool.translate([bx+xwidth-radius,by+radius,bz-zdepth]),
838 gcpy              tool.translate([bx+radius,by+yheight-radius,bz-zdepth]),
839 gcpy              tool.translate([bx+xwidth-radius,by+yheight-radius,bz-
840 gcpy                  zdepth])
841 gcpy          )
842 gcpy          return toolpath
843 gcpy      def cutrectanglerounddxf(self, tool_num, bx, by, bz, xwidth,
844 gcpy          yheight, zdepth, radius):
845 gcpy          toolpath = self.cutrectangleround(tool_num, bx, by, bz,
846 gcpy              xwidth, yheight, zdepth, radius)
847 gcpy          self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
848 gcpy              radius)
849 gcpy          return toolpath
```

3.4.3.2.3 Keyhole toolpath and undercut tooling The first topologically unusual toolpath is cutkeyhole toolpath 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 and which 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 3.3.2.2

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:

```
847 gcpy      def cutkeyholegcdfx(self, kh_tool_num, kh_start_depth,
848 gcpy          kh_max_depth, kht_direction, kh_distance):
849 gcpy          toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
850 gcpy              kh_max_depth, 90, kh_distance)
851 gcpy          elif (kht_direction == "S"):
852 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
853 gcpy                  kh_max_depth, 270, kh_distance)
852 gcpy          elif (kht_direction == "E"):
853 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
853 gcpy                  kh_max_depth, 0, kh_distance)
```

```
854 gcpy          elif (kht_direction == "W"):  
855 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth, kh_max_depth, kh_distance)  
856 gcpy          if self.generatepaths == True:  
857 gcpy              self.toolpaths = union([self.toolpaths, toolpath])  
858 gcpy              return toolpath  
859 gcpy          else:  
860 gcpy              return cube([0.01, 0.01, 0.01])  
  
107 gcpscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth, kht_direction, kh_distance){  
108 gcpscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth, kht_direction, kh_distance);  
109 gcpscad }
```

cutKHgcdxf The original version of the command, cutKHgcdxf retains an interface which allows calling it for arbitrary beginning and ending points of an arc.

Note that code is still present for the partial calculation of one quadrant (for the case of all nodes within the quadrant). The first task is to place a circle at the origin which is invariant of angle:

```
862 gcpy          def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth, kh_angle, kh_distance):  
863 gcpy              oXpos = self.xpos()  
864 gcpy              oYpos = self.ypos()  
865 gcpy              self.dxfKH(kh_tool_num, self.xpos(), self.ypos(), kh_start_depth, kh_max_depth, kh_angle, kh_distance)  
866 gcpy              toolpath = self.cutline(self.xpos(), self.ypos(), - kh_max_depth)  
867 gcpy              self.setxpos(oXpos)  
868 gcpy              self.setypos(oYpos)  
869 gcpy              if self.generatepaths == False:  
870 gcpy                  return toolpath  
871 gcpy              else:  
872 gcpy                  return cube([0.001, 0.001, 0.001])  
  
874 gcpy          def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth, kh_max_depth, kh_angle, kh_distance):  
875 gcpy              oXpos = self.xpos()  
876 gcpy              oYpos = self.ypos()  
877 gcpy              #Circle at entry hole  
878 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(kh_tool_num, 7), 0, 90)  
879 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(kh_tool_num, 7), 90, 180)  
880 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(kh_tool_num, 7), 180, 270)  
881 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(kh_tool_num, 7), 270, 360)
```

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

```
882 gcpy #pre-calculate needed values  
883 gcpy     r = self.tool_radius(kh_tool_num, 7)  
884 gcpy     # print(r)  
885 gcpy     rt = self.tool_radius(kh_tool_num, 1)  
886 gcpy     # print(rt)  
887 gcpy     ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.tool_radius(kh_tool_num, 7))**2)  
888 gcpy     # print(ro)  
889 gcpy     angle = math.degrees(math.acos(ro/rt))  
890 gcpy #Outlines of entry hole and slot  
891 gcpy     if (kh_angle == 0):  
892 gcpy #Lower left of entry hole  
893 gcpy         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self.tool_radius(kh_tool_num, 1), 180, 270)  
894 gcpy #Upper left of entry hole  
895 gcpy         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self.tool_radius(kh_tool_num, 1), 90, 180)  
896 gcpy #Upper right of entry hole  
897 gcpy #         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt, 41.810, 90)  
898 gcpy         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt, angle, 90)
```

```

899 gcpy #Lower right of entry hole
900 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                          270, 360-angle)
901 gcpy #          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(),
                          self.tool_radius(kh_tool_num, 1), 270, 270+math.acos(math.
                          radians(self.tool_diameter(kh_tool_num, 5)/self.tool_diameter(
                          kh_tool_num, 1))))
902 gcpy #Actual line of cut
903 gcpy #          self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
                          self.xpos()+kh_distance, self.ypos())
904 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
905 gcpy          self.dxfarc(kh_tool_num, self.xpos()+kh_distance, self.
                          ypos(), self.tool_diameter(kh_tool_num, (
                          kh_max_depth+4.36))/2, 0, 90)
906 gcpy #lower right of end of slot
907 gcpy          self.dxfarc(kh_tool_num, self.xpos()+kh_distance, self.
                          ypos(), self.tool_diameter(kh_tool_num, (
                          kh_max_depth+4.36))/2, 270, 360)
908 gcpy #upper right slot
909 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))
910 gcpy #          self.dxfline(kh_tool_num, self.xpos()+(sqrt((self.
                          tool_diameter(kh_tool_num, 1)^2)-(self.tool_diameter(kh_tool_num
                          , 5)^2))/2), self.ypos()+self.tool_diameter(kh_tool_num, (
                          kh_max_depth))/2, ((kh_max_depth-6.34))/2)^2-(self.
                          tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.xpos
                          ()+kh_distance, self.ypos()+self.tool_diameter(kh_tool_num, (
                          kh_max_depth))/2, kh_tool_num)
911 gcpy #end position at top of slot
912 gcpy #lower right slot
913 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))
914 gcpy #          dxfline(kh_tool_num, self.xpos()+(sqrt((self.tool_diameter
                          (kh_tool_num, 1)^2)-(self.tool_diameter(kh_tool_num, 5)^2))/2),
                          self.ypos()-self.tool_diameter(kh_tool_num, (kh_max_depth))/2, (
                          (kh_max_depth-6.34))/2)^2-(self.tool_diameter(kh_tool_num, (
                          kh_max_depth-6.34))/2)^2, self.xpos()+kh_distance, self.ypos()-
                          self.tool_diameter(kh_tool_num, (kh_max_depth))/2, KH_tool_num)
915 gcpy #end position at top of slot
916 gcpy #          hull(){
917 gcpy #              translate([xpos(), ypos(), zpos()]){
918 gcpy #                  keyhole_shaft(6.35, 9.525);
919 gcpy #              }
920 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
921 gcpy #                  keyhole_shaft(6.35, 9.525);
922 gcpy #              }
923 gcpy #          }
924 gcpy #          hull(){
925 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
926 gcpy #                  keyhole_shaft(6.35, 9.525);
927 gcpy #              }
928 gcpy #              translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
          {
929 gcpy #                  keyhole_shaft(6.35, 9.525);
930 gcpy #              }
931 gcpy #          }
932 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
933 gcpy #          cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
          feed);
934 gcpy #          setxpos(getxpos()-kh_distance);
935 gcpy #      } else if (kh_angle > 0 && kh_angle < 90) {
936 gcpy #          //echo(kh_angle);
937 gcpy #          dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
938 gcpy #          dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
939 gcpy #          dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, kh_angle+asin((tool_diameter(KH_tool_num, (
          kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth
          ))/2)), 90+kh_angle, KH_tool_num);
940 gcpy #          dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, 270+kh_angle, 360+kh_angle-asin((tool_diameter
          (KH_tool_num, (kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num
          , (kh_max_depth))/2)), KH_tool_num);

```

```

941 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
942 gcpy #  getypos()+(kh_distance*sin(kh_angle)), tool_diameter(KH_tool_num
    , (kh_max_depth+4.36))/2, 0+kh_angle, 90+kh_angle, KH_tool_num);
943 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);
944 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))),
945 gcpy #  getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(
    kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
    /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
946 gcpy #  getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
    , (kh_max_depth+4.36))/2)*sin(kh_angle)),
947 gcpy #  getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
    , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
948 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
949 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
950 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
    +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
951 gcpy #//echo(kh_angle);
952 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
    kh_distance*sin(kh_angle)), -kh_max_depth, feed);
953 gcpy #      toolpath = toolpath.union(self.cutline(self.xpos()+
    kh_distance, self.ypos(), -kh_max_depth))
954 gcpy      elif (kh_angle == 90):
955 gcpy #Lower left of entry hole
956 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
    (kh_tool_num, 1), 180, 270)
957 gcpy #Lower right of entry hole
958 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
    (kh_tool_num, 1), 270, 360)
959 gcpy #left slot
960 gcpy      self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
    oYpos+kh_distance)
961 gcpy #right slot
962 gcpy      self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
    oYpos+kh_distance)
963 gcpy #upper left of end of slot
964 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
    90, 180)
965 gcpy #upper right of end of slot
966 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
    0, 90)
967 gcpy #Upper right of entry hole
968 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
969 gcpy #Upper left of entry hole
970 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
    180)
971 gcpy #      toolpath = toolpath.union(self.cutline(oXpos, oYpos+
    kh_distance, -kh_max_depth))
972 gcpy      elif (kh_angle == 180):
973 gcpy #Lower right of entry hole
974 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
    (kh_tool_num, 1), 270, 360)
975 gcpy #Upper right of entry hole
976 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
    (kh_tool_num, 1), 0, 90)
977 gcpy #Upper left of entry hole
978 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
    angle)
979 gcpy #Lower left of entry hole
980 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
    270)
981 gcpy #upper slot
982 gcpy      self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
    kh_distance, oYpos-r)
983 gcpy #lower slot
984 gcpy      self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
    kh_distance, oYpos+r)
985 gcpy #upper left of end of slot
986 gcpy      self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
    90, 180)
987 gcpy #lower left of end of slot
988 gcpy      self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
    180, 270)
989 gcpy #      toolpath = toolpath.union(self.cutline(oXpos-
    kh_distance, oYpos, -kh_max_depth))

```

```

990 gcpy          elif (kh_angle == 270):
991 gcpy #Upper left of entry hole
992 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                             (kh_tool_num, 1), 90, 180)
993 gcpy #Upper right of entry hole
994 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                             (kh_tool_num, 1), 0, 90)
995 gcpy #left slot
996 gcpy          self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                             oYpos-kh_distance)
997 gcpy #right slot
998 gcpy          self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                             oYpos-kh_distance)
999 gcpy #lower left of end of slot
1000 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                             180, 270)
1001 gcpy #lower right of end of slot
1002 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                             270, 360)
1003 gcpy #lower right of entry hole
1004 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                             angle)
1005 gcpy #lower left of entry hole
1006 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                             360)
1007 gcpy #          toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                             kh_distance, -kh_max_depth))
1008 gcpy #          print(self.zpos())
1009 gcpy #          self.setxpos(oXpos)
1010 gcpy #          self.setypos(oYpos)
1011 gcpy #          if self.generatepaths == False:
1012 gcpy #              return toolpath
1013 gcpy
1014 gcpy # } else if (kh_angle == 90) {
1015 gcpy # //Lower left of entry hole
1016 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
1017 gcpy # //Lower right of entry hole
1018 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
1019 gcpy # //Upper right of entry hole
1020 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
                             KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
1021 gcpy # //Upper left of entry hole
1022 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180-acos(tool_diameter(
                             KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180, KH_tool_num
                             );
1023 gcpy # //Actual line of cut
1024 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
                             );
1025 gcpy # //upper right of slot
1026 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                             KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
1027 gcpy # //upper left of slot
1028 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                             KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
1029 gcpy # //right of slot
1030 gcpy # dxfline(
1031 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1032 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,
1033 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1034 gcpy #     //end position at top of slot
1035 gcpy #     getypos()+kh_distance,
1036 gcpy #     KH_tool_num);
1037 gcpy # dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))
                             /2, getypos()+(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
                             tool_diameter(KH_tool_num, 5)^2))/2), getxpos()-tool_diameter(
                             KH_tool_num, (kh_max_depth+6.35))/2, getypos()+kh_distance,
                             KH_tool_num);
1038 gcpy # hull(){
1039 gcpy #     translate([xpos(), ypos(), zpos()]){
1040 gcpy #         keyhole_shaft(6.35, 9.525);
1041 gcpy #     }
1042 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1043 gcpy #         keyhole_shaft(6.35, 9.525);
1044 gcpy #     }
1045 gcpy # }
1046 gcpy # hull(){

```

```

1047 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
1048 gcpy #             keyhole_shaft(6.35, 9.525);
1049 gcpy #         }
1050 gcpy #         translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
1051 gcpy #     {
1052 gcpy #         keyhole_shaft(6.35, 9.525);
1053 gcpy #     }
1054 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1055 gcpy #     cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
1056 gcpy #         feed);
1057 gcpy #     setypos(getypos()-kh_distance);
1058 gcpy # } else if (kh_angle == 180) {
1059 gcpy #     //Lower right of entry hole
1060 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
1061 gcpy #     //Upper right of entry hole
1062 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
1063 gcpy #     //Upper left of entry hole
1064 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 90, 90+acos(
1065 gcpy #         tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
1066 gcpy #         KH_tool_num);
1067 gcpy #     //Lower left of entry hole
1068 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270-acos(tool_diameter(
1069 gcpy #         KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num
1070 gcpy #     );
1071 gcpy #     //upper left of slot
1072 gcpy #     dxline(
1073 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
1074 gcpy #             tool_diameter(KH_tool_num, 5)^2))/2),
1075 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1076 gcpy #         //((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
1077 gcpy #             kh_max_depth-6.34))/2)^2,
1078 gcpy #         getxpos()-kh_distance,
1079 gcpy #         //end position at top of slot
1080 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1081 gcpy #         KH_tool_num);
1082 gcpy #     //lower right slot
1083 gcpy #     dxline(
1084 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
1085 gcpy #             tool_diameter(KH_tool_num, 5)^2))/2),
1086 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1087 gcpy #         //((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
1088 gcpy #             kh_max_depth-6.34))/2)^2,
1089 gcpy #         getxpos()-kh_distance,
1090 gcpy #         //end position at top of slot
1091 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1092 gcpy #         KH_tool_num);
1093 gcpy #     hull(){
1094 gcpy #         translate([xpos(), ypos(), zpos()]){
1095 gcpy #             keyhole_shaft(6.35, 9.525);
1096 gcpy #         }
1097 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
1098 gcpy #             keyhole_shaft(6.35, 9.525);
1099 gcpy #         }
1100 gcpy #     }
1101 gcpy #     hull(){
1102 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
1103 gcpy #             keyhole_shaft(6.35, 9.525);
1104 gcpy #         }
1105 gcpy #         translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
1106 gcpy #     {
1107 gcpy #         keyhole_shaft(6.35, 9.525);
1108 gcpy #     }
1109 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1110 gcpy #     cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
1111 gcpy #         feed);
1112 gcpy #     setxpos(getxpos()+kh_distance);
1113 gcpy # } else if (kh_angle == 270) {

```

```

1108 gcpy # //Upper right of entry hole
1109 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
1110 gcpy # //Upper left of entry hole
1111 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
1112 gcpy # //lower right of slot
1113 gcpy # dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
1114 gcpy # //lower left of slot
1115 gcpy # dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
1116 gcpy # //Actual line of cut
1117 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
);
1118 gcpy # //right of slot
1119 gcpy # dxfline(
1120 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1121 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,
1122 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1123 gcpy # //end position at top of slot
1124 gcpy #     getypos()-kh_distance,
1125 gcpy #     KH_tool_num);
1126 gcpy # //left of slot
1127 gcpy # dxfline(
1128 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1129 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,
1130 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1131 gcpy # //end position at top of slot
1132 gcpy #     getypos()-kh_distance,
1133 gcpy #     KH_tool_num);
1134 gcpy # //Lower right of entry hole
1135 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
);
1136 gcpy # //Lower left of entry hole
1137 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
KH_tool_num);
1138 gcpy # hull(){
1139 gcpy #     translate([xpos(), ypos(), zpos()]){
1140 gcpy #         keyhole_shaft(6.35, 9.525);
1141 gcpy #     }
1142 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1143 gcpy #         keyhole_shaft(6.35, 9.525);
1144 gcpy #     }
1145 gcpy # }
1146 gcpy # hull(){
1147 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1148 gcpy #         keyhole_shaft(6.35, 9.525);
1149 gcpy #     }
1150 gcpy #     translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
{
1151 gcpy #         keyhole_shaft(6.35, 9.525);
1152 gcpy #     }
1153 gcpy # }
1154 gcpy # cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1155 gcpy # cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
feed);
1156 gcpy # setypos(getypos()+kh_distance);
1157 gcpy # }
1158 gcpy #}

```

Dovetail joinery and tooling One focus of this project from the beginning has been cutting joinery. The first such toolpath to be developed is half-blind dovetails, since they are intrinsically simple to calculate since their geometry is dictated by the geometry of the tool.

BlocksCAD project page at: <https://www.blocks3d.com/community/projects/1941456>
and discussion at: <https://community.carbide3d.com/t/tool-paths-for-different-sized-dovetail-bit/89098>

Making such cuts will require dovetail tooling such as:

- 808079 <https://www.amanatool.com/45828-carbide-tipped-dovetail-8-deg-x-1-2-dia-x-825-x-1.html>

- 814 <https://www.leevalley.com/en-us/shop/tools/power-tool-accessories/router-bits/30172-dovetail-bits?item=18J1607>

Two commands are required:

```
1160 gcpy      def cut_pins(self, Joint_Width, stockZthickness,
                    Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                    DTT_angle):
1161 gcpy      DT0 = math.tan(math.radians(DTT_angle)) * (stockZthickness
                    * Proportion)
1162 gcpy      DTR = DTT_diameter/2 - DT0
1163 gcpy      cpr = self.rapidXY(0, stockZthickness + Spacing/2)
1164 gcpy      ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                    stockZthickness * Proportion)
1165 gcpy #      ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
                    Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
                    Proportion))
1166 gcpy      i = 1
1167 gcpy      while i < Number_of_Dovetails * 2:
1168 gcpy #          print(i)
1169 gcpy          ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                    Number_of_Dovetails * 2)), self.ypos(), -
                    stockZthickness * Proportion))
1170 gcpy          ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                    Number_of_Dovetails * 2)), (stockZthickness +
                    Spacing) + (stockZthickness * Proportion) - (
                    DTT_diameter/2), -(stockZthickness * Proportion)))
1171 gcpy          ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                    Number_of_Dovetails * 2)), stockZthickness + Spacing
                    /2, -(stockZthickness * Proportion)))
1172 gcpy          ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
                    Joint_Width / (Number_of_Dovetails * 2)),
                    stockZthickness + Spacing/2, -(stockZthickness *
                    Proportion)))
1173 gcpy          self.dxfrectangleround(self.currenttoolnumber(),
                    i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
                    stockZthickness + (Spacing/2) - DTR,
                    DTR * 2,
                    (stockZthickness * Proportion) + Spacing/2 + DTR *
                    2 - (DTT_diameter/2),
                    DTR)
1178 gcpy          i += 2
1179 gcpy      self.rapidZ(0)
1180 gcpy      return ctp
1181 gcpy
```

and

```
1183 gcpy      def cut_tails(self, Joint_Width, stockZthickness,
                    Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                    DTT_angle):
1184 gcpy      DT0 = math.tan(math.radians(DTT_angle)) * (stockZthickness
                    * Proportion)
1185 gcpy      DTR = DTT_diameter/2 - DT0
1186 gcpy      cpr = self.rapidXY(0, 0)
1187 gcpy      ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                    stockZthickness * Proportion)
1188 gcpy      ctp = ctp.union(self.cutlinedxfgc(
                    Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
                    - DT0),
                    self.ypos(),
                    -stockZthickness * Proportion))
1191 gcpy      i = 1
1192 gcpy      while i < Number_of_Dovetails * 2:
1193 gcpy          ctp = ctp.union(self.cutlinedxfgc(
                    i * (Joint_Width / (Number_of_Dovetails * 2)) - (
                    DTT_diameter - DT0),
                    stockZthickness * Proportion - DTT_diameter / 2,
                    -(stockZthickness * Proportion)))
1197 gcpy          ctp = ctp.union(self.cutarcCWdxf(180, 90,
                    i * (Joint_Width / (Number_of_Dovetails * 2)),
                    stockZthickness * Proportion - DTT_diameter / 2,
                    self.ypos(),
                    DTT_diameter - DT0, 0, 1))
1202 gcpy          ctp = ctp.union(self.cutarcCWdxf(90, 0,
                    i * (Joint_Width / (Number_of_Dovetails * 2)),
                    stockZthickness * Proportion - DTT_diameter / 2,
                    DTT_diameter - DT0, 0, 1))
1207 gcpy          ctp = ctp.union(self.cutlinedxfgc(
```

```
1208 gcpy          i * (Joint_Width / (Number_of_Dovetails * 2)) + (
                    DTT_diameter - DT0),
1209 gcpy          0,
1210 gcpy          -(stockZthickness * Proportion)))
1211 gcpy          ctp = ctp.union(self.cutlinedxfgc(
1212 gcpy          (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                    - (DTT_diameter - DT0),
1213 gcpy          0,
1214 gcpy          -(stockZthickness * Proportion)))
1215 gcpy          i += 2
1216 gcpy          self.rapidZ(0)
1217 gcpy          self.rapidXY(0, 0)
1218 gcpy          ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
                    -stockZthickness * Proportion))
1219 gcpy          self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
1220 gcpy          self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
                    stockZthickness + DTR)
1221 gcpy          self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
                    DTR, DTR, 90, 180)
1222 gcpy          self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
                    DTR * 2, Joint_Width, stockZthickness + DTR * 2)
1223 gcpy          i = 0
1224 gcpy          while i < Number_of_Dovetails * 2:
1225 gcpy              ctp = ctp.union(self.cutline(i * (Joint_Width / (
                    Number_of_Dovetails * 2)), stockZthickness + DT0, -(
                    stockZthickness * Proportion)))
1226 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                    Number_of_Dovetails * 2)), stockZthickness + DT0, -(
                    stockZthickness * Proportion)))
1227 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                    Number_of_Dovetails * 2)), 0, -(stockZthickness *
                    Proportion)))
1228 gcpy              self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
                    / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
1229 gcpy              self.dxfline(self.currenttoolnumber(),
1230 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                    ,
1231 gcpy                  0,
1232 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                    , stockZthickness * Proportion - DTT_diameter /
                    2)
1233 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
                    Joint_Width / (Number_of_Dovetails * 2)),
                    stockZthickness * Proportion - DTT_diameter / 2, (
                    Joint_Width / (Number_of_Dovetails * 2)) - DTR, 90,
                    180)
1234 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
                    Joint_Width / (Number_of_Dovetails * 2)),
                    stockZthickness * Proportion - DTT_diameter / 2, (
                    Joint_Width / (Number_of_Dovetails * 2)) - DTR, 0,
                    90)
1235 gcpy              self.dxfline(self.currenttoolnumber(),
1236 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                    - DTR,
1237 gcpy                  0,
1238 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                    - DTR, stockZthickness * Proportion -
                    DTT_diameter / 2)
1239 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 2) * (
                    Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
                    180, 270)
1240 gcpy              i += 2
1241 gcpy              self.dxfarc(self.currenttoolnumber(), Joint_Width,
                    stockZthickness + DTR, DTR, 0, 90)
1242 gcpy              self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
                    stockZthickness + DTR, Joint_Width + DTR, 0)
1243 gcpy              self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,
                    270, 360)
1244 gcpy              return ctp
```

which are used as:

```
toolpaths = gcp.cut_pins(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, Proportion, DTT_dia
toolpaths = toolpaths.union(gcp.cut_tails(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, P
```

Future versions may adjust the parameters passed in, having them calculate from the speci-
fications for the currently active dovetail tool.

Full-blind box joints BlocksCAD project page at: <https://www.blocks3d.com/community/projects/1943966> and discussion at: <https://community.carbide3d.com/t/full-blind-box-joints-in-ca53329>

Full-blind box joints will require 3 separate tools:

- small V tool — this will be needed to make a cut along the edge of the joint
- small square tool — this should be the same diameter as the small V tool
- large V tool — this will facilitate the stock being of a greater thickness and avoid the need to make multiple cuts to cut the blind miters at the ends of the joint

Two different versions of the commands will be necessary, one for each orientation:

- horizontal
- vertical

and then the internal commands for each side will in turn need separate versions:

```

1246 gcpy      def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
              width, thickness, largeVdiameter, smallDiameter,
              normalormirror = "Default", squaretool = 102, smallV = 390,
              largeV = 301):
1247 gcpy      Number_of_Pins = int(((width - thickness * 2) / (
              smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
1248 gcpy #      print("Number of Pins: ", Number_of_Pins)
1249 gcpy      self.movetosafeZ()
1250 gcpy      self.toolchange(squaretool, 17000)
1251 gcpy      toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
              orientation, side, width, thickness, Number_of_Pins,
              largeVdiameter, smallDiameter)
1252 gcpy      self.movetosafeZ()
1253 gcpy      self.toolchange(smallV, 17000)
1254 gcpy      toolpath = toolpath.union(self.
              Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
              , width, thickness, Number_of_Pins, largeVdiameter,
              smallDiameter))
1255 gcpy      self.toolchange(largeV, 17000)
1256 gcpy      toolpath = toolpath.union(self.
              Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
              , width, thickness, Number_of_Pins, largeVdiameter,
              smallDiameter))
1257 gcpy      return toolpath
1258 gcpy
1259 gcpy      def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
              side, width, thickness, Number_of_Pins, largeVdiameter,
              smallDiameter, normalormirror = "Default"):
1260 gcpy #      Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
              "Upper"
1261 gcpy #      Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
              Right"
1262 gcpy      if (orientation == "Vertical"):
1263 gcpy          if (normalormirror == "Default" and side != "Both"):
1264 gcpy              if (side == "Left"):
1265 gcpy                  normalormirror = "Even"
1266 gcpy              if (side == "Right"):
1267 gcpy                  normalormirror = "Odd"
1268 gcpy      if (orientation == "Horizontal"):
1269 gcpy          if (normalormirror == "Default" and side != "Both"):
1270 gcpy              if (side == "Lower"):
1271 gcpy                  normalormirror = "Even"
1272 gcpy              if (side == "Upper"):
1273 gcpy                  normalormirror = "Odd"
1274 gcpy      Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
              1.1
1275 gcpy      Finger-Origin = width/2 - Finger_Width/2
1276 gcpy      rapid = self.rapidZ(0)
1277 gcpy      self.setdxcolor("Cyan")
1278 gcpy      rapid = rapid.union(self.rapidXY(bx, by))
1279 gcpy      toolpath = (self.Finger_Joint_square(bx, by, orientation,
              side, width, thickness, Number_of_Pins, Finger-Origin,
              smallDiameter))
1280 gcpy      if (orientation == "Vertical"):
1281 gcpy          if (side == "Both"):
1282 gcpy              toolpath = self.cutrectangleroundxf(self.
              currenttoolnum, bx - (thickness - smallDiameter
              /2), by-smallDiameter/2, 0, (thickness * 2) -
              smallDiameter, width+smallDiameter, (
              smallDiameter / 2) / math.tan(math.radians(45)),
              smallDiameter/2)

```

```

1283 gcpy          if (side == "Left"):
1284 gcpy              toolpath = self.cutrectanglerounddx(
                    self.currenttoolnum, bx - (smallDiameter/2), by -
                    smallDiameter/2, 0, thickness, width+
                    smallDiameter, ((smallDiameter / 2) / math.tan(
                    math.radians(45))), smallDiameter/2)
1285 gcpy          if (side == "Right"):
1286 gcpy              toolpath = self.cutrectanglerounddx(
                    self.currenttoolnum, bx - (thickness - smallDiameter
                    /2), by-smallDiameter/2, 0, thickness, width+
                    smallDiameter, ((smallDiameter / 2) / math.tan(
                    math.radians(45))), smallDiameter/2)
1287 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
                    orientation, side, width, thickness, Number_of_Pins,
                    Finger_Origin, smallDiameter))
1288 gcpy          if (orientation == "Horizontal"):
1289 gcpy              if (side == "Both"):
1290 gcpy                  toolpath = self.cutrectanglerounddx(
                    self.currenttoolnum,
                    bx-smallDiameter/2,
                    by - (thickness - smallDiameter/2),
                    0,
                    width+smallDiameter,
                    (thickness * 2) - smallDiameter,
                    (smallDiameter / 2) / math.tan(math.radians(45)
                    ),
                    smallDiameter/2)
1291 gcpy
1292 gcpy              if (side == "Lower"):
1293 gcpy                  toolpath = self.cutrectanglerounddx(
                    self.currenttoolnum,
                    bx - (smallDiameter/2),
                    by - smallDiameter/2,
                    0,
                    width+smallDiameter,
                    thickness,
                    ((smallDiameter / 2) / math.tan(math.radians
                    (45))),
                    smallDiameter/2)
1294 gcpy
1295 gcpy              if (side == "Upper"):
1296 gcpy                  toolpath = self.cutrectanglerounddx(
                    self.currenttoolnum,
                    bx - smallDiameter/2,
                    by - (thickness - smallDiameter/2),
                    0,
                    width+smallDiameter,
                    thickness,
                    ((smallDiameter / 2) / math.tan(math.radians
                    (45))),
                    smallDiameter/2)
1297 gcpy
1298 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
                    orientation, side, width, thickness, Number_of_Pins,
                    Finger_Origin, smallDiameter))
1299 gcpy
1300 gcpy          return toolpath
1301 gcpy
1302 gcpy          def Finger_Joint_square(self, bx, by, orientation, side, width,
1303 gcpy              thickness, Number_of_Pins, Finger_Origin, smallDiameter,
1304 gcpy              normalormirror = "Default"):
1305 gcpy              jointdepth = -(thickness - (smallDiameter / 2) / math.tan(
1306 gcpy                  math.radians(45)))
1307 gcpy              # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
1308 gcpy                  "Upper"
1309 gcpy              # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
1310 gcpy                  Right"
1311 gcpy              if (orientation == "Vertical"):
1312 gcpy                  if (normalormirror == "Default" and side != "Both"):
1313 gcpy                      if (side == "Left"):
1314 gcpy                          normalormirror = "Even"
1315 gcpy                      if (side == "Right"):
1316 gcpy                          normalormirror = "Odd"
1317 gcpy              if (orientation == "Horizontal"):
1318 gcpy                  if (normalormirror == "Default" and side != "Both"):
1319 gcpy                      if (side == "Lower"):
1320 gcpy                          normalormirror = "Even"
1321 gcpy                      if (side == "Upper"):
1322 gcpy                          normalormirror = "Odd"
1323 gcpy              radius = smallDiameter/2
1324 gcpy              jointwidth = thickness - smallDiameter
1325 gcpy              toolpath = self.currenttool()

```

```

1341 gcpy          rapid = self.rapidZ(0)
1342 gcpy          self.setdxfcolor("Blue")
1343 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
,1000))

1344 gcpy          self.beginpolyline(self.currenttool())
1345 gcpy          if (orientation == "Vertical"):
1346 gcpy              rapid = rapid.union(self.rapidXY(bx, by + Finger_Origin
))
1347 gcpy              self.addvertex(self.currenttoolnumber(), self.xpos(),
self.ypos())
1348 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(
jointdepth,1000))
1349 gcpy          i = 0
1350 gcpy          while i <= Number_of_Pins - 1:
1351 gcpy              if (side == "Right"):
1352 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + smallDiameter +
radius/5, jointdepth))
1353 gcpy              if (side == "Left" or side == "Both"):
1354 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius,
jointdepth))
1355 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos() + jointwidth, self.ypos(),
jointdepth))
1356 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius/5,
jointdepth))
1357 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos() - jointwidth, self.ypos(),
jointdepth))
1358 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius,
jointdepth))
1359 gcpy              if (side == "Left"):
1360 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + smallDiameter +
radius/5, jointdepth))
1361 gcpy              if (side == "Right" or side == "Both"):
1362 gcpy                  if (i < (Number_of_Pins - 1)):
1363 gcpy                      # print(i)
1364 gcpy                      toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius,
jointdepth))
1365 gcpy                      toolpath = toolpath.union(self.cutvertexdx(
self.xpos() - jointwidth, self.ypos(),
jointdepth))
1366 gcpy                      toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius/5,
jointdepth))
1367 gcpy                      toolpath = toolpath.union(self.cutvertexdx(
self.xpos() + jointwidth, self.ypos(),
jointdepth))
1368 gcpy                      toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + radius,
jointdepth))

1369 gcpy              i += 1
1370 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
"Upper"
1371 gcpy          if (orientation == "Horizontal"):
1372 gcpy              rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
))
1373 gcpy              self.addvertex(self.currenttoolnumber(), self.xpos(),
self.ypos())
1374 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(
jointdepth,1000))
1375 gcpy          i = 0
1376 gcpy          while i <= Number_of_Pins - 1:
1377 gcpy              if (side == "Upper"):
1378 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos() + smallDiameter + radius/5, self
.ypos(), jointdepth))
1379 gcpy              if (side == "Lower" or side == "Both"):
1380 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos() + radius, self.ypos(),
jointdepth))
1381 gcpy                  toolpath = toolpath.union(self.cutvertexdx(
self.xpos(), self.ypos() + jointwidth,

```

```

                                jointdepth))
1382 gcpy                    toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + radius/5, self.ypos(),
                                jointdepth))
1383 gcpy                    toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() - jointwidth,
                                jointdepth))
1384 gcpy                    toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + radius, self.ypos(),
                                jointdepth))
1385 gcpy                    if (side == "Lower"):
1386 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + smallDiameter + radius/5, self
                                .ypos(), jointdepth))
1387 gcpy                    if (side == "Upper" or side == "Both"):
1388 gcpy                        if (i < (Number_of_Pins - 1)):
1389 gcpy                            # print(i)
1390 gcpy                            toolpath = toolpath.union(self.cutvertexdxf
                                (self.xpos() + radius, self.ypos(),
                                jointdepth))
1391 gcpy                            toolpath = toolpath.union(self.cutvertexdxf
                                (self.xpos(), self.ypos() - jointwidth,
                                jointdepth))
1392 gcpy                            toolpath = toolpath.union(self.cutvertexdxf
                                (self.xpos() + radius/5, self.ypos(),
                                jointdepth))
1393 gcpy                            toolpath = toolpath.union(self.cutvertexdxf
                                (self.xpos(), self.ypos() + jointwidth,
                                jointdepth))
1394 gcpy                            toolpath = toolpath.union(self.cutvertexdxf
                                (self.xpos() + radius, self.ypos(),
                                jointdepth))

1395 gcpy                            i += 1
1396 gcpy                    self.closepolyline(self.currenttoolnumber())
1397 gcpy                    return toolpath
1398 gcpy
1399 gcpy                    def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
                                side, width, thickness, Number_of_Pins, largeVdiameter,
                                smallDiameter):
1400 gcpy                        rapid = self.rapidZ(0)
1401 gcpy                        # rapid = rapid.union(self.rapidXY(bx, by))
1402 gcpy                        self.setdxfcolor("Red")
1403 gcpy                        if (orientation == "Vertical"):
1404 gcpy                            rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
                                /6))
1405 gcpy                            toolpath = self.cutlineZgcfeed(-thickness,1000)
1406 gcpy                            toolpath = self.cutlinedxfgc(bx, by + width +
                                smallDiameter/6, - thickness)
1407 gcpy                        if (orientation == "Horizontal"):
1408 gcpy                            rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
                                by))
1409 gcpy                            toolpath = self.cutlineZgcfeed(-thickness,1000)
1410 gcpy                            toolpath = self.cutlinedxfgc(bx + width - smallDiameter
                                /6, by, -thickness)
1411 gcpy                        # rapid = self.rapidZ(0)
1412 gcpy                        return toolpath
1413 gcpy
1414 gcpy                    def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
                                side, width, thickness, Number_of_Pins, largeVdiameter,
                                smallDiameter):
1415 gcpy                        radius = smallDiameter/2
1416 gcpy                        rapid = self.rapidZ(0)
1417 gcpy                        # rapid = rapid.union(self.rapidXY(bx, by))
1418 gcpy                        # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                                "Upper"
1419 gcpy                        # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                                Right"
1420 gcpy                        if (orientation == "Vertical"):
1421 gcpy                            rapid = rapid.union(self.rapidXY(bx, by))
1422 gcpy                            toolpath = self.cutlineZgcfeed(-thickness,1000)
1423 gcpy                            toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                                thickness, -thickness))
1424 gcpy                            rapid = self.rapidZ(0)
1425 gcpy                            rapid = rapid.union(self.rapidXY(bx, by + width -
                                thickness))
1426 gcpy                            self.setdxfcolor("Blue")
1427 gcpy                            toolpath = toolpath.union(self.cutlineZgcfeed(-
                                thickness,1000))

```

```

1428 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
1429 gcpy          width, -thickness))
1430 gcpy          if (side == "Left" or side == "Both"):
1431 gcpy              rapid = self.rapidZ(0)
1432 gcpy              self.setdxfc("DarkGray")
1433 gcpy              rapid = rapid.union(self.rapidXY(bx+thickness-(
1434 gcpy                  smallDiameter / 2) / math.tan(math.radians(45)),
1435 gcpy                  by - radius/2))
1436 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-(
1437 gcpy                  smallDiameter / 2) / math.tan(math.radians(45))
1438 gcpy                  ,10000))
1439 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx+
1440 gcpy                  thickness-(smallDiameter / 2) / math.tan(math.
1441 gcpy                      radians(45)), by + width + radius/2, -(
1442 gcpy                          smallDiameter / 2) / math.tan(math.radians(45)))
1443 gcpy                  )
1444 gcpy              rapid = self.rapidZ(0)
1445 gcpy              self.setdxfc("Green")
1446 gcpy              rapid = rapid.union(self.rapidXY(bx+thickness/2, by
1447 gcpy                  +width))
1448 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
1449 gcpy                  thickness/2,1000))
1450 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx+
1451 gcpy                  thickness/2, by +thickness, -thickness/2))
1452 gcpy          if (side == "Right" or side == "Both"):
1453 gcpy              rapid = self.rapidZ(0)
1454 gcpy              self.setdxfc("DarkGray")
1455 gcpy              rapid = rapid.union(self.rapidXY(bx-(thickness-(
1456 gcpy                  smallDiameter / 2) / math.tan(math.radians(45)))
1457 gcpy                  , by - radius/2))
1458 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-(
1459 gcpy                  smallDiameter / 2) / math.tan(math.radians(45))
1460 gcpy                  ,10000))
1461 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx-(
1462 gcpy                  thickness-(smallDiameter / 2) / math.tan(math.
1463 gcpy                      radians(45))), by + width + radius/2, -(
1464 gcpy                          smallDiameter / 2) / math.tan(math.radians(45)))
1465 gcpy                  )
1466 gcpy              rapid = self.rapidZ(0)
1467 gcpy              self.setdxfc("Green")
1468 gcpy              rapid = rapid.union(self.rapidXY(bx-thickness/2, by
1469 gcpy                  +width))
1470 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
1471 gcpy                  thickness/2,1000))
1472 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx-
1473 gcpy                  thickness/2, by + width -thickness, -thickness
1474 gcpy                      /2))
1475 gcpy              rapid = self.rapidZ(0)
1476 gcpy              rapid = rapid.union(self.rapidXY(bx-thickness/2, by
1477 gcpy                  ))
1478 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
1479 gcpy                  thickness/2,1000))
1480 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx-
1481 gcpy                  thickness/2, by +thickness, -thickness/2))
1482 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
1483 gcpy          "Upper"
1484 gcpy          if (orientation == "Horizontal"):
1485 gcpy              rapid = rapid.union(self.rapidXY(bx, by))
1486 gcpy              self.setdxfc("Blue")
1487 gcpy              toolpath = self.cutlineZgcfeed(-thickness,1000)
1488 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx +
1489 gcpy                  thickness, by, -thickness))
1490 gcpy              rapid = rapid.union(self.rapidZ(0))
1491 gcpy              rapid = rapid.union(self.rapidXY(bx + width - thickness
1492 gcpy                  , by))
1493 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
1494 gcpy                  thickness,1000))
1495 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx + width,
1496 gcpy                  by, -thickness))
1497 gcpy          if (side == "Lower" or side == "Both"):

```

```
1470 gcpy          rapid = self.rapidZ(0)
1471 gcpy          self.setdxfcolor("Dark_Gray")
1472 gcpy          rapid = rapid.union(self.rapidXY(bx - radius, by+
            thickness-(smallDiameter / 2) / math.tan(math.
            radians(45))))
1473 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-(
            smallDiameter / 2) / math.tan(math.radians(45))
            ,10000))
1474 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            width + radius, by+thickness-(smallDiameter / 2)
            / math.tan(math.radians(45)), -(smallDiameter /
            2) / math.tan(math.radians(45))))
1475 gcpy          rapid = self.rapidZ(0)
1476 gcpy          self.setdxfcolor("Green")
1477 gcpy          rapid = rapid.union(self.rapidXY(bx+width, by+
            thickness/2))
1478 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
            thickness/2,1000))
1479 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            width -thickness, by+thickness/2, -thickness/2))
1480 gcpy          rapid = self.rapidZ(0)
1481 gcpy          rapid = rapid.union(self.rapidXY(bx, by+thickness
            /2))
1482 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
            thickness/2,1000))
1483 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            thickness, by+thickness/2, -thickness/2))
1484 gcpy          if (side == "Upper" or side == "Both"):
1485 gcpy              rapid = self.rapidZ(0)
1486 gcpy              self.setdxfcolor("Dark_Gray")
1487 gcpy              rapid = rapid.union(self.rapidXY(bx - radius, by-(
            thickness-(smallDiameter / 2) / math.tan(math.
            radians(45))))))
1488 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-(
            smallDiameter / 2) / math.tan(math.radians(45))
            ,10000))
1489 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            width + radius, by-(thickness-(smallDiameter /
            2) / math.tan(math.radians(45))), -(
            smallDiameter / 2) / math.tan(math.radians(45)))
            )
1490 gcpy          rapid = self.rapidZ(0)
1491 gcpy          self.setdxfcolor("Green")
1492 gcpy          rapid = rapid.union(self.rapidXY(bx+width, by-
            thickness/2))
1493 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
            thickness/2,1000))
1494 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            width -thickness, by-thickness/2, -thickness/2))
1495 gcpy          rapid = self.rapidZ(0)
1496 gcpy          rapid = rapid.union(self.rapidXY(bx, by-thickness
            /2))
1497 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
            thickness/2,1000))
1498 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
            thickness, by-thickness/2, -thickness/2))
1499 gcpy          rapid = self.rapidZ(0)
1500 gcpy          return toolpath
```

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 by each operation, causing it to be instantiated on the 3D stage unless the Boolean generatepaths is True.

```
1425 gcpy          def stockandtoolpaths(self, option = "stockandtoolpaths"):
1426 gcpy              if option == "stock":
1427 gcpy                  if self.generatepaths == False:
1428 gcpy                      show(self.stock)
1429 gcpy                  # print("Outputting stock")
1430 gcpy              else:
1431 gcpy                  return self.stock
1432 gcpy              elif option == "toolpaths":
1433 gcpy                  if self.generatepaths == False:
1434 gcpy                      show(self.toolpaths)
1435 gcpy              else:
1436 gcpy                  return self.toolpaths
```



```
1437 gcpy          elif option == "rapids":
1438 gcpy              if self.generatepaths == False:
1439 gcpy                  show(self.rapids)
1440 gcpy              else:
1441 gcpy                  return self.rapids
1442 gcpy          else:
1443 gcpy              part = self.stock.difference(self.toolpaths)
1444 gcpy              if self.generatepaths == False:
1445 gcpy                  show(part)
1446 gcpy              else:
1447 gcpy                  return part
```

It is convenient to have specific commands reflecting the possible options:

```
111 gcpscad module stockandtoolpaths(){
112 gcpscad     gcp.stockandtoolpaths();
113 gcpscad }
114 gcpscad
115 gcpscad module stockwotoolpaths(){
116 gcpscad     gcp.stockandtoolpaths("stock");
117 gcpscad }
118 gcpscad
119 gcpscad module outputtoolpaths(){
120 gcpscad     gcp.stockandtoolpaths("toolpaths");
121 gcpscad }
122 gcpscad
123 gcpscad module outputrapids(){
124 gcpscad     gcp.stockandtoolpaths("rapids");
125 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) G53G0Z-5.000
toolchange(...);	(TOOL/MILL, 3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
cutoneaxis_setfeed(...);	(PREPOSITION FOR RAPID PLUNGE) G0X0Y0 Z0.25 G1Z0F100 G1 X109.5 Y75 Z-8.35F400 Z9
cutwithfeed(...);	
closegcodefile();	M05 M02

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

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

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

A further consideration is that rather than hard-coding all possibilities or any changes, having an option for a "post-processor" will be far more flexible.

Described at: <https://carbide3d.com/hub/faq/create-pro-custom-post-processor/> the necessary hooks would be:

- onOpen
- onClose
- onSection (which is where tool changes are defined, since "section" in this case is segmented per tool)

3.5.2 DXF Overview

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

```
0
SECTION
2
ENTITIES
0
LWPOLYLINE
90
2
70
0
43
0
10
-31.375
20
-34.9152
10
-31.375
20
-18.75
0
ARC
10
-54.75
```

```

20
-37.5
40
4
50
0
51
90
0
ENDSEC
0
EOF

```

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.

```

1449 gcpy      def writegc(self, *arguments):
1450 gcpy          if self.generategcode == True:
1451 gcpy              line_to_write = ""
1452 gcpy              for element in arguments:
1453 gcpy                  line_to_write += element
1454 gcpy              self.gc.write(line_to_write)
1455 gcpy              self.gc.write("\n")
1456 gcpy
1457 gcpy      def writedxf(self, toolnumber, *arguments):
1458 gcpy #          global dxfclosed
1459 gcpy          line_to_write = ""
1460 gcpy          for element in arguments:
1461 gcpy              line_to_write += element
1462 gcpy          if self.generateddxf == True:
1463 gcpy              if self.dxfclosed == False:
1464 gcpy                  self.dxf.write(line_to_write)
1465 gcpy                  self.dxf.write("\n")
1466 gcpy          if self.generateddxfs == True:
1467 gcpy              self.writedxfs(toolnumber, line_to_write)
1468 gcpy
1469 gcpy      def writedxfs(self, toolnumber, line_to_write):
1470 gcpy #          print("Processing writing toolnumber", toolnumber)
1471 gcpy #          line_to_write = ""
1472 gcpy #          for element in arguments:
1473 gcpy #              line_to_write += element
1474 gcpy          if (toolnumber == 0):
1475 gcpy              return
1476 gcpy          elif self.generateddxfs == True:
1477 gcpy              if (self.large_square_tool_num == toolnumber):
1478 gcpy                  self.dxf_lgsq.write(line_to_write)
1479 gcpy                  self.dxf_lgsq.write("\n")
1480 gcpy              if (self.small_square_tool_num == toolnumber):
1481 gcpy                  self.dxf_smsq.write(line_to_write)
1482 gcpy                  self.dxf_smsq.write("\n")
1483 gcpy              if (self.large_ball_tool_num == toolnumber):
1484 gcpy                  self.dxf_lgbl.write(line_to_write)
1485 gcpy                  self.dxf_lgbl.write("\n")
1486 gcpy              if (self.small_ball_tool_num == toolnumber):
1487 gcpy                  self.dxf_smb1.write(line_to_write)
1488 gcpy                  self.dxf_smb1.write("\n")
1489 gcpy              if (self.large_V_tool_num == toolnumber):
1490 gcpy                  self.dxf_lgV.write(line_to_write)
1491 gcpy                  self.dxf_lgV.write("\n")
1492 gcpy              if (self.small_V_tool_num == toolnumber):
1493 gcpy                  self.dxf_smV.write(line_to_write)
1494 gcpy                  self.dxf_smV.write("\n")
1495 gcpy              if (self.DT_tool_num == toolnumber):
1496 gcpy                  self.dxfDT.write(line_to_write)
1497 gcpy                  self.dxfDT.write("\n")
1498 gcpy              if (self.KH_tool_num == toolnumber):
1499 gcpy                  self.dxfKH.write(line_to_write)
1500 gcpy                  self.dxfKH.write("\n")
1501 gcpy              if (self.Roundover_tool_num == toolnumber):
1502 gcpy                  self.dxfRt.write(line_to_write)
1503 gcpy                  self.dxfRt.write("\n")
1504 gcpy              if (self.MISC_tool_num == toolnumber):
1505 gcpy                  self.dxfMt.write(line_to_write)

```

```
1506 gcpy self.dxfMt.write("\n")
```

which commands will accept a series of arguments and then write them out to a file object for the appropriate file. Note that the DXF files for specific tools will expect that the tool numbers be set in the matching variables from the template. Further note that while it is possible to use tools which are not so defined, the toolpaths will not be written into DXF files for any tool numbers which do not match the variables from the template (but will appear in the main .dxf).

opengcodefile For writing to files it will be necessary to have commands for opening the files: opengcodefile
opendxfile and opendxfile which will set 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:

```
127 gcpscad module opendxfile(basefilename){
128 gcpscad     gcp.opendxfile(basefilename);
129 gcpscad }
130 gcpscad
131 gcpscad module opendxfiles(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) {
132 gcpscad     gcp.opendxfiles(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)
    ;
133 gcpscad }
```

opengcodefile With matching OpenSCAD commands: opengcodefile for OpenSCAD:

```
135 gcpscad module opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed) {
136 gcpscad     gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed);
137 gcpscad }
```

and Python:

```
1508 gcpy     def opengcodefile(self, basefilename = "export",
1509 gcpy         currenttoolnum = 102,
1510 gcpy         toolradius = 3.175,
1511 gcpy         plunge = 400,
1512 gcpy         feed = 1600,
1513 gcpy         speed = 10000
1514 gcpy         ):
1515 gcpy         self.basefilename = basefilename
1516 gcpy         self.currenttoolnum = currenttoolnum
1517 gcpy         self.toolradius = toolradius
1518 gcpy         self.plunge = plunge
1519 gcpy         self.feed = feed
1520 gcpy         self.speed = speed
1521 gcpy         if self.generategcode == True:
1522 gcpy             self.gcodefilename = basefilename + ".nc"
1523 gcpy             self.gc = open(self.gcodefilename, "w")
1524 gcpy
1525 gcpy     def opendxfile(self, basefilename = "export"):
1526 gcpy         self.basefilename = basefilename
1527 gcpy         # global generatedxfs
1528 gcpy         # global dxfclosed
1529 gcpy         self.dxfclosed = False
1530 gcpy         self.dxfcolor = "Black"
1531 gcpy         if self.generatedxif == True:
1532 gcpy             self.generatedxfs = False
1533 gcpy             self.dxffilename = basefilename + ".dxf"
1534 gcpy             self.dxf = open(self.dxffilename, "w")
1535 gcpy             self.dxfpreamble(-1)
1536 gcpy
1537 gcpy     def opendxfiles(self, basefilename = "export",
1538 gcpy         large_square_tool_num = 0,
1539 gcpy         small_square_tool_num = 0,
1540 gcpy         large_ball_tool_num = 0,
1541 gcpy         small_ball_tool_num = 0,
1542 gcpy         large_V_tool_num = 0,
1543 gcpy         small_V_tool_num = 0,
1544 gcpy         DT_tool_num = 0,
```

```
1545 gcpy                KH_tool_num = 0,
1546 gcpy                Roundover_tool_num = 0,
1547 gcpy                MISC_tool_num = 0):
1548 gcpy #                global generatedxfs
1549 gcpy                self.basefilename = basefilename
1550 gcpy                self.generatedxfs = True
1551 gcpy                self.large_square_tool_num = large_square_tool_num
1552 gcpy                self.small_square_tool_num = small_square_tool_num
1553 gcpy                self.large_ball_tool_num = large_ball_tool_num
1554 gcpy                self.small_ball_tool_num = small_ball_tool_num
1555 gcpy                self.large_V_tool_num = large_V_tool_num
1556 gcpy                self.small_V_tool_num = small_V_tool_num
1557 gcpy                self.DT_tool_num = DT_tool_num
1558 gcpy                self.KH_tool_num = KH_tool_num
1559 gcpy                self.Roundover_tool_num = Roundover_tool_num
1560 gcpy                self.MISC_tool_num = MISC_tool_num
1561 gcpy                if self.generatedxf == True:
1562 gcpy                    if (large_square_tool_num > 0):
1563 gcpy                        self.dxfllgsqfilename = basefilename + str(
                                large_square_tool_num) + ".dxf"
1564 gcpy #                        print("Opening ", str(self.dxfllgsqfilename))
1565 gcpy                        self.dxfllgsq = open(self.dxfllgsqfilename, "w")
1566 gcpy                    if (small_square_tool_num > 0):
1567 gcpy #                        print("Opening small square")
1568 gcpy                        self.dxfllmsqfilename = basefilename + str(
                                small_square_tool_num) + ".dxf"
1569 gcpy                        self.dxfllmsq = open(self.dxfllmsqfilename, "w")
1570 gcpy                    if (large_ball_tool_num > 0):
1571 gcpy #                        print("Opening large ball")
1572 gcpy                        self.dxfllgblfilename = basefilename + str(
                                large_ball_tool_num) + ".dxf"
1573 gcpy                        self.dxfllgbl = open(self.dxfllgblfilename, "w")
1574 gcpy                    if (small_ball_tool_num > 0):
1575 gcpy #                        print("Opening small ball")
1576 gcpy                        self.dxfllmblfilename = basefilename + str(
                                small_ball_tool_num) + ".dxf"
1577 gcpy                        self.dxfllmbl = open(self.dxfllmblfilename, "w")
1578 gcpy                    if (large_V_tool_num > 0):
1579 gcpy #                        print("Opening large V")
1580 gcpy                        self.dxfllgVfilename = basefilename + str(
                                large_V_tool_num) + ".dxf"
1581 gcpy                        self.dxfllgV = open(self.dxfllgVfilename, "w")
1582 gcpy                    if (small_V_tool_num > 0):
1583 gcpy #                        print("Opening small V")
1584 gcpy                        self.dxfllsmVfilename = basefilename + str(
                                small_V_tool_num) + ".dxf"
1585 gcpy                        self.dxfllsmV = open(self.dxfllsmVfilename, "w")
1586 gcpy                    if (DT_tool_num > 0):
1587 gcpy #                        print("Opening DT")
1588 gcpy                        self.dxfllDTfilename = basefilename + str(DT_tool_num
                                ) + ".dxf"
1589 gcpy                        self.dxfllDT = open(self.dxfllDTfilename, "w")
1590 gcpy                    if (KH_tool_num > 0):
1591 gcpy #                        print("Opening KH")
1592 gcpy                        self.dxfllKHfilename = basefilename + str(KH_tool_num
                                ) + ".dxf"
1593 gcpy                        self.dxfllKH = open(self.dxfllKHfilename, "w")
1594 gcpy                    if (Roundover_tool_num > 0):
1595 gcpy #                        print("Opening Rt")
1596 gcpy                        self.dxfllRtfilename = basefilename + str(
                                Roundover_tool_num) + ".dxf"
1597 gcpy                        self.dxfllRt = open(self.dxfllRtfilename, "w")
1598 gcpy                    if (MISC_tool_num > 0):
1599 gcpy #                        print("Opening Mt")
1600 gcpy                        self.dxfllMtfilename = basefilename + str(
                                MISC_tool_num) + ".dxf"
1601 gcpy                        self.dxfllMt = open(self.dxfllMtfilename, "w")
```

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

```
1602 gcpy                if (large_square_tool_num > 0):
1603 gcpy                    self.dxfpreamble(large_square_tool_num)
1604 gcpy                if (small_square_tool_num > 0):
1605 gcpy                    self.dxfpreamble(small_square_tool_num)
1606 gcpy                if (large_ball_tool_num > 0):
1607 gcpy                    self.dxfpreamble(large_ball_tool_num)
1608 gcpy                if (small_ball_tool_num > 0):
```

```
1609 gcpy            self.dxfpreamble(small_ball_tool_num)
1610 gcpy            if (large_V_tool_num > 0):
1611 gcpy                self.dxfpreamble(large_V_tool_num)
1612 gcpy            if (small_V_tool_num > 0):
1613 gcpy                self.dxfpreamble(small_V_tool_num)
1614 gcpy            if (DT_tool_num > 0):
1615 gcpy                self.dxfpreamble(DT_tool_num)
1616 gcpy            if (KH_tool_num > 0):
1617 gcpy                self.dxfpreamble(KH_tool_num)
1618 gcpy            if (Roundover_tool_num > 0):
1619 gcpy                self.dxfpreamble(Roundover_tool_num)
1620 gcpy            if (MISC_tool_num > 0):
1621 gcpy                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 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:

- writedxflgbl
- Ball nose, large (lgbl) writedxflgbl
- writedxfsmb1
- Ball nose, small (smb1) writedxfsmb1
- writedxflgsq
- Square, large (lgsq) writedxflgsq
- writedxfsmsq
- Square, small (smsq) writedxfsmsq
- writedxflgV
- V, large (lgV) writedxflgV
- writedxfsmV
- V, small (smV) writedxfsmV
- writedxfKH
- Keyhole (KH) writedxfKH
- writedxfDT
- Dovetail (DT) writedxfDT

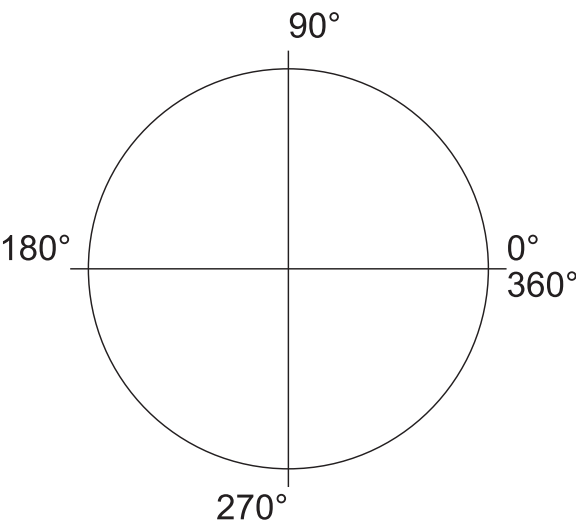
dxfpreamble This module requires that the tool number be passed in, and after writing out dxfpreamble, that value will be used to write out to the appropriate file with a series of if statements.

```
1623 gcpy    def dxfpreamble(self, tn):
1624 gcpy #        self.writedxf(tn, str(tn))
1625 gcpy        self.writedxf(tn, "0")
1626 gcpy        self.writedxf(tn, "SECTION")
1627 gcpy        self.writedxf(tn, "2")
1628 gcpy        self.writedxf(tn, "ENTITIES")
```

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

- dxfline
- a line dxfline
- beginpolyline
- connected lines beginpolyline/addvertex/closepolyline
- addvertex
- arc dxfarc
- closepolyline
- circle — a notable option would be for the arc to close on itself, creating a circle dxfcircle
- dxfcircle

DXF orders arcs counter-clockwise:



Note that arcs of greater than 90 degrees are not rendered accurately (in certain applications at least), 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:

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

The DXF file format supports colors defined by AutoCAD’s indexed color system:

Color Code	Color Name
0	Black (or Foreground)
1	Red
2	Yellow
3	Green
4	Cyan
5	Blue
6	Magenta
7	White (or Background)
8	Dark Gray
9	Light Gray

Color codes 10–255 represent additional colors, with hues varying based on RGB values. Obviously, a command to manage adding the colour commands would be:

```
1630 gcpy      def setdxffcolor(self, color):
1631 gcpy          self.dxffcolor = color
1632 gcpy
1633 gcpy      def writedxffcolor(self, tn):
1634 gcpy          self.writedxff(tn, "8")
1635 gcpy          if (self.dxffcolor == "Black"):
1636 gcpy              self.writedxff(tn, "Layer_Black")
1637 gcpy          if (self.dxffcolor == "Red"):
1638 gcpy              self.writedxff(tn, "Layer_Red")
1639 gcpy          if (self.dxffcolor == "Yellow"):
1640 gcpy              self.writedxff(tn, "Layer_Yellow")
1641 gcpy          if (self.dxffcolor == "Green"):
1642 gcpy              self.writedxff(tn, "Layer_Green")
1643 gcpy          if (self.dxffcolor == "Cyan"):
1644 gcpy              self.writedxff(tn, "Layer_Cyan")
1645 gcpy          if (self.dxffcolor == "Blue"):
1646 gcpy              self.writedxff(tn, "Layer_Blue")
1647 gcpy          if (self.dxffcolor == "Magenta"):
1648 gcpy              self.writedxff(tn, "Layer_Magenta")
1649 gcpy          if (self.dxffcolor == "White"):
1650 gcpy              self.writedxff(tn, "Layer_White")
1651 gcpy          if (self.dxffcolor == "Dark_Gray"):
1652 gcpy              self.writedxff(tn, "Layer_Dark_Gray")
1653 gcpy          if (self.dxffcolor == "Light_Gray"):
1654 gcpy              self.writedxff(tn, "Layer_Light_Gray")
1655 gcpy
1656 gcpy          self.writedxff(tn, "62")
1657 gcpy          if (self.dxffcolor == "Black"):
1658 gcpy              self.writedxff(tn, "0")
1659 gcpy          if (self.dxffcolor == "Red"):
1660 gcpy              self.writedxff(tn, "1")
1661 gcpy          if (self.dxffcolor == "Yellow"):
```

```
1662 gcpy                self.writedxf(tn, "2")
1663 gcpy                if (self.dxfcolor == "Green"):
1664 gcpy                self.writedxf(tn, "3")
1665 gcpy                if (self.dxfcolor == "Cyan"):
1666 gcpy                self.writedxf(tn, "4")
1667 gcpy                if (self.dxfcolor == "Blue"):
1668 gcpy                self.writedxf(tn, "5")
1669 gcpy                if (self.dxfcolor == "Magenta"):
1670 gcpy                self.writedxf(tn, "6")
1671 gcpy                if (self.dxfcolor == "White"):
1672 gcpy                self.writedxf(tn, "7")
1673 gcpy                if (self.dxfcolor == "Dark_Gray"):
1674 gcpy                self.writedxf(tn, "8")
1675 gcpy                if (self.dxfcolor == "Light_Gray"):
1676 gcpy                self.writedxf(tn, "9")

```

```
139 gpcpscad module setdxfcolor(color){
140 gpcpscad     gcp.setdxfcolor(color);
141 gpcpscad }
```

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 Keyhole

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.

```
1678 gcpy    def dxfline(self, tn, xbegin, ybegin, xend, yend):
1679 gcpy        self.writedxf(tn, "0")
1680 gcpy        self.writedxf(tn, "LINE")
1681 gcpy #
1682 gcpy        self.writedxfcolor(tn)
1683 gcpy #
1684 gcpy        self.writedxf(tn, "10")
1685 gcpy        self.writedxf(tn, str(xbegin))
1686 gcpy        self.writedxf(tn, "20")
1687 gcpy        self.writedxf(tn, str(ybegin))
1688 gcpy        self.writedxf(tn, "30")
1689 gcpy        self.writedxf(tn, "0.0")
1690 gcpy        self.writedxf(tn, "11")
1691 gcpy        self.writedxf(tn, str(xend))
1692 gcpy        self.writedxf(tn, "21")
1693 gcpy        self.writedxf(tn, str(yend))
1694 gcpy        self.writedxf(tn, "31")
1695 gcpy        self.writedxf(tn, "0.0")

```

In addition to dxfline which allows creating a line without consideration of context, there is also a dxfpolyline which will create a continuous/joined sequence of line segments which requires beginning it, adding vertexes, and then when done, ending the sequence. First, begin the polyline:

```
1697 gcpy    def beginpolyline(self, tn):#, xbegin, ybegin
1698 gcpy        self.writedxf(tn, "0")
1699 gcpy        self.writedxf(tn, "POLYLINE")
1700 gcpy        self.writedxf(tn, "8")
1701 gcpy        self.writedxf(tn, "default")
1702 gcpy        self.writedxf(tn, "66")
1703 gcpy        self.writedxf(tn, "1")
1704 gcpy #
1705 gcpy        self.writedxfcolor(tn)
1706 gcpy #
1707 gcpy #        self.writedxf(tn, "10")
1708 gcpy #        self.writedxf(tn, str(xbegin))
1709 gcpy #        self.writedxf(tn, "20")
1710 gcpy #        self.writedxf(tn, str(ybegin))
1711 gcpy #        self.writedxf(tn, "30")
1712 gcpy #        self.writedxf(tn, "0.0")

```

```
1713 gcpy          self.writedxf(tn, "70")
1714 gcpy          self.writedxf(tn, "0")
```

then add as many vertexes as are wanted:

```
1715 gcpy          def addvertex(self, tn, xend, yend):
1716 gcpy              self.writedxf(tn, "0")
1717 gcpy              self.writedxf(tn, "VERTEX")
1718 gcpy              self.writedxf(tn, "8")
1719 gcpy              self.writedxf(tn, "default")
1720 gcpy              self.writedxf(tn, "70")
1721 gcpy              self.writedxf(tn, "32")
1722 gcpy              self.writedxf(tn, "10")
1723 gcpy              self.writedxf(tn, str(xend))
1724 gcpy              self.writedxf(tn, "20")
1725 gcpy              self.writedxf(tn, str(yend))
1726 gcpy              self.writedxf(tn, "30")
1727 gcpy              self.writedxf(tn, "0.0")
```

then end the sequence:

```
1729 gcpy          def closepolyline(self, tn):
1730 gcpy              self.writedxf(tn, "0")
1731 gcpy              self.writedxf(tn, "SEQEND")
```

For arcs, 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 (G2 vs. G3).

```
1733 gcpy          def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
                             endangle):
1734 gcpy              if (self.generatedxf == True):
1735 gcpy                  self.writedxf(tn, "0")
1736 gcpy                  self.writedxf(tn, "ARC")
1737 gcpy #
1738 gcpy                  self.writedxfcolor(tn)
1739 gcpy #
1740 gcpy                  self.writedxf(tn, "10")
1741 gcpy                  self.writedxf(tn, str(xcenter))
1742 gcpy                  self.writedxf(tn, "20")
1743 gcpy                  self.writedxf(tn, str(ycenter))
1744 gcpy                  self.writedxf(tn, "40")
1745 gcpy                  self.writedxf(tn, str(radius))
1746 gcpy                  self.writedxf(tn, "50")
1747 gcpy                  self.writedxf(tn, str(anglebegin))
1748 gcpy                  self.writedxf(tn, "51")
1749 gcpy                  self.writedxf(tn, str(endangle))
1750 gcpy
1751 gcpy          def gcdearc(self, tn, xcenter, ycenter, radius, anglebegin,
                             endangle):
1752 gcpy              if (self.generategcode == True):
1753 gcpy                  self.writegc(tn, "(0)")
```

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.

```
1755 gcpy          def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1756 gcpy #              global toolpath
1757 gcpy #              toolpath = self.currenttool()
1758 gcpy #              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1759 gcpy              self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1760 gcpy              if (self.zpos == ez):
1761 gcpy                  self.settzpos(0)
1762 gcpy              else:
1763 gcpy                  self.settzpos((self.zpos()-ez)/90)
1764 gcpy #              self.setxpos(ex)
1765 gcpy #              self.setypos(ey)
1766 gcpy #              self.setzpos(ez)
1767 gcpy              if self.generatepaths == True:
1768 gcpy                  print("Unioning cutarcNECCdxf toolpath")
1769 gcpy                  self.arcloop(1, 90, xcenter, ycenter, radius)
1770 gcpy #                  self.toolpaths = self.toolpaths.union(toolpath)
1771 gcpy              else:
1772 gcpy                  toolpath = self.arcloop(1, 90, xcenter, ycenter, radius
```

```

        )
1773 gcpy #         print("Returning cutarcNECCdxf toolpath")
1774 gcpy         return toolpath
1775 gcpy
1776 gcpy     def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1777 gcpy #         global toolpath
1778 gcpy #         toolpath = self.currenttool()
1779 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1780 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 90, 180)
1781 gcpy         if (self.zpos == ez):
1782 gcpy             self.settzpos(0)
1783 gcpy         else:
1784 gcpy             self.settzpos((self.zpos()-ez)/90)
1785 gcpy #         self.setxpos(ex)
1786 gcpy #         self.setypos(ey)
1787 gcpy #         self.setzpos(ez)
1788 gcpy         if self.generatepaths == True:
1789 gcpy             self.arcloop(91, 180, xcenter, ycenter, radius)
1790 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1791 gcpy         else:
1792 gcpy             toolpath = self.arcloop(91, 180, xcenter, ycenter,
radius)
1793 gcpy             return toolpath
1794 gcpy
1795 gcpy     def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1796 gcpy #         global toolpath
1797 gcpy #         toolpath = self.currenttool()
1798 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1799 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1800 gcpy         if (self.zpos == ez):
1801 gcpy             self.settzpos(0)
1802 gcpy         else:
1803 gcpy             self.settzpos((self.zpos()-ez)/90)
1804 gcpy #         self.setxpos(ex)
1805 gcpy #         self.setypos(ey)
1806 gcpy #         self.setzpos(ez)
1807 gcpy         if self.generatepaths == True:
1808 gcpy             self.arcloop(181, 270, xcenter, ycenter, radius)
1809 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1810 gcpy         else:
1811 gcpy             toolpath = self.arcloop(181, 270, xcenter, ycenter,
radius)
1812 gcpy             return toolpath
1813 gcpy
1814 gcpy     def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1815 gcpy #         global toolpath
1816 gcpy #         toolpath = self.currenttool()
1817 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1818 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 270, 360)
1819 gcpy         if (self.zpos == ez):
1820 gcpy             self.settzpos(0)
1821 gcpy         else:
1822 gcpy             self.settzpos((self.zpos()-ez)/90)
1823 gcpy #         self.setxpos(ex)
1824 gcpy #         self.setypos(ey)
1825 gcpy #         self.setzpos(ez)
1826 gcpy         if self.generatepaths == True:
1827 gcpy             self.arcloop(271, 360, xcenter, ycenter, radius)
1828 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1829 gcpy         else:
1830 gcpy             toolpath = self.arcloop(271, 360, xcenter, ycenter,
radius)
1831 gcpy             return toolpath
1832 gcpy
1833 gcpy     def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1834 gcpy #         global toolpath
1835 gcpy #         toolpath = self.currenttool()
1836 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1837 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1838 gcpy         if (self.zpos == ez):

```

```

1839 gcpy             self.settzpos(0)
1840 gcpy             else:
1841 gcpy                 self.settzpos((self.zpos()-ez)/90)
1842 gcpy #             self.setxpos(ex)
1843 gcpy #             self.setypos(ey)
1844 gcpy #             self.setzpos(ez)
1845 gcpy             if self.generatepaths == True:
1846 gcpy                 self.narcloop(89, 0, xcenter, ycenter, radius)
1847 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1848 gcpy             else:
1849 gcpy                 toolpath = self.narcloop(89, 0, xcenter, ycenter,
1850 gcpy                                     radius)
1851 gcpy                 return toolpath
1852 gcpy             def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1853 gcpy #                 global toolpath
1854 gcpy #                 toolpath = self.currenttool()
1855 gcpy #                 toolpath = toolpath.translate([self.xpos(), self.ypos(),
1856 gcpy #                                     self.zpos()])
1857 gcpy                 self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1858 gcpy                                     radius, 270, 360)
1859 gcpy                 if (self.zpos == ez):
1860 gcpy                     self.settzpos(0)
1861 gcpy                 else:
1862 gcpy                     self.settzpos((self.zpos()-ez)/90)
1863 gcpy #                 self.setxpos(ex)
1864 gcpy #                 self.setypos(ey)
1865 gcpy #                 self.setzpos(ez)
1866 gcpy                 if self.generatepaths == True:
1867 gcpy                     self.narcloop(359, 270, xcenter, ycenter, radius)
1868 gcpy #                     self.toolpaths = self.toolpaths.union(toolpath)
1869 gcpy                 else:
1870 gcpy                     toolpath = self.narcloop(359, 270, xcenter, ycenter,
1871 gcpy                                     radius)
1872 gcpy                     return toolpath
1873 gcpy             def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1874 gcpy #                 global toolpath
1875 gcpy #                 toolpath = self.currenttool()
1876 gcpy #                 toolpath = toolpath.translate([self.xpos(), self.ypos(),
1877 gcpy #                                     self.zpos()])
1878 gcpy                 self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1879 gcpy                                     radius, 180, 270)
1880 gcpy                 if (self.zpos == ez):
1881 gcpy                     self.settzpos(0)
1882 gcpy                 else:
1883 gcpy                     self.settzpos((self.zpos()-ez)/90)
1884 gcpy #                 self.setxpos(ex)
1885 gcpy #                 self.setypos(ey)
1886 gcpy #                 self.setzpos(ez)
1887 gcpy                 if self.generatepaths == True:
1888 gcpy                     self.narcloop(269, 180, xcenter, ycenter, radius)
1889 gcpy #                     self.toolpaths = self.toolpaths.union(toolpath)
1890 gcpy                 else:
1891 gcpy                     toolpath = self.narcloop(269, 180, xcenter, ycenter,
1892 gcpy                                     radius)
1893 gcpy                     return toolpath
1894 gcpy             def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1895 gcpy #                 global toolpath
1896 gcpy #                 toolpath = self.currenttool()
1897 gcpy #                 toolpath = toolpath.translate([self.xpos(), self.ypos(),
1898 gcpy #                                     self.zpos()])
1899 gcpy                 self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1900 gcpy                                     radius, 90, 180)
1901 gcpy                 if (self.zpos == ez):
1902 gcpy                     self.settzpos(0)
1903 gcpy                 else:
1904 gcpy                     self.settzpos((self.zpos()-ez)/90)
1905 gcpy #                 self.setxpos(ex)
1906 gcpy #                 self.setypos(ey)
1907 gcpy #                 self.setzpos(ez)
1908 gcpy                 if self.generatepaths == True:
1909 gcpy                     self.narcloop(179, 90, xcenter, ycenter, radius)
1910 gcpy #                     self.toolpaths = self.toolpaths.union(toolpath)
1911 gcpy                 else:
1912 gcpy                     toolpath = self.narcloop(179, 90, xcenter, ycenter,
1913 gcpy                                     radius)

```

```
1907 gcpy                                return toolpath
```

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

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

```
1909 gcpy      def arcCCGc(self, ex, ey, ez, xcenter, ycenter, radius):
1910 gcpy          self.writegc("G03_X", str(ex), "_Y", str(ey), "_Z", str(ez)
                  , "_R", str(radius))

1911 gcpy
1912 gcpy      def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1913 gcpy          self.writegc("G02_X", str(ex), "_Y", str(ey), "_Z", str(ez)
                  , "_R", str(radius))
```

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

```
1915 gcpy      def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1916 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1917 gcpy          if self.generatepaths == True:
1918 gcpy              self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
                  )
1919 gcpy          else:
1920 gcpy              return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1921 gcpy
1922 gcpy      def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1923 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1924 gcpy          if self.generatepaths == False:
1925 gcpy              return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1926 gcpy
1927 gcpy      def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1928 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1929 gcpy          if self.generatepaths == False:
1930 gcpy              return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1931 gcpy
1932 gcpy      def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1933 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1934 gcpy          if self.generatepaths == False:
1935 gcpy              return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1936 gcpy
1937 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1938 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1939 gcpy          if self.generatepaths == False:
1940 gcpy              return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1941 gcpy
1942 gcpy      def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1943 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1944 gcpy          if self.generatepaths == False:
1945 gcpy              return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1946 gcpy
1947 gcpy      def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1948 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1949 gcpy          if self.generatepaths == False:
1950 gcpy              return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
                  radius)

1951 gcpy
1952 gcpy      def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                  :
1953 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1954 gcpy          if self.generatepaths == False:
1955 gcpy              return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
                  radius)
```

```
143 gpcscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
144 gpcscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
145 gpcscad }
146 gpcscad
147 gpcscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
148 gpcscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
149 gpcscad }
150 gpcscad
151 gpcscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
152 gpcscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
153 gpcscad }
154 gpcscad
155 gpcscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
156 gpcscad     gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
157 gpcscad }
```

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 `closedxfile`. In some instances it may be necessary to write `closedxfile` additional information, depending on the file format. Note that these commands will need to be within the `gcodepreview` class.

```
1957 gcpy      def dxfpreamble(self, tn):
1958 gcpy #          self.writedxf(tn, str(tn))
1959 gcpy          self.writedxf(tn, "0")
1960 gcpy          self.writedxf(tn, "ENDSEC")
1961 gcpy          self.writedxf(tn, "0")
1962 gcpy          self.writedxf(tn, "EOF")

1964 gcpy      def gcodepreamble(self):
1965 gcpy          self.writegc("Z12.700")
1966 gcpy          self.writegc("M05")
1967 gcpy          self.writegc("M02")
```

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

```
1969 gcpy      def closegcodefile(self):
1970 gcpy          if self.generategcode == True:
1971 gcpy              self.gcodepreamble()
1972 gcpy              self.gc.close()
1973 gcpy
1974 gcpy      def closedxfile(self):
1975 gcpy          if self.generatedxfile == True:
1976 gcpy #              global dxfclose
1977 gcpy              self.dxfpreamble(-1)
1978 gcpy #              self.dxfclosed = True
1979 gcpy              self.dxf.close()
1980 gcpy
1981 gcpy      def closedxfiles(self):
1982 gcpy          if self.generatedxfs == True:
1983 gcpy              if (self.large_square_tool_num > 0):
1984 gcpy                  self.dxfpreamble(self.large_square_tool_num)
1985 gcpy              if (self.small_square_tool_num > 0):
1986 gcpy                  self.dxfpreamble(self.small_square_tool_num)
1987 gcpy              if (self.large_ball_tool_num > 0):
1988 gcpy                  self.dxfpreamble(self.large_ball_tool_num)
1989 gcpy              if (self.small_ball_tool_num > 0):
1990 gcpy                  self.dxfpreamble(self.small_ball_tool_num)
1991 gcpy              if (self.large_V_tool_num > 0):
1992 gcpy                  self.dxfpreamble(self.large_V_tool_num)
1993 gcpy              if (self.small_V_tool_num > 0):
1994 gcpy                  self.dxfpreamble(self.small_V_tool_num)
1995 gcpy              if (self.DT_tool_num > 0):
1996 gcpy                  self.dxfpreamble(self.DT_tool_num)
1997 gcpy              if (self.KH_tool_num > 0):
1998 gcpy                  self.dxfpreamble(self.KH_tool_num)
1999 gcpy              if (self.Roundover_tool_num > 0):
2000 gcpy                  self.dxfpreamble(self.Roundover_tool_num)
2001 gcpy              if (self.MISC_tool_num > 0):
2002 gcpy                  self.dxfpreamble(self.MISC_tool_num)
2003 gcpy
2004 gcpy              if (self.large_square_tool_num > 0):
2005 gcpy                  self.dxf.close()
```

```
2006 gcpy          if (self.small_square_tool_num > 0):
2007 gcpy              self.dxfsmsq.close()
2008 gcpy          if (self.large_ball_tool_num > 0):
2009 gcpy              self.dxfldb.close()
2010 gcpy          if (self.small_ball_tool_num > 0):
2011 gcpy              self.dxfdb.close()
2012 gcpy          if (self.large_V_tool_num > 0):
2013 gcpy              self.dxfdbV.close()
2014 gcpy          if (self.small_V_tool_num > 0):
2015 gcpy              self.dxfdbV.close()
2016 gcpy          if (self.DT_tool_num > 0):
2017 gcpy              self.dxfDT.close()
2018 gcpy          if (self.KH_tool_num > 0):
2019 gcpy              self.dxfKH.close()
2020 gcpy          if (self.Roundover_tool_num > 0):
2021 gcpy              self.dxfRt.close()
2022 gcpy          if (self.MISC_tool_num > 0):
2023 gcpy              self.dxfMt.close()
```

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

```
159 gcpyscad module closegcodefile(){
160 gcpyscad     gcp.closegcodefile();
161 gcpyscad }
162 gcpyscad
163 gcpyscad module closedxfiles(){
164 gcpyscad     gcp.closedxfiles();
165 gcpyscad }
166 gcpyscad
167 gcpyscad module closedxfile(){
168 gcpyscad     gcp.closedxfile();
169 gcpyscad }
```

Input Files

With all other features in place, it becomes possible to read in a G-code file and then create a 3D preview of how it will cut.
First, a template file will be necessary:

```
1 gcpgcpy from openscad import *
2 gcpgcpy #nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview/
           refs/heads/main/gcodepreview.py")
3 gcpgcpy
4 gcpgcpy from gcodepreview import *
5 gcpgcpy
6 gcpgcpy gc_file = "filename_of_G-code_file_to_process.nc"
7 gcpgcpy
8 gcpgcpy gcp = gcodepreview(True, False, False)
9 gcpgcpy
10 gcpgcpy gcp.previewgcodefile(gc_file)
```

previewgcodefile Which simply needs to call the previewgcodefile command:

```
2025 gcpy          def previewgcodefile(self, gc_file):
2026 gcpy              gc_file = open(gc_file, 'r')
2027 gcpy              gcfilecontents = []
2028 gcpy              with gc_file as file:
2029 gcpy                  for line in file:
2030 gcpy                      command = line
2031 gcpy                      gcfilecontents.append(line)
2032 gcpy
2033 gcpy              numlinesfound = 0
2034 gcpy              for line in gcfilecontents:
2035 gcpy                  print(line)
2036 gcpy                  if line[:10] == "(stockMin:":
2037 gcpy                      subdivisions = line.split()
2038 gcpy                      extentleft = float(subdivisions[0][10:-3])
2039 gcpy                      extentfb = float(subdivisions[1][: -3])
2040 gcpy                      extentd = float(subdivisions[2][: -3])
2041 gcpy                      numlinesfound = numlinesfound + 1
2042 gcpy                  if line[:13] == "(STOCK/BLOCK,":
2043 gcpy                      subdivisions = line.split()
```

```

2044 gcpy          sizeX = float(subdivisions[0][13:-1])
2045 gcpy          sizeY = float(subdivisions[1][:-1])
2046 gcpy          sizeZ = float(subdivisions[4][:-1])
2047 gcpy          numlinesfound = numlinesfound + 1
2048 gcpy          if line[:3] == "G21":
2049 gcpy              units = "mm"
2050 gcpy              numlinesfound = numlinesfound + 1
2051 gcpy          if numlinesfound >=3:
2052 gcpy              break
2053 gcpy #          print(numlinesfound)
2054 gcpy
2055 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
                extentfb, extentd)

2056 gcpy
2057 gcpy          commands = []
2058 gcpy          for line in gcfilecontents:
2059 gcpy              Xc = 0
2060 gcpy              Yc = 0
2061 gcpy              Zc = 0
2062 gcpy              Fc = 0
2063 gcpy              Xp = 0.0
2064 gcpy              Yp = 0.0
2065 gcpy              Zp = 0.0
2066 gcpy              if line == "G53G0Z-5.000\n":
2067 gcpy                  self.movetosafeZ()
2068 gcpy              if line[:3] == "M6T":
2069 gcpy                  tool = int(line[3:])
2070 gcpy                  self.toolchange(tool)
2071 gcpy              if line[:2] == "G0":
2072 gcpy                  machinestate = "rapid"
2073 gcpy              if line[:2] == "G1":
2074 gcpy                  machinestate = "cutline"
2075 gcpy              if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
                "X" or line[:1] == "Y" or line[:1] == "Z":
2076 gcpy                  if "F" in line:
2077 gcpy                      Fplus = line.split("F")
2078 gcpy                      Fc = 1
2079 gcpy                      fr = float(Fplus[1])
2080 gcpy                      line = Fplus[0]
2081 gcpy                  if "Z" in line:
2082 gcpy                      Zplus = line.split("Z")
2083 gcpy                      Zc = 1
2084 gcpy                      Zp = float(Zplus[1])
2085 gcpy                      line = Zplus[0]
2086 gcpy                  if "Y" in line:
2087 gcpy                      Yplus = line.split("Y")
2088 gcpy                      Yc = 1
2089 gcpy                      Yp = float(Yplus[1])
2090 gcpy                      line = Yplus[0]
2091 gcpy                  if "X" in line:
2092 gcpy                      Xplus = line.split("X")
2093 gcpy                      Xc = 1
2094 gcpy                      Xp = float(Xplus[1])
2095 gcpy              if Zc == 1:
2096 gcpy                  if Yc == 1:
2097 gcpy                      if Xc == 1:
2098 gcpy                          if machinestate == "rapid":
2099 gcpy                              command = "rapidXYZ(" + str(Xp) + "
                    ,\u" + str(Yp) + ",\u" + str(Zp) +
                    ")"
2100 gcpy                              self.rapidXYZ(Xp, Yp, Zp)
2101 gcpy                          else:
2102 gcpy                              command = "cutlineXYZ(" + str(Xp) +
                    ",\u" + str(Yp) + ",\u" + str(Zp) +
                    + ")"
2103 gcpy                              self.cutlineXYZ(Xp, Yp, Zp)
2104 gcpy                  else:
2105 gcpy                      if machinestate == "rapid":
2106 gcpy                          command = "rapidYZ(" + str(Yp) + ",
                    \u" + str(Zp) + ")"
2107 gcpy                          self.rapidYZ(Yp, Zp)
2108 gcpy                      else:
2109 gcpy                          command = "cutlineYZ(" + str(Yp) +
                    ",\u" + str(Zp) + ")"
2110 gcpy                          self.cutlineYZ(Yp, Zp)
2111 gcpy                  else:
2112 gcpy                      if Xc == 1:
2113 gcpy                          if machinestate == "rapid":

```

```
2114 gcpy                                command = "rapidXZ(" + str(Xp) + ",
                                         " + str(Zp) + ")"
2115 gcpy                                self.rapidXZ(Xp, Zp)
2116 gcpy                                else:
2117 gcpy                                command = "cutlineXZ(" + str(Xp) +
                                         ", " + str(Zp) + ")"
                                         self.cutlineXZ(Xp, Zp)
2118 gcpy
2119 gcpy                                else:
2120 gcpy                                if machinestate == "rapid":
2121 gcpy                                    command = "rapidZ(" + str(Zp) + ")"
2122 gcpy                                    self.rapidZ(Zp)
2123 gcpy                                else:
2124 gcpy                                    command = "cutlineZ(" + str(Zp) + "
                                         )"
                                         self.cutlineZ(Zp)
2125 gcpy
2126 gcpy                                else:
2127 gcpy                                    if Yc == 1:
2128 gcpy                                        if Xc == 1:
2129 gcpy                                            if machinestate == "rapid":
2130 gcpy                                                command = "rapidXY(" + str(Xp) + ",
                                         " + str(Yp) + ")"
                                         self.rapidXY(Xp, Yp)
2131 gcpy                                else:
2132 gcpy                                    command = "cutlineXY(" + str(Xp) +
                                         ", " + str(Yp) + ")"
                                         self.cutlineXY(Xp, Yp)
2133 gcpy
2134 gcpy                                else:
2135 gcpy                                    if machinestate == "rapid":
2136 gcpy                                        command = "rapidY(" + str(Yp) + ")"
2137 gcpy                                        self.rapidY(Yp)
2138 gcpy                                else:
2139 gcpy                                    command = "cutlineY(" + str(Yp) + "
                                         )"
                                         self.cutlineY(Yp)
2140 gcpy
2141 gcpy                                else:
2142 gcpy                                    if Xc == 1:
2143 gcpy                                        if machinestate == "rapid":
2144 gcpy                                            command = "rapidX(" + str(Xp) + ")"
2145 gcpy                                            self.rapidX(Xp)
2146 gcpy                                else:
2147 gcpy                                    command = "cutlineX(" + str(Xp) + "
                                         )"
                                         self.cutlineX(Xp)
2148 gcpy                                commands.append(command)
2149 gcpy                                print(line)
2150 gcpy                                print(command)
2151 gcpy #                                print(machinestate, Xc, Yc, Zc)
2152 gcpy #                                print(Xp, Yp, Zp)
2153 gcpy #                                print("/n")
2154 gcpy
2155 gcpy                                for command in commands:
2156 gcpy #                                print(command)
2157 gcpy
2158 gcpy                                show(self.stockandtoolpaths())
2159 gcpy
2160 gcpy
```

Future considerations:

- Multiple Preview Modes:
- Fast Preview: Write all movements with both begin and end positions into a list for a specific tool — as this is done, check for a previous movement between those positions and compare depths and tool number — keep only the deepest movement for a given tool.
- Motion Preview: Work up a 3D model of the machine and actually show the stock in relation to it,

4 Notes

Other Resources

Coding Style

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

Red Flags to avoid include:

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

Coding References

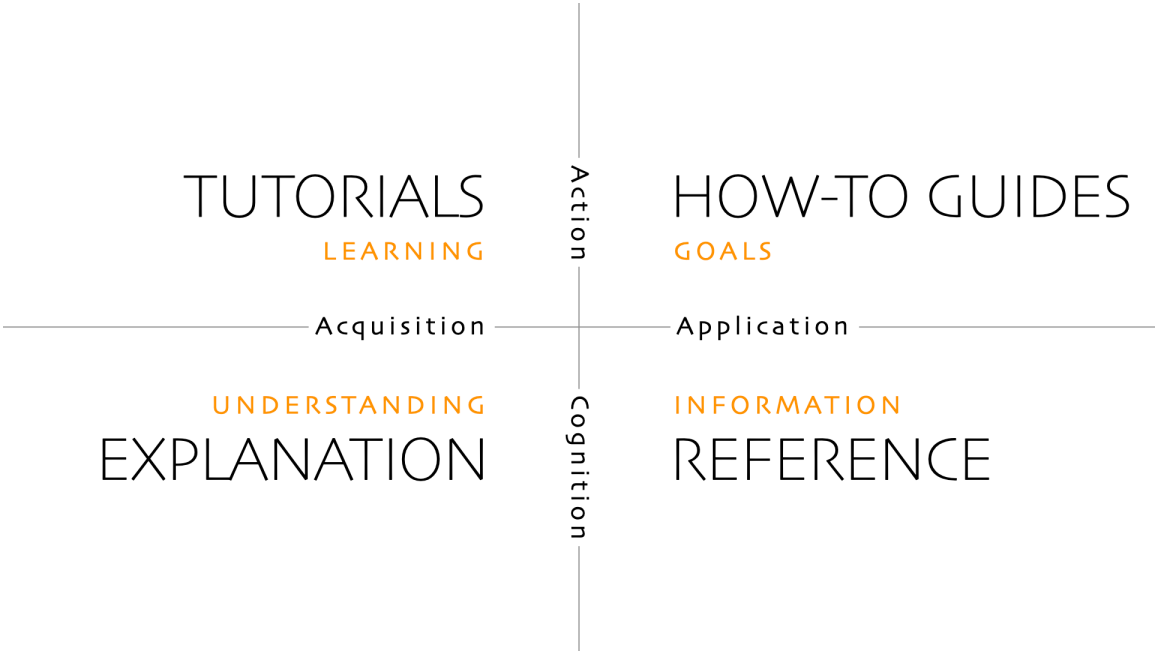
<https://thewhitetulip.gitbook.io/py/06-file-handling>

Documentation Style

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

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

resulting in a matrix of:



where:

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

Straddling the boundary between coding and documenation are docstrings and general coding style with the latter discussed at: <https://peps.python.org/pep-0008/>

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?

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:

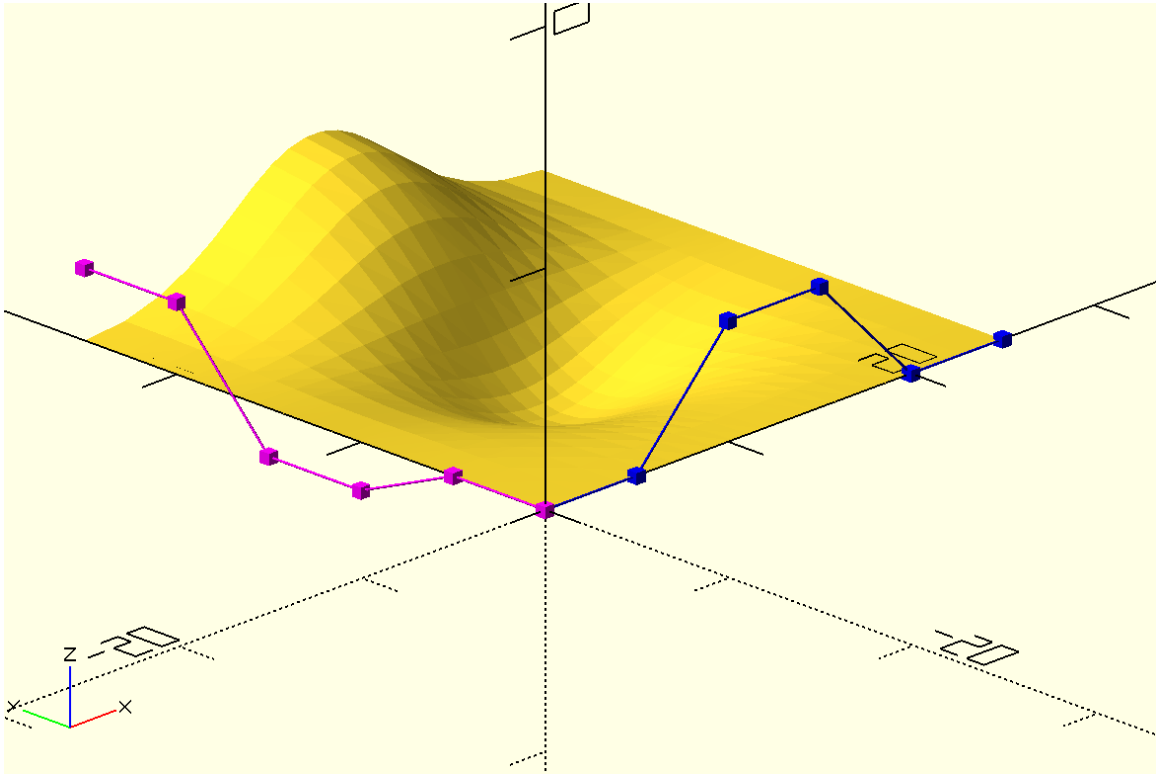
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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/

Mathematics

<https://elementsofprogramming.com/>

References

[ConstGeom]	Walmsley, Brian. <i>Construction Geometry</i> . 2d ed., Centennial College Press, 1981.
[MkCalc]	Horvath, Joan, and Rich Cameron. <i>Make: Calculus: Build models to learn, visualize, and explore</i> . First edition., Make: Community LLC, 2022.
[MkGeom]	Horvath, Joan, and Rich Cameron. <i>Make: Geometry: Learn by 3D Printing, Coding and Exploring</i> . First edition., Make: Community LLC, 2021.
[MkTrig]	Horvath, Joan, and Rich Cameron. <i>Make: Trigonometry: Build your way from triangles to analytic geometry</i> . First edition., Make: Community LLC, 2023.
[PractShopMath]	Begnal, Tom. <i>Practical Shop Math: Simple Solutions to Workshop Fractions, Formulas + Geometric Shapes</i> . Updated edition, Spring House Press, 2018.
[RS274]	Thomas R. Kramer, Frederick M. Proctor, Elena R. Messina. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374 https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3
[SoftwareDesign]	Ousterhout, John K. <i>A Philosophy of Software Design</i> . First Edition., Yaknyam Press, Palo Alto, Ca., 2018

Command Glossary

settool settool(102). 25

setupstock setupstock(200, 100, 8.35, "Top", "Lower-left", 8.35). 23

Index

- addvertex, 70
- ballnose, 32
- beginpolyline, 70
- bowl tool, 32
- closedxfile, 77, 78
- closegcodefile, 77, 78
- closepolyline, 70
- currenttoolnum, 24, 29
- currenttoolnumber, 29
- currenttoolshape, 34
- cut..., 38, 41
- cutarcCC, 43
- cutarcCW, 43
- cutkeyhole toolpath, 50
- cutKHgcdxf, 51
- cutline, 41
- cutrectangleround, 49
- dovetail, 34
- dxfar, 70
- dxfcircle, 70
- dxflin, 70
- dxfpreamble, 77, 78
- dxfpreamble, 70
- dxfwrite, 70
- endmill square, 31
- endmill v, 32
- feed, 38
- flat V, 32
- gcodepreview, 23
 - writeln, 67
- gcp.setupstock, 25
- generatepaths, 19
- init, 23
- initializemachinestate(), 25
- keyhole, 33
- mpx, 24
- mpy, 24
- mpz, 24
- opendxfile, 68
- opengcodefile, 68
- plunge, 38
- previewgcodefile, 78
- rapid..., 38
- rcl, 38
- settool, 29
- setupstock, 25
 - gcodepreview, 25
- setxpos, 25
- setypos, 25
- setzpos, 25
- speed, 38
- stepsizearc, 21
- stepsizeroundover, 21
- stockandtoolpaths, 19
- subroutine
 - gcodepreview, 25
 - writeln, 67
- tapered ball, 32
- threadmill, 33
- tool diameter, 37
- tool radius, 38
- toolchange, 34
- toolpaths, 19
- tpzinc, 24
- writedxDT, 70
- writedxKH, 70
- writedxflgl, 70
- writedxflgsq, 70
- writedxflgV, 70
- writedxfsmb, 70
- writedxfsmsq, 70
- writedxfsmV, 70
- xpos, 24
- ypos, 24
- zpos, 24

Routines

- addvertex, 70
- ballnose, 32
- beginpolyline, 70
- bowl tool, 32
- closedxfile, 77, 78
- closegcodefile, 77, 78
- closepolyline, 70
- currenttoolnumber, 29
- cut..., 38, 41
- cutarcCC, 43
- cutarcCW, 43
- cutkeyhole toolpath, 50
- cutKHgcdxf, 51
- cutline, 41
- cutrectangleround, 49
- dovetail, 34
- dxfar, 70
- dxfcircle, 70
- dxfline, 70
- dxfpreamble, 77, 78
- dxfpreamble, 70
- dxfwrite, 70
- endmill square, 31
- endmill v, 32
- flat V, 32
- gcodepreview, 23, 25
- gcp.setupstock, 25
- init, 23
- initializemachinestate(), 25
- keyhole, 33
- opendxfile, 68
- opengcodefile, 68
- previewgcodefile, 78
- rapid..., 38
- rcl, 38
- settool, 29
- setupstock, 25
- setxpos, 25
- setypos, 25
- setzpos, 25
- stockandtoolpaths, 19
- tapered ball, 32
- threadmill, 33
- tool diameter, 37
- tool radius, 38
- toolchange, 34
- writedxfDT, 70
- writedxfKH, 70
- writedxfgbl, 70
- writedxfgsq, 70
- writedxfV, 70
- writedxfsmbl, 70
- writedxfmsq, 70
- writedxfsmV, 70
- writeln, 67
- xpos, 24
- ypos, 24
- zpos, 24

Variables

currenttoolnum, 24, 29	plunge, 38
currenttoolshape, 34	
feed, 38	speed, 38
	stepsizearc, 21
generatepaths, 19	stepsizeroundover, 21
mpx, 24	
mpy, 24	toolpaths, 19
mpz, 24	tpzinc, 24